Geometric Deep Learning
Grids, Groups, Graphs, Geodesics, and Gauges

Michael M. Bronstein[1], Joan Bruna[2], Taco Cohen[3], Petar Veličković[4]

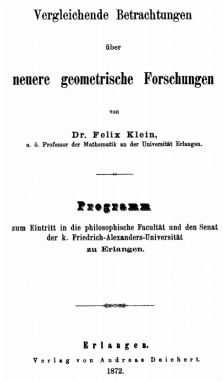
May 4, 2021

[1]Imperial College London / USI IDSIA / Twitter
[2]New York University
[3]Qualcomm AI Research. Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.
[4]DeepMind

Contents

Preface

For nearly two millenia since Euclid's *Elements*, the word 'geometry' has been synonymous with *Euclidean geometry*, as no other types of geometry existed. Euclid's monopoly came to an end in the nineteenth century, with examples of non-Euclidean geometries constructed by Lobachevesky, Bolyai, Gauss, and Riemann. Towards the end of that century, these studies had diverged into disparate fields, with mathematicians and philosophers debating the validity of and relations between these geometries as well as the nature of the "one true geometry".


Vergleichende Betrachtungen
über
neuere geometrische Forschungen
von
Dr. Felix Klein,
o. ö. Professor der Mathematik an der Universität Erlangen.
Programm
zum Eintritt in die philosophische Facultät und den Senat
der k. Friedrich-Alexanders-Universität
zu Erlangen.
Erlangen.
Verlag von Andreas Deichert.
1872.

According to a popular belief, the Erlangen Programme was delivered in Klein's inaugural address in October 1872. Klein indeed gave such a talk (though on December 7 of the same year), but it was for a non-mathematical audience and concerned primarily his ideas of mathematical education. What is now called the 'Erlangen Programme' was actually a research prospectus brochure *Vergleichende Betrachtungen über neuere geometrische Forschungen* ("A comparative review of recent researches in geometry") he prepared as part of his professor appointment. See Tobies (2019).

A way out of this pickle was shown by a young mathematician Felix Klein, appointed in 1872 as professor in the small Bavarian University of Erlangen. In a research prospectus, which entered the annals of mathematics as the *Erlangen Programme*, Klein proposed approaching geometry as the study of *invariants*, i.e. properties unchanged under some class of transformations, called the *symmetries* of the geometry. This approach created clarity by showing that various geometries known at the time could be defined by an appropriate choice of symmetry transformations, formalized using the language of group theory. For instance, Euclidean geometry is concerned with lengths and angles, because these properties are preserved by the group of Euclidean transformations (rotations and translations), while affine geometry studies parallelism, which is preserved by the group of affine transformations. The relation between these geometries is immediately apparent when considering the respective groups, because the Euclidean group is a subgroup of the affine group, which in turn is a subgroup of the group of projective transformations.

The impact of the Erlangen Programme on geometry was very profound. Furthermore, it spilled to other fields, especially physics, where symmetry principles allowed to derive conservation laws from first principles of symmetry (an astonishing result known as Noether's Theorem), and even enabled the classification of elementary particles as irreducible representations of the symmetry group. *Category theory*, now pervasive in pure mathematics, can be "regarded as a continuation of the Klein Erlangen Programme, in the sense that a geometrical space with its group of transformations is generalized to a category with its algebra of mappings", in the words of its creators Samuel Eilenber and Saunders Mac Lane.

Yes, categories, morphisms, natural transformations, all seem to fit nicely to the topic

See Marquis (2009).

At the time of writing, the state of the field of deep learning is somewhat

reminiscent of the field of geometry in the nineteenth century. There is a veritable zoo of neural network architectures for various kinds of data, but few unifying principles. As in times past, this makes it difficult to understand the relations between various methods, inevitably resulting in the reinvention and re-branding of the same concepts in different application domains. For a novice trying to learn the field, absorbing the sheer volume of redundant ideas is a true nightmare.

In this text, we make a modest attempt to apply the Erlangen Programme mindset to the domain of deep learning, with the ultimate goal of obtaining a systematisation of this field and 'connecting the dots'. We call this geometrisation attempt 'Geometric Deep Learning', and true to the spirit of Felix Klein, propose to derive different inductive biases and network architectures implementing them from first principles of symmetry and invariance. In particular, we focus on a large class of neural networks designed for analysing unstructured sets, grids, graphs, and manifolds, and show that they can be understood in a unified manner as methods that respect the structure and symmetries of these domains.

We believe this text would appeal to a broad audience of deep learning researchers, practitioners, and enthusiasts. A novice may use it as an overview and introduction to Geometric Deep Learning. A seasoned deep learning expert may discover new ways of deriving familiar architectures from basic principles and perhaps some surprising connections. Practitioners may get new insights on how to solve problems in their respective fields.

With such a fast-paced field as modern machine learning, the risk of writing a text like this is that it becomes obsolete and irrelevant before it sees the light of day. Having focused on foundations, our hope is that the key concepts we discuss will transcend their specific realisations — or, as Claude Adrien Helvétius put it, *"la connaissance de certains principes supplée facilement à la connoissance de certains faits."*

"The knowledge of certain principles easily compensates the lack of knowledge of certain facts." (Helvétius, 1759)

Notation

Ω, u	Domain, point on domain
$x(u) \in \mathcal{X}(\Omega, \mathcal{C})$	Signal on the domain of the form $x : \Omega \to \mathcal{C}$
$f(x) \in \mathcal{F}(\mathcal{X}(\Omega))$	Functions on signals on the domain of the form $f : \mathcal{X}(\Omega) \to \mathcal{Y}$
$\mathfrak{G}, \mathfrak{g}$	Group, element of the group
$\mathfrak{g}.u, \rho(\mathfrak{g})$	Group action, group representation
$\mathbf{X} \in \mathcal{C}^{\lvert\Omega\rvert \times s}$	Matrix representing a signal on a discrete domain
$\mathbf{x}_u \in \mathcal{C}^s$	Vector representing a discrete domain signal $\mathbf{X}$ on element $u \in \Omega$
$x_{uj} \in \mathcal{C}$	Scalar representing the jth component of a discrete domain signal $\mathbf{X}$ on element $u \in \Omega$
$\mathbf{F}(\mathbf{X})$	Function on discrete domain signals that returns another discrete domain signal, as a matrix
$\tau : \Omega \to \Omega$	Automorphism of the domain
$\eta : \Omega \to \Omega'$	Isomorphism between two different domains
$\sigma : \mathcal{C} \to \mathcal{C}'$	Activation function (point-wise non-linearity)
$G = (\mathcal{V}, \mathcal{E})$	Graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$
$\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$	Mesh with nodes $\mathcal{V}$, edges $\mathcal{E}$, and faces $\mathcal{F}$
$x \star \theta$	Convolution with filter θ
S_v	Shift operator
φ_i	Basis function
$T_u\Omega, T\Omega$	Tangent space at u, tangent bundle
$X \in T_u\Omega$	Tangent vector
$g_u(X, Y) = \langle X, Y \rangle_u$	Riemannian metric
$\ell(\gamma), \ell_{uv}$	Length of a curve γ, discrete metric on edge (u, v)

1 Introduction

The last decade has witnessed an experimental revolution in data science and machine learning, epitomised by deep learning methods. Indeed, many high-dimensional learning tasks previously thought to be beyond reach – such as computer vision, playing Go, or protein folding – are in fact feasible with appropriate computational scale. Remarkably, the essence of deep learning is built from two simple algorithmic principles: first, the notion of representation or *feature learning*, whereby adapted, often hierarchical, features capture the appropriate notion of regularity for each task, and second, learning by local gradient-descent, typically implemented as *backpropagation*.

While learning generic functions in high dimensions is a cursed estimation problem, most tasks of interest are not generic, and come with essential pre-defined regularities arising from the underlying low-dimensionality and structure of the physical world. This text is concerned with exposing these regularities through unified geometric principles that can be applied throughout a wide spectrum of applications.

Isn't this exploiting domain knowledge of the system ?

Exploiting the known symmetries of a large system is a powerful and classical remedy against the curse of dimensionality, and forms the basis of most physical theories. Deep learning systems are no exception, and since the early days researchers have adapted neural networks to exploit the low-dimensional geometry arising from physical measurements, e.g. grids in images, sequences in time-series, or position and momentum in molecules, and their associated symmetries, such as translation or rotation. Throughout our exposition, we will describe these models, as well as many others, as natural instances of the same underlying principle of geometric regularity.

exploit low dimensional symmetry in deep learning systems

the question is "can we unify this principle across all geometries" ? find proper abstractions ?

Such a 'geometric unification' endeavour in the spirit of the Erlangen Program serves a dual purpose: on one hand, it provides a common mathematical framework to study the most successful neural network architectures, such as CNNs, RNNs, GNNs, and Transformers. On the other, it gives a constructive procedure to incorporate prior physical knowledge into neural architectures and provide principled way to build future architectures yet to be invented.

Generativity of the 'geometric unification' principle

Before proceeding, it is worth noting that our work concerns *representation learning architectures* and exploiting the symmetries of data therein. The many exciting *pipelines* where such representations may be used (such as

The core generalizing principle has 2 arms here : representation learning architectures + exploit the symmetries. And the attempt will be to establish all the current architectures like CNN, RNN, GAN, VAE etc. as concrete instances of this abstraction.

self-supervised learning, generative modelling, or reinforcement learning) are *not* our central focus. Hence, we will not review in depth influential neural pipelines such as variational autoencoders (Kingma and Welling, 2013), generative adversarial networks (Goodfellow et al., 2014), normalising flows (Rezende and Mohamed, 2015), deep Q-networks (Mnih et al., 2015), proximal policy optimisation (Schulman et al., 2017), or deep mutual information maximisation (Hjelm et al., 2019). That being said, we believe that the principles we will focus on are of significant importance in all of these areas.

The same applies for techniques used for *optimising* or *regularising* our architectures, such as Adam (Kingma and Ba, 2014), dropout (Srivastava et al., 2014) or batch normalisation (Ioffe and Szegedy, 2015).

Further, while we have attempted to cast a reasonably wide net in order to illustrate the power of our geometric blueprint, our work does not attempt to accurately summarise the *entire* existing wealth of research on Geometric Deep Learning. Rather, we study several well-known architectures in-depth in order to demonstrate the principles and ground them in existing research, with the hope that we have left sufficient references for the reader to meaningfully apply these principles to any future geometric deep architecture they encounter or devise.

2 Learning in High Dimensions

Supervised machine learning, in its simplest formalisation, considers a set of N observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ drawn *i.i.d.* from an underlying data distribution P defined over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are respectively the data and the label domains. The defining feature in this setup is that $\mathcal{X}$ is a *high-dimensional space*: one typically assumes $\mathcal{X} = \mathbb{R}^d$ to be a Euclidean space of large dimension d.

Let us further assume that the labels y are generated by an unknown function f, such that $y_i = f(x_i)$, and the learning problem reduces to estimating the function f using a parametrised function class $\mathcal{F} = \{f_{\boldsymbol{\theta} \in \Theta}\}$. Neural networks are a common realisation of such parametric function classes, in which case $\boldsymbol{\theta} \in \Theta$ corresponds to the network weights. In this idealised setup, there is no noise in the labels, and modern deep learning systems typically operate in the so-called *interpolating regime*, where the estimated $\hat{f} \in \mathcal{F}$ satisfies $\hat{f}(x_i) = f(x_i)$ for all $i = 1, \ldots, N$. The performance of a learning algorithm is measured in terms of the *expected performance* on new samples drawn from

Statistical learning theory is concerned with more refined notions of generalisation based on *concentration inequalities*; we will review some of these in future work.

P, using some *loss* $L(\cdot,\cdot)$

$$\mathcal{R}(\tilde{f}) := \mathbb{E}_P \; L(\tilde{f}(x), f(x)),$$

with the squared-loss $L(y,y') = \frac{1}{2}|y-y'|^2$ being among the most commonly used ones.

A successful learning scheme thus needs to encode the appropriate notion of regularity or *inductive bias* for f, imposed through the construction of the function class $\mathcal{F}$ and the use of *regularisation*. We briefly introduce this concept in the following section.

2.1 Inductive Bias via Function Regularity

Modern machine learning operates with large, high-quality datasets, which, together with appropriate computational resources, motivate the design of rich function classes $\mathcal{F}$ with the capacity to interpolate such large data. This mindset plays well with neural networks, since even the simplest choices of architecture yields a *dense* class of functions. The capacity to approximate almost arbitrary functions is the subject of various *Universal Approximation Theorems*; several such results were proved and popularised in the 1990s by applied mathematicians and computer scientists (see e.g. Cybenko (1989); Hornik (1991); Barron (1993); Leshno et al. (1993); Maiorov (1999); Pinkus (1999)).

A set $\mathcal{A} \subset \mathcal{X}$ is said to be *dense* in $\mathcal{X}$ if its closure $\mathcal{A} \cup \{\lim_{i\to\infty} a_i : a_i \in \mathcal{A}\} = \mathcal{X}$. This implies that any point in $\mathcal{X}$ is arbitrarily close to a point in $\mathcal{A}$. A typical Universal Approximation result shows that the class of functions represented e.g. by a two-layer perceptron, $f(\mathbf{x}) = \mathbf{c}^\top \mathrm{sign}(\mathbf{A}\mathbf{x}+\mathbf{b})$ is dense in the space of continuous functions on $\mathbb{R}^d$.

Universal Approximation, however, does not imply an *absence* of inductive bias. Given a hypothesis space $\mathcal{F}$ with universal approximation, we can define a complexity measure $c : \mathcal{F} \to \mathbb{R}_+$ and redefine our interpolation problem as

$$\tilde{f} \in \arg\min_{g\in\mathcal{F}} c(g) \quad \text{s.t.} \quad g(x_i) = f(x_i) \quad \text{for } \; i = 1,\dots,N,$$

i.e., we are looking for the most regular functions within our hypothesis class. For standard function spaces, this complexity measure can be defined as a *norm*, making $\mathcal{F}$ a *Banach space* and allowing to leverage a plethora of theoretical results in functional analysis. In low dimensions, splines are a workhorse for function approximation. They can be formulated as above, with a norm capturing the classical notion of smoothness, such as the squared-norm of second-derivatives $\int_{-\infty}^{+\infty} |f''(x)|^2 \mathrm{d}x$ for cubic splines.

Is this done through regularization in neural networks ?

Informally, a norm $\|x\|$ can be regarded as a "length" of vector x. A *Banach space* is a complete vector space equipped with a norm.

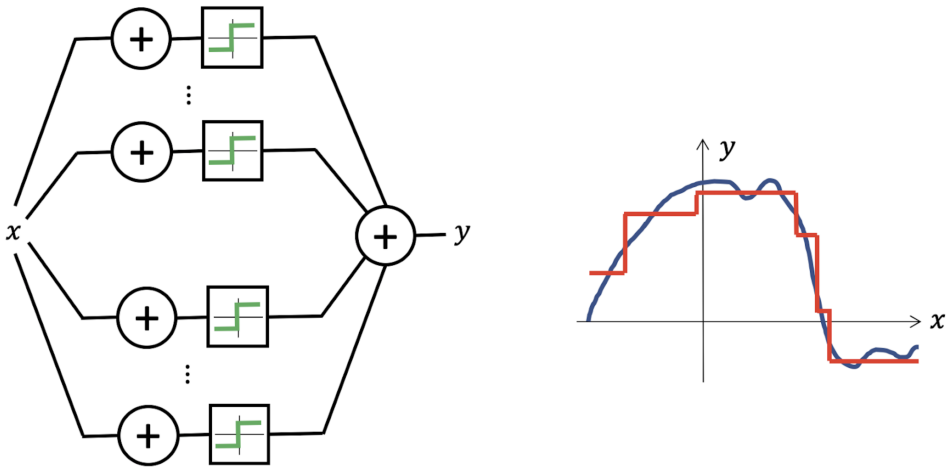


Figure 1: Multilayer Perceptrons (Rosenblatt, 1958), the simplest feed-forward neural networks, are universal approximators: with just one hidden layer, they can represent combinations of step functions, allowing to approximate any continuous function with arbitrary precision.

In the case of neural networks, the complexity measure c can be expressed in terms of the network weights, i.e. $c(f_{\boldsymbol{\theta}}) = c(\boldsymbol{\theta})$. The L_2-norm of the network weights, known as *weight decay*, or the so-called *path-norm* (Neyshabur et al., 2015) are popular choices in deep learning literature. From a Bayesian perspective, such complexity measures can also be interpreted as the negative log of the prior for the function of interest. More generally, this complexity can be enforced *explicitly* by incorporating it into the empirical loss (resulting in the so-called Structural Risk Minimisation), or *implicitly*, as a result of a certain optimisation scheme. For example, it is well-known that gradient-descent on an under-determined least-squares objective will choose interpolating solutions with minimal L_2 norm. The extension of such implicit regularisation results to modern neural networks is the subject of current studies (see e.g. Blanc et al. (2020); Shamir and Vardi (2020); Razin and Cohen (2020); Gunasekar et al. (2017)). All in all, a natural question arises: how to define effective priors that capture the expected regularities and complexities of real-world prediction tasks?

initial weights as priors ?

explicit versus implicit regularization

weight decay = explicit regularization
optimization scheme = implicit regularization

2.2 The Curse of Dimensionality

While interpolation in low-dimensions (with $d = 1, 2$ or 3) is a classic signal processing task with very precise mathematical control of estimation errors using increasingly sophisticated regularity classes (such as spline interpolants, wavelets, curvelets, or ridgelets), the situation for high-dimensional problems is entirely different.

In order to convey the essence of the idea, let us consider a classical notion of regularity that can be easily extended to high dimensions: 1-Lipschitz-functions $f : \mathcal{X} \to \mathbb{R}$, i.e. functions satisfying $|f(x) - f(x')| \leq \|x - x'\|$ for all $x, x' \in \mathcal{X}$. This hypothesis only asks the target function to be *locally* smooth, i.e., if we perturb the input x slightly (as measured by the norm $\|x - x'\|$), the output $f(x)$ is not allowed to change much. If our only knowledge of the target function f is that it is 1-Lipschitz, how many observations do we expect to require to ensure that our estimate $\tilde{f}$ will be close to f? Figure 2 reveals that the general answer is necessarily exponential in the dimension d, signaling that the Lipschitz class grows 'too quickly' as the input dimension increases: in many applications with even modest dimension d, the number of samples would be bigger than the number of atoms in the universe. The situation is not better if one replaces the Lipschitz class by a global smoothness hypothesis, such as the Sobolev Class $\mathcal{H}^s(\Omega_d)$. Indeed, classic results (Tsybakov, 2008) establish a minimax rate of approximation and learning for the Sobolev class of the order $\epsilon^{-d/s}$, showing that the extra smoothness assumptions on f only improve the statistical picture when $s \propto d$, an unrealistic assumption in practice.

curse of dimensionality that says that the number of samples grows exponentially to the dimension even for Lipschitz classes of functions

A function f is in the *Sobolev class* $\mathcal{H}^s(\Omega_d)$ if $f \in L^2(\Omega_d)$ and the generalised s-th order derivative is square-integrable: $\int |\omega|^{2s+1} |\hat{f}(\omega)|^2 d\omega < \infty$, where $\hat{f}$ is the Fourier transform of f; see Section 4.2.

Fully-connected neural networks define function spaces that enable more flexible notions of regularity, obtained by considering complexity functions c on their weights. In particular, by choosing a sparsity-promoting regularisation, they have the ability to break this curse of dimensionality (Bach, 2017). However, this comes at the expense of making strong assumptions on the nature of the target function f, such as that f depends on a collection of low-dimensional projections of the input (see Figure 3). In most real-world applications (such as computer vision, speech analysis, physics, or chemistry), functions of interest tend to exhibits complex long-range correlations that cannot be expressed with low-dimensional projections (Figure 3), making this hypothesis unrealistic. It is thus necessary to define an alternative source of regularity, by exploiting the spatial structure of the physical domain and the geometric priors of f, as we describe in the next Section 3.

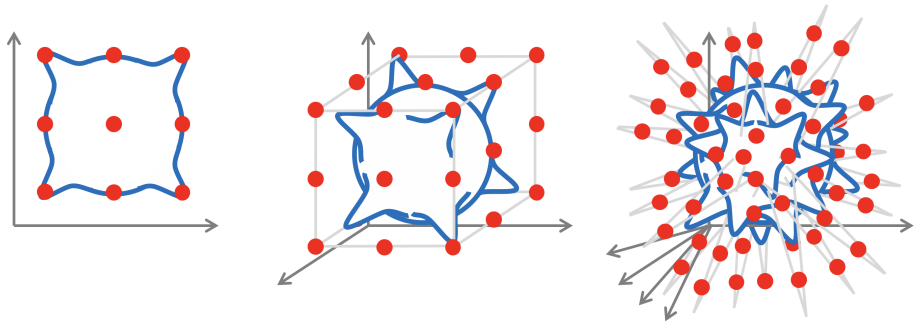


Figure 2: We consider a Lipschitz function $f(x) = \sum_{j=1}^{2^d} z_j \phi(x - x_j)$ where $z_j = \pm 1$, $x_j \in \mathbb{R}^d$ is placed in each quadrant, and ϕ a locally supported Lipschitz 'bump'. Unless we observe the function in most of the 2^d quadrants, we will incur in a constant error in predicting it. This simple geometric argument can be formalised through the notion of *Maximum Discrepancy* (von Luxburg and Bousquet, 2004), defined for the Lipschitz class as $\kappa(d) = \mathbb{E}_{x,x'} \sup_{f \in \mathrm{Lip}(1)} \left| \frac{1}{N} \sum_l f(x_l) - \frac{1}{N} \sum_l f(x'_l) \right| \simeq N^{-1/d}$, which measures the largest expected discrepancy between two independent N-sample expectations. Ensuring that $\kappa(d) \simeq \epsilon$ requires $N = \Theta(\epsilon^{-d})$; the corresponding sample $\{x_l\}_l$ defines an ϵ-net of the domain. For a d-dimensional Euclidean domain of diameter 1, its size grows exponentially as ϵ^{-d}.

3 Geometric Priors

Note: Geometric deep learning is not learning from generic high dimensional data. We consider data that comes from the structure of the underlying domain and on which we can apply principles of symmetry and scale separation, commonly referred to as Geometric Priors

Using geometric priors we hope to reduce the size and complexity of the hypothesis class, thereby reducing the statistical error. This should address the curse of dimensionality.

For more details see AMMI Lecture 3 (Geometric Priors 1) by Taco Cohen (https://youtu.be/fWBrupgU4X8)

Modern data analysis is synonymous with high-dimensional learning. While the simple arguments of Section 2.1 reveal the impossibility of learning from generic high-dimensional data as a result of the curse of dimensionality, there is hope for physically-structured data, where we can employ two fundamental principles: *symmetry* and *scale separation*. In the settings considered in this text, this additional structure will usually come from the structure of the domain underlying the input signals: we will assume that our machine learning system operates on *signals* (functions) on some domain Ω. While in many cases linear combinations of points on Ω is not well-defined, we can linearly combine signals on it, i.e., the space of signals forms a vector space. Moreover, since we can define an inner product between signals, this space is a *Hilbert space*.

In the setting of geometric deep learning usually the data "lives" in the domain and is called the signal

So even though the domain may not have any structure, the signal will have some structure which we can use. In fact it can be shown that the space of signals forms a vector space, in fact Hilbert Space

Ω must be a vector space in order for an expression $\alpha u + \beta v$ to make sense.

and hence can be linearly combined

Ω is the domain of our data. But linear combinations of data may not be well-defined in many cases. Hence we consider that our machine learning system operates on a higher order abstraction that are functions and we call them signals. Signals can be linearly combined and the space of signals form a vector space. Also we can define inner product between signals - hence the space is a Hilbert space

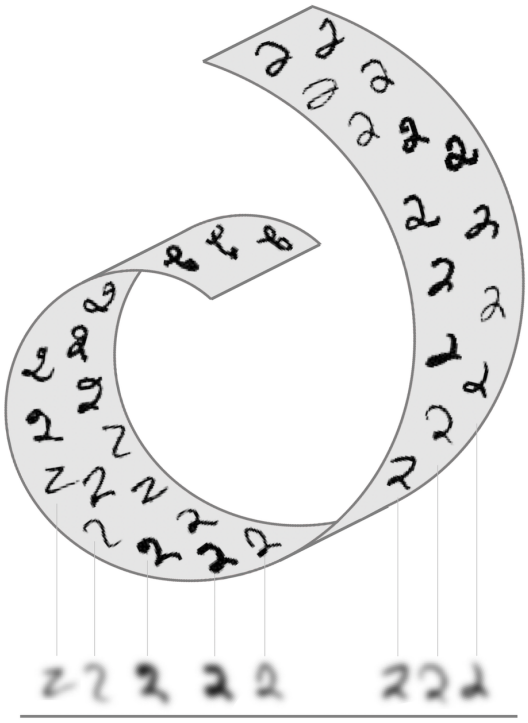


Figure 3: If the unknown function f is presumed to be well approximated as $f(\mathbf{x}) \approx g(\mathbf{A}\mathbf{x})$ for some unknown $\mathbf{A} \in \mathbb{R}^{k\times d}$ with $k \ll d$, then shallow neural networks can capture this inductive bias, see e.g. Bach (2017). In typical applications, such dependency on low-dimensional projections is unrealistic, as illustrated in this example: a low-pass filter projects the input images to a low-dimensional subspace; while it conveys most of the semantics, substantial information is lost.

The space of $\mathcal{C}$-valued signals on Ω (for Ω a set, possibly with additional structure, and $\mathcal{C}$ a vector space, whose dimensions are called *channels*)

$$\mathcal{X}(\Omega, \mathcal{C}) = \{x : \Omega \to \mathcal{C}\} \tag{1}$$

is a function space that has a vector space structure. Addition and scalar multiplication of signals is defined as:

$$(\alpha x + \beta y)(u) = \alpha x(u) + \beta y(u) \quad \text{for all} \quad u \in \Omega,$$

with real scalars α, β. Given an inner product $\langle v, w \rangle_{\mathcal{C}}$ on $\mathcal{C}$ and a measure μ on Ω (with respect to which we can define an integral), we can define an inner product on $\mathcal{X}(\Omega, \mathcal{C})$ as

$$\langle x, y \rangle = \int_{\Omega} \langle x(u),\, y(u) \rangle_{\mathcal{C}} \, \mathrm{d}\mu(u). \tag{2}$$

When Ω has some additional structure, we may further restrict the kinds of signals in $\mathcal{X}(\Omega, \mathcal{C})$. For example, when Ω is a smooth manifold, we may require the signals to be smooth. Whenever possible, we will omit the range $\mathcal{C}$ for brevity.

When the domain Ω is discrete, μ can be chosen as the *counting measure*, in which case the integral becomes a sum. In the following, we will omit the measure and use $\mathrm{d}u$ for brevity.

a concrete example of a signal being defined on a domain Ω, we say the signal "lives" in the domain

As a typical illustration, take $\Omega = \mathbb{Z}_n \times \mathbb{Z}_n$ to be a two-dimensional $n \times n$ grid, x an RGB image (i.e. a signal $x : \Omega \to \mathbb{R}^3$), and f a function (such as a single-layer Perceptron) operating on $3n^2$-dimensional inputs. As we will see in the following with greater detail, the domain Ω is usually endowed with certain geometric structure and symmetries. Scale separation results from our ability to preserve important characteristics of the signal when transferring it onto a coarser version of the domain (in our example, subsampling the image by coarsening the underlying grid).

On the contrary there can be cases where the signal _is_ the domain e.g. graphs without edge or node features. In such cases we use adjacency matrix to form the signal.

symmetry and scale separation

We will show that both principles, to which we will generically refer as *geometric priors*, are prominent in most modern deep learning architectures. In the case of images considered above, geometric priors are built into Convolutional Neural Networks (CNNs) in the form of convolutional filters with shared weights (exploiting translational symmetry) and pooling (exploiting scale separation). Extending these ideas to other domains such as graphs and manifolds and showing how geometric priors emerge from fundamental principles is the main goal of Geometric Deep Learning and the *leitmotif* of our text.

Symmetry and scale separation gives rise to Geometric Priors. In CNN we have geometric priors built in via convolutional filters with shared weights (sharing of weights is possible because of translation symmetry) and pooling (which is possible for scale separation)

3.1 Symmetries, Representations, and Invariance

Informally, a *symmetry* of an object or system is a transformation that leaves a certain property of said object or system unchanged or *invariant*. Such transformations may be either smooth, continuous, or discrete. Symmetries are ubiquitous in many machine learning tasks. For example, in computer vision the object category is unchanged by shifts, so shifts are symmetries in the problem of visual object classification. In computational chemistry, the task of predicting properties of molecules independently of their orientation in space requires *rotational invariance*. Discrete symmetries emerge naturally when describing particle systems where particles do not have canonical ordering and thus can be arbitrarily permuted, as well as in many dynamical systems, via the time-reversal symmetry (such as systems in detailed balance or the Newton's second law of motion). As we will see in Section 4.1, permutation symmetries are also central to the analysis of graph-structured data.

Permutation symmetry is a transformation where permuting the order of the vertices does not change the graph

a typical example of composition of 2 symmetries is Rotation and Translation : Rotate + Translate will have the same effect as Translate + Rotate

Symmetry groups The set of symmetries of an object satisfies a number of properties. First, symmetries may be combined to obtain new symmetries: if $\mathfrak{g}$ and $\mathfrak{h}$ are two symmetries, then their compositions $\mathfrak{g} \circ \mathfrak{h}$ and $\mathfrak{h} \circ \mathfrak{g}$ are also symmetries. The reason is that if both transformations leave the object invariant, then so does the composition of transformations, and hence the composition is also a symmetry. Furthermore, symmetries are always invertible, and the inverse is also a symmetry. This shows that the collection of all symmetries form an algebraic object known as a *group*. Since these objects will be a centerpiece of the mathematical model of Geometric Deep Learning, they deserve a formal definition and detailed discussion:

Symmetries can be composed and the set of all symmetries of an object form a group as described here. We talk about symmetry groups.

We will follow the juxtaposition notation convention used in group theory, $\mathfrak{g} \circ \mathfrak{h} = \mathfrak{gh}$, which should be read right-to-left: we first apply $\mathfrak{h}$ and then $\mathfrak{g}$. The order is important, as in many cases symmetries are non-commutative. Readers familiar with Lie groups might be disturbed by our choice to use the Fraktur font to denote group elements, as it is a common notation of Lie algebras.

a symmetry is a transformation that preserves some property or structure, and the set of all such transformations for a given structure forms a symmetry group. - Page 21, Section 3.2

A *group* is a set $\mathfrak{G}$ along with a binary operation $\circ : \mathfrak{G} \times \mathfrak{G} \to \mathfrak{G}$ called *composition* (for brevity, denoted by juxtaposition $\mathfrak{g} \circ \mathfrak{h} = \mathfrak{g}\mathfrak{h}$) satisfying the following axioms:

Associativity: $(\mathfrak{g}\mathfrak{h})\mathfrak{k} = \mathfrak{g}(\mathfrak{h}\mathfrak{k})$ for all $\mathfrak{g}, \mathfrak{h}, \mathfrak{k} \in \mathfrak{G}$.

Identity: there exists a unique $\mathfrak{e} \in \mathfrak{G}$ satisfying $\mathfrak{e}\mathfrak{g} = \mathfrak{g}\mathfrak{e} = \mathfrak{g}$ for all $\mathfrak{g} \in \mathfrak{G}$.

Inverse: For each $\mathfrak{g} \in \mathfrak{G}$ there is a unique inverse $\mathfrak{g}^{-1} \in \mathfrak{G}$ such that $\mathfrak{g}\mathfrak{g}^{-1} = \mathfrak{g}^{-1}\mathfrak{g} = \mathfrak{e}$.

Closure: The group is closed under composition, i.e., for every $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$, we have $\mathfrak{g}\mathfrak{h} \in \mathfrak{G}$.

Note that *commutativity* is not part of this definition, i.e. we may have $\mathfrak{g}\mathfrak{h} \neq \mathfrak{h}\mathfrak{g}$. Groups for which $\mathfrak{g}\mathfrak{h} = \mathfrak{h}\mathfrak{g}$ for all $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$ are called commutative or *Abelian*.

After the Norwegian mathematician Niels Henrik Abel (1802–1829).

Though some groups can be very large and even infinite, they often arise from compositions of just a few elements, called *group generators*. Formally, $\mathfrak{G}$ is said to be *generated* by a subset $S \subseteq \mathfrak{G}$ (called the group *generator*) if every element $\mathfrak{g} \in \mathfrak{G}$ can be written as a finite composition of the elements of S and their inverses. For instance, the symmetry group of an equilateral triangle (dihedral group D_3) is generated by a 60° rotation and a reflection (Figure 4). The 1D *translation group*, which we will discuss in detail in the following, is generated by infinitesimal displacements; this is an example of a *Lie group* of differentiable symmetries.

Lie groups have a differentiable manifold structure. One such example that we will study in Section 4.3 is the special orthogonal group $\mathrm{SO}(3)$, which is a 3-dimensional manifold.

rotational symmetry, reflection symmetry together define a symmetry group

Here are the objects belonging to this group :-

- a triangle can belong to this symmetry group since triangles have rotational and reflection symmetries

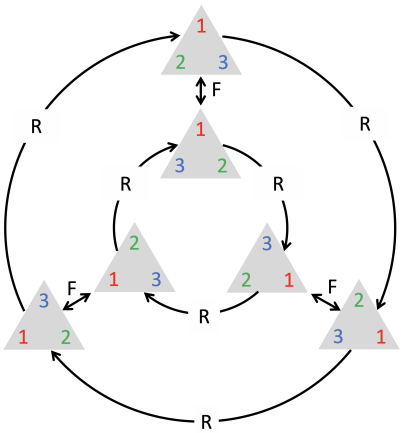
- also a sequence of 3 elements will belong to this same symmetry group as they can be permuted in any way using rotation and reflection

Note that here we have defined a group as an abstract object, without saying what the group elements *are* (e.g. transformations of some domain), only how they *compose*. Hence, very different kinds of objects may have the same symmetry group. For instance, the aforementioned group of rotational and reflection symmetries of a triangle is the same as the group of permutations of a sequence of three elements (we can permute the corners in the triangle in any way using a rotation and reflection – see Figure 4).

The diagram shown in Figure 4 (where each node is associated with a group element, and each arrow with a generator), is known as the *Cayley diagram*.

By groups we mean symmetry groups

Group Actions and Group Representations Rather than considering groups as abstract entities, we are mostly interested in how groups *act* on data. Since we assumed that there is some domain Ω underlying our data, we will study how the group acts on Ω (e.g. translation of points of the plane), and from there obtain actions of the same group on the space of signals $\mathcal{X}(\Omega)$ (e.g. translations of planar images and feature maps).



	id	R	R^2	F	FR	FR^2
id	id	R	R^2	F	FR	FR^2
R	R	R^2	id	RF	RFR	RFR^2
R^2	R^2	id	R	R^2F	R^2FR	R^2FR^2
F	F	FR	FR^2	id	R	FR
FR	FR	FR^2	F	FRF	FRFR	$FRFR^2$
FR^2	FR^2	F	FR	FR^2F	FR^2FR	FR^2FR^2

Figure 4: Left: an equilateral triangle with corners labelled by $1, 2, 3$, and all possible rotations and reflections of the triangle. The group D_3 of rotation/reflection symmetries of the triangle is generated by only two elements (rotation by $60°$ R and reflection F) and is the same as the group Σ_3 of permutations of three elements. Right: the multiplication table of the group D_3. The element in the row $\mathfrak{g}$ and column $\mathfrak{h}$ corresponds to the element $\mathfrak{gh}$.

Technically, what we define here is a *left* group action.

A *group action* of $\mathfrak{G}$ on a set Ω is defined as a mapping $(\mathfrak{g}, u) \mapsto \mathfrak{g}.u$ associating a group element $\mathfrak{g} \in \mathfrak{G}$ and a point $u \in \Omega$ with some other point on Ω in a way that is compatible with the group operations, i.e., $\mathfrak{g}.(\mathfrak{h}.u) = (\mathfrak{gh}).u$ for all $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$ and $u \in \Omega$. We shall see numerous instances of group actions in the following sections. For example, in the plane the *Euclidean group* $\mathrm{E}(2)$ is the group of transformations of $\mathbb{R}^2$ that preserves Euclidean distances, and consists of translations, rotations, and reflections. The same group, however, can also act on the space of *images* on the plane (by translating, rotating and flipping the grid of pixels), as well as on the representation spaces learned by a neural network. More precisely, if we have a group $\mathfrak{G}$ acting on Ω, we automatically obtain an action of $\mathfrak{G}$ on the space $\mathcal{X}(\Omega)$:

Distance-preserving transformations are called *isometries*. According to Klein's Erlangen Programme, the classical Euclidean geometry arises from this group.

In this example, translation, rotation and reflection define the symmetry group. The objects belonging to this group are:

- Euclidean group
- images on the plane
- representation spaces learned by a NN

$$(\mathfrak{g}.x)(u) = x(\mathfrak{g}^{-1}u). \tag{3}$$

Action of G on space X(Ω) → ← *New signal after applying the transformation (inverse)*

(linear action g.x, linear in x)

Due to the inverse on $\mathfrak{g}$, this is indeed a valid group action, in that we have $(\mathfrak{g}.(\mathfrak{h}.x))(u) = ((\mathfrak{gh}).x)(u)$.

The most important kind of group actions, which we will encounter repeatedly throughout this text, are *linear* group actions, also known as *group representations*. The action on signals in equation (3) is indeed linear, in the

sense that

Linear combination of signals .. remember signals form a vector space

$$\mathfrak{g}.(\alpha x + \beta x') = \alpha(\mathfrak{g}.x) + \beta(\mathfrak{g}.x')$$

Intuition: when x is a planar image, g.x is the action which maps the image as per the translation defined by g. We could also interpret this as a map from G to the matrix that defines the image in the translated coordinates

for any scalars α, β and signals $x, x' \in \mathcal{X}(\Omega)$. We can describe linear actions either as maps $(\mathfrak{g}, x) \mapsto \mathfrak{g}.x$ that are linear in x, or equivalently, by currying, as a map $\rho : \mathfrak{G} \to \mathbb{R}^{n\times n}$ that assigns to each group element $\mathfrak{g}$ an (invertible) matrix $\rho(\mathfrak{g})$. The dimension n of the matrix is in general arbitrary and not necessarily related to the dimensionality of the group or the dimensionality of Ω, but in applications to deep learning n will usually be the dimensionality of the feature space on which the group acts. For instance, we may have the group of 2D translations acting on a space of images with n pixels.

Invertible matrix is one which is its own inverse

When Ω is infinte, the space of signals $\mathcal{X}(\Omega)$ is infinite dimensional, in which case $\rho(\mathfrak{g})$ is a linear operator on this space, rather than a finite dimensional matrix. In practice, one must always discretise to a finite grid, though.

As with a general group action, the assignment of matrices to group elements should be compatible with the group action. More specifically, the matrix representing a composite group element $\mathfrak{gh}$ should equal the matrix product of the representation of $\mathfrak{g}$ and $\mathfrak{h}$:

> A n-dimensional real *representation* of a group $\mathfrak{G}$ is a map $\rho : \mathfrak{G} \to \mathbb{R}^{n\times n}$, assigning to each $\mathfrak{g} \in \mathfrak{G}$ an *invertible* matrix $\rho(\mathfrak{g})$, and satisfying the condition $\rho(\mathfrak{gh}) = \rho(\mathfrak{g})\rho(\mathfrak{h})$ for all $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$. A representation is called *unitary* or *orthogonal* if the matrix $\rho(\mathfrak{g})$ is unitary or orthogonal for all $\mathfrak{g} \in \mathfrak{G}$.

Similarly, a complex representation is a map $\rho : \mathfrak{G} \to \mathbb{C}^{n\times n}$ satisfying the same equation.

Written in the language of group representations, the action of $\mathfrak{G}$ on signals $x \in \mathcal{X}(\Omega)$ is defined as $\rho(\mathfrak{g})x(u) = x(\mathfrak{g}^{-1}u)$. We again verify that

$$(\rho(\mathfrak{g})(\rho(\mathfrak{h})x))(u) = (\rho(\mathfrak{gh})x)(u).$$

ρ (g) is an invertible matrix which acts on the signal x(u)

alternative representation of group actions than the one used in (3) above

translation => shift

the input image x is not just a d-dimensional vector, but a signal defined on some domain Ω, which in this case is a two-dimensional grid. The structure of the domain is captured by a symmetry group $\mathfrak{G}$ — the group of 2D translations in our example — which acts on the points on the domain. In the space of signals $\mathcal{X}(\Omega)$, the group actions (elements of the group, $\mathfrak{g}\in\mathfrak{G}$) on the underlying domain are manifested through what is called the group representation $\rho(\mathfrak{g})$ — in our case, it is simply the shift operator, a d×d matrix that acts on a d-dimensional vector

Invariant and Equivariant functions The symmetry of the domain Ω underlying the signals $\mathcal{X}(\Omega)$ imposes structure on the function f defined on such signals. It turns out to be a powerful inductive bias, improving learning efficiency by reducing the space of possible interpolants, $\mathcal{F}(\mathcal{X}(\Omega))$, to those which satisfy the symmetry priors. Two important cases we will be exploring in this text are *invariant* and *equivariant* functions.

very important concept to understand that the group symmetry gives a powerful inductive bias and reduces the space of applicable functions thereby making learning more efficient

In general, f depends both on the signal an the domain, i.e., $\mathcal{F}(\mathcal{X}(\Omega), \Omega)$. We will often omit the latter dependency for brevity.

> A function $f : \mathcal{X}(\Omega) \to \mathcal{Y}$ is $\mathfrak{G}$-*invariant* if $f(\rho(\mathfrak{g})x) = f(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega)$, i.e., its output is unaffected by the group action on the input.

Geometric priors turns out to be a powerful inductive bias (implicit regularization ?) that improves learning.

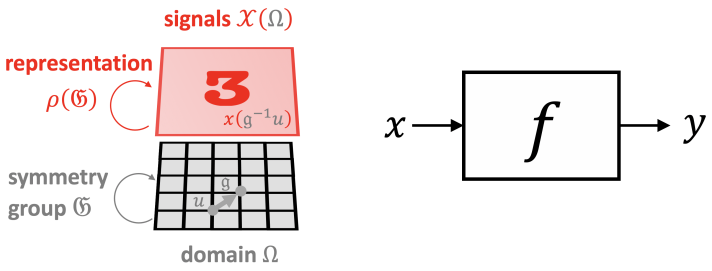


Figure 5: Three spaces of interest in Geometric Deep Learning: the (physical) *domain* Ω, the space of *signals* $\mathcal{X}(\Omega)$, and the *hypothesis* class $\mathcal{F}(\mathcal{X}(\Omega))$. Symmetries of the domain Ω (captured by the group $\mathfrak{G}$) act on signals $x \in \mathcal{X}(\Omega)$ through group representations $\rho(\mathfrak{g})$, imposing structure on the functions $f \in \mathcal{F}(\mathcal{X}(\Omega))$ acting on such signals.

Note that signal processing books routinely use the term 'shift-invariance' referring to shift-equivariance, e.g. Linear Shift-invariant Systems.

A classical example of invariance is *shift-invariance*, arising in computer vision and pattern recognition applications such as image classification. The function f in this case (typically implemented as a Convolutional Neural Network) inputs an image and outputs the probability of the image to contain an object from a certain class (e.g. cat or dog). It is often reasonably assumed that the classification result should not be affected by the position of the object in the image, i.e., the function f must be shift-invariant. Multi-layer Perceptrons, which can approximate any smooth function, do not have this property – one of the reasons why early attempts to apply these architectures to problems of pattern recognition in the 1970s failed. The development of neural network architectures with local weight sharing, as epitomised by Convolutional Neural Networks, was, among other reasons, motivated by the need for shift-invariant object classification.

shift invariance serving as the geometric prior and acts as the induction bias with CNNs

If we however take a closer look at the convolutional layers of CNNs, we will find that they are not shift-invariant but *shift-equivariant*: in other words, a shift of the input to a convolutional layer produces a shift in the output feature maps by the same amount.

More generally, we might have $f : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$ with input and output spaces having different domains Ω, Ω' and representations ρ, ρ' of the same group $\mathfrak{G}$. In this case, equivariance is defined as $f(\rho(\mathfrak{g})x) = \rho'(\mathfrak{g})f(x)$.

A function $f : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega)$ is $\mathfrak{G}$-*equivariant* if $f(\rho(\mathfrak{g})x) = \rho(\mathfrak{g})f(x)$ for all $\mathfrak{g} \in \mathfrak{G}$, i.e., group action on the input affects the output in the same way.

Resorting again to computer vision, a prototypical application requiring

shift-equivariance is image segmentation, where the output of f is a pixel-wise image mask. Obviously, the segmentation mask must follow shifts in the input image. In this example, the domains of the input and output are the same, but since the input has three color channels while the output has one channel per class, the representations $(\rho, \mathcal{X}(\Omega, \mathcal{C}))$ and $(\rho', \mathcal{X}(\Omega, \mathcal{C}'))$ are somewhat different.

general blueprint of deep learning architectures

However, even the previous use case of image classification is usually implemented as a sequence of convolutional (shift-equivariant) layers, followed by global pooling (which is shift-invariant). As we will see in Section 3.5, this is a general blueprint of a majority of deep learning architectures, including CNNs and Graph Neural Networks (GNNs).

3.2 Isomorphisms and Automorphisms

very nice definition of symmetry and symmetry group

Subgroups and Levels of structure As mentioned before, a symmetry is a transformation that preserves some property or structure, and the set of all such transformations for a given structure forms a symmetry group. It happens often that there is not one but multiple structures of interest, and so we can consider several *levels of structure* on our domain Ω. Hence, what counts as a symmetry depends on the structure under consideration, but in all cases a symmetry is an invertible map that respects this structure.

Invertible and structure-preserving maps between different objects often go under the generic name of *isomorphisms* (Greek for 'equal shape'). An isomorphism from an object to itself is called an *automorphism*, or symmetry.

On the most basic level, the domain Ω is a *set*, which has a minimal amount of structure: all we can say is that the set has some *cardinality*. Self-maps that preserve this structure are *bijections* (invertible maps), which we may consider as set-level symmetries. One can easily verify that this is a group by checking the axioms: a compositions of two bijections is also a bijection (closure), the associativity stems from the associativity of the function composition, the map $\tau(u) = u$ is the identity element, and for every τ the inverse exists by definition, satisfying $(\tau \circ \tau^{-1})(u) = (\tau^{-1} \circ \tau)(u) = u$.

For a finite set, the cardinality is the number of elements ('size') of the set, and for infinite sets the cardinality indicates different kinds of infinities, such as the countable infinity of the natural numbers, or the uncountable infinity of the continuum $\mathbb{R}$.

Depending on the application, there may be further levels of structure. For instance, if Ω is a topological space, we can consider maps that preserve *continuity*: such maps are called *homeomorphisms* and in addition to simple bijections between sets, are also continuous and have continuous inverse. Intuitively, continuous functions are well-behaved and map points in a neighbourhood (open set) around a point u to a neighbourhood around $\tau(u)$.

Every differentiable function is continuous. If the map is continuously differentiable 'sufficiently many times', it is said to be *smooth*.

One can further demand that the map and its inverse are (continuously) *differentiable*, i.e., the map and its inverse have a derivative at every point (and the derivative is also continuous). This requires further differentiable structure that comes with differentiable manifolds, where such maps are called *diffeomorphisms* and denoted by $\mathrm{Diff}(\Omega)$. Additional examples of structures we will encounter include *distances* or *metrics* (maps preserving them are called *isometries*) or *orientation* (to the best of our knowledge, orientation-preserving maps do not have a common Greek name).

A *metric* or *distance* is a function $d : \Omega \times \Omega \to [0, \infty)$ satisfying for all $u, v, w \in \Omega$:

Identity of indiscernibles: $d(u, v) = 0$ iff $u = v$.

Symmetry: $d(u, v) = d(v, u)$.

Triangle inequality: $d(u, v) \leq d(u, w) + d(w, v)$.

A space equipped with a metric (Ω, d) is called a *metric space*.

The right level of structure to consider depends on the problem. For example, when segmenting histopathology slide images, we may wish to consider flipped versions of an image as equivalent (as the sample can be flipped when put under the microscope), but if we are trying to classify road signs, we would only want to consider orientation-preserving transformations as symmetries (since reflections could change the meaning of the sign).

More structure => More constraints => Smaller symmetry groups => Stronger Inductive Bias => Better Learning (?)

As we add levels of structure to be preserved, the symmetry group will get smaller. Indeed, adding structure is equivalent to selecting a *subgroup*, which is a subset of the larger group that satisfies the axioms of a group by itself:

Let $(\mathfrak{G}, \circ)$ be a group and $\mathfrak{H} \subseteq \mathfrak{G}$ a subset. $\mathfrak{H}$ is said to be a *subgroup* of $\mathfrak{G}$ if $(\mathfrak{H}, \circ)$ constitutes a group with the same operation.

For instance, the group of Euclidean isometries $\mathrm{E}(2)$ is a subgroup of the group of planar diffeomorphisms $\mathrm{Diff}(2)$, and in turn the group of orientation-preserving isometries $\mathrm{SE}(2)$ is a subgroup of $\mathrm{E}(2)$. This hierarchy of structure follows the Erlangen Programme philosophy outlined in the Preface: in Klein's construction, the Projective, Affine, and Euclidean geometries

have increasingly more invariants and correspond to progressively smaller groups.

Isomorphisms and Automorphisms We have described symmetries as structure preserving and invertible maps *from an object to itself*. Such maps are also known as *automorphisms*, and describe a way in which an object is equivalent it itself. However, an equally important class of maps are the so-called *isomorphisms*, which exhibit an equivalence between two non-identical objects. These concepts are often conflated, but distinguishing them is necessary to create clarity for our following discussion.

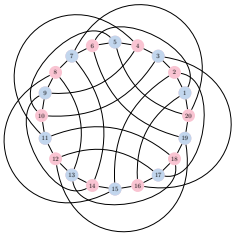
Difference between an isomorphism and automorphism

To understand the difference, consider a set $\Omega = \{0, 1, 2\}$. An automorphism of the set Ω is a bijection $\tau : \Omega \to \Omega$ such as a cyclic shift $\tau(u) = u + 1 \mod 3$. Such a map preserves the cardinality property, and maps Ω onto itself. If we have another set $\Omega' = \{a, b, c\}$ with the same number of elements, then a bijection $\eta : \Omega \to \Omega'$ such as $\eta(0) = a$, $\eta(1) = b$, $\eta(2) = c$ is a *set isomorphism*.

automorphism and isomorphism with graphs

As we will see in Section 4.1 for graphs, the notion of structure includes not just the number of nodes, but also the connectivity. An isomorphism $\eta : \mathcal{V} \to \mathcal{V}'$ between two graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is thus a bijection between the nodes that maps pairs of connected nodes to pairs of connected nodes, and likewise for pairs of non-connected nodes. Two isomorphic graphs are thus structurally identical, and differ only in the way their nodes are ordered. On the other hand, a graph automorphism or symmetry is a map $\tau : \mathcal{V} \to \mathcal{V}$ maps the nodes of the graph back to itself, while preserving the connectivity. A graph with a non-trivial automorphism (i.e., $\tau \neq \mathrm{id}$) presents symmetries.

I.e., $(\eta(u), \eta(v)) \in \mathcal{V}'$ iff $(u, v) \in \mathcal{V}$.

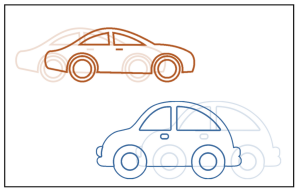


The *Folkman graph* (Folkman, 1967) is a beautiful example of a graph with 3840 automorphisms, exemplified by the many symmetric ways to draw it.

3.3 Deformation Stability

The symmetry formalism introduced in Sections 3.1–3.2 captures an idealised world where we know exactly which transformations are to be considered as symmetries, and we want to respect these symmetries *exactly*. For instance in computer vision, we might assume that planar translations are exact symmetries. However, the real world is noisy and this model falls short in two ways.

Firstly, while these simple groups provide a way to understand *global* sym-



Two objects moving at different velocities in a video define a transformation outside the translation group.

metries of the domain Ω (and by extension, of signals on it, $\mathcal{X}(\Omega)$), they do not capture *local* symmetries well. For instance, consider a video scene with several objects, each moving along its own different direction. At subsequent frames, the resulting scene will contain approximately the same semantic information, yet no global translation explains the transformation from one frame to another. In other cases, such as a deformable 3D object viewed by a camera, it is simply very hard to describe the group of transformations that preserve the object identity. These examples illustrate that in reality we are more interested in a far larger set of transformations where global, exact invariance is replaced by a local, inexact one. In our discussion, we will distinguish between two scenarios: the setting where the domain Ω is fixed, and signals $x \in \mathcal{X}(\Omega)$ are undergoing deformations, and the setting where the domain Ω itself may be deformed.

Domain Ω is fixed but the signal gets deformed

Stability to signal deformations In many applications, we know a priori that a small deformation of the signal x should not change the output of $f(x)$, so it is tempting to consider such deformations as symmetries. For instance, we could view small diffeomorphisms $\tau \in \mathrm{Diff}(\Omega)$, or even small bijections, as symmetries. However, small deformations can be composed to form large deformations, so "small deformations" do not form a group, and we cannot ask for invariance or equivariance to small deformations only. Since large deformations can can actually materially change the semantic content of the input, it is not a good idea to use the full group $\mathrm{Diff}(\Omega)$ as symmetry group either.

reminder: map and its inverse are continuously differentiable (diffeomorphisms)

E.g., the composition of two ϵ-isometries is a 2ϵ-isometry, violating the closure property.

A better approach is to quantify how "far" a given $\tau \in \mathrm{Diff}(\Omega)$ is from a given symmetry subgroup $\mathfrak{G} \subset \mathrm{Diff}(\Omega)$ (e.g. translations) with a complexity measure $c(\tau)$, so that $c(\tau) = 0$ whenever $\tau \in \mathfrak{G}$. We can now replace our previous definition of exact invariance and equivarance under group actions with a 'softer' notion of *deformation stability* (or *approximate invariance*):

Can we say simply that we are trying to put a bound on the translation ?

$$\|f(\rho(\tau)x) - f(x)\| \leq Cc(\tau)\|x\|, \ , \ \forall x \in \mathcal{X}(\Omega) \tag{4}$$

where $\rho(\tau)x(u) = x(\tau^{-1}u)$ as before, and where C is some constant independent of the signal x. A function $f \in \mathcal{F}(\mathcal{X}(\Omega))$ satisfying the above equation is said to be *geometrically stable*. We will see examples of such functions in the next Section 3.4.

Since $c(\tau) = 0$ for $\tau \in \mathfrak{G}$, this definition generalises the $\mathfrak{G}$-invariance property defined above. Its utility in applications depends on introducing an

appropriate deformation cost. In the case of images defined over a continuous Euclidean plane, a popular choice is $c^2(\tau) := \int_\Omega \|\nabla\tau(u)\|^2 \mathrm{d}u$, which measures the 'elasticity' of τ, i.e., how different it is from the displacement by a constant vector field. This deformation cost is in fact a norm often called the *Dirichlet energy*, and can be used to quantify how far τ is from the translation group.

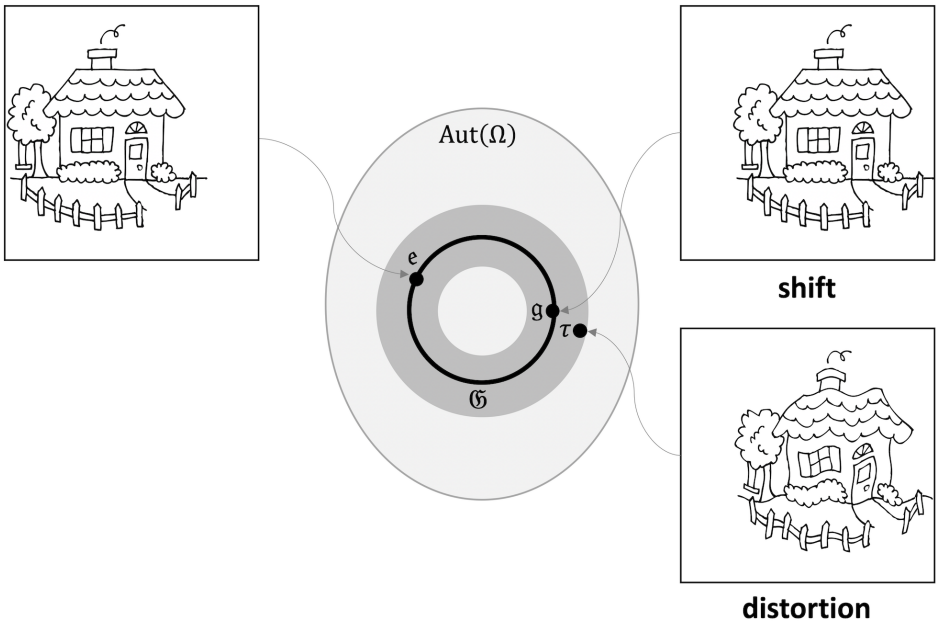


Figure 6: The set of all bijective mappings from Ω into itself forms the *set automorphism group* $\mathrm{Aut}(\Omega)$, of which a symmetry group $\mathfrak{G}$ (shown as a circle) is a subgroup. Geometric Stability extends the notion of $\mathfrak{G}$-invariance and equivariance to 'transformations around $\mathfrak{G}$' (shown as gray ring), quantified in the sense of some metric between transformations. In this example, a smooth distortion of the image is close to a shift.

Domain Ω itself gets deformed

Stability to domain deformations In many applications, the object being deformed is not the signal, but the geometric domain Ω itself. Canonical instances of this are applications dealing with graphs and manifolds: a graph can model a social network at different instance of time containing slightly different social relations (follow graph), or a manifold can model a 3D object undergoing non-rigid deformations. This deformation can be quantified

as follows. If $\mathcal{D}$ denotes the space of all possible variable domains (such as the space of all graphs, or the space of Riemannian manifolds), one can define for $\Omega, \tilde{\Omega} \in \mathcal{D}$ an appropriate metric ('distance') $d(\Omega, \tilde{\Omega})$ satisfying $d(\Omega, \tilde{\Omega}) = 0$ if Ω and $\tilde{\Omega}$ are equivalent in some sense: for example, the graph edit distance vanishes when the graphs are isomorphic, and the Gromov-Hausdorff distance between Riemannian manifolds equipped with geodesic distances vanishes when two manifolds are isometric.

The graph edit distance measures the minimal cost of making two graphs isomorphic by a sequences of graph edit operations. The Gromov-Hausdorff distance measures the smallest possible metric distortion of a correspondence between two metric spaces, see Gromov (1981).

A common construction of such distances between domains relies on some family of invertible mapping $\eta : \Omega \to \tilde{\Omega}$ that try to 'align' the domains in a way that the corresponding structures are best preserved. For example, in the case of graphs or Riemannian manifolds (regarded as metric spaces with the geodesic distance), this alignment can compare pair-wise adjacency or distance structures (d and $\tilde{d}$, respectively),

$$d_{\mathcal{D}}(\Omega, \tilde{\Omega}) = \inf_{\eta \in \mathfrak{G}} \|d - \tilde{d} \circ (\eta \times \eta)\|$$

where $\mathfrak{G}$ is the group of isomorphisms such as bijections or isometries, and the norm is defined over the product space $\Omega \times \Omega$. In other words, a distance between elements of $\Omega, \tilde{\Omega}$ is 'lifted' to a distance between the domains themselves, by accounting for all the possible alignments that preserve the internal structure. Given a signal $x \in \mathcal{X}(\Omega)$ and a deformed domain $\tilde{\Omega}$, one can then consider the deformed signal $\tilde{x} = x \circ \eta^{-1} \in \mathcal{X}(\tilde{\Omega})$.

Two graphs can be aligned by the Quadratic Assignment Problem (QAP), which considers in its simplest form two graphs $G, \tilde{G}$ of the same size n, and solves $\min_{\mathbf{P} \in \Sigma_n} \mathrm{trace}(\mathbf{A}\mathbf{P}\tilde{\mathbf{A}}\mathbf{P}^\top)$, where $\mathbf{A}, \tilde{\mathbf{A}}$ are the respective adjacency matrices and Σ_n is the group of $n \times n$ permutation matrices. The graph edit distance can be associated with such QAP (Bougleux et al., 2015).

By slightly abusing the notation, we define $\mathcal{X}(\mathcal{D}) = \{(\mathcal{X}(\Omega), \Omega) : \Omega \in \mathcal{D}\}$ as the ensemble of possible input signals defined over a varying domain. A function $f : \mathcal{X}(\mathcal{D}) \to \mathcal{Y}$ is stable to domain deformations if

$$\|f(x, \Omega) - f(\tilde{x}, \tilde{\Omega})\| \leq C\|x\| d_{\mathcal{D}}(\Omega, \tilde{\Omega}) \tag{5}$$

for all $\Omega, \tilde{\Omega} \in \mathcal{D}$, and $x \in \mathcal{X}(\Omega)$. We will discuss this notion of stability in the context of manifolds in Sections 4.4–4.6, where isometric deformations play a crucial role. Furthermore, it can be shown that the stability to domain deformations is a natural generalisation of the stability to signal deformations, by viewing the latter in terms of deformations of the volume form Gama et al. (2019).

3.4 Scale Separation

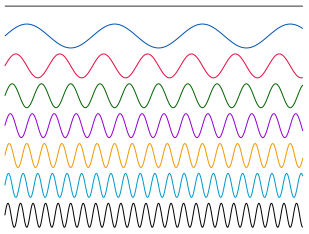
While deformation stability substantially strengthens the global symmetry priors, it is not sufficient in itself to overcome the curse of dimensionality, in

the sense that, informally speaking, there are still "too many" functions that respect (4) as the size of the domain grows. A key insight to overcome this curse is to exploit the multiscale structure of physical tasks. Before describing multiscale representations, we need to introduce the main elements of Fourier transforms, which rely on frequency rather than scale.

Fourier Transform and Global invariants Arguably the most famous signal decomposition is the *Fourier transform*, the cornerstone of harmonic analysis. The classical one-dimensional Fourier transform

$$\hat{x}(\xi) = \int_{-\infty}^{+\infty} x(u) e^{-\mathrm{i}\xi u} \mathrm{d}u$$

expresses the function $x(u) \in L^2(\Omega)$ on the domain $\Omega = \mathbb{R}$ as a linear combination of orthogonal oscillating *basis functions* $\varphi_\xi(u) = e^{\mathrm{i}\xi u}$, indexed by their rate of oscillation (or *frequency*) ξ. Such an organisation into frequencies reveals important information about the signal, e.g. its smoothness and localisation. The Fourier basis itself has a deep geometric foundation and can be interpreted as the natural vibrations of the domain, related to its geometric structure (see e.g. Berger (2012)).



Fourier basis functions have global support. As a result, local signals produce energy across all frequencies.

The Fourier transform plays a crucial role in signal processing as it offers a dual formulation of *convolution*,

$$(x \star \theta)(u) = \int_{-\infty}^{+\infty} x(v)\theta(u-v)\mathrm{d}v$$

a standard model of linear signal filtering (here and in the following, x denotes the signal and θ the filter). As we will show in the following, the convolution operator is diagonalised in the Fourier basis, making it possible to express convolution as the product of the respective Fourier transforms,

$$\widehat{(x \star \theta)}(\xi) = \hat{x}(\xi) \cdot \hat{\theta}(\xi),$$

a fact known in signal processing as the Convolution Theorem.

In the following, we will use convolution and (*cross-*)*correlation*

$$(x \star \theta)(u) = \int_{-\infty}^{+\infty} x(v)\theta(u+v)\mathrm{d}v$$

interchangeably, as it is common in machine learning: the difference between the two is whether the filter is reflected, and since the filter is typically learnable, the distinction is purely notational.

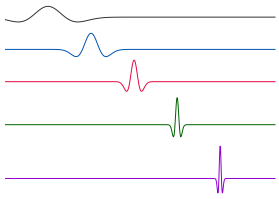
As it turns out, many fundamental differential operators such as the Laplacian are described as convolutions on Euclidean domains. Since such differential operators can be defined intrinsically over very general geometries, this provides a formal procedure to extend Fourier transforms beyond Euclidean domains, including graphs, groups and manifolds. We will discuss this in detail in Section 4.4.

An essential aspect of Fourier transforms is that they reveal *global* properties of the signal and the domain, such as smoothness or conductance. Such global behavior is convenient in presence of global symmetries of the domain such as translation, but not to study more general diffeomorphisms. This requires a representation that trades off spatial and frequential localisation, as we see next.

Fourier frequency based representation reveal global properties such as smoothness or conductance. These are useful in the presence of global symmetries of the domain like translation. OTOH local invariance cannot be identified using the Fourier representation - we need a different representation called scale based representation.

Multiscale representations The notion of local invariance can be articulated by switching from a Fourier frequency-based representation to a *scale-based* representation, the cornerstone of multi-scale decomposition methods such as *wavelets*. The essential insight of multi-scale methods is to decompose functions defined over the domain Ω into elementary functions that are localised *both in space and frequency*. In the case of wavelets, this is achieved by correlating a translated and dilated filter (*mother wavelet*) ψ, producing a combined spatio-frequency representation called a *continuous wavelet transform*

See Mallat (1999) for a comperehensive introduction.



Contrary to Fourier, wavelet atoms are localised and multi-scale, allowing to capture fine details of the signal with atoms having small spatial support and coarse details with atoms having large spatial support. The term *atom* here is synonymous with 'basis element' in Fourier analysis, with the caveat that wavelets are redundant (over-complete).

$$(W_\psi x)(u,\xi) = \xi^{-1/2} \int_{-\infty}^{+\infty} \psi\left(\frac{v-u}{\xi}\right) x(v)\mathrm{d}v.$$

The translated and dilated filters are called *wavelet atoms*; their spatial position and dilation correspond to the coordinates u and ξ of the wavelet transform. These coordinates are usually sampled dyadically ($\xi = 2^{-j}$ and $u = 2^{-j}k$), with j referred to as *scale*. Multi-scale signal representations bring important benefits in terms of capturing regularity properties beyond global smoothness, such as piece-wise smoothness, which made them a popular tool in signal and image processing and numerical analysis in the 90s.

useful for capturing local invariance and other locality properties

Deformation stability of Multiscale representations: The benefit of multiscale localised wavelet decompositions over Fourier decompositions is revealed when considering the effect of small deformations 'nearby' the underlying symmetry group. Let us illustrate this important concept in the Euclidean domain and the translation group. Since the Fourier representation diagonalises the shift operator (which can be thought of as convolution, as we will see in more detail in Section 4.2), it is an efficient representation for translation transformations. However, Fourier decompositions are unstable under high-frequency deformations. In contrast, wavelet decompositions offer a stable representation in such cases.

Indeed, let us consider $\tau \in \mathrm{Aut}(\Omega)$ and its associated linear representation $\rho(\tau)$. When $\tau(u) = u - v$ is a shift, as we will verify in Section 4.2, the operator $\rho(\tau) = S_v$ is a *shift operator* that commutes with convolution. Since convolution operators are diagonalised by the Fourier transform, the action of shift in the frequency domain amounts to shifting the complex phase of the Fourier transform,

$$\widehat{(S_v x)}(\xi) = e^{-\mathrm{i}\xi v}\hat{x}(\xi).$$

Thus, the *Fourier modulus* $f(x) = |\hat{x}|$ removing the complex phase is a simple shift-invariant function, $f(S_v x) = f(x)$. However, if we have only approximate translation, $\tau(u) = u - \tilde{\tau}(u)$ with $\|\nabla\tau\|_\infty = \sup_{u\in\Omega}\|\nabla\tilde{\tau}(u)\| \leq \epsilon$, the situation is entirely different: it is possible to show that

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

irrespective of how small ϵ is (i.e., how close is τ to being a shift). Consequently, such Fourier representation is *unstable under deformations*, however small. This unstability is manifested in general domains and non-rigid transformations; we will see another instance of this unstability in the analysis of 3d shapes using the natural extension of Fourier transforms described in Section 4.4.

Wavelets offer a remedy to this problem that also reveals the power of multiscale representations. In the above example, we can show (Mallat, 2012) that the wavelet decomposition $W_\psi x$ is *approximately equivariant* to deformations,

$$\frac{\|\rho(\tau)(W_\psi x) - W_\psi(\rho(\tau)x)\|}{\|x\|} = \mathcal{O}(\epsilon).$$

This notation implies that $\rho(\tau)$ acts on the spatial coordinate of $(W_\psi x)(u, \xi)$.

In other words, decomposing the signal information into scales using localised filters rather than frequencies turns a global unstable representation into a family of locally stable features. Importantly, such measurements at different scales are not yet invariant, and need to be progressively processed towards the low frequencies, hinting at the deep compositional nature of modern neural networks, and captured in our Blueprint for Geometric Deep Learning, presented next.

Scale Separation Prior: We can build from this insight by considering a multiscale coarsening of the data domain Ω into a hierarchy $\Omega_1, \ldots, \Omega_J$. As it turns out, such coarsening can be defined on very general domains,

Here J is the scale

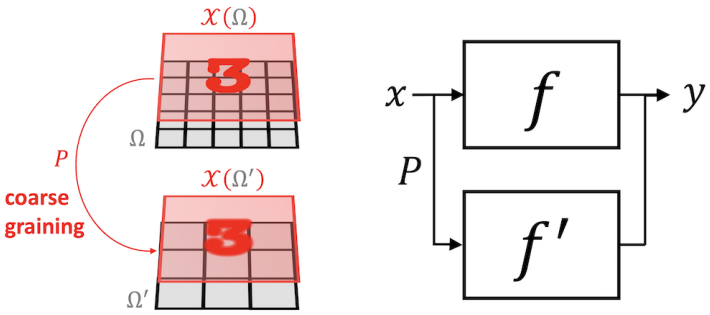


Figure 7: Illustration of Scale Separation for image classification tasks. The classifier f' defined on signals on the coarse grid $\mathcal{X}(\Omega')$ should satisfy $f \approx f' \circ P$, where $P : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$.

pooling in CNN

including grids, graphs, and manifolds. Informally, a coarsening assimilates nearby points $u, u' \in \Omega$ together, and thus only requires an appropriate notion of *metric* in the domain. If $\mathcal{X}_j(\Omega_j, \mathcal{C}_j) := \{x_j : \Omega_j \to \mathcal{C}_j\}$ denotes signals defined over the coarsened domain Ω_j, we informally say that a function $f : \mathcal{X}(\Omega) \to \mathcal{Y}$ is *locally stable* at scale j if it admits a factorisation of the form $f \approx f_j \circ P_j$, where $P_j : \mathcal{X}(\Omega) \to \mathcal{X}_j(\Omega_j)$ is a non-linear *coarse graining* and $f_j : \mathcal{X}_j(\Omega_j) \to \mathcal{Y}$. In other words, while the target function f might depend on complex long-range interactions between features over the whole domain, in locally-stable functions it is possible to *separate* the interactions across scales, by first focusing on localised interactions that are then propagated towards the coarse scales.

Fast Multipole Method (FMM) is a numerical technique originally developed to speed up the calculation of long-ranged forces in n-body problems. FMM groups sources that lie close together and treats them as a single source.

The crux of it - the mathematical properties that come into play in local pooling in CNNs

Such principles are of fundamental importance in many areas of physics and mathematics, as manifested for instance in statistical physics in the so-called renormalisation group, or leveraged in important numerical algorithms such as the Fast Multipole Method. In machine learning, multiscale representations and local invariance are the fundamental mathematical principles underpinning the efficiency of Convolutional Neural Networks and Graph Neural Networks and are typically implemented in the form of *local pooling*. In future work, we will further develop tools from computational harmonic analysis that unify these principles across our geometric domains and will shed light onto the statistical learning benefits of scale separation.

3.5 The Blueprint of Geometric Deep Learning

The geometric principles of Symmetry, Geometric Stability, and Scale Separation discussed in Sections 3.1–3.4 can be combined to provide a universal blueprint for learning stable representations of high-dimensional data. These representations will be produced by functions f operating on signals $\mathcal{X}(\Omega, \mathcal{C})$ defined on the domain Ω, which is endowed with a symmetry group $\mathfrak{G}$.

The geometric priors we have described so far do not prescribe a specific *architecture* for building such representation, but rather a series of necessary conditions. However, they hint at an axiomatic construction that provably satisfies these geometric priors, while ensuring a highly expressive representation that can approximate any target function satisfying such priors.

A simple initial observation is that, in order to obtain a highly expressive representation, we are required to introduce a non-linear element, since if f is linear and $\mathfrak{G}$-invariant, then for all $x \in \mathcal{X}(\Omega)$,

a nice mathematical explanation of why we need the non-linearity

explained beautifully by Bruno in the lecture 4 (Geometric Priors II) under the section Combining Invariance with Scale Separation

$$f(x) = \frac{1}{\mu(\mathfrak{G})} \int_{\mathfrak{G}} f(\mathfrak{g}.x) \mathrm{d}\mu(\mathfrak{g}) = f\left(\frac{1}{\mu(\mathfrak{G})} \int_{\mathfrak{G}} (\mathfrak{g}.x) \mathrm{d}\mu(\mathfrak{g}) \right),$$

Here, $\mu(\mathfrak{g})$ is known as the *Haar measure* of the group $\mathfrak{G}$, and the integral is performed over the entire group.

which indicates that F only depends on x through the $\mathfrak{G}$-*average* $Ax = \frac{1}{\mu(\mathfrak{G})} \int_{\mathfrak{G}} (\mathfrak{g}.x) \mathrm{d}\mu(\mathfrak{g})$. In the case of images and translation, this would entail using only the average RGB color of the input!

While this reasoning shows that the family of *linear invariants* is not a very rich object, the family of *linear equivariants* provides a much more powerful tool, since it enables the construction of rich and stable features by composition with appropriate non-linear maps, as we will now explain. Indeed, if $B : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega, \mathcal{C}')$ is $\mathfrak{G}$-equivariant satisfying $B(\mathfrak{g}.x) = \mathfrak{g}.B(x)$ for all $x \in \mathcal{X}$ and $\mathfrak{g} \in \mathfrak{G}$, and $\sigma : \mathcal{C}' \to \mathcal{C}''$ is an arbitrary (non-linear) map, then we easily verify that the composition $U := (\boldsymbol{\sigma} \circ B) : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega, \mathcal{C}'')$ is also $\mathfrak{G}$-equivariant, where $\boldsymbol{\sigma} : \mathcal{X}(\Omega, \mathcal{C}') \to \mathcal{X}(\Omega, \mathcal{C}'')$ is the element-wise instantiation of σ given as $(\boldsymbol{\sigma}(x))(u) := \sigma(x(u))$.

Linear equivariants enable construction of rich features through composition with non linear maps

Composition U is also G-equivariant

U(g.x) = (σ ∘ B)(g . x)
= σ ∘ B(g . x)
= g . (σ ∘ B(x))
= g . (σ ∘ B)(x)
= g . U(x)

This simple property allows us to define a very general family of $\mathfrak{G}$-invariants, by composing U with the group averages $A \circ U : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{C}''$. A natural question is thus whether any $\mathfrak{G}$-invariant function can be approximated at arbitrary precision by such a model, for appropriate choices of B and σ. It is not hard to adapt the standard Universal Approximation Theorems from unstructured vector inputs to show that shallow 'geometric' networks are

Here we get a general family of G-invariants by composing U with the Group Average

But the question is can we approximate any G-invariant function with this model ?

Universal approximation theorem says so and we can adjust the group average to a general non-linear invariant

But as we know global invariance has tension with deformation stability. Hence we look for local equivariance

(A ∘ U)(x) = (A ∘ U) (g . x), since g . U(x) = U (g . x) from G-equivariance

G-invariance

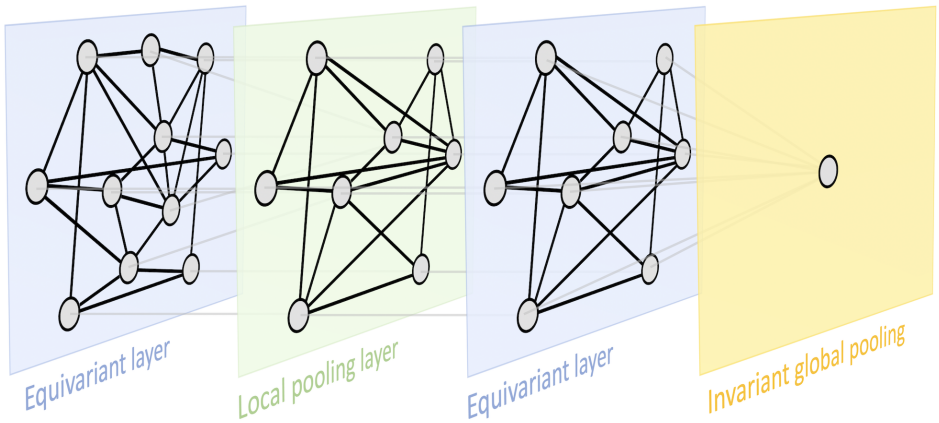


Figure 8: Geometric Deep Learning blueprint, exemplified on a graph. A typical Graph Neural Network architecture may contain permutation equivariant layers (computing node-wise features), local pooling (graph coarsening), and a permutation-invariant global pooling layer (readout layer).

Such proofs have been demonstrated, for example, for the Deep Sets model by Zaheer et al. (2017).

also universal approximators, by properly generalising the group average to a general non-linear invariant. However, as already described in the case of Fourier versus Wavelet invariants, there is a fundamental tension between shallow global invariance and deformation stability. This motivates an alternative representation, which considers instead *localised* equivariant maps. Assuming that Ω is further equipped with a distance metric d, we call an equivariant map U localised if $(Ux)(u)$ depends only on the values of $x(v)$ for $\mathcal{N}_u = \{v : d(u,v) \leq r\}$, for some small radius r; the latter set $\mathcal{N}_u$ is called the *receptive field*.

Fourier : symmetry
Wavelet : scale

Meaningful metrics can be defined on grids, graphs, manifolds, and groups. A notable exception are sets, where there is no predefined notion of metric.

A single layer of local equivariant map U cannot approximate functions with long-range interactions, but a composition of several local equivariant maps $U_J \circ U_{J-1} \cdots \circ U_1$ increases the receptive field while preserving the stability properties of local equivariants. The receptive field is further increased by interleaving downsampling operators that coarsen the domain (again assuming a metric structure), completing the parallel with Multiresolution Analysis (MRA, see e.g. Mallat (1999)).

The term 'receptive field' originated in the neuroscience literature, referring to the spatial domain that affects the output of a given neuron.

In summary, the geometry of the input domain, with knowledge of an underyling symmetry group, provides three key building blocks: (i) a local equivariant map, (ii) a global invariant map, and (iii) a coarsening operator.

These building blocks provide a rich function approximation space with prescribed invariance and stability properties by combining them together in a scheme we refer to as the *Geometric Deep Learning Blueprint* (Figure 8).

Geometric Deep Learning Blueprint

Let Ω and Ω' be domains, $\mathfrak{G}$ a symmetry group over Ω, and write $\Omega' \subseteq \Omega$ if Ω' can be considered a compact version of Ω.

We define the following building blocks:

Linear $\mathfrak{G}$-equivariant layer $B : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega', \mathcal{C}')$ satisfying $B(\mathfrak{g}.x) = \mathfrak{g}.B(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$.

Nonlinearity $\sigma : \mathcal{C} \to \mathcal{C}'$ applied element-wise as $(\boldsymbol{\sigma}(x))(u) = \sigma(x(u))$.

Local pooling (coarsening) $P : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega', \mathcal{C})$, such that $\Omega' \subseteq \Omega$.

$\mathfrak{G}$-invariant layer (global pooling) $A : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{Y}$ satisfying $A(\mathfrak{g}.x) = A(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$.

Using these blocks allows constructing $\mathfrak{G}$-invariant functions $f : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{Y}$ of the form

$$f = A \circ \boldsymbol{\sigma}_J \circ B_J \circ P_{J-1} \circ \ldots \circ P_1 \circ \boldsymbol{\sigma}_1 \circ B_1$$

where the blocks are selected such that the output space of each block matches the input space of the next one. Different blocks may exploit different choices of symmetry groups $\mathfrak{G}$.

This is simply an awesome formalization

Different settings of Geometric Deep Learning One can make an important distinction between the setting when the domain Ω is assumed to be *fixed* and one is only interested in varying input signals defined on that domain, or the domain is part of the input as *varies* together with signals defined on it. A classical instance of the former case is encountered in computer vision applications, where images are assumed to be defined on a fixed domain

(grid). Graph classification is an example of the latter setting, where both the structure of the graph as well as the signal defined on it (e.g. node features) are important. In the case of varying domain, geometric stability (in the sense of insensitivity to the deformation of Ω) plays a crucial role in Geometric Deep Learning architectures.

This blueprint has the right level of generality to be used across a wide range of geometric domains. Different Geometric Deep Learning methods thus differ in their choice of the domain, symmetry group, and the specific implementation details of the aforementioned building blocks. As we will see in the following, a large class of deep learning architectures currently in use fall into this scheme and can thus be derived from common geometric principles.

The blueprint offers the right level of generalization - specializing each of its components will yield a different class of deep learning architectures.

In the following sections (4.1–4.6) we will describe the various geometric domains focusing on the '5G', and in Sections 5.1–5.8 the specific implementations of Geometric Deep Learning on these domains.

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

4 Geometric Domains: the 5 Gs

The main focus of our text will be on graphs, grids, groups, geodesics, and gauges. In this context, by 'groups' we mean global symmetry transformations in homogeneous space, by 'geodesics' metric structures on manifolds, and by 'gauges' local reference frames defined on tangent bundles (and vec-

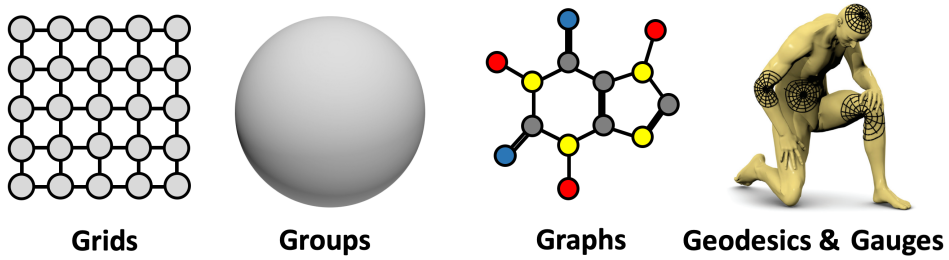


Figure 9: The 5G of Geometric Deep Learning: grids, groups & homogeneous spaces with global symmetry, graphs, geodesics & metrics on manifolds, and gauges (frames for tangent or feature spaces).

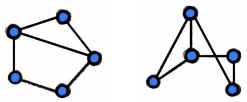
tor bundles in general). These notions will be explained in more detail later. In the next sections, we will discuss in detail the main elements in common and the key distinguishing features between these structures and describe the symmetry groups associated with them. Our exposition is not in the order of generality – in fact, grids are particular cases of graphs – but a way to highlight important concepts underlying our Geometric Deep Learning blueprint.

4.1 Graphs and Sets

In multiple branches of science, from sociology to particle physics, graphs are used as models of systems of relations and interactions. From our perspective, graphs give rise to a very basic type of invariance modelled by the group of permutations. Furthermore, other objects of interest to us, such as grids and sets, can be obtained as a particular case of graphs.

A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a collection of *nodes* $\mathcal{V}$ and *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ between pairs of nodes. For the purpose of the following discussion, we will further assume the nodes to be endowed with s-dimensional *node features*, denoted by $\mathbf{x}_u$ for all $u \in \mathcal{V}$. Social networks are perhaps among the most commonly studied examples of graphs, where nodes represent users, edges correspond to friendship relations between them, and node features model user properties such as age, profile picture, etc. It is also often possible to endow the edges, or entire graphs, with features; but as this does not alter the main findings of this section, we will defer discussing it to future work.

Depending on the application field, nodes may also be called *vertices*, and edges are often referred to as *links* or *relations*. We will use these terms interchangeably.



Isomorphism is an edge-preserving bijection between two graphs. Two isomorphic graphs shown here are identical up to reordering of their nodes.

The key structural property of graphs is that the nodes in $\mathcal{V}$ are usually not assumed to be provided in any particular order, and thus any operations performed on graphs should not depend on the ordering of nodes. The desirable property that functions acting on graphs should satisfy is thus *permutation invariance*, and it implies that for any two *isomorphic* graphs, the outcomes of these functions are identical. We can see this as a particular setting of our blueprint, where the domain $\Omega = \mathcal{G}$ and the space $\mathcal{X}(\mathcal{G}, \mathbb{R}^d)$ is that of d-dimensional node-wise signals. The symmetry we consider is given by the *permutation group* $\mathfrak{G} = \Sigma_n$, whose elements are all the possible orderings of the set of node indices $\{1, \dots, n\}$.

Let us first illustrate the concept of permutation invariance on *sets*, a special case of graphs without edges (i.e., $\mathcal{E} = \emptyset$). By stacking the node features as rows of the $n \times d$ matrix $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top$, we do effectively specify an ordering of the nodes. The action of the permutation $\mathfrak{g} \in \Sigma_n$ on the set of nodes amounts to the reordering of the rows of $\mathbf{X}$, which can be represented as an $n \times n$ *permutation matrix* $\rho(\mathfrak{g}) = \mathbf{P}$, where each row and column contains exactly one 1 and all the other entries are zeros.

There are exactly $n!$ such permutations, so Σ_n is, even for modest n, a very large group.

Usual definition of permutation matrix (check wikipedia for details)

A function f operating on this set is then said to be *permutation invariant* if, for any such permutation matrix $\mathbf{P}$, it holds that $f(\mathbf{PX}) = f(\mathbf{X})$. One simple such function is

$$f(\mathbf{X}) = \phi\left(\sum_{u \in \mathcal{V}} \psi\left(\mathbf{x}_u\right)\right), \tag{6}$$

where the function ψ is independently applied to every node's features, and ϕ is applied on its *sum-aggregated* outputs: as sum is independent of the order in which its inputs are provided, such a function is invariant with respect to the permutation of the node set, and is hence guaranteed to always return the same output, no matter how the nodes are permuted.

e.g. predicting the total energy of a molecular graph, we need to ensure that the output is unaffected by a different ordering of the input nodes. We call such functions f permutation-invariant

f(PX, PAP') = f(X, A)

Functions like the above provide a 'global' graph-wise output, but very often, we will be interested in functions that act 'locally', in a node-wise manner. For example, we may want to apply some function to *update* the features in every node, obtaining the set of *latent* node features. If we stack these latent features into a matrix $\mathbf{H} = \mathbf{F}(\mathbf{X})$ is no longer permutation invariant: the order of the rows of $\mathbf{H}$ should be *tied* to the order of the rows of $\mathbf{X}$, so that we know which output node feature corresponds to which input node. We need instead a more fine-grained notion of *permutation equivariance*, stating that, once we "commit" to a permutation of inputs, it consistently permutes the resulting objects. Formally, $\mathbf{F}(\mathbf{X})$ is a *permutation equivariant* function

also it may be difficult to compute the global function if the graph is a huge one

If we want to make node-wise predictions e.g. to detect malicious users in a social network we want a function that changes in the same way with reordering of the nodes. In other words permutation-equivariant

F(PX, PAP') = PF(X, A)

We use the bold notation for our function $\mathbf{F}(\mathbf{X})$ to emphasise it outputs node-wise vector features and is hence a matrix-valued function.

if, for any permutation matrix $\mathbf{P}$, it holds that $\mathbf{F}(\mathbf{PX}) = \mathbf{PF}(\mathbf{X})$. A shared node-wise linear transform

$$\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}) = \mathbf{X}\boldsymbol{\Theta} \tag{7}$$

specified by a weight matrix $\boldsymbol{\Theta} \in \mathbb{R}^{d \times d'}$, is one possible construction of such a permutation equivariant function, producing in our example latent features of the form $\mathbf{h}_u = \boldsymbol{\Theta}^\top \mathbf{x}_u$.

This construction arises naturally from our Geometric Deep Learning blueprint. We can first attempt to characterise *linear equivariants* (functions of the form $\mathbf{FPX} = \mathbf{PFX}$), for which it is easy to verify that any such map can be written as a linear combination of two *generators*, the identity $\mathbf{F}_1\mathbf{X} = \mathbf{X}$ and the average $\mathbf{F}_2\mathbf{X} = \frac{1}{n}\mathbf{1}\mathbf{1}^\top\mathbf{X} = \frac{1}{n}\sum_{u=1}^{n}\mathbf{x}_u$. As will be described in Section 5.4, the popular Deep Sets (Zaheer et al., 2017) architecture follows precisely this blueprint.

We can now generalise the notions of permutation invariance and equivariance from sets to graphs. In the generic setting $\mathcal{E} \neq \emptyset$, the graph connectivity can be represented by the $n \times n$ *adjacency matrix* $\mathbf{A}$, defined as

$$a_{uv} = \begin{cases} 1 & (u,v) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

When the graph is *undirected*, i.e. $(u,v) \in \mathcal{E}$ iff $(v,u) \in \mathcal{E}$, the adjacency matrix is *symmetric*, $\mathbf{A} = \mathbf{A}^\top$.

Note that now the adjacency and feature matrices $\mathbf{A}$ and $\mathbf{X}$ are "synchronised", in the sense that a_{uv} specifies the adjacency information between the nodes described by the uth and vth rows of $\mathbf{X}$. Therefore, applying a permutation matrix $\mathbf{P}$ to the node features $\mathbf{X}$ automatically implies applying it to $\mathbf{A}$'s rows and columns, $\mathbf{PAP}^\top$. We say that (a graph-wise function) f is *permutation invariant* if

uth and vth rows of X represent the feature vectors for node u and v respectively

$\mathbf{PAP}^\top$ is the representation of Σ_n acting on matrices.

$$f(\mathbf{PX}, \mathbf{PAP}^\top) = f(\mathbf{X}, \mathbf{A}) \tag{9}$$

and (a node-wise function) $\mathbf{F}$ is *permutation equivariant* if

As a way to emphasise the fact that our functions operating over graphs now need to take into account the adjacency information, we use the notation $f(\mathbf{X}, \mathbf{A})$.

$$\mathbf{F}(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{PF}(\mathbf{X}, \mathbf{A}) \tag{10}$$

for any permutation matrix $\mathbf{P}$.

Explanation of how applying P to X and the same P to PAP' keeps both the transformed feature matrix X and the transformed adjacency matrix "synchronized"

Here again, we can first characterise linear equivariant functions. As observed by Maron et al. (2018), any linear $\mathbf{F}$ satisfying equation (10) can be expressed as a linear combination of fifteen linear generators; remarkably, this corresponds to the *Bell number* B_4, which counts the number of ways to partition a set of 4 elements, in this case given by the 4-indices $(u,v),(u',v')$ indexing a linear map acting on the adjacency matrix.

Let P be a permutation matrix such that $[x_1, \ldots x_n]P = [x_{\sigma^{-1}(1)}, \ldots, x_{\sigma^{-1}(n)}]$, where σ is a permutation - that is, each row vector is sent to the one with $\sigma(i)^{\text{th}}$ component the original i^{th} component. Then, treating a matrix A as a row vector, $A_{ij} = (AP)_{i\sigma(j)} = (AP)^T_{\sigma(j)i}$, so $(PAP^T)_{\sigma(i)\sigma(j)} = ((A^TP)^TP)_{\sigma(i)\sigma(j)} = (A^TP)^T_{\sigma(i)j} = (A^TP)_{j\sigma(i)} = A^T_{ji} = A_{ij}$. Therefore PAP^T re-indexes A by a permutation σ of the indices, as required.

(see https://math.stackexchange.com/a/711621)

If P switches two rows, then $P^2 = I$. Combine this with $PP^T = I$ to get $P = P^T$.

(see https://math.stackexchange.com/a/1312826)

this family of generators is *independent of* n. Amongst these generators, our blueprint specifically advocates for those that are also *local*, i.e., whereby the output on node u directly depends on its neighbouring nodes in the graph. We can formalise this constraint explicitly in our model construction, by defining what it means for a node to be neighbouring another.

Often, the node u itself is included in its own neighbourhood.

A (undirected) *neighbourhood* of node u, sometimes also called *1-hop*, is defined as

$$\mathcal{N}_u = \{v : (u, v) \in \mathcal{E} \text{ or } (v, u) \in \mathcal{E}\} \tag{11}$$

and the *neighbourhood features* as the multiset

A *multiset*, denoted $\{\!\{ \dots \}\!\}$, is a set where the same element can appear more than once. This is the case here because the features of different nodes can be equal.

$$\mathbf{X}_{\mathcal{N}_u} = \{\!\{\mathbf{x}_v : v \in \mathcal{N}_u\}\!\}. \tag{12}$$

Operating on 1-hop neighbourhoods aligns well with the *locality* aspect of our blueprint: namely, defining our metric over graphs as the *shortest path distance* between nodes using edges in $\mathcal{E}$.

The GDL blueprint thus yields a general recipe for constructing permutation equivariant functions on graphs, by specifying a *local* function ϕ that operates over the features of a node and its neighbourhood, $\phi(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$. Then, a permutation equivariant function $\mathbf{F}$ can be constructed by applying ϕ to every node's neighbourhood in isolation (see Figure 10):

F has to be permutation equivariant and that is possible only if ϕ is permutation invariant

$$\mathbf{F}(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} — & \phi(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & — \\ — & \phi(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & — \\ & \vdots & \\ — & \phi(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & — \end{bmatrix} \tag{13}$$

As $\mathbf{F}$ is constructed by applying a shared function ϕ to each node locally, its permutation equivariance rests on ϕ's output being independent on the ordering of the nodes in $\mathcal{N}_u$. Thus, if ϕ is built to be permutation invariant, then this property is satisfied. As we will see in future work, the choice of ϕ plays a crucial role in the expressive power of such a scheme. When ϕ is injective, it is equivalent to one step of the *Weisfeiler-Lehman graph isomorphism test*, a classical algorithm in graph theory providing a necessary condition for two graphs to be isomorphic by an iterative color refinement procedure.

It is also worth noticing that the difference between functions defined on sets and more general graphs in this example is that in the latter case we need to explicitly account for the structure of the domain. As a consequence, graphs stand apart in the sense that the domain becomes *part of the input* in machine

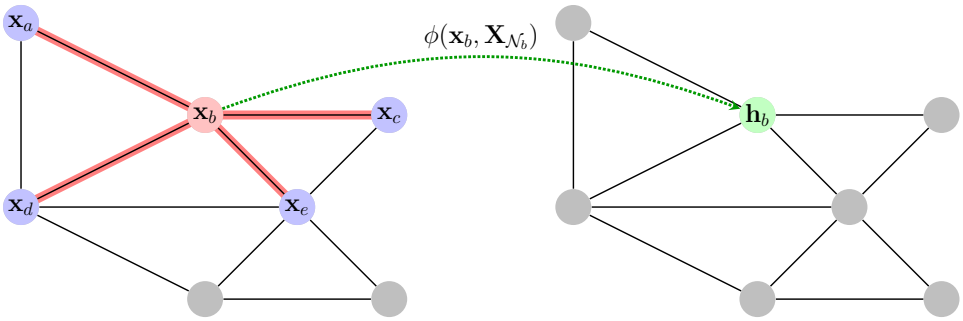


Figure 10: An illustration of constructing permutation-equivariant functions over graphs, by applying a permutation-invariant function ϕ to every neighbourhood. In this case, ϕ is applied to the features $\mathbf{x}_b$ of node b as well as the multiset of its neighbourhood features, $\mathbf{X}_{\mathcal{N}_b} = \{\!\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}\!\}$. Applying ϕ in this manner to every node's neighbourhood recovers the rows of the resulting matrix of latents features $\mathbf{H} = \mathbf{F}(\mathbf{X}, \mathbf{A})$.

learning problems, whereas when dealing with sets and grids (both particular cases of graphs) we can specify only the features and assume the domain to be *fixed*. This distinction will be a recurring motif in our discussion. As a result, the notion of geometric stability (invariance to domain deformation) is crucial in most problems of learning on graphs. It straightforwardly follows from our construction that permutation invariant and equivariant functions produce identical outputs on isomorphic (topologically-equivalent) graphs. These results can be generalised to approximately isomorphic graphs, and several results on stability under graph perturbations exist (Levie et al., 2018). We will return to this important point in our discussion on manifolds, which we will use as an vehicle to study such invariance in further detail.

Second, due to their additional structure, graphs and grids, unlike sets, can be coarsened in a non-trivial way, giving rise to a variety of pooling operations.

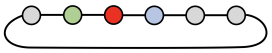
More precisely, we cannot define a non-trivial coarsening assuming set structure alone. There exist established approaches that infer topological structure from unordered sets, and those can admit non-trivial coarsening.

4.2 Grids and Euclidean spaces

The second type of objects we consider are grids. It is fair to say that the impact of deep learning was particularly dramatic in computer vision, natural language processing, and speech recognition. These applications all share a geometric common denominator: an underlying grid structure. As already

translation invariance is a much stronger geometric prior than permutation invariance for grids

mentioned, grids are a particular case of graphs with special adjacency. However, since the order of nodes in a grid is fixed, machine learning models for signals defined on grids are no longer required to account for permutation invariance, and have a stronger geometric prior: translation invariance.



Circulant matrices and Convolutions Let us dwell on this point in more detail. Assuming for simplicity periodic boundary conditions, we can think of a one-dimensional grid as a *ring graph* with nodes indexed by $0, 1, \dots, n-1$ modulo n (which we will omit for notation brevity) and the adjacency matrix with elements $a_{u,u+1 \bmod n} = 1$ and zero otherwise. There are two main differences from the general graph case we have discussed before. First, each node u has identical connectivity, to its neighbours $u-1$ and $u+1$, and thus structure-wise indistinguishable from the others. Second and more importantly, since the nodes of the grid have a fixed ordering, we also have a fixed ordering of the *neighbours:* we can call $u-1$ the 'left neighbour' and $u+1$ the 'right neighbour'. If we use our previous recipe for designing a equivariant function $\mathbf{F}$ using a local aggregation function ϕ, we now have $\mathbf{f}(\mathbf{x}_u) = \phi(\mathbf{x}_{u-1}, \mathbf{x}_u, \mathbf{x}_{u+1})$ at every node of the grid: ϕ does not need to be permutation invariant anymore. For a particular choice of a linear transformation $\phi(\mathbf{x}_{u-1}, \mathbf{x}_u, \mathbf{x}_{u+1}) = \theta_{-1}\mathbf{x}_{u-1} + \theta_0\mathbf{x}_u + \theta_1\mathbf{x}_{u+1}$, we can write $\mathbf{F}(\mathbf{X})$ as a matrix product,

As we will see later, this makes the grid a homogeneous space.

so ordering is always maintained. The result is that phi no longer has to be permutation invariant

$$\mathbf{F}(\mathbf{X}) = \begin{bmatrix} \theta_0 & \theta_1 & & & \theta_{-1} \\ \theta_{-1} & \theta_0 & \theta_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \theta_{-1} & \theta_0 & \theta_1 \\ \theta_1 & & & \theta_{-1} & \theta_0 \end{bmatrix} \begin{bmatrix} — & \mathbf{x}_0 & — \\ — & \mathbf{x}_1 & — \\ & \vdots & \\ — & \mathbf{x}_{n-2} & — \\ — & \mathbf{x}_{n-1} & — \end{bmatrix}$$

Note this very special multi-diagonal structure with one element repeated along each diagonal, sometimes referred to as "weight sharing" in the machine learning literature.

More generally, given a vector $\boldsymbol{\theta} = (\theta_0, \dots, \theta_{n-1})$, a *circulant matrix* $\mathbf{C}(\boldsymbol{\theta}) = (\theta_{u-v \bmod n})$ is obtained by appending circularly shifted versions of the vector $\boldsymbol{\theta}$. Circulant matrices are synonymous with discrete convolutions,

Because of the periodic boundary conditions, it is a *circular* or *cyclic convolution*. In signal processing, $\boldsymbol{\theta}$ is often referred to as the "filter," and in CNNs, its coefficients are learnable.

$$(\mathbf{x} \star \boldsymbol{\theta})_u = \sum_{v=0}^{n-1} x_{v \bmod n}\ \theta_{u-v \bmod n}$$

check out https://arxiv.org/abs/1805.05533 for a nice proof and a tutorial on circulant matrix

as one has $\mathbf{C}(\boldsymbol{\theta})\mathbf{x} = \mathbf{x} \star \boldsymbol{\theta}$. A particular choice of $\boldsymbol{\theta} = (0, 1, 0, \dots, 0)^\top$ yields a special circulant matrix that shifts vectors to the right by one position. This matrix is called the (right) *shift* or *translation operator* and denoted by $\mathbf{S}$.

Intuitive explanation: Left multiplication by S amounts to row circular permutation. Right multiplication by S amounts to column circular permutation. And both result in the same matrix. Hence SC(θ) = C(θ)S

The left shift operator is given by $\mathbf{S}^\top$. Obviously, shifting left and then right (or vice versa) does not do anything, which means $\mathbf{S}$ is *orthogonal*: $\mathbf{S}^\top\mathbf{S} = \mathbf{S}\mathbf{S}^\top = \mathbf{I}$.

Circulant matrices can be characterised by their *commutativity* property: the product of circulant matrices is commutative, i.e. $\mathbf{C}(\boldsymbol{\theta})\mathbf{C}(\boldsymbol{\eta}) = \mathbf{C}(\boldsymbol{\eta})\mathbf{C}(\boldsymbol{\theta})$ for any $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$. Since the shift is a circulant matrix, we get the familiar *translation* or *shift equivariance* of the convolution operator,

Takeaway: Commutativity of circulant matrices gives translation or shift equivariance of the convolution operator

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

Such commutativity property should not be surprising, since the underlying symmetry group (the translation group) is Abelian. Moreover, the opposite direction appears to be true as well, i.e. a matrix is circulant iff it commutes with shift. This, in turn, allows us to *define* convolution as a translation equivariant linear operation, and is a nice illustration of the power of geometric priors and the overall philosophy of Geometric ML: convolution emerges from the first principle of translational symmetry.

a circulant matrix commutes with shift. Conversely if a matrix commutes with shift, it's circulant. Proof see Section 3 of https://arxiv.org/abs/1805.05533

Takeaway: Usually when we study CNNs we take for granted the convolution operator as if it appeared from thin air. With this interpretation convolution emerges from the first principle of translation symmetry. Again goes to show the power of symmetry based geometry or geometric priors

Note that unlike the situation on sets and graphs, the number of linearly independent shift-equivariant functions (convolutions) *grows* with the size of the domain (since we have one degree of freedom in each diagonal of a circulant matrix). However, the scale separation prior guarantees filters can be *local*, resulting in the same $\Theta(1)$-parameter complexity per layer, as we will verify in Section 5.1 when discussing the use of these principles in the implementation of Convolutional Neural Network architectures.

Derivation of the discrete Fourier transform We have already mentioned the Fourier transform and its connection to convolution: the fact that the Fourier transform diagonalises the convolution operation is an important property used in signal processing to perform convolution in the frequency domain as an element-wise product of the Fourier transforms. However, textbooks usually only state this fact, rarely explaining *where* the Fourier transform comes from and what is so *special* about the Fourier basis. Here we can show it, demonstrating once more how foundational are the basic principles of symmetry.

1. Circular convolution of 2 vectors can be written as a matrix vector product with a circulant matrix

2. Simultaneous diagonalization of circulant matrices yields the DFT

see Sec 4.2 of https://arxiv.org/abs/1805.05533

Convolution Theorem

For this purpose, recall a fact from linear algebra that (diagonalisable) matrices are *jointly diagonalisable* iff they mutually commute. In other words, there exists a common eigenbasis for all the circulant matrices, in which

We must additionally assume distinct eigenvalues, otherwise there might be multiple possible diagonalisations. This assumption is satisfied with our choice of $\mathbf{S}$.

they differ only by their eigenvalues. We can therefore pick one circulant matrix and compute its eigenvectors—we are assured that these will be the eigenvectors of all other circulant matrices as well. It is convenient to pick the shift operator, for which the eigenvectors happen to be the discrete Fourier basis

$\mathbf{S}$ is orthogonal but non-symmetric, hence, its eigenvectors are orthogonal but the eigenvalues are complex (roots of unity).

$$\boldsymbol{\varphi}_k = \frac{1}{\sqrt{n}}\left(1, e^{\frac{2\pi i k}{n}}, e^{\frac{4\pi i k}{n}}, \dots, e^{\frac{2\pi i (n-1) k}{n}}\right)^\top, \qquad k = 0, 1, \dots, n-1,$$

which we can arrange into an $n \times n$ Fourier matrix $\boldsymbol{\Phi} = (\boldsymbol{\varphi}_0, \dots, \boldsymbol{\varphi}_{n-1})$. Multiplication by $\boldsymbol{\Phi}^*$ gives the Discrete Fourier Transform (DFT), and by $\boldsymbol{\Phi}$ the inverse DFT,

Note that the eigenvectors are complex, so we need to take complex conjugation when transposing $\boldsymbol{\Phi}$.

$$\hat{x}_k = \frac{1}{\sqrt{n}} \sum_{u=0}^{n-1} x_u e^{-\frac{2\pi i k u}{n}} \qquad\qquad x_u = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \hat{x}_k e^{+\frac{2\pi i k u}{n}}.$$

Since all circulant matrices are jointly diagonalisable, they are also diagonalised by the Fourier transform and differ only in their eigenvalues. Since the eigenvalues of the circulant matrix $\mathbf{C}(\boldsymbol{\theta})$ are the Fourier transform of the filter (see e.g. Bamieh (2018)), $\hat{\boldsymbol{\theta}} = \boldsymbol{\Phi}^* \boldsymbol{\theta}$, we obtain the Convolution Theorem:

Since the Fourier transform is an orthogonal matrix ($\boldsymbol{\Phi}^*\boldsymbol{\Phi} = \mathbf{I}$), geometrically it acts as a change of the system of coordinates that amounts to an n-dimensional rotation. In this system of coordinates ("Fourier domain"), the action of a circulant $\mathbf{C}$ matrix becomes element-wise product.

$$\mathbf{C}(\boldsymbol{\theta})\mathbf{x} = \boldsymbol{\Phi} \begin{bmatrix} \hat{\theta}_0 & & \\ & \ddots & \\ & & \hat{\theta}_{n-1} \end{bmatrix} \boldsymbol{\Phi}^* \mathbf{x} = \boldsymbol{\Phi}(\hat{\boldsymbol{\theta}} \odot \hat{\mathbf{x}})$$

Because the Fourier matrix $\boldsymbol{\Phi}$ has a special algebraic structure, the products $\boldsymbol{\Phi}^\star \mathbf{x}$ and $\boldsymbol{\Phi}\mathbf{x}$ can be computed with $\mathcal{O}(n \log n)$ complexity using a Fast Fourier Transform (FFT) algorithm. This is one of the reasons why frequency-domain filtering is so popular in signal processing; furthermore, the filter is typically designed directly in the frequency domain, so the Fourier transform $\hat{\boldsymbol{\theta}}$ is never explicitly computed.

Less computational complexity in the Fourier domain

Besides the didactic value of the derivation of the Fourier transform and convolution we have done here, it provides a scheme to generalise these concepts to graphs. Realising that the adjacency matrix of the ring graph is exactly the shift operator, one can can develop the graph Fourier transform and an analogy of the convolution operator by computing the eigenvectors of the adjacency matrix (see e.g. Sandryhaila and Moura (2013)). Early

attempts to develop graph neural networks by analogy to CNNs, sometimes termed 'spectral GNNs', exploited this exact blueprint. We will see in Sections 4.4–4.6 that this analogy has some important limitations. The first limitation comes from the fact that a grid is fixed, and hence all signals on it can be represented in the same Fourier basis. In contrast, on general graphs, the Fourier basis depends on the structure of the graph. Hence, we cannot directly compare Fourier transforms on two different graphs — a problem that translated into a lack of generalisation in machine learning problems. Secondly, multi-dimensional grids, which are constructed as tensor products of one-dimensional grids, retain the underlying structure: the Fourier basis elements and the corresponding frequencies (eigenvalues) can be organised in multiple dimensions. In images, for example, we can naturally talk about horizontal and vertical frequency and filters have a notion of *direction*. On graphs, the structure of the Fourier domain is one-dimensional, as we can only organise the Fourier basis functions by the magnitude of the corresponding frequencies. As a result, graph filters are oblivious of direction or *isotropic*.

In graph signal processing, the eigenvectors of the *graph Laplacian* are often used as an alternative of the adjacency matrix to construct the graph Fourier transform, see Shuman et al. (2013). On grids, both matrices have joint eigenvectors, but on graphs they results in somewhat different though related constructions.

Derivation of the continuous Fourier transform For the sake of completeness, and as a segway for the next discussion, we repeat our analysis in the continuous setting. Like in Section 3.4, consider functions defined on $\Omega = \mathbb{R}$ and the translation operator $(S_v f)(u) = f(u - v)$ shifting f by some position v. Applying S_v to the Fourier basis functions $\varphi_\xi(u) = e^{\mathrm{i}\xi u}$ yields, by associativity of the exponent,

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

i.e., $\varphi u_\xi(u)$ is the complex eigenvector of S_v with the complex eigenvalue $e^{-\mathrm{i}\xi v}$ – exactly mirroring the situation we had in the discrete setting. Since S_v is a unitary operator (i.e., $\|S_v x\|_p = \|x\|_p$ for any p and $x \in L_p(\mathbb{R})$), any eigenvalue λ must satisfy $|\lambda| = 1$, which corresponds precisely to the eigenvalues $e^{-i\xi v}$ found above. Moreover, the spectrum of the translation operator is *simple*, meaning that two functions sharing the same eigenvalue must necessarily be collinear. Indeed, suppose that $S_v f = e^{-\mathrm{i}\xi_0 v} f$ for some ξ_0. Taking the Fourier transform in both sides, we obtain

$$\forall\, \xi\ ,\ e^{-\mathrm{i}\xi v} \hat{f}(\xi) = e^{-\mathrm{i}\xi_0 v} \hat{f}(\xi)\ ,$$

which implies that $\hat{f}(\xi) = 0$ for $\xi \neq \xi_0$, thus $f = \alpha \varphi_{\xi_0}$.

For a general linear operator C that is translation equivariant ($S_v C = C S_v$), we have

$$S_v C e^{\mathrm{i}\xi u} = C S_v e^{\mathrm{i}\xi u} = e^{-\mathrm{i}\xi v} C e^{\mathrm{i}\xi u},$$

Eigenfunction is synonymous with 'eigenvector' and is used when referring to eigenvectors of continuous operators.

implying that $Ce^{\mathrm{i}\xi u}$ is also an eigenfunction of S_v with eigenvalue $e^{-\mathrm{i}\xi v}$, from where it follows from the simplicity of spectrum that $Ce^{\mathrm{i}\xi u} = \beta\varphi_\xi(u)$; in other words, the Fourier basis is the eigenbasis of all translation equivariant operators. As a result, C is *diagonal* in the Fourier domain and can be expressed as $Ce^{\mathrm{i}\xi u} = \hat{p}_C(\xi)e^{\mathrm{i}\xi u}$, where $\hat{p}_C(\xi)$ is a *transfer function* acting on different frequencies ξ. Finally, for an arbitrary function $x(u)$, by linearity,

$$\begin{aligned}(Cx)(u) &= C\int_{-\infty}^{+\infty} \hat{x}(\xi)e^{\mathrm{i}\xi u}\mathrm{d}\xi = \int_{-\infty}^{+\infty} \hat{x}(\xi)\hat{p}_C(\xi)e^{\mathrm{i}\xi u}\mathrm{d}\xi \\ &= \int_{-\infty}^{+\infty} p_C(v)x(u-v)\mathrm{d}v \;= (x \star p_C)(u),\end{aligned}$$

The spectral characterisation of the translation group is a particular case of a more general result in Functional Analysis, the *Stone's Theorem*, which derives an equivalent characterisation for any one-parameter unitary group.

where $p_C(u)$ is the inverse Fourier transform of $\hat{p}_C(\xi)$. It thus follows that every linear translation equivariant operator is a convolution.

4.3 Groups and Homogeneous spaces

Our discussion of grids highlighted how shifts and convolutions are intimately connected: convolutions are linear shift-equivariant operations, and vice versa, any shift-equivariant linear operator is a convolution. Furthermore, shift operators can be jointly diagonalised by the Fourier transform. As it turns out, this is part of a far larger story: both convolution and the Fourier transform can be defined *for any group of symmetries* that we can sum or integrate over.

Technically, we need the group to be *locally compact*, so that there exists a left-invariant Haar measure. Integrating with respect to this measure, we can "shift" the integrand by any group element and obtain the same result, just as how we have

$$\int_{-\infty}^{+\infty} x(u)\mathrm{d}u = \int_{-\infty}^{+\infty} x(u-v)\mathrm{d}u$$

for functions $x : \mathbb{R} \to \mathbb{R}$.

Consider the Euclidean domain $\Omega = \mathbb{R}$. We can understand the convolution as a pattern matching operation: we match shifted copies of a filter $\theta(u)$ with an input signal $x(u)$. The value of the convolution $(x \star \theta)(u)$ at a point u is the inner product of the signal x with the filter *shifted by* u,

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

Note that what we define here is not convolution but *cross-correlation*, which is tacitly used in deep learning under the name 'convolution'. We do it for consistency with the following discussion, since in our notation $(\rho(\mathfrak{g})x)(u) = x(u-v)$ and $(\rho(\mathfrak{g}^{-1})x)(u) = x(u+v)$.

Note that in this case u is both *a point on the domain* $\Omega = \mathbb{R}$ and also *an element of the translation group*, which we can identify with the domain itself, $\mathfrak{G} = \mathbb{R}$. We will now show how to generalise this construction, by simply replacing the translation group by another group $\mathfrak{G}$ acting on Ω.

Group convolution As discussed in Section 3, the action of the group $\mathfrak{G}$ on the domain Ω induces a representation ρ of $\mathfrak{G}$ on the space of signals $\mathcal{X}(\Omega)$ via $\rho(\mathfrak{g})x(u) = x(\mathfrak{g}^{-1}u)$. In the above example, $\mathfrak{G}$ is the translation group whose elements act by shifting the coordinates, $u + v$, whereas $\rho(\mathfrak{g})$ is the shift operator acting on signals as $(S_v x)(u) = x(u - v)$. Finally, in order to apply a filter to the signal, we invoke our assumption of $\mathcal{X}(\Omega)$ being a Hilbert space, with an inner product

$$\langle x, \theta \rangle = \int_\Omega x(u)\theta(u)\mathrm{d}u,$$

The integration is done w.r.t. an invariant measure μ on Ω. In case μ is discrete, this means summing over Ω.

where we assumed, for the sake of simplicity, scalar-valued signals, $\mathcal{X}(\Omega, \mathbb{R})$; in general the inner product has the form of equation (2).

Having thus defined how to transform signals and match them with filters, we can define the *group convolution* for signals on Ω,

$$(x \star \theta)(\mathfrak{g}) = \langle x, \rho(\mathfrak{g})\theta \rangle = \int_\Omega x(u)\theta(\mathfrak{g}^{-1}u)\mathrm{d}u. \tag{14}$$

Note that $x \star \theta$ takes values on the *elements* $\mathfrak{g}$ *of our group* $\mathfrak{G}$ rather than points on the domain Ω. Hence, the next layer, which takes $x \star \theta$ as input, should act on signals defined *on to the group* $\mathfrak{G}$, a point we will return to shortly.

Just like how the traditional Euclidean convolution is shift-equivariant, the more general group convolution is $\mathfrak{G}$-*equivariant*. The key observation is that matching the signal x with a $\mathfrak{g}$-transformed filter $\rho(\mathfrak{g})\theta$ is the same as matching the inverse transformed signal $\rho(\mathfrak{g}^{-1})x$ with the untransformed filter θ. Mathematically, this can be expressed as $\langle x, \rho(\mathfrak{g})\theta \rangle = \langle \rho(\mathfrak{g}^{-1})x, \theta \rangle$. With this insight, $\mathfrak{G}$-equivariance of the group convolution (14) follows immediately from its definition and the defining property $\rho(\mathfrak{h}^{-1})\rho(\mathfrak{g}) = \rho(\mathfrak{h}^{-1}\mathfrak{g})$ of group representations,

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

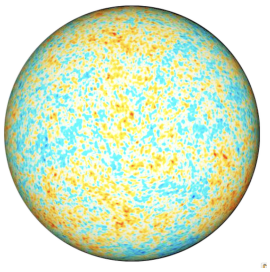
Let us look at some examples. The case of one-dimensional grid we have studied above is obtained with the choice $\Omega = \mathbb{Z}_n = \{0, \dots, n-1\}$ and the cyclic shift group $\mathfrak{G} = \mathbb{Z}_n$. The group elements in this case are cyclic shifts of indices, i.e., an element $\mathfrak{g} \in \mathfrak{G}$ can be identified with some $u = 0, \dots, n-1$ such that $\mathfrak{g}.v = v - u \bmod n$, whereas the inverse element is $\mathfrak{g}^{-1}.v = v + u \bmod n$. Importantly, in this example the elements of the *group* (shifts) are also elements of the *domain* (indices). We thus can, with some

abuse of notation, identify the two structures (i.e., $\Omega = \mathfrak{G}$); our expression for the group convolution in this case

$$(x \star \theta)(\mathfrak{g}) = \sum_{v=0}^{n-1} x_v \, \theta_{\mathfrak{g}^{-1}v},$$

Actually here again, this is cross-correlation.

leads to the familiar convolution $(x \star \theta)_u = \sum_{v=0}^{n-1} x_v \, \theta_{v+u \bmod n}$.

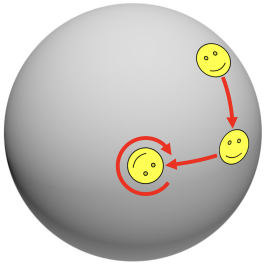


Cosmic microwave background radiation, captured by the Planck space observatory, is a signal on $\mathbb{S}^2$.

Spherical convolution Now consider the two-dimensional sphere $\Omega = \mathbb{S}^2$ with the group of rotations, the *special orthogonal group* $\mathfrak{G} = \mathrm{SO}(3)$. While chosen for pedagogical reason, this example is actually very practical and arises in numerous applications. In astrophysics, for example, observational data often naturally has spherical geometry. Furthermore, spherical symmetries are very important in applications in chemistry when modeling molecules and trying to predict their properties, e.g. for the purpose of virtual drug screening.

Representing a point on the sphere as a three-dimensional unit vector $\mathbf{u} : \|\mathbf{u}\| = 1$, the action of the group can be represented as a 3×3 orthogonal matrix $\mathbf{R}$ with $\det(\mathbf{R}) = 1$. The spherical convolution can thus be written as the inner product between the signal and the rotated filter,

$$(x \star \theta)(\mathbf{R}) = \int_{\mathbb{S}^2} x(\mathbf{u}) \theta(\mathbf{R}^{-1}\mathbf{u}) \mathrm{d}\mathbf{u}.$$



The action of $\mathrm{SO}(3)$ group on $\mathbb{S}^2$. Note that three types of rotation are possible; the $\mathrm{SO}(3)$ is a three-dimensional manifold.

The first thing to note is than now the group is not identical to the domain: the group $\mathrm{SO}(3)$ is a Lie group that is in fact a three-dimensional manifold, whereas $\mathbb{S}^2$ is a two-dimensional one. Consequently, in this case, unlike the previous example, the convolution is a function *on* $\mathrm{SO}(3)$ *rather than on* Ω.

This has important practical consequences: in our Geometric Deep Learning blueprint, we concatenate multiple equivariant maps ("layers" in deep learning jargon) by applying a subsequent operator to the output of the previous one. In the case of translations, we can apply multiple convolutions in sequence, since their outputs are all defined on the same domain Ω. In the general setting, since $x \star \theta$ is a function on $\mathfrak{G}$ rather than on Ω, we cannot use exactly the same operation subsequently—it means that the next operation

has to deal with *signals on* $\mathfrak{G}$, i.e. $x \in \mathcal{X}(\mathfrak{G})$. Our definition of group convolution allows this case: we take as domain $\Omega = \mathfrak{G}$ acted on by $\mathfrak{G}$ itself via the group action $(\mathfrak{g}, \mathfrak{h}) \mapsto \mathfrak{g}\mathfrak{h}$ defined by the composition operation of $\mathfrak{G}$. This yields the representation $\rho(\mathfrak{g})$ acting on $x \in \mathcal{X}(\mathfrak{G})$ by $(\rho(\mathfrak{g})x)(\mathfrak{h}) = x(\mathfrak{g}^{-1}\mathfrak{h})$. Just like before, the inner product is defined by integrating the point-wise product of the signal and the filter over the domain, which now equals $\Omega = \mathfrak{G}$. In our example of spherical convolution, a second layer of convolution would thus have the form

The representation of $\mathfrak{G}$ acting on functions defined on $\mathfrak{G}$ itself is called the *regular representation* of $\mathfrak{G}$.

$$((x \star \theta) \star \phi)(\mathbf{R}) = \int_{\mathrm{SO}(3)} (x \star \theta)(\mathbf{Q})\phi(\mathbf{R}^{-1}\mathbf{Q})\mathrm{d}\mathbf{Q}.$$

Since convolution involves inner product that in turn requires integrating over the domain Ω, we can only use it on domains Ω that are small (in the discrete case) or low-dimensional (in the continuous case). For instance, we can use convolutions on the plane $\mathbb{R}^2$ (two dimensional) or special orthogonal group $\mathrm{SE}(3)$ (three dimensional), or on the finite set of nodes of a graph (n-dimensional), but we cannot in practice perform convolution on the group of permutations Σ_n, which has $n!$ elements. Likewise, integrating over higher-dimensional groups like the affine group (containing translations, rotations, shearing and scaling, for a total of 6 dimensions) is not feasible in practice. Nevertheless, as we have seen in Section 5.3, we can still build equivariant convolutions for large groups $\mathfrak{G}$ by working with signals defined on low-dimensional spaces Ω on which $\mathfrak{G}$ acts. Indeed, it is possible to show that any equivariant linear map $f : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$ between two domains Ω, Ω' can be written as a generalised convolution similar to the group convolution discussed here.

Second, we note that the Fourier transform we derived in the previous section from the shift-equivariance property of the convolution can also be extended to a more general case by projecting the signal onto the matrix elements of irreducible representations of the symmetry group. We will discuss this in future work. In the case of $\mathrm{SO}(3)$ studied here, this gives rise to the *spherical harmonics* and *Wigner D-functions*, which find wide applications in quantum mechanics and chemistry.

Finally, we point to the assumption that has so far underpinned our discussion in this section: whether Ω was a grid, plane, or the sphere, we could transform every point into any other point, intuitively meaning that all the points on the domain "look the same." A domain Ω with such property is

called a *homogeneous space*, where for any $u, v \in \Omega$ there exists $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g}.u = v$. In the next section we will try to relax this assumption.

The additional properties, $\mathfrak{e}.u = u$ and $\mathfrak{g}(\mathfrak{h}.u) = (\mathfrak{g}\mathfrak{h}).u$ are tacitly assumed here.

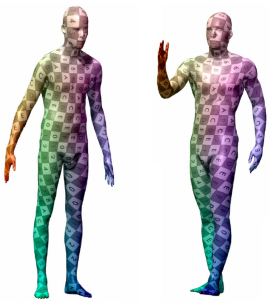
4.4 Geodesics and Manifolds

As well as the group of rotations $\mathrm{SO}(3)$, by virtue of it being a *Lie group*.

In our last example, the sphere $\mathbb{S}^2$ was a *manifold*, albeit a special one with a global symmetry group due to its homogeneous structure. Unfortunately, this is not the case for the majority of manifolds, which typically do not have global symmetries. In this case, we cannot straightforwardly define an action of $\mathfrak{G}$ on the space of signals on Ω and use it to 'slide' filters around in order to define a convolution as a direct generalisation of the classical construction. Nevertheless, manifolds do have two types of invariance that we will explore in this section: transformations preserving metric structure and local reference frame change.

While for many machine learning readers manifolds might appear as somewhat exotic objects, they are in fact very common in various scientific domains. In physics, manifolds play a central role as the model of our Universe — according to Einstein's General Relativity Theory, gravity arises from the curvature of the space-time, modeled as a pseudo-Riemannian manifold. In more 'prosaic' fields such as computer graphics and vision, manifolds are a common mathematical model of 3D shapes. The broad spectrum of applications of such models ranges from virtual and augmented reality and special effects obtained by means of 'motion capture' to structural biology dealing with protein interactions that stick together ('bind' in chemical jargon) like pieces of 3D puzzle. The common denominator of these applications is the use of a manifold to represent the boundary surface of some 3D object.

The term '3D' is somewhat misleading and refers to the *embedding space*. The shapes themselves are 2D manifolds (surfaces).



The human body is an example of a non-rigid object deforming in a nearly-isometric way.

There are several reasons why such models are convenient. First, they offer a compact description of the 3D object, eliminating the need to allocate memory to 'empty space' as is required in grid-based representations. Second, they allow to ignore the internal structure of the object. This is a handy property for example in structural biology where the internal folding of a protein molecule is often irrelevant for interactions that happen on the molecular surface. Third and most importantly, one often needs to deal with *deformable objects* that undergo non-rigid deformations. Our own body is one such example, and many applications in computer graphics and vision, such as the aforementioned motion capture and virtual avatars, require *deformation invariance*. Such deformations can be modelled very well as transformations

that preserve the intrinsic structure of a (Riemannian) manifold, namely the distances between points measured *along* the manifold, without regard to the way the manifold is embedded in the ambient space.

We should emphasise that manifolds fall under the setting of *varying domains* in our Geometric Deep Learning blueprint, and in this sense are similar to graphs. We will highlight the importance of the notion of invariance to domain deformations – what we called 'geometric stability' in Section 3.3. Since differential geometry is perhaps less familiar to the machine learning audience, we will introduce the basic concepts required for our discussion and refer the reader to Penrose (2005) for their detailed exposition.

Riemannian manifolds Since the formal definition of a manifold is somewhat involved, we prefer to provide an intuitive picture at the expense of some precision. In this context, we can think of a (differentiable or smooth) manifold as a smooth multidimensional curved surface that is *locally Euclidean*, in the sense that any small neighbourhood around any point it can be deformed to a neighbourhood of $\mathbb{R}^s$; in this case the manifold is said to be *s-dimensional*. This allows us to locally approximate the manifold around point u through the *tangent space* $T_u\Omega$. The latter can be visualised by thinking of a prototypical two-dimensional manifold, the sphere, and attaching a plane to it at a point: with sufficient zoom, the spherical surface will seem planar (Figure 11). The collection of all tangent spaces is called the *tangent bundle*, denoted $T\Omega$; we will dwell on the concept of bundles in more detail in Section 4.5.

By 'smooth' we mean differentiable suffient number of times, which is tacitly assumed for convenience. 'Deformed' here means *diffeomorphic*, i.e., we can map between the two neighbourhoods using a smooth and invertible map with smooth inverse.

Formally, the tangent bundle is the *disjoint union* $T\Omega = \bigsqcup_{u\in\Omega} T_u\Omega$.

A *tangent vector*, which we denote by $X \in T_u\Omega$, can be thought of as a local displacement from point u. In order to measure the *lengths* of tangent vectors and *angles* between them, we need to equip the tangent space with additional structure, expressed as a positive-definite bilinear function $g_u : T_u\Omega \times T_u\Omega \to \mathbb{R}$ depending smoothly on u. Such a function is called a *Riemannian metric*, in honour of Bernhardt Riemann who introduced the concept in 1856, and can be thought of as an inner product on the tangent space, $\langle X, Y\rangle_u = g_u(X, Y)$, which is an expression of the angle between any two tangent vectors $X, Y \in T_u\Omega$. The metric also induces a norm $\|X\|_u = g_u^{1/2}(X, X)$ allowing to locally measure lengths of vectors.

A bilinear function g is said to be *positive-definite* if $g(X, X) > 0$ for any non-zero vector $X \neq 0$. If g is expressed as a matrix $\mathbf{G}$, it means $\mathbf{G} \succ 0$. The determinant $|\mathbf{G}|^{1/2}$ provides a local volume element, which does not depend on the choice of the basis.

We must stress that tangent vectors are abstract geometric entities that exists in their own right and are *coordinate-free*. If we are to express a tangent vector

Unfortunately, too often vectors are identified with their coordinates. To emphasise this important difference, we use X to denote a tangent vector and $\mathbf{x}$ to denote its coordinates.

X numerically as an array of numbers, we can only represent it as a list of coordinates $\mathbf{x} = (x_1, \dots, x_s)$ *relative to some local basis* $\{X_1, \dots X_s\} \subseteq T_u\Omega$. Similarly, the metric can be expressed as an $s \times s$ matrix $\mathbf{G}$ with elements $g_{ij} = g_u(X_i, X_j)$ in that basis. We will return to this point in Section 4.5.

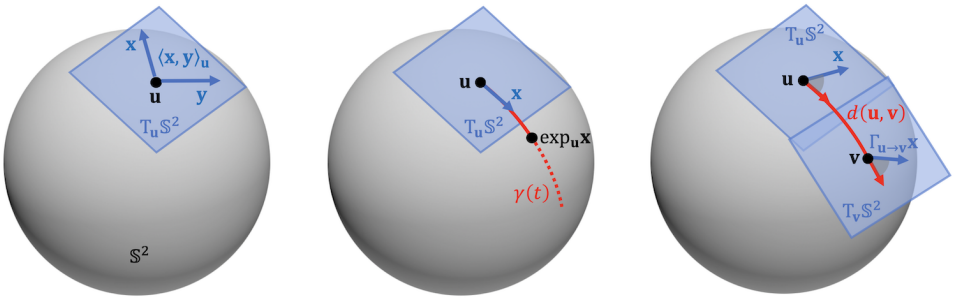


Figure 11: Basic notions of Riemannian geometry illustrated on the example of the two-dimensional sphere $\mathbb{S}^2 = \{\mathbf{u} \in \mathbb{R}^3 : \|\mathbf{u}\| = 1\}$, realised a subset (sub-manifold) of $\mathbb{R}^3$. The tangent space to the sphere is given as $T_{\mathbf{u}}\mathbb{S}^2 = \{\mathbf{x} \in \mathbb{R}^3 : \mathbf{x}^\top\mathbf{u} = 0\}$ and is a 2D plane – hence this is a 2-dimensional manifold. The Riemannian metric is simply the Euclidean inner product restricted to the tangent plane, $\langle \mathbf{x}, \mathbf{y}\rangle_{\mathbf{u}} = \mathbf{x}^\top\mathbf{y}$ for any $\mathbf{x}, \mathbf{x} \in T_{\mathbf{u}}\mathbb{S}^2$. The exponential map is given by $\exp_{\mathbf{u}}(\mathbf{x}) = \cos(\|\mathbf{x}\|)\mathbf{u} + \frac{\sin(\|\mathbf{x}\|)}{\|\mathbf{x}\|}\mathbf{x}$, for $\mathbf{x} \in T_{\mathbf{u}}\mathbb{S}^2$. Geodesics are great arcs of length $d(\mathbf{u}, \mathbf{v}) = \cos^{-1}(\mathbf{u}^\top\mathbf{v})$.

A manifold equipped with a metric is called a *Riemannian manifold* and properties that can be expressed entirely in terms of the metric are said to be *intrinsic*. This is a crucial notion for our discussion, as according to our template, we will be seeking to construct functions acting on signals defined on Ω that are invariant to metric-preserving transformations called *isometries* that deform the manifold without affecting its local structure. If such functions can be expressed in terms of intrinsic quantities, they are automatically guaranteed to be isometry-invariant and thus unaffected by isometric deformations. These results can be further extended to dealing with approximate isometries; this is thus an instance of the geometric stability (domain deformation) discussed in our blueprint.

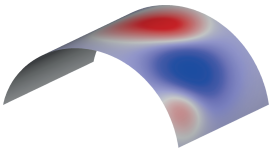
This result is known as the *Embedding Theorem*, due to Nash (1956). The art of origami is a manifestation of different isometric embeddings of the planar surface in $\mathbb{R}^3$ (Figure: Shutterstock/300 librarians).

While, as we noted, the definition of a Riemannian manifold does not require a geometric realisation in any space, it turns out that any smooth Riemannian manifold can be realised as a subset of a Euclidean space of sufficiently high dimension (in which case it is said to be 'embedded' in that space) by using

the structure of the Euclidean space to induce a Riemannian metric. Such an embedding is however not necessarily unique – as we will see, two different isometric realisations of a Riemannian metric are possible.

Scalar and Vector fields Since we are interested in signals defined on Ω, we need to provide the proper notion of scalar- and vector-valued functions on manifolds. A (smooth) *scalar field* is a function of the form $x : \Omega \to \mathbb{R}$. Scalar fields form a vector space $\mathcal{X}(\Omega, \mathbb{R})$ that can be equipped with the inner product

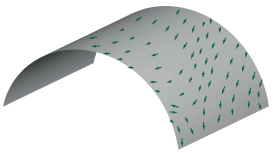
$$\langle x, y \rangle = \int_\Omega x(u) y(u) \mathrm{d}u, \tag{15}$$



Example of a scalar field.

where $\mathrm{d}u$ is the volume element induced by the Riemannian metric. A (smooth) *tangent vector field* is a function of the form $X : \Omega \to T\Omega$ assigning to each point a tangent vector in the respective tangent space, $u \mapsto X(u) \in T_u\Omega$. Vector fields also form a vector space $\mathcal{X}(\Omega, T\Omega)$ with the inner product defined through the Riemannian metric,

$$\langle X, Y \rangle = \int_\Omega g_u(X(u), Y(u)) \mathrm{d}u. \tag{16}$$



Example of a vector field. The fields are typically assumed to be of the same regularity class (smoothness) as the manifold itself.

Intrinsic gradient Another way to think of (and actually *define*) vector fields is as a generalised notion of derivative. In classical calculus, one can locally linearise a (smooth) function through the *differential* $\mathrm{d}x(u) = x(u + \mathrm{d}u) - x(u)$, which provides the change of the value of the function x at point u as a result of an inifinitesimal displacement $\mathrm{d}u$. However, in our case the naïve use of this definition is impossible, since expressions of the form "$u + \mathrm{d}u$" are meaningless on manifolds due to the lack of a global vector space structure.

The solution is to use tangent vectors as a model of local infinitesimal displacement. Given a smooth scalar field $x \in \mathcal{X}(\Omega, \mathbb{R})$, we can think of a (smooth) vector field as a linear map $Y : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, \mathbb{R})$ satisfying the properties of a *derivation*: $Y(c) = 0$ for any constant c (corresponding to the intuition that constant functions have vanishing derivatives), $Y(x + z) = Y(x) + Y(z)$ (linearity), and $Y(xz) = Y(x)z + xY(z)$ (*product* or *Leibniz rule*), for any smooth scalar fields $x, z \in \mathcal{X}(\Omega, \mathbb{R})$. It can be shown that one can use these properties to define vector fields axiomatically. The differential $\mathrm{d}x(Y) = Y(x)$ can be viewed as an operator $(u, Y) \mapsto Y(x)$ and interpreted as follows: the

change of x as the result of displacement $Y \in T_u\Omega$ at point u is given by $\mathrm{d}_u x(Y)$. It is thus an extension of the classical notion of *directional derivative*.

Importantly, this construction *does not use the Riemannian metric whatsoever* and can thus can be extended to a more general construction of bundles discussed in the Section 4.5.

Alternatively, at each point u the differential can be regarded as a *linear functional* $\mathrm{d}x_u : T_u\Omega \to \mathbb{R}$ acting on tangent vectors $X \in T_u\Omega$. Linear functionals on a vector space are called *dual vectors* or *covectors*; if in addition we are given an inner product (Riemannian metric), a dual vector can always be represented as

$$\mathrm{d}x_u(X) = g_u(\nabla x(u), X).$$

This is a consequence of the Riesz-Fréchet Representation Theorem, by which every dual vector can be expressed as an inner product with a vector.

The representation of the differential at point u is a tangent vector $\nabla x(u) \in T_u\Omega$ called the (intrinsic) *gradient* of x; similarly to the gradient in classical calculus, it can be thought of as the direction of the steepest increase of x. The gradient considered as an *operator* $\nabla : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, T\Omega)$ assigns at each point $x(u) \mapsto \nabla x(u) \in T_u\Omega$; thus, the gradient of a scalar field x is a vector field ∇x.

Geodesics Now consider a smooth curve $\gamma : [0, T] \to \Omega$ on the manifold with endpoints $u = \gamma(0)$ and $v = \gamma(T)$. The derivative of the curve at point t is a tangent vector $\gamma'(t) \in T_{\gamma(t)}\Omega$ called the *velocity vector*. Among all the curves connecting points u and v, we are interested in those of *minimum length*, i.e., we are seeking γ minimising the length functional

It is tacitly assumed that curves are given in *arclength parametrisation*, such that $\|\gamma'\| = 1$ (constant velocity).

$$\ell(\gamma) = \int_0^T \|\gamma'(t)\|_{\gamma(t)} \mathrm{d}t = \int_0^T g^{1/2}_{\gamma(t)}(\gamma'(t), \gamma'(t)) \mathrm{d}t.$$

Such curves are called *geodesics* (from the Greek γεοδαισία, literally 'division of Earth') and they play important role in differential geometry. Crucially to our discussion, the way we defined geodesics is intrinsic, as they depend solely on the Riemannian metric (through the length functional).

Readers familiar with differential geometry might recall that geodesics are a more general concept and their definition in fact does not necessarily require a Riemannian metric but a *connection* (also called a *covariant derivative*, as it generalises the notion of derivative to vector and tensor fields), which is defined axiomatically, similarly to our construction of the differential. Given a Riemannian metric, there exists a unique special connection called the *Levi-Civita connection* which is often tacitly assumed in Riemannian geometry. Geodesics arising from this connection are the length-minimising curves we have defined above.

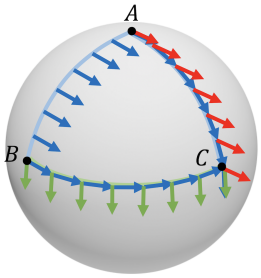
The Levi-Civita connection is torsion-free and compatible with the metric. The Fundamental Theorem of Riemannian geometry guarantees its existence and uniqueness.

We will show next how to use geodesics to define a way to transport tangent vectors on the manifold (parallel transport), create local intrinsic maps from the manifold to the tangent space (exponential map), and define distances (geodesic metric). This will allow us to construct convolution-like operations by applying a filter locally in the tangent space.

Parallel transport One issue we have already encountered when dealing with manifolds is that we cannot directly add or subtract two points $u, v \in \Omega$. The same problem arises when trying to compare tangent vectors at different points: though they have the same dimension, they belong to *different spaces*, e.g. $X \in T_u\Omega$ and $Y \in T_v\Omega$, and thus not directly comparable. Geodesics provide a mechanism to move vectors from one point to another, in the following way: let γ be a geodesic connecting points $u = \gamma(0)$ and $v = \gamma(T)$ and let $X \in T_u\Omega$. We can define a new set of tangent vectors along the geodesic, $X(t) \in T_{\gamma(t)}\Omega$ such that the length of $X(t)$ and the angle (expressed through the Riemannian metric) between it and the velocity vector of the curve is constant,

$$g_{\gamma(t)}(X(t), \gamma'(t)) = g_u(X, \gamma'(0)) = \text{const}, \qquad \|X(t)\|_{\gamma(t)} = \|X\|_u = \text{const}.$$

As a result, we get a unique vector $X(T) \in T_v\Omega$ at the end point v.



Euclidean transport of a vector from A to C makes no sense on the sphere, as the resulting vectors (red) are not in the tangent plane. Parallel transport from A to C (blue) rotates the vector along the path. It is path dependent: going along the path BC and ABC produces different results.

The map $\Gamma_{u\to v}(X) : T_u\Omega \to T_u\Omega$ and $T_v\Omega$ defined as $\Gamma_{u\to v}(X) = X(T)$ using the above notation is called *parallel transport* or *connection*; the latter term implying it is a mechanism to 'connect' between the tangent spaces $T_u\Omega$ and $T_v\Omega$. Due to the angle and length preservation conditions, parallel transport amounts to only rotation of the vector, so it can be associated with an element of the special orthogonal group $\mathrm{SO}(s)$ (called the *structure group* of the tangent bundle), which we will denote by $\mathfrak{g}_{u\to v}$ and discuss in further detail in Section 4.5.

Assuming that the manifold is orientable, otherwise $\mathrm{O}(s)$.

As we mentioned before, a connection can be defined axiomatically independently of the Riemannian metric, providing thus an abstract notion of parallel transport along any smooth curve. The result of such transport, however, depends on the path taken.

Exponential map Locally around a point u, it is always possible to define a unique geodesic in a given direction $X \in T_u\Omega$, i.e. such that $\gamma(0) = u$ and $\gamma'(0) = X$. When $\gamma_X(t)$ is defined for all $t \geq 0$ (that is, we can shoot the

geodesic from a point u for as long as we like), the manifold is said to be *geodesically complete* and the exponential map is defined on the whole tangent space. Since compact manifolds are geodesically complete, we can tacitly assume this convenient property.

This definition of geodesic provided a point and a direction gives a natural mapping from (a subset of) the tangent space $T_u\Omega$ to Ω called the *exponential map* $\exp : B_r(0) \subset T_u\Omega \to \Omega$, which is defined by taking a unit step along the geodesic in the direction X, i.e., $\exp_u(X) = \gamma_X(1)$. The exponential map $\exp_u$ is a local diffeomorphism, as it deforms the neighbourhood $B_r(0)$ (a ball or radius r) of the origin on $T_u\Omega$ into a neighbourhood of u. Conversely, one can also regard the exponential map as an intrinsic local deformation ('flattening') of the manifold into the tangent space.

Note that geodesic completeness does not necessarily guarantee that $\exp$ is a global diffeomorphism – the largest radius r about u for which $\exp_u(B_r(0) \subseteq T_u\Omega)$ is mapped diffeomorphically is called the *injectivity radius*.

Geodesic distances A result known as the Hopf-Rinow Theorem guarantees that geodesically complete manifolds are also *complete metric spaces*, in which one can realise a distance (called the *geodesic distance* or *metric*) between any pair of points u, v as the length of the shortest path between them

$$d_g(u,v) = \min_\gamma \ell(\gamma) \qquad \text{s.t.} \qquad \gamma(0) = u,\ \gamma(T) = v,$$

which exists (i.e., the minimum is attained).

Hopf-Rinow Theorem thus estabilishes the equivalence between geodesic and metric completeness, the latter meaning every Cauchy sequence converges in the geodesic distance metric.

Note that the term 'metric' is used in two senses: Riemannian metric g and distance d. To avoid confusion, we will use the term 'distance' referring to the latter. Our notation d_g makes the distance depend on the Riemannian metric g, though the definition of geodesic length L.

Isometries Consider now a deformation of our manifold Ω into another manifold $\tilde{\Omega}$ with a Riemannian metric h, which we assume to be a diffeomorphism $\eta : (\Omega, g) \to (\tilde{\Omega}, h)$ between the manifolds. Its differential $\mathrm{d}\eta : T\Omega \to T\tilde{\Omega}$ defines a map between the respective tangent bundles (referred to as *pushforward*), such that at a point u, we have $\mathrm{d}\eta_u : T_u\Omega \to T_{\eta(u)}\tilde{\Omega}$, interpreted as before: if we make a small displacement from point u by tangent vector $X \in T_u\Omega$, the map η will be displaced from point $\eta(u)$ by tangent vector $\mathrm{d}\eta_u(X) \in T_{\eta(u)}\tilde{\Omega}$.

Pushforward and pullback are adjoint operators $\langle \eta^*\alpha, X\rangle = \langle \alpha, \eta_* X\rangle$ where $\alpha \in T^*\Omega$ is a *dual vector field*, defined at each point as a linear functional acting on $T_u\Omega$ and the inner products are defined respectively on vector and dual vector fields.

Since the pushforward provides a mechanism to associate tangent vectors on the two manifolds, it allows to *pullback* the metric h from $\tilde{\Omega}$ to Ω,

$$(\eta^* h)_u(X,Y) = h_{\eta(u)}(\mathrm{d}\eta_u(X), \mathrm{d}\eta_u(Y))$$

If the pullback metric coincides at every point with that of Ω, i.e., $g = \eta^* h$, the map η is called (a Riemannian) *isometry*. For two-dimensional manifolds

(surfaces), isometries can be intuitively understood as inelastic deformations that deform the manifold without 'stretching' or 'tearing' it.

By virtue of their definition, isometries preserve intrinsic structures such as geodesic distances, which are expressed entirely in terms of the Riemannian metric. Therefore, we can also understand isometries from the position of metric geometry, as distance-preserving maps ('metric isometries') *between metric spaces* $\eta : (\Omega, d_g) \to (\tilde{\Omega}, d_h)$, in the sense that

$$d_g(u, v) = d_h(\eta(u), \eta(v))$$

for all $u, v \in \Omega$, or more compactly, $d_g = d_h \circ (\eta \times \eta)$. In other words, Riemannian isometries are also metric isometries. On *connected* manifolds, the converse is also true: every metric isometry is also a Riemannian isometry.

This result is known as the Myers–Steenrod Theorem. We tacitly assume our manifolds to be connected.

In our Geometric Deep Learning blueprint, η is a model of domain deformations. When η is an isometry, any intrinsic quantities are unaffected by such deformations. One can generalise exact (metric) isometries through the notions of *metric dilation*

$$\mathrm{dil}(\eta) = \sup_{u \neq v \in \Omega} \frac{d_h(\eta(u), \eta(v))}{d_g(u, v)}$$

or *metric distortion*

$$\mathrm{dis}(\eta) = \sup_{u, v \in \Omega} |d_h(\eta(u), \eta(v)) - d_g(u, v)|,$$

The Gromov-Hausdorff distance between metric spaces, which we mentioned in Section 3.2, can be expressed as the smallest possible metric distortion.

which capture the relative and absolute change of the geodesic distances under η, respectively. The condition (5) for the stability of a function $f \in \mathcal{F}(\mathcal{X}(\Omega))$ under domain deformation can be rewritten in this case as

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

Intrinsic symmetries A particular case of the above is a diffeomorphism of the domain itself (what we termed *automorphism* in Section 3.2), which we will denote by $\tau \in \mathrm{Diff}(\Omega)$. We will call it a Riemannian (self-)isometry if the pullback metric satisfies $\tau^* g = g$, or a metric (self-)isometry if $d_g = d_g \circ (\tau \times \tau)$. Not surprisingly, isometries form a group with the composition operator denoted by $\mathrm{Iso}(\Omega)$ and called the *isometry group*; the identity element is the map $\tau(u) = u$ and the inverse always exists (by definition of τ as a diffeomorphism). Self-isometries are thus *intrinsic symmetries* of manifolds.

Continuous symmetries on manifolds are infinitesimally generated by special tangent vector fields called *Killing fields*, named after Wilhelm Killing.

Fourier analysis on Manifolds We will now show how to construct intrinsic convolution-like operations on manifolds, which, by construction, will be invariant to isometric deformations. For this purpose, we have two options: One is to use an analogy of the Fourier transform, and define the convolution as a product in the Fourier domain. The other is to define the convolution spatially, by correlating a filter locally with the signal. Let us discuss the spectral approach first.

We remind that in the Euclidean domain the Fourier transform is obtained as the eigenvectors of circulant matrices, which are jointly diagonalisable due to their commutativity. Thus, any circulant matrix and in particular, differential operator, can be used to define an analogy of the Fourier transform on general domains. In Riemannian geometry, it is common to use the orthogonal eigenbasis of the Laplacian operator, which we will define here.

For this purpose, recall our definition of the intrinsic gradient operator $\nabla : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, T\Omega)$, producing a tangent vector field that indicates the local direction of steepest increase of a scalar field on the manifold. In a similar manner, we can define the *divergence operator* $\nabla^* : \mathcal{X}(\Omega, T\Omega) \to \mathcal{X}(\Omega, \mathbb{R})$. If we think of a tangent vector field as a flow on the manifold, the divergence measures the net flow of a field at a point, allowing to distinguish between field 'sources' and 'sinks'. We use the notation ∇^* (as opposed to the common div) to emphasise that the two operators are adjoint,

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

where we use the inner products (15) and (16) between scalar and vector fields.

The *Laplacian* (also known as the *Laplace-Beltrami operator* in differential geometry) is an operator on $\mathcal{X}(\Omega)$ defined as $\Delta = \nabla^* \nabla$, which can be interpreted as the difference between the average of a function on an infinitesimal sphere around a point and the value of the function at the point itself. It is one of the most important operators in mathematical physics, used to describe phenomena as diverse as heat diffusion, quantum oscillations, and wave propagation. Importantly in our context, the Laplacian is intrinsic, and thus invariant under isometries of Ω.

From this interpretation it is also clear that the Laplacian is isotropic. We will see in Section 4.6 that it is possible to define *anisotropic Laplacians* (see (Andreux et al., 2014; Boscaini et al., 2016b)) of the form $\nabla^*(A(u)\nabla)$, where $A(u)$ is a position-dependent tensor determining local direction.

It is easy to see that the Laplacian is self-adjoint ('symmetric'),

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

The quadratic form on the left in the above expression is actually the already familiar Dirichlet energy,

$$c^2(x) = \|\nabla x\|^2 = \langle \nabla x, \nabla x \rangle = \int_\Omega \|\nabla x(u)\|_u^2 \mathrm{d}u = \int_\Omega g_u(\nabla x(u), \nabla x(u)) \mathrm{d}u$$

measuring the smoothness of x.

The Laplacian operator admits an eigedecomposition

$$\Delta \varphi_k = \lambda_k \varphi_k, \qquad k = 0, 1, \dots$$

with countable spectrum if the manifold is compact (which we tacitly assume), and orthogonal eigenfunctions, $\langle \varphi_k, \varphi_l \rangle = \delta_{kl}$, due to the self-adjointness of Δ. The Laplacian eigenbasis can also be constructed as a set of orthogonal minimisers of the Dirichlet energy,

$$\varphi_{k+1} = \arg\min_{\varphi} \|\nabla \varphi\|^2 \qquad \text{s.t.} \qquad \|\varphi\| = 1 \ \text{ and } \ \langle \varphi, \varphi_j \rangle = 0$$

for $j = 0, \dots, k$, allowing to interpret it as the smoothest orthogonal basis on Ω. The eigenfunctions $\varphi_0, \varphi_1, \dots$ and the corresponding eigenvalues $0 = \lambda_0 \leq \lambda_1 \leq \dots$ can be interpreted as the analogy of the atoms and frequencies in the classical Fourier transform.

In fact $e^{\mathrm{i}\xi u}$ are the eigenfunctions of the Euclidean Laplacian $\frac{\mathrm{d}^2}{\mathrm{d}u^2}$.

This orthogonal basis allows to expand square-integrable functions on Ω into *Fourier series*

$$x(u) = \sum_{k \geq 0} \langle x, \varphi_k \rangle \varphi_k(u)$$

where $\hat{x}_k = \langle x, \varphi_k \rangle$ are referred to as the *Fourier coefficient* or the (generalised) Fourier transform of x. Truncating the Fourier series results in an approximation error that can be bounded (Aflalo and Kimmel, 2013) by

Note that this Fourier transform has a discrete index, since the spectrum is discrete due to the compactness of Ω.

$$\left\| x - \sum_{k=0}^{N} \langle x, \varphi_k \rangle \varphi_k \right\|^2 \leq \frac{\|\nabla x\|^2}{\lambda_{N+1}}.$$

Aflalo et al. (2015) further showed that no other basis attains a better error, making the Laplacian eigenbasis *optimal* for representing smooth signals on manifolds.

Spectral Convolution on Manifolds *Spectral convolution* can be defined as the product of Fourier transforms of the signal x and the filter θ,

$$(x \star \theta)(u) = \sum_{k\geq 0} (\hat{x}_k \cdot \hat{\theta}_k)\varphi_k(u). \tag{17}$$

Note that here we use what is a *property* of the classical Fourier transform (the Convolution Theorem) as a way to *define* a non-Euclidean convolution. By virtue of its construction, the spectral convolution is intrinsic and thus isometry-invariant. Furthermore, since the Laplacian operator is isotropic, it has no sense of direction; in this sense, the situation is similar to that we had on graphs in Section 4.1 due to permutation invariance of neighbour aggregation.

Figure 12: Instability of spectral filters under domain perturbation. Left: a signal $\mathbf{x}$ on the mesh Ω. Middle: result of spectral filtering in the eigenbasis of the Laplacian Δ on Ω. Right: the same spectral filter applied to the eigenvectors of the Laplacian $\tilde{\Delta}$ of a nearly-isometrically perturbed domain $\tilde{\Omega}$ produces a very different result.

In practice, a direct computation of (17) appears to be prohibitively expensive due to the need to diagonalise the Laplacian. Even worse, it turns out geometrically unstable: the higher-frequency eigenfunctions of the Laplacian can change dramatically as a result of even small near-isometric perturbations of the domain Ω (see Figure 12). A more stable solution is provided by realising the filter as a *spectral transfer function* of the form $\hat{p}(\Delta)$,

$$(\hat{p}(\Delta)x)(u) = \sum_{k\geq 0} \hat{p}(\lambda_k)\langle x, \varphi_k\rangle \varphi_k(u) \tag{18}$$

$$= \int_\Omega x(v) \sum_{k\geq 0} \hat{p}(\lambda_k)\varphi_k(v)\varphi_k(u)\,\mathrm{d}v \tag{19}$$

which can be interpreted in two manners: either as a spectral filter (18), where we identify $\hat{\theta}_k = \hat{p}(\lambda_k)$, or as a spatial filter (19) with a position-dependent kernel $\theta(u, v) = \sum_{k\geq 0} \hat{p}(\lambda_k)\varphi_k(v)\varphi_k(u)$. The advantage of this

formulation is that $\hat{p}(\lambda)$ can be parametrised by a small number of coefficients, and choosing parametric functions such as polynomials $\hat{p}(\lambda) = \sum_{l=0}^{r} \alpha_l \lambda^l$ allows for efficiently computing the filter as

$$(\hat{p}(\Delta)x)(u) = \sum_{k\geq 0} \sum_{l=0}^{r} \alpha_l \lambda_k^l \langle x, \varphi_k \rangle \varphi_k(u) = \sum_{l=0}^{r} \alpha_l (\Delta^l x)(u),$$

avoiding the spectral decomposition altogether. We will discuss this construction in further detail in Section 4.6.

Geometric Deep Learning methods based on spectral convolution expressed through the Fourier transform are often referred to as 'spectral' and opposed to 'spatial' methods we have seen before in the context of graphs. We see here that these two views may be equivalent, so this dichotomy is somewhat artificial and not completely appropriate.

Spatial Convolution on Manifolds A second alternative is to attempt defining convolution on manifolds is by matching a filter at different points, like we did in formula (14),

$$(x \star \theta)(u) = \int_{T_u\Omega} x(\exp_u Y)\theta_u(Y)\mathrm{d}Y, \tag{20}$$

where we now have to use the exponential map to access the values of the scalar field x from the tangent space, and the filter θ_u is defined in the tangent space at each point and hence position-dependent. If one defines the filter intrinsically, such a convolution would be isometry-invariant, a property we mentioned as crucial in many computer vision and graphics applications.

We need, however, to note several substantial differences from our previous construction in Sections 4.2–4.3. First, because a manifold is generally not a homogeneous space, we do not have anymore a global group structure allowing us have a shared filter (i.e., the same θ at every u rather than θ_u in expression (20)) defined at one point and then move it around. An analogy of this operation on the manifold would require parallel transport, allowing to apply a shared θ, defined as a function on $T_u\Omega$, at some other $T_v\Omega$. However, as we have seen, this in general will depend on the path between u and v, so *the way we move the filter around matters*.Third, since we can use the exponential map only locally, the filter must be *local*, with support bounded by the injectivity radius. Fourth and most crucially, we cannot work with $\theta(X)$, as X is an abstract geometric object: in order for it to be used for computations, we must represent it *relative to some local basis* $\omega_u : \mathbb{R}^s \to T_u\Omega$, as an s-dimensional array of coordinates $\mathbf{x} = \omega_u^{-1}(X)$. This allows us to rewrite the convolution (20) as

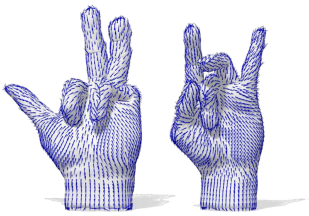
$$(x \star \theta)(u) = \int_{[0,1]^s} x(\exp_u(\omega_u \mathbf{y}))\theta(\mathbf{y})\mathrm{d}\mathbf{y}, \tag{21}$$

with the filter defined on the unit cube. Since the exponential map is intrinsic (through the definition of geodesic), the resulting convolution is isometry-invariant.

Yet, this tacitly assumed we can carry the frame ω_u along to another manifold, i.e. $\omega'_u = \mathrm{d}\eta_u \circ \omega_u$. Obtaining such a frame (or *gauge*, in physics terminology) given only a the manifold Ω in a consistent manner is however fraught with difficulty. First, a smooth global gauge may not exist: this is the situation on manifolds that are not *parallelisable*, in which case one cannot define a smooth non-vanishing tangent vector field. Second, we do not have a canonical gauge on manifolds, so this choice is arbitrary; since our convolution depends on ω, if one chose a different one, we would obtain different results.

The sphere $\mathbb{S}^2$ is an example of a non-parallelisable manifold, a result of the *Poincaré-Hopf Theorem*, which is colloquially stated as 'one cannot comb a hairy ball without creating a cowlick.'

We should note that this is a case where practice diverges from theory: in practice, it is possible to build frames that are mostly smooth, with a limited number of singularities, e.g. by taking the intrinsic gradient of some intrinsic scalar field on the manifold. Moreover, such constructions are stable, i.e., the frames constructed this way will be identical on isometric manifolds and similar on approximately isometric ones. Such approaches were in fact employed in the early works on deep learning on manifolds (Masci et al., 2015; Monti et al., 2017).



Example of stable gauges constructed on nearly-isometric manifolds (only one axis is shown) using the GFrames algorithm of Melzi et al. (2019).

Nevertheless, this solution is not entirely satisfactory because near singularities, the filter orientation (being defined in a fixed manner relative to the gauge) will vary wildly, leading to a non-smooth feature map even if the input signal and filter are smooth. Moreover, there is no clear reason why a given direction at some point u should be considered equivalent to another direction at an altogether different point v. Thus, despite *practical* alternatives, we will look next for a more *theoretically* well-founded approach that would be altogether independent on the choice of gauge.

4.5 Gauges and Bundles

Historically, fibre bundles arose first in modern differential geometry of Élie Cartan (who however did not define them explicitly), and were then further developed as a standalone object in the field of topology in the 1930s.

The notion of gauge, which we have defined as a frame for the tangent space, is quite a bit more general in physics: it can refer to a frame for any *vector bundle*, not just the tangent bundle. Informally, a vector bundle describes a family of vector spaces parametrised by another space and consists of a *base space* Ω with an identical vector space $\mathbb{V}$ (called the *fibre*) attached to each position $u \in \Omega$ (for the tangent bundle these are the tangent spaces

$T_u\Omega$). Roughly speaking, a bundle looks as a product $\Omega \times \mathbb{V}$ locally around u, but globally might be 'twisted' and have an overall different structure. In Geometric Deep Learning, fibres can be used to model the feature spaces at each point in the manifold Ω, with the dimension of the fibre being equal to the number of feature channels. In this context, a new and fascinating kind of symmetry, called *gauge symmetry* may present itself.

Let us consider again an s-dimensional manifold Ω with its tangent bundle $T\Omega$, and a vector field $X : \Omega \to T\Omega$ (which in this terminology is referred to as a *section* on the tangent bundle). Relative to a gauge ω for the tangent bundle, X is represented as a function $\mathbf{x} : \Omega \to \mathbb{R}^s$. However it is important to realise that what we are really interested in is the underlying geometrical object (vector field), whose representation as a function $\mathbf{x} \in \mathcal{X}(\Omega, \mathbb{R}^s)$ *depends on the choice of gauge* ω. If we change the gauge, we also need to change $\mathbf{x}$ so as to preserve the underlying vector field being represented.

Tangent bundles and the Structure group When we change the gauge, we need to apply at each point an invertible matrix that maps the old gauge to the new one. This matrix is unique for every pair of gauges at each point, but possibly different at different points. In other words, a *gauge transformation* is a mapping $\mathfrak{g} : \Omega \to \mathrm{GL}(s)$, where $\mathrm{GL}(s)$ is the *general linear group* of invertible $s \times s$ matrices. It acts on the gauge $\omega_u : \mathbb{R}^s \to T_u\Omega$ to produce a new gauge $\omega'_u = \omega_u \circ \mathfrak{g}_u : \mathbb{R}^s \to T_u\Omega$. The gauge transformation acts on a coordinate vector field at each point via $\mathbf{x}'(u) = \mathfrak{g}_u^{-1}\mathbf{x}(u)$ to produce the coordinate representation $\mathbf{x}'$ of X relative to the new gauge. The underlying vector field remains unchanged:

$$X(u) = \omega'_u(\mathbf{x}'(u)) = \omega_u(\mathfrak{g}_u\mathfrak{g}_u^{-1}\mathbf{x}(u)) = \omega_u(\mathbf{x}(u)) = X(u),$$

which is exactly the property we desired. More generally, we may have a field of geometric quantities that transform according to a representation ρ of $\mathrm{GL}(s)$, e.g. a field of 2-tensors (matrices) $\mathbf{A}(u) \in \mathbb{R}^{s\times s}$ that transform like $\mathbf{A}'(u) = \rho_2(\mathfrak{g}_u^{-1})\mathbf{A}(u) = \rho_1(\mathfrak{g}_u)\mathbf{A}(u)\rho_1(\mathfrak{g}_u^{-1})$. In this case, the gauge transformation $\mathfrak{g}_u$ acts via $\rho(\mathfrak{g}_u)$.

Sometimes we may wish to restrict attention to frames with a certain property, such as orthogonal frames, right-handed frames, etc. Unsurprisingly, we are interested in a set of some property-preserving transformations that form a group. For instance, the group that preserves orthogonality is the orthogonal group $\mathrm{O}(s)$ (rotations and reflections), and the group that additionally

preserves orientation or 'handedness' is $\mathrm{SO}(s)$ (pure rotations). Thus, in general we have a group $\mathfrak{G}$ called the *structure group* of the bundle, and a gauge transformation is a map $\mathfrak{g} : \Omega \to \mathfrak{G}$. A key observation is that in all cases with the given property, for any two frames at a given point there exists exactly one gauge transformation relating them.

We use s to denote the dimension of the base space Ω and d referring to the dimension of the fibre. For tangent bundles, $d = s$ is the dimension of the underlying manifold. For RGB image, $s = 2$ and $d = 3$.

As mentioned before, gauge theory extends beyond tangent bundles, and in general, we can consider a bundle of vector spaces whose structure and dimensions are not necessarily related to those of the base space Ω. For instance, a color image pixel has a position $u \in \Omega = \mathbb{Z}^2$ on a 2D grid and a value $\mathbf{x}(u) \in \mathbb{R}^3$ in the RGB space, so the space of pixels can be viewed as a vector bundle with base space $\mathbb{Z}^2$ and a fibre $\mathbb{R}^3$ attached at each point. It is customary to express an RGB image relative to a gauge that has basis vectors for R, G, and B (in that order), so that the coordinate representation of the image looks like $\mathbf{x}(u) = (r(u), g(u), b(u))^\top$. But we may equally well permute the basis vectors (color channels) independently at each position, as long as we remember the frame (order of channels) in use at each point. As a computational operation this is rather pointless, but as we will see shortly it is conceptually useful to think about gauge transformations for the space of RGB colors, because it allows us to express a gauge symmetry – in this case, an equivalence between colors – and make functions defined on images respect this symmetry (treating each color equivalently).

In this example we have chosen $\mathfrak{G} = \Sigma_3$ the permutations of the 3 color channels as the structure group of the bundle. Other choices, such as a Hue rotation $\mathfrak{G} = \mathrm{SO}(2)$ are also possible.

As in the case of a vector field on a manifold, an RGB gauge transformation changes the numerical representation of an image (permuting the RGB values independently at each pixel) but not the underlying image. In machine learning applications, we are interested in constructing functions $f \in \mathcal{F}(\mathcal{X}(\Omega))$ on such images (e.g. to perform image classification or segmentation), implemented as layers of a neural network. It follows that if, for whatever reason, we were to apply a gauge transformation to our image, we would need to also change the function f (network layers) so as to preserve their meaning. Consider for simplicity a 1×1 convolution, i.e. a map that takes an RGB pixel $\mathbf{x}(u) \in \mathbb{R}^3$ to a feature vector $\mathbf{y}(u) \in \mathbb{R}^C$. According to our Geometric Deep Learning blueprint, the output is associated with a group representation ρ_{out}, in this case a C-dimensional representation of the structure group $\mathfrak{G} = \Sigma_3$ (RGB channel permutations), and similarly the input is associated with a representation $\rho_{\text{in}}(\mathfrak{g}) = \mathfrak{g}$. Then, if we apply a gauge transformation to the input, we would need to change the linear map (1×1 convolution) $f : \mathbb{R}^3 \to \mathbb{R}^C$ to $f' = \rho_{\text{out}}^{-1}(\mathfrak{g}) \circ f \circ \rho_{\text{in}}(\mathfrak{g})$ so that the output feature vector $\mathbf{y}(u) = f(\mathbf{x}(u))$ transforms like $\mathbf{y}'(u) = \rho_{\text{out}}(\mathfrak{g}_u)\mathbf{y}(u)$ at every

point. Indeed we verify:

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

Here the notation $\rho^{-1}(\mathfrak{g})$ should be understood as the inverse of the group representation (matrix) $\rho(\mathfrak{g})$.

Gauge Symmetries To say that we consider gauge transformations to be symmetries is to say that any two gauges related by a gauge transformation are to be considered equivalent. For instance, if we take $\mathfrak{G} = \mathrm{SO}(d)$, any two right-handed orthogonal frames are considered equivalent, because we can map any such frame to any other such frame by a rotation. In other words, there are no distinguished local directions such as "up" or "right". Similarly, if $\mathfrak{G} = \mathrm{O}(d)$ (the orthogonal group), then any left and right handed orthogonal frame are considered equivalent. In this case, there is no preferred orientation either. In general, we can consider a group $\mathfrak{G}$ and a collection of frames at every point u such that for any two of them there is a unique $\mathfrak{g}(u) \in \mathfrak{G}$ that maps one frame onto the other.

Regarding gauge transformations as symmetries in our Geometric Deep Learning blueprint, we are interested in making the functions f acting on signals defined on Ω and expressed with respect to the gauge should equivariant to such transformation. Concretely, this means that if we apply a gauge transformation to the input, the output should undergo the same transformation (perhaps acting via a different representation of $\mathfrak{G}$). We noted before that when we change the gauge, the function f should be changed as well, but for a gauge equivariant map this is not the case: changing the gauge leaves the mapping invariant. To see this, consider again the RGB color space example. The map $f : \mathbb{R}^3 \to \mathbb{R}^C$ is equivariant if $f \circ \rho_{\text{in}}(\mathfrak{g}) = \rho_{\text{out}}(\mathfrak{g}) \circ f$, but in this case the gauge transformation applied to f has no effect: $\rho_{\text{out}}^{-1}(\mathfrak{g}) \circ f \circ \rho_{\text{in}}(\mathfrak{g}) = f$. In other words, the coordinate expression of a gauge equivariant map is independent of the gauge, in the same way that in the case of graph, we applied the same function regardless of how the input nodes were permuted. However, unlike the case of graphs and other examples covered so far, gauge transformations act *not on* Ω but separately *on each of the feature vectors* $\mathbf{x}(u)$ by a transformation $\mathfrak{g}(u) \in \mathfrak{G}$ for each $u \in \Omega$.

Further considerations enter the picture when we look at filters on manifolds with a larger spatial support. Let us first consider an easy example of a mapping $f : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, \mathbb{R})$ from scalar fields to scalar fields on an s-dimensional manifold Ω. Unlike vectors and other geometric quantities, scalars do not have an orientation, so a scalar field $x \in \mathcal{X}(\Omega, \mathbb{R})$ is *invariant* to gauge transformations (it transforms according to the trivial representation

$\rho(\mathfrak{g}) = 1$). Hence, any linear map from scalar fields to scalar fields is *gauge equivariant* (or invariant, which is the same in this case). For example, we could write f similarly to (19), as a convolution-like operation with a position-dependent filter $\theta : \Omega \times \Omega \to \mathbb{R}$,

$$(x \star \theta)(u) = \int_\Omega \theta(u, v) x(v) \mathrm{d}v. \tag{22}$$

This implies that we have a potentially different filter $\theta_u = \theta(u, \cdot)$ at each point, i.e., no spatial weight sharing — which gauge symmetry alone does not provide.

Consider now a more interesting case of a mapping $f : \mathcal{X}(\Omega, T\Omega) \to \mathcal{X}(\Omega, T\Omega)$ from vector fields to vector fields. Relative to a gauge, the input and output vector fields $X, Y \in \mathcal{X}(\Omega, T\Omega)$ are vector-valued functions $\mathbf{x}, \mathbf{y} \in \mathcal{X}(\Omega, \mathbb{R}^s)$. A general linear map between such functions can be written using the same equation we used for scalars (22), only replacing the scalar kernel by a matrix-valued one $\mathbf{\Theta} : \Omega \times \Omega \to \mathbb{R}^{s \times s}$. The matrix $\mathbf{\Theta}(u, v)$ should map tangent vectors in $T_v\Omega$ to tangent vectors in $T_u\Omega$, but these points have *different gauges* that we may change *arbitrarily and independently.* That is, the filter would have to satisfy $\mathbf{\Theta}(u, v) = \rho^{-1}(\mathfrak{g}(u))\mathbf{\Theta}(u, v)\rho(\mathfrak{g}(v))$ for all $u, v \in \Omega$, where ρ denotes the action of $\mathfrak{G}$ on vectors, given by an $s \times s$ rotation matrix. Since $\mathfrak{g}(u)$ and $\mathfrak{g}(v)$ can be chosen freely, this is an overly strong constraint on the filter.

Indeed $\mathbf{\Theta}$ would have to be zero in this case

A better approach is to first transport the vectors to a common tangent space by means of the connection, and then impose gauge equivariance w.r.t. a single gauge transformation at one point only. Instead of (22), we can then define the following map between vector fields,

$$(\mathbf{x} \star \mathbf{\Theta})(u) = \int_\Omega \mathbf{\Theta}(u, v)\rho(\mathfrak{g}_{v \to u})\mathbf{x}(v)\mathrm{d}v, \tag{23}$$

where $\mathfrak{g}_{v \to u} \in \mathfrak{G}$ denotes the parallel transport from v to u along the geodesic connecting these two points; its representation $\rho(\mathfrak{g}_{v \to u})$ is an $s \times s$ rotation matrix rotating the vector as it moves between the points. Note that this geodesic is assumed to be unique, which is true only locally and thus the filter must have a local support. Under a gauge transformation $\mathfrak{g}_u$, this element transforms as $\mathfrak{g}_{u \to v} \mapsto \mathfrak{g}_u^{-1}\mathfrak{g}_{u \to v}\mathfrak{g}_v$, and the field itself transforms as $\mathbf{x}(v) \mapsto \rho(\mathfrak{g}_v)\mathbf{x}(v)$. If the filter commutes with the structure group representation $\mathbf{\Theta}(u, v)\rho(\mathfrak{g}_u) = \rho(\mathfrak{g}_u)\mathbf{\Theta}(u, v)$, equation (23) defines a *gauge-equivariant convolution*, which transforms as

$$(\mathbf{x}' \star \mathbf{\Theta})(u) = \rho^{-1}(\mathfrak{g}_u)(\mathbf{x} \star \mathbf{\Theta})(u).$$

under the aforementioned transformation.

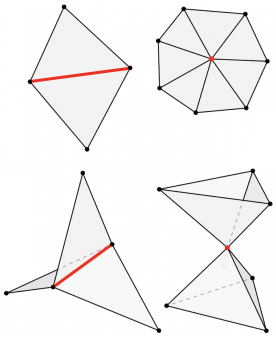
4.6 Geometric graphs and Meshes

We will conclude our discussion of different geometric domains with *geometric graphs* (i.e., graphs that can be realised in some geometric space) and *meshes*. In our '5G' of geometric domains, meshes fall somewhere between graphs and manifolds: in many regards, they are similar to graphs, but their additional structure allows to also treat them similarly to continuous objects. For this reason, we do not consider meshes as a standalone object in our scheme, and in fact, will emphasise that many of the constructions we derive in this section for meshes are directly applicable to general graphs as well.

As we already mentioned in Section 4.4, two-dimensional manifolds (surfaces) are a common way of modelling 3D objects (or, better said, the boundary surfaces of such objects). In computer graphics and vision applications, such surfaces are often discretised as *triangular meshes*, which can be roughly thought of as a piece-wise planar approximation of a surface obtained by gluing triangles together along their edges. Meshes are thus (undirected) *graphs with additional structure*: in addition to nodes and edges, a mesh $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ also have ordered triplets of nodes forming *triangular faces* $\mathcal{F} = \{(u, v, q) : u, v, q \in \mathcal{V} \text{ and } (u, v), (u, q), (q, v) \in \mathcal{E}\}$; the order of the nodes defines the face *orientation*.

Triangular meshes are examples of topological structures known as *simplicial complexes*.

It is further assumed that that each edge is shared by exactly two triangles, and the boundary of all triangles incident on each node forms a single loop of edges. This condition guarantees that 1-hop neighbourhoods around each node are disk-like and the mesh thus constitutes a *discrete manifold* – such meshes are referred to as *manifold meshes*. Similarly to Riemannian manifolds, we can define a *metric* on the mesh. In the simplest instance, it can be induced from the embedding of the mesh nodes $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and expressed through the Euclidean length of the edges, $\ell_{uv} = \|\mathbf{x}_u - \mathbf{x}_v\|$. A metric defined in this way automatically satisfies properties such as the *triangle inequality*, i.e., expressions of the form $\ell_{uv} \leq \ell_{uq} + \ell_{vq}$ for any $(u, v, q) \in \mathcal{F}$ and any combination of edges. Any property that can be expressed solely in terms of ℓ is *intrinsic*, and any deformation of the mesh preserving ℓ is an *isometry* – these notions are already familiar to the reader from our discussion in Section 4.4.



Examples of manifold (top) and non-manifold (bottom) edges and nodes. For manifolds with boundary, one further defines *boundary edges* that belong to exactly one triangle.

Laplacian matrices By analogy to our treatment of graphs, let us assume a (manifold) mesh with n nodes, each associated with a d-dimensional feature vector, which we can arrange (assuming some arbitrary ordering) into an $n \times d$ matrix $\mathbf{X}$. The features can represent the geometric coordinates of the nodes as well as additional properties such as colors, normals, etc, or in specific applications such as chemistry where geometric graphs model molecules, properties such as the atomic number.

Let us first look at the spectral convolution (17) on meshes, which we remind the readers, arises from the Laplacian operator. Considering the mesh as a discretisation of an underlying continuous surface, we can discretise the Laplacian as

$$(\boldsymbol{\Delta}\mathbf{X})_u = \sum_{v \in \mathcal{N}_u} w_{uv}(\mathbf{x}_u - \mathbf{x}_v), \tag{24}$$

or in matrix-vector notation, as an $n \times n$ symmetric matrix $\boldsymbol{\Delta} = \mathbf{D} - \mathbf{W}$, where $\mathbf{D} = \mathrm{diag}(d_1, \dots, d_n)$ is called the *degree matrix* and $d_u = \sum_v w_{uv}$ the *degree* of node u. It is easy to see that equation (24) performs local permutation-invariant aggregation of neighbour features $\phi(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u}) = d_u \mathbf{x}_u - \sum_{v \in \mathcal{N}_u} w_{uv} \mathbf{x}_v$, and $\mathbf{F}(\mathbf{X}) = \boldsymbol{\Delta}\mathbf{X}$ is in fact an instance of our general blueprint (13) for constructing permutation-equivariant functions on graphs.

Note that insofar there is nothing *specific to meshes* in our definition of Laplacian in (24); in fact, this construction is valid for arbitrary graphs as well, with edge weights identified with the adjacency matrix, $\mathbf{W} = \mathbf{A}$, i.e., $w_{uv} = 1$ if $(u, v) \in \mathcal{E}$ and zero otherwise. Laplacians constructed in this way are often called *combinatorial*, to reflect the fact that they merely capture the connectivity structure of the graph. For geometric graphs (which do not necessarily have the additional structure of meshes, but whose nodes do have spatial coordinates that induces a metric in the form of edge lengths), it is common to use weights inversely related to the metric, e.g. $w_{uv} \propto e^{-\ell_{uv}}$.

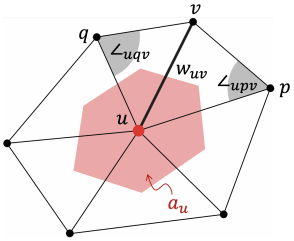
The degree in this case equals the number of neighbours.

If the graph is directed, the corresponding Laplacian is non-symmetric.

On meshes, we can exploit the additional structure afforded by the faces, and define the edge weights in equation (24) using the *cotangent formula* (Pinkall and Polthier, 1993; Meyer et al., 2003)

$$w_{uv} = \frac{\cot \angle_{uqv} + \cot \angle_{upv}}{2a_u} \tag{25}$$

where $\angle_{uqv}$ and $\angle_{upv}$ are the two angles in the triangles (u, q, v) and (u, p, v) opposite the shared edge (u, v), and a_u is the local area element, typically



The earliest use of this formula dates back to the PhD thesis of MacNeal (1949), who developed it to solve PDEs on the Caltech Electric Analog Computer.

computed as the area of the polygon constructed upon the barycenters of the triangles (u, p, q) sharing the node u and given by $a_u = \frac{1}{3}\sum_{v,q:(u,v,q)\in\mathcal{F}} a_{uvq}$.

The cotangent Laplacian can be shown to have multiple convenient properties (see e.g. Wardetzky et al. (2007)): it is a *positive-semidefinite* matrix, $\mathbf{\Delta} \succcurlyeq 0$ and thus has non-negative eigenvalues $\lambda_1 \leq \ldots \leq \lambda_n$ that can be regarded as an analogy of frequency, it is symmetric and thus has orthogonal eigenvectors, and it is *local* (i.e., the value of $(\mathbf{\Delta X})_u$ depends only on 1-hop neighbours, $\mathcal{N}_u$). Perhaps the most important property is the convergence of the cotangent mesh Laplacian matrix $\mathbf{\Delta}$ to the continuous operator Δ when the mesh is infinitely refined (Wardetzky, 2008). Equation (25) constitutes thus an appropriate *discretisation* of the Laplacian operator defined on Riemannian manifolds in Section 4.4.

Some technical conditions must be imposed on the refinement, to avoid e.g. triangles becoming pathological. One such example is a bizarre triangulation of the cylinder known in German as the *Schwarzscher Stiefel* (Schwarz's boot) or in English literature as the 'Schwarz lantern', proposed in 1880 by Hermann Schwarz, a German mathematician known from the Cauchy-Schwarz inequality fame.

While one expects the Laplacian to be intrinsic, this is not very obvious from equation (25), and it takes some effort to express the cotangent weights entirely in terms of the discrete metric ℓ as

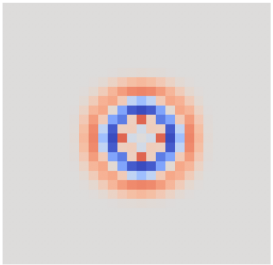
$$w_{uv} = \frac{-\ell_{uv}^2 + \ell_{vq}^2 + \ell_{uq}^2}{8a_{uvq}} + \frac{-\ell_{uv}^2 + \ell_{vp}^2 + \ell_{up}^2}{8a_{uvp}}$$

where the area of the triangles a_{ijk} is given as

$$a_{uvq} = \sqrt{s_{uvq}(s_{uvq} - \ell_{uv})(s_{uvq} - \ell_{vq})(s_{uvq} - \ell_{uq})}$$

using *Heron's semiperimeter formula* with $s_{uvq} = \frac{1}{2}(\ell_{uv} + \ell_{uq} + \ell_{vq})$. This endows the Laplacian (and any quantities associated with it, such as its eigenvectors and eigenvalues) with *isometry invariance*, a property for which it is so loved in geometry processing and computer graphics (see an excellent review by Wang and Solomon (2019)): any deformation of the mesh that does not affect the metric ℓ (does not 'stretch' or 'squeeze' the edges of the mesh) does not change the Laplacian.

Finally, as we already noticed, the definition of the Laplacian (25) is invariant to the permutation of nodes in $\mathcal{N}_u$, as it involves aggregation in the form of summation. While on general graphs this is a necessary evil due to the lack of canonical ordering of neighbours, on meshes we can order the 1-hop neighbours according to some orientation (e.g., clock-wise), and the only ambiguity is the selection of the first node. Thus, instead of any possible permutation we need to account for *cyclic shifts* (rotations), which intuitively corresponds to the ambiguity arising from $\mathrm{SO}(2)$ gauge transformations



Laplacian-based filters are isotropic. In the plane, such filters have radial symmetry.

discussed in Section 4.5. For a fixed gauge, it is possible to define an *anisotropic Laplacian* that is sensitive to local directions and amounts to changing the metric or the weights w_{uv}. Constructions of this kind were used to design shape descriptors by Andreux et al. (2014); Boscaini et al. (2016b) and in early Geometric Deep Learning architectures on meshes by Boscaini et al. (2016a).

Spectral analysis on meshes The orthogonal eigenvectors $\boldsymbol{\Phi} = (\boldsymbol{\varphi}_1, \dots, \boldsymbol{\varphi}_n)$ diagonalising the Laplacian matrix ($\boldsymbol{\Delta} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top$, where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ is the diagonal matrix of Laplacian eigenvalues), are used as the non-Euclidean analogy of the Fourier basis, allowing to perform spectral convolution on the mesh as the product of the respective Fourier transforms,

$$\mathbf{X} \star \boldsymbol{\theta} = \boldsymbol{\Phi} \operatorname{diag}(\boldsymbol{\Phi}^\top \boldsymbol{\theta})(\boldsymbol{\Phi}^\top \mathbf{X}) = \boldsymbol{\Phi} \operatorname{diag}(\hat{\boldsymbol{\theta}})\hat{\mathbf{X}},$$

where the filter $\hat{\boldsymbol{\theta}}$ is designed directly in the Fourier domain. Again, nothing in this formula is specific to meshes, and one can use the Laplacian matrix of a generic (undirected) graph. It is tempting to exploit this spectral definition of convolution to generalise CNNs to graphs, which in fact was done by one of the authors of this text, Bruna et al. (2013). However, it appears that the non-Euclidean Fourier transform is extremely sensitive to even minor perturbations of the underlying mesh or graph (see Figure 12 in Section 4.4) and thus can only be used when one has to deal with different signals on a *fixed* domain, but not when one wishes to generalise across *different domains*. Unluckily, many computer graphics and vision problems fall into the latter category, where one trains a neural network on one set of 3D shapes (meshes) and test on a different set, making the Fourier transform-based approach inappropriate.

The fact that the graph is assumed to be undirected is important: in this case the Laplacian is symmetric and has orthogonal eigenvectors.

As noted in Section 4.4, it is preferable to use spectral filters of the form (18) applying some transfer function $\hat{p}(\lambda)$ to the Laplacian matrix,

$$\hat{p}(\boldsymbol{\Delta})\mathbf{X} = \boldsymbol{\Phi}\hat{p}(\boldsymbol{\Lambda})\boldsymbol{\Phi}^\top \mathbf{X} = \boldsymbol{\Phi} \operatorname{diag}(\hat{p}(\lambda_1), \dots, \hat{p}(\lambda_n))\hat{\mathbf{X}}.$$

When $\hat{p}$ can be expressed in terms of matrix-vector products, the eigendecomposition of the $n \times n$ matrix $\boldsymbol{\Delta}$ can be avoided altogether. For example, Defferrard et al. (2016) used *polynomials* of degree r as filter functions,

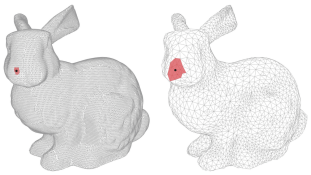
In the general case, the complexity of eigendecomposition is $\mathcal{O}(n^3)$.

$$\hat{p}(\boldsymbol{\Delta})\mathbf{X} = \sum_{k=0}^{r} \alpha_k \boldsymbol{\Delta}^k \mathbf{X} = \alpha_0 \mathbf{X} + \alpha_1 \boldsymbol{\Delta}\mathbf{X} + \ldots + \alpha_r \boldsymbol{\Delta}^r \mathbf{X},$$

amounting to the multiplication of the $n \times d$ feature matrix $\mathbf{X}$ by the $n \times n$ Laplacian matrix r times. Since the Laplacian is typically sparse (with $\mathcal{O}(|\mathcal{E}|)$ non-zero elements) this operation has low complexity of $\mathcal{O}(|\mathcal{E}|dr) \sim \mathcal{O}(|\mathcal{E}|)$. Furthermore, since the Laplacian is local, a polynomial filter of degree r is localised in r-hop neighbourhood.

Meshes are nearly-regular graphs, with each node having $\mathcal{O}(1)$ neighbours, resulting in $\mathcal{O}(n)$ non-zeros in $\mathbf{\Delta}$.

However, this exact property comes at a disadvantage when dealing with meshes, since the actual support of the filter (i.e., the radius it covers) depends on the *resolution* of the mesh. One has to bear in mind that meshes arise from the discretisation of some underlying continuous surface, and one may have two different meshes $\mathcal{T}$ and $\mathcal{T}'$ representing *the same object*. In a finer mesh, one might have to use larger neighbourhoods (thus, larger degree r of the filter) than in a coarser one.



Two-hop neighbourhoods on meshes of different resolution.

For this reason, in computer graphics applications it is more common to use *rational filters*, since they are resolution-independent. There are many ways to define such filters (see, e.g. Patanè (2020)), the most common being as a polynomial of some rational function, e.g., $\frac{\lambda-1}{\lambda+1}$. More generally, one can use a complex function, such as the *Cayley transform* $\frac{\lambda-\mathrm{i}}{\lambda+\mathrm{i}}$ that maps the real line into the unit circle in the complex plane. Levie et al. (2018) used spectral filters expressed as *Cayley polynomials*, real rational functions with complex coefficients $\alpha_l \in \mathbb{C}$,

Cayley transform is a particular case of a *Möbius transformation*. When applied to the Laplacian (a positive-semindefinite matrix), it maps its non-negative eigenvalues to the complex half-circle.

$$\hat{p}(\lambda) = \mathrm{Re}\left(\sum_{l=0}^{r} \alpha_l \left(\frac{\lambda - \mathrm{i}}{\lambda + \mathrm{i}}\right)^l\right).$$

When applied to matrices, the computation of the Cayley polynomial requires matrix inversion,

$$\hat{p}(\mathbf{\Delta}) = \mathrm{Re}\left(\sum_{l=0}^{r} \alpha_l (\mathbf{\Delta} - \mathrm{i}\mathbf{I})^l (\mathbf{\Delta} + \mathrm{i}\mathbf{I})^{-l}\right),$$

In signal processing, polynomial filters are termed *finite impulse response* (FIR), whereas rational filters are *infinite impulse response* (IIR).

which can be carried out approximately with linear complexity. Unlike polynomial filters, rational filters do not have a local support, but have exponential decay (Levie et al., 2018). A crucial difference compared to the direct computation of the Fourier transform is that polynomial and rational filters are stable under approximate isometric deformations of the underlying graph or mesh – various results of this kind were shown e.g. by Levie et al. (2018, 2019); Gama et al. (2020); Kenlay et al. (2021).

Meshes as operators and Functional maps The paradigm of functional maps suggests thinking of meshes as *operators*. As we will show, this allows obtaining more interesting types of invariance exploiting the additional structure of meshes. For the purpose of our discussion, assume the mesh $\mathcal{T}$ is constructed upon embedded nodes with coordinates $\mathbf{X}$. If we construct an intrinsic operator like the Laplacian, it can be shown that it encodes completely the structure of the mesh, and one can recover the mesh (up to its isometric embedding, as shown by Zeng et al. (2012)). This is also true for some other operators (see e.g. Boscaini et al. (2015); Corman et al. (2017); Chern et al. (2018)), so we will assume a general operator, or $n \times n$ matrix $\mathbf{Q}(\mathcal{T}, \mathbf{X})$, as a representation of our mesh.

In this view, the discussion of Section 4.1 of learning functions of the form $f(\mathbf{X}, \mathcal{T})$ can be rephrased as learning functions of the form $f(\mathbf{Q})$. Similar to graphs and sets, the nodes of meshes also have no canonical ordering, i.e., functions on meshes must satisfy the permutation invariance or equivariance conditions,

$$\begin{aligned} f(\mathbf{Q}) &= f(\mathbf{P}\mathbf{Q}\mathbf{P}^\top) \\ \mathbf{P}\mathbf{F}(\mathbf{Q}) &= \mathbf{F}(\mathbf{P}\mathbf{Q}\mathbf{P}^\top) \end{aligned}$$

for any permutation matrix $\mathbf{P}$. However, compared to general graphs we now have more structure: we can assume that our mesh arises from the discretisation of some underlying continuous surface Ω. It is thus possible to have a different mesh $\mathcal{T}' = (\mathcal{V}', \mathcal{E}', \mathcal{F}')$ with n' nodes and coordinates $\mathbf{X}'$ representing the same object Ω as $\mathcal{T}$. Importantly, the meshes $\mathcal{T}$ and $\mathcal{T}'$ can have a different connectivity structure and even different number of nodes ($n' \neq n$). Therefore, we cannot think of these meshes as isomorphic graphs with mere reordering of nodes and consider the permutation matrix $\mathbf{P}$ as correspondence between them.

Functional maps were introduced by Ovsjanikov et al. (2012) as a generalisation of the notion of correspondence to such settings, replacing the correspondence between *points* on two domains (a map $\eta : \Omega \to \Omega'$) with correspondence between *functions* (a map $\mathbf{C} : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$, see Figure 13). A *functional map* is a linear operator $\mathbf{C}$, represented as a matrix $n' \times n$, establishing correspondence between signals $\mathbf{x}'$ and $\mathbf{x}$ on the respective domains as

$$\mathbf{x}' = \mathbf{C}\mathbf{x}.$$

In most cases the functional map is implemented in the spectral domain, as a $k \times k$ map $\hat{\mathbf{C}}$ between the Fourier coefficients, $\mathbf{x}' = \mathbf{\Phi}'\hat{\mathbf{C}}\mathbf{\Phi}^\top\mathbf{x}$, where $\mathbf{\Phi}$ and $\mathbf{\Phi}'$ are the respective $n \times k$ and $n' \times k$ matrices of the (truncated) Laplacian eigenbases, with $k \ll n, n'$.

Rustamov et al. (2013) showed that in order to guarantee *area-preserving* mapping, the functional map must be orthogonal, $\mathbf{C}^\top\mathbf{C} = \mathbf{I}$, i.e., be an

element of the orthogonal group $\mathbf{C} \in \mathrm{O}(n)$. In this case, we can invert the map using $\mathbf{C}^{-1} = \mathbf{C}^\top$.

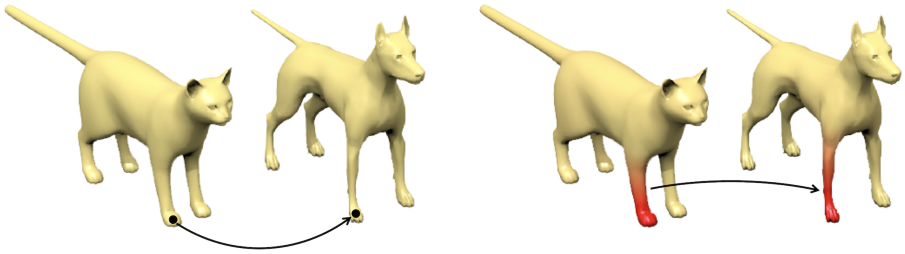


Figure 13: Pointwise map (left) vs functional map (right).

The functional map also establishes a relation between the operator representation of meshes,

$$\mathbf{Q}' = \mathbf{C}\mathbf{Q}\mathbf{C}^\top, \qquad \mathbf{Q} = \mathbf{C}^\top\mathbf{Q}'\mathbf{C},$$

which we can interpret as follows: given an operator representation $\mathbf{Q}$ of $\mathcal{T}$ and a functional map $\mathbf{C}$, we can construct its representation $\mathbf{Q}'$ of $\mathcal{T}'$ by first mapping the signal from $\mathcal{T}'$ to $\mathcal{T}$ (using $\mathbf{C}^\top$), applying the operator $\mathbf{Q}$, and then mapping back to $\mathcal{T}'$ (using $\mathbf{C}$) This leads us to a more general class of *remeshing invariant* (or equivariant) functions on meshes, satisfying

Note that we read these operations *right-to-left*.

$$\begin{aligned} f(\mathbf{Q}) &= f(\mathbf{C}\mathbf{Q}\mathbf{C}^\top) = f(\mathbf{Q}') \\ \mathbf{C}\mathbf{F}(\mathbf{Q}) &= \mathbf{F}(\mathbf{C}\mathbf{Q}\mathbf{C}^\top) = \mathbf{F}(\mathbf{Q}') \end{aligned}$$

for any $\mathbf{C} \in \mathrm{O}(n)$. It is easy to see that the previous setting of permutation invariance and equivariance is a particular case, which can be thought of as a trivial remeshing in which only the order of nodes is changed.

This follows from the orthogonality of permutation matrices, $\mathbf{P}^\top\mathbf{P} = \mathbf{I}$.

Wang et al. (2019a) showed that given an eigendecomposition of the operator $\mathbf{Q} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$, any remeshing invariant (or equivariant) function can be expressed as $f(\mathbf{Q}) = f(\mathbf{\Lambda})$ and $\mathbf{F}(\mathbf{Q}) = \mathbf{V}\mathbf{F}(\mathbf{\Lambda})$, or in other words, remeshing-invariant functions *involve only the spectrum of* $\mathbf{Q}$. Indeed, functions of Laplacian eigenvalues have been proven in practice to be robust to surface discretisation and perturbation, explaining the popularity of spectral constructions based on Laplacians in computer graphics, as well as in deep learning on graph (Defferrard et al., 2016; Levie et al., 2018). Since this result refers to a generic operator $\mathbf{Q}$, multiple choices are available besides the ubiquitous Laplacian – notable examples include the Dirac (Liu et al., 2017; Kostrikov et al., 2018) or Steklov (Wang et al., 2018) operators, as well as learnable parametric operators (Wang et al., 2019a).

5 Geometric Deep Learning Models

Having thoroughly studied various instantiations of our Geometric Deep Learning blueprint (for different choices of domain, symmetry group, and notions of locality), we are ready to discuss how enforcing these prescriptions can yield some of the most popular deep learning architectures.

Our exposition, once again, will not be in strict order of generality. We initially cover three architectures for which the implementation follows nearly-directly from our preceding discussion: convolutional neural networks (CNNs), group-equivariant CNNs, and graph neural networks (GNNs).

We will then take a closer look into variants of GNNs for cases where a graph structure is not known upfront (i.e. unordered sets), and through our discussion we will describe the popular Deep Sets and Transformer architectures as instances of GNNs.

Following our discussion on geometric graphs and meshes, we first describe equivariant message passing networks, which introduce explicit geometric symmetries into GNN computations. Then, we show ways in which our theory of geodesics and gauge symmetries can be materialised within deep learning, recovering a family of intrinsic mesh CNNs (including Geodesic CNNs, MoNet and gauge-equivariant mesh CNNs).

Lastly, we look back on the grid domain from a *temporal* angle. This discussion will lead us to recurrent neural networks (RNNs). We will demonstrate a manner in which RNNs are translation equivariant over temporal grids, but also study their stability to time warping transformations. This property is highly desirable for properly handling long-range dependencies, and enforcing class invariance to such transformations yields exactly the class of gated RNNs (including popular RNN models such as the LSTM or GRU).

While we hope the above canvasses most of the key deep learning architectures in use at the time of writing, we are well aware that novel neural network instances are proposed daily. Accordingly, rather than aiming to cover every possible architecture, we hope that the following sections are illustrative enough, to the point that the reader is able to easily categorise any future Geometric Deep Learning developments using the lens of invariances and symmetries.

How is C(θ) a circulant matrix ?

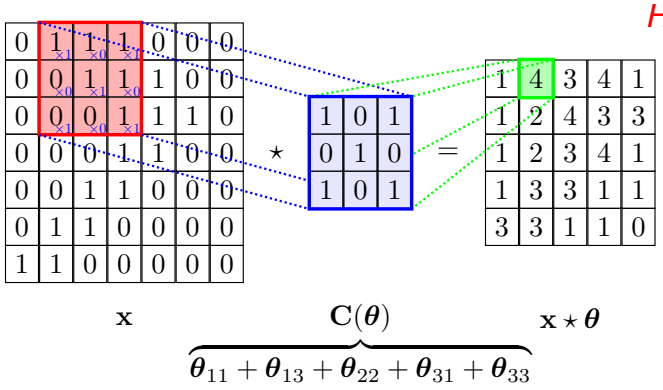


Figure 14: The process of convolving an image $\mathbf{x}$ with a filter $\mathbf{C}(\boldsymbol{\theta})$. The filter parameters $\boldsymbol{\theta}$ can be expressed as a linear combination of generators $\boldsymbol{\theta}_{vw}$.

5.1 Convolutional Neural Networks

Convolutional Neural Networks are perhaps the earliest and most well known example of deep learning architectures following the blueprint of Geometric Deep Learning outlined in Section 3.5. In Section 4.2 we have fully characterised the class of linear and local translation equivariant operators, given by convolutions $\mathbf{C}(\boldsymbol{\theta})\mathbf{x} = \mathbf{x} \star \boldsymbol{\theta}$ with a localised filter $\boldsymbol{\theta}$. Let us first focus on scalar-valued ('single-channel' or 'grayscale') discretised images, where the domain is the grid $\Omega = [H] \times [W]$ with $\mathbf{u} = (u_1, u_2)$ and $\mathbf{x} \in \mathcal{X}(\Omega, \mathbb{R})$.

Recall, $\mathbf{C}(\boldsymbol{\theta})$ is a *circulant* matrix with parameters $\boldsymbol{\theta}$.

Any convolution with a compactly supported filter of size $H^f \times W^f$ can be written as a linear combination of generators $\boldsymbol{\theta}_{1,1}, \dots, \boldsymbol{\theta}_{H^f,W^f}$, given for example by the unit peaks $\boldsymbol{\theta}_{vw}(u_1, u_2) = \delta(u_1 - v, u_2 - w)$. Any local linear equivariant map is thus expressible as

$$\mathbf{F}(\mathbf{x}) = \sum_{v=1}^{H^f} \sum_{w=1}^{W^f} \alpha_{vw} \mathbf{C}(\boldsymbol{\theta}_{vw})\mathbf{x} \,, \tag{26}$$

Note that we usually imagine $\mathbf{x}$ and $\boldsymbol{\theta}_{vw}$ as 2D matrices, but in this equation, both $\mathbf{x}$ and $\boldsymbol{\theta}_{vw}$ have their two coordinate dimensions *flattened* into one—making $\mathbf{x}$ a vector, and $\mathbf{C}(\boldsymbol{\theta}_{vw})$ a matrix.

which, in coordinates, corresponds to the familiar 2D convolution (see Figure 14 for an overview):

$$\mathbf{F}(\mathbf{x})_{uv} = \sum_{a=1}^{H^f} \sum_{b=1}^{W^f} \alpha_{ab} x_{u+a,v+b} \,. \tag{27}$$

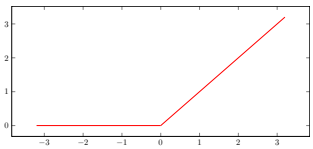
Other choices of the basis $\boldsymbol{\theta}_{vw}$ are also possible and will yield equivalent operations (for potentially different choices of α_{vw}). A popular example are *directional derivatives*: $\boldsymbol{\theta}_{vw}(u_1,u_2) = \delta(u_1,u_2) - \delta(u_1-v,u_2-w), (v,w) \neq (0,0)$ taken together with the local average $\boldsymbol{\theta}_0(u_1,u_2) = \frac{1}{H_f W_f}$. In fact, directional derivatives can be considered a grid-specific analogue of diffusion processes on graphs, which we recover if we assume each pixel to be a node connected to its immediate neighbouring pixels in the grid.

When the scalar input channel is replaced by multiple channels (e.g., RGB colours, or more generally an arbitrary number of *feature maps*), the convolutional filter becomes a *convolutional tensor* expressing arbitrary linear combinations of input features into output feature maps. In coordinates, this can be expressed as

$$\mathbf{F}(\mathbf{x})_{uvj} = \sum_{a=1}^{H^f} \sum_{b=1}^{W^f} \sum_{c=1}^{M} \alpha_{jabc} x_{u+a,v+b,c} \;,\; j \in [N] \;, \tag{28}$$

where M and N are respectively the number of input and output channels. This basic operation encompasses a broad class of neural network architectures, which, as we will show in the next section, have had a profound impact across many areas of computer vision, signal processing, and beyond. Here, rather than dissecting the myriad of possible architectural variants of CNNs, we prefer to focus on some of the essential innovations that enabled their widespread use.

Efficient multiscale computation As discussed in the GDL template for general symmetries, extracting translation invariant features out of the convolutional operator $\mathbf{F}$ requires a non-linear step. Convolutional features are processed through a non-linear *activation function* σ, acting element-wise on the input—i.e., $\sigma : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega)$, as $\sigma(\mathbf{x})(u) = \sigma(\mathbf{x}(u))$. Perhaps the most popular example at the time of writing is the Rectified Linear Unit (ReLU): $\sigma(x) = \max(x, 0)$. This non-linearity effectively *rectifies* the signals, pushing their energy towards lower frequencies, and enabling the computation of high-order interactions across scales by iterating the construction.



ReLU, often considered a 'modern' architectural choice, was already used in the Neocognitron (Fukushima and Miyake, 1982). Rectification is equivalent to the principle of demodulation, which is fundamental in electrical engineering as the basis for many transmission protocols, such as FM radio; and also has a prominent role in models for neuronal activity.

Already in the early works of Fukushima and Miyake (1982) and LeCun et al. (1998), CNNs and similar architectures had a multiscale structure, where after each convolutional layer (28) one performs a grid coarsening $\mathbf{P} : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$, where the grid Ω' has coarser resolution than Ω. This enables

multiscale filters with effectively increasing receptive field, yet retaining a constant number of parameters per scale. Several signal coarsening strategies $\mathbf{P}$ (referred to as *pooling*) may be used, the most common are applying a low-pass anti-aliasing filter (e.g. local average) followed by grid downsampling, or non-linear max-pooling.

In summary, a 'vanilla' CNN layer can be expressed as the composition of the basic objects already introduced in our Geometric Deep Learning blueprint:

$$\mathbf{h} = \mathbf{P}(\sigma(\mathbf{F}(\mathbf{x}))) \,, \tag{29}$$

i.e. an equivariant linear layer $\mathbf{F}$, a coarsening operation $\mathbf{P}$, and a non-linearity σ. It is also possible to perform translation invariant *global* pooling operations within CNNs. Intuitively, this involves each pixel—which, after several convolutions, summarises a *patch* centered around it—*proposing* the final representation of the image, with the ultimate choice being guided by a form of aggregation of these proposals. A popular choice here is the average function, as its outputs will retain similar magnitudes irrespective of the image size (Springenberg et al., 2014).

max-pooling, average-pooling, global pooling (max or average) is translation invariant

CNNs which only consist of the operations mentioned in this paragraph are often dubbed "all-convolutional". In contrast, many CNNs *flatten* the image across the spatial axes and pass them to an MLP classifier, once sufficient equivariant and coarsening layers have been applied. This loses translation invariance.

Prominent examples following this CNN blueprint (some of which we will discuss next) are displayed in Figure 15.

Deep and Residual Networks A CNN architecture, in its simplest form, is therefore specified by hyperparameters $(H_k^f, W_k^f, N_k, p_k)_{k \leq K}$, with $M_{k+1} = N_k$ and $p_k = 0, 1$ indicating whether grid coarsening is performed or not. While all these hyperparameters are important in practice, a particularly important question is to understand the role of depth K in CNN architectures, and what are the fundamental tradeoffs involved in choosing such a key hyperparameter, especially in relation to the filter sizes (H_k^f, W_k^f).

While a rigorous answer to this question is still elusive, mounting empirical evidence collected throughout the recent years suggests a favourable tradeoff towards deeper (large K) yet thinner (small (H_k^f, W_k^f)) models . In this context, a crucial insight by He et al. (2016) was to reparametrise each convolutional layer to model a *perturbation* of the previous features, rather than a generic non-linear transformation:

$$\mathbf{h} = \mathbf{P}\left(\mathbf{x} + \sigma(\mathbf{F}(\mathbf{x}))\right) \,. \tag{30}$$

Historically, ResNet models are predated by *highway networks* (Srivastava et al., 2015), which allow for more general *gating* mechanisms to control the residual information flow.

The resulting *residual* networks provide several key advantages over the previous formulation. In essence, the residual parametrisation is consistent

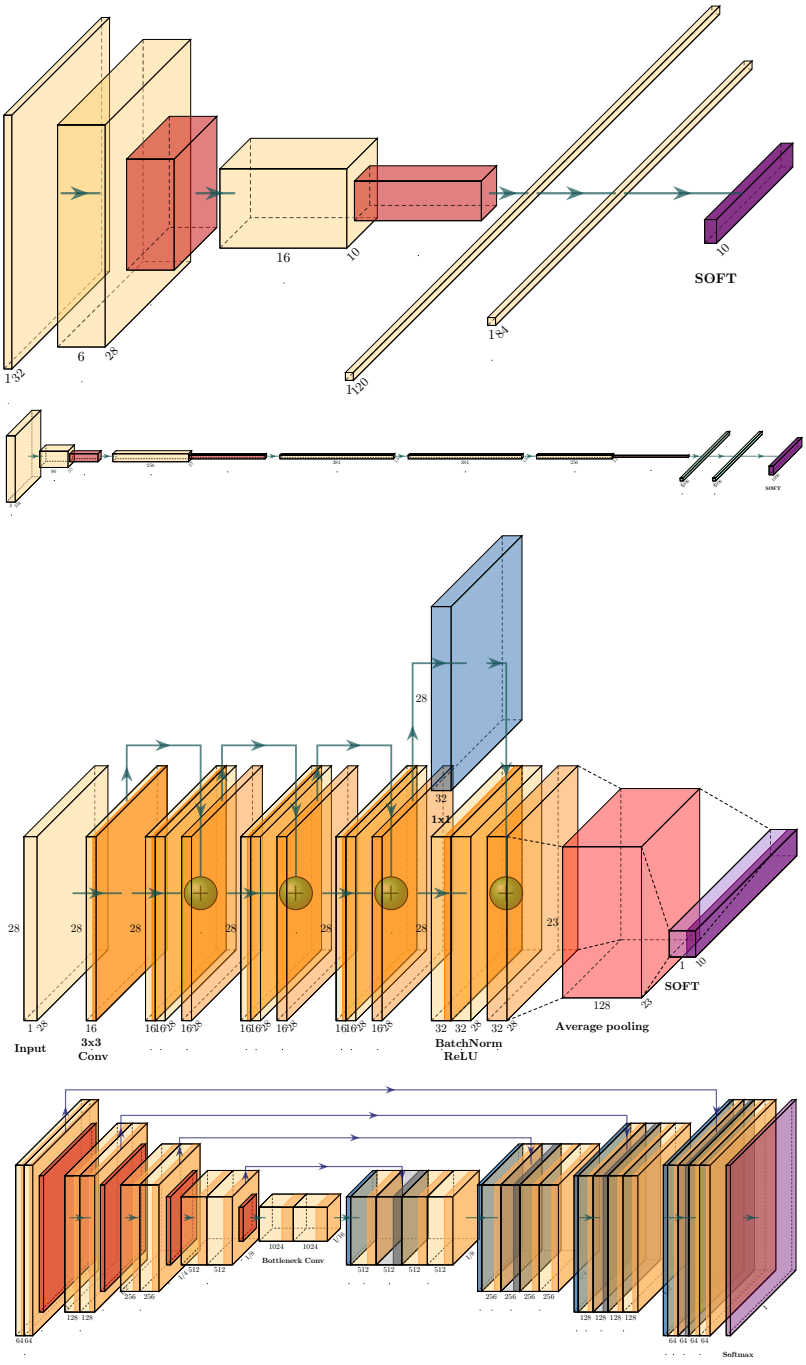


Figure 15: Prominent examples of CNN architectures. **Top-to-bottom**: LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012), ResNet (He et al., 2016) and U-Net (Ronneberger et al., 2015). Drawn using the PlotNeuralNet package (Iqbal, 2018).

with the view that the deep network is a discretisation of an underlying continuous dynamical system, modelled as an ordinary differential equation (ODE). Crucially, learning a dynamical system by modeling its velocity turns out to be much easier than learning its position directly. In our learning setup, this translates into an optimisation landscape with more favorable geometry, leading to the ability to train much deeper architectures than was possible before. As will be discussed in future work, learning using deep neural networks defines a non-convex optimisation problem, which can be efficiently solved using gradient-descent methods under certain simplifying regimes. The key advantage of the ResNet parametrisation has been rigorously analysed in simple scenarios (Hardt and Ma, 2016), and remains an active area of theoretical investigation. Finally, Neural ODEs (Chen et al., 2018) are a recent popular architecture that pushes the analogy with ODEs even further, by learning the parameters of the ODE $\dot{\mathbf{x}} = \sigma(\mathbf{F}(\mathbf{x}))$ directly and relying on standard numerical integration.

In this case, the ResNet is performing a Forward Euler discretisation of an ODE: $\dot{\mathbf{x}} = \sigma(\mathbf{F}(\mathbf{x}))$

Normalisation Another important algorithmic innovation that boosted the empirical performance of CNNs significantly is the notion of *normalisation*. In early models of neural activity, it was hypothesised that neurons perform some form of local 'gain control', where the layer coefficients $\mathbf{x}_k$ are replaced by $\tilde{\mathbf{x}}_k = \sigma_k^{-1} \odot (\mathbf{x}_k - \mu_k)$. Here, μ_k and σ_k encode the first and second-order moment information of $\mathbf{x}_k$, respectively. Further, they can be either computed globally or locally.

In the context of Deep Learning, this principle was widely adopted through the *batch normalisation* layer (Ioffe and Szegedy, 2015), followed by several variants (Ba et al., 2016; Salimans and Kingma, 2016; Ulyanov et al., 2016; Cooijmans et al., 2016; Wu and He, 2018). Despite some attempts to rigorously explain the benefits of normalisation in terms of better conditioned optimisation landscapes (Santurkar et al., 2018), a general theory that can provide guiding principles is still missing at the time of writing.

We note that normalising activations of neural networks has seen attention even before the advent of batch normalisation. See, e.g., Lyu and Simoncelli (2008).

Data augmentation While CNNs encode the geometric priors associated with translation invariance and scale separation, they do not explicitly account for other known transformations that preserve semantic information, e.g lightning or color changes, or small rotations and dilations. A pragmatic approach to incorporate these priors with minimal architectural changes is to perform *data augmentation*, where one manually performs said transfor-

mations to the input images and adds them into the training set.

Data augmentation has been empirically successful and is widely used—not only to train state-of-the-art vision architectures, but also to prop up several developments in self-supervised and causal representation learning (Chen et al., 2020; Grill et al., 2020; Mitrovic et al., 2020). However, it is provably sub-optimal in terms of sample complexity (Mei et al., 2021); a more efficient strategy considers instead architectures with richer invariance groups—as we discuss next.

5.2 Group-equivariant CNNs

As discussed in Section 4.3, we can generalise the convolution operation from signals on a Euclidean space to signals on any *homogeneous space* Ω acted upon by a group $\mathfrak{G}$. By analogy to the Euclidean convolution where a *translated* filter is matched with the signal, the idea of group convolution is to move the filter around the domain using the group action, e.g. by rotating and translating. By virtue of the *transitivity* of the group action, we can move the filter to any position on Ω. In this section, we will discuss several concrete examples of the general idea of group convolution, including implementation aspects and architectural choices.

Recall that a homogeneous space is a set Ω equipped with a transitive group action, meaning that for any $u, v \in \Omega$ there exists $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g}.u = v$.

Discrete group convolution We begin by considering the case where the domain Ω as well as the group $\mathfrak{G}$ are discrete. As our first example, we consider medical volumetric images represented as signals of on 3D grids with discrete translation and rotation symmetries. The domain is the 3D cubical grid $\Omega = \mathbb{Z}^3$ and the images (e.g. MRI or CT 3D scans) are modelled as functions $x : \mathbb{Z}^3 \to \mathbb{R}$, i.e. $x \in \mathcal{X}(\Omega)$. Although in practice such images have support on a finite cuboid $[W] \times [H] \times [D] \subset \mathbb{Z}^3$, we instead prefer to view them as functions on $\mathbb{Z}^3$ with appropriate zero padding. As our symmetry, we consider the group $\mathfrak{G} = \mathbb{Z}^3 \rtimes O_h$ of distance- and orientation-preserving transformations on $\mathbb{Z}^3$. This group consists of translations ($\mathbb{Z}^3$) and the discrete rotations O_h generated by 90 degree rotations about the three axes (see Figure 16).

DNA is a biopolymer molecule made of four repeating units called *nucleotides* (Cytosine, Guanine, Adenine, and Thymine), arranged into two strands coiled around each other in a double helix, where each nucleotide occurs opposite of the complementary one (*base pairs* A/T and C/G).

As our second example, we consider DNA sequences made up of four letters: C, G, A, and T. The sequences can be represented on the 1D grid $\Omega = \mathbb{Z}$ as signals $x : \mathbb{Z} \to \mathbb{R}^4$, where each letter is one-hot coded in $\mathbb{R}^4$. Naturally, we

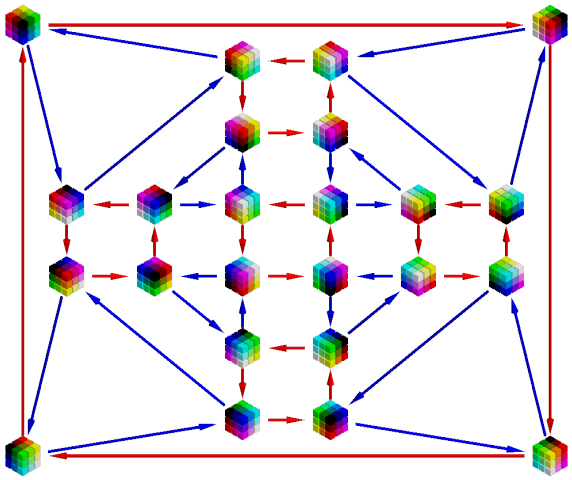
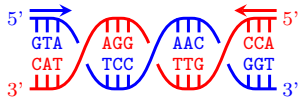


Figure 16: A 3×3 filter, rotated by all 24 elements of the discrete rotation group O_h, generated by 90-degree rotations about the vertical axis (red arrows), and 120-degree rotations about a diagonal axis (blue arrows).

have a discrete 1D translation symmetry on the grid, but DNA sequences have an additional interesting symmetry. This symmetry arises from the way DNA is physically embodied as a double helix, and the way it is read by the molecular machinery of the cell. Each strand of the double helix begins with what is called the $5'$-end and ends with a $3'$-end, with the $5'$ on one strand complemented by a $3'$ on the other strand. In other words, the two strands have an opposite orientation. Since the DNA molecule is always read off starting at the $5'$-end, but we do not know which one, a sequence such as ACCCTGG is equivalent to the reversed sequence with each letter replaced by its complement, CCAGGGT. This is called *reverse-complement symmetry* of the letter sequence. We thus have the two-element group $\mathbb{Z}_2 = \{0, 1\}$ corresponding to the identity 0 and reverse-complement transformation 1 (and composition $1 + 1 = 0 \mod 2$). The full group combines translations and reverse-complement transformations.



A schematic of the DNA's double helix structure, with the two strands coloured in blue and red. Note how the sequences in the helices are complementary and read in reverse (from 5' to 3').

In our case, the group convolution (14) we defined in Section 4.3 is given as

$$(x \star \theta)(\mathfrak{g}) = \sum_{u \in \Omega} x_u \rho(\mathfrak{g}) \theta_u, \tag{31}$$

the inner product between the (single-channel) input signal x and a filter θ transformed by $\mathfrak{g} \in \mathfrak{G}$ via $\rho(\mathfrak{g})\theta_u = \theta_{\mathfrak{g}^{-1}u}$, and the output $x \star \theta$ is a function

on $\mathfrak{G}$. Note that since Ω is discrete, we have replaced the integral from equation (14) by a sum.

Transform+Convolve approach We will show that the group convolution can be implemented in two steps: a filter transformation step, and a translational convolution step. The filter transformation step consists of creating rotated (or reverse-complement transformed) copies of a basic filter, while the translational convolution is the same as in standard CNNs and thus efficiently computable on hardware such as GPUs. To see this, note that in both of our examples we can write a general transformation $\mathfrak{g} \in \mathfrak{G}$ as a transformation $\mathfrak{h} \in \mathfrak{H}$ (e.g. a rotation or reverse-complement transformation) followed by a translation $\mathfrak{k} \in \mathbb{Z}^d$, i.e. $\mathfrak{g} = \mathfrak{k}\mathfrak{h}$ (with juxtaposition denoting the composition of the group elements $\mathfrak{k}$ and $\mathfrak{h}$). By properties of the group representation, we have $\rho(\mathfrak{g}) = \rho(\mathfrak{k}\mathfrak{h}) = \rho(\mathfrak{k})\rho(\mathfrak{h})$. Thus,

$$\begin{aligned}(x \star \theta)(\mathfrak{k}\mathfrak{h}) &= \sum_{u \in \Omega} x_u \rho(\mathfrak{k})\rho(\mathfrak{h})\theta_u \\ &= \sum_{u \in \Omega} x_u (\rho(\mathfrak{h})\theta)_{u-\mathfrak{k}}\end{aligned} \tag{32}$$

We recognise the last equation as the standard (planar Euclidean) convolution of the signal x and the transformed filter $\rho(\mathfrak{h})\theta$. Thus, to implement group convolution for these groups, we take the canonical filter θ, create transformed copies $\theta_{\mathfrak{h}} = \rho(\mathfrak{h})\theta$ for each $\mathfrak{h} \in \mathfrak{H}$ (e.g. each rotation $\mathfrak{h} \in O_h$ or reverse-complement DNA symmetry $\mathfrak{h} \in \mathbb{Z}_2$), and then convolve x with each of these filters: $(x \star \theta)(\mathfrak{k}\mathfrak{h}) = (x \star \theta_{\mathfrak{h}})(\mathfrak{k})$. For both of our examples, the symmetries act on filters by simply permuting the filter coefficients, as shown in Figure 16 for discrete rotations. Hence, these operations can be implemented efficiently using an indexing operation with pre-computed indices.

While we defined the feature maps output by the group convolution $x \star \theta$ as functions on $\mathfrak{G}$, the fact that we can split $\mathfrak{g}$ into $\mathfrak{h}$ and $\mathfrak{k}$ means that we can also think of them as a stack of Euclidean feature maps (sometimes called *orientation channels*), with one feature map per filter transformation / orientation $\mathfrak{k}$. For instance, in our first example we would associate to each filter rotation (each node in Figure 16) a feature map, which is obtained by convolving (in the traditional translational sense) the rotated filter. These feature maps can thus still be stored as a $W \times H \times C$ array, where the number

of channels C equals the number of independent filters times the number of transformations $\mathfrak{h} \in \mathfrak{H}$ (e.g. rotations).

As shown in Section 4.3, the group convolution is equivariant: $(\rho(\mathfrak{g})x) \star \theta = \rho(\mathfrak{g})(x \star \theta)$. What this means in terms of orientation channels is that under the action of $\mathfrak{h}$, each orientation channel is transformed, and the orientation channels themselves are permuted. For instance, if we associate one orientation channel per transformation in Figure 16 and apply a rotation by 90 degrees about the z-axis (corresponding to the red arrows), the feature maps will be permuted as shown by the red arrows. This description makes it clear that a group convolutional neural network bears much similarity to a traditional CNN. Hence, many of the network design patterns discussed in the Section 5.1, such as residual networks, can be used with group convolutions as well.

Spherical CNNs in the Fourier domain For the continuous symmetry group of the sphere that we saw in Section 4.3, it is possible to implement the convolution in the spectral domain, using the appropriate Fourier transform (we remind the reader that the convolution on $\mathbb{S}^2$ is a function on $\mathrm{SO}(3)$, hence we need to define the Fourier transform on both these domains in order to implement multi-layer spherical CNNs). *Spherical harmonics* are an orthogonal basis on the 2D sphere, analogous to the classical Fourier basis of complex exponential. On the special orthogonal group, the Fourier basis is known as the *Wigner D-functions*. In both cases, the Fourier transforms (coefficients) are computed as the inner product with the basis functions, and an analogy of the Convolution Theorem holds: one can compute the convolution in the Fourier domain as the element-wise product of the Fourier transforms. Furthermore, FFT-like algorithms exist for the efficient computation of Fourier transform on $\mathbb{S}^2$ and $\mathrm{SO}(3)$. We refer for further details to Cohen et al. (2018).

5.3 Graph Neural Networks

Graph Neural Networks (GNNs) are the realisation of our Geometric Deep Learning blueprint on graphs leveraging the properties of the permutation group. GNNs are among the most general class of deep learning architectures currently in existence, and as we will see in this text, most other deep learning architectures can be understood as a special case of the GNN with

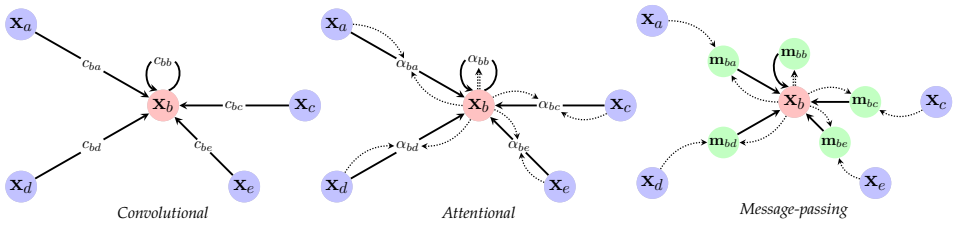


Figure 17: A visualisation of the dataflow for the three flavours of GNN layers, g. We use the neighbourhood of node b from Figure 10 to illustrate this. Left-to-right: **convolutional**, where sender node features are multiplied with a constant, c_{uv}; **attentional**, where this multiplier is *implicitly* computed via an attention mechanism of the receiver over the sender: $\alpha_{uv} = a(\mathbf{x}_u, \mathbf{x}_v)$; and **message-passing**, where vector-based messages are computed based on both the sender and receiver: $\mathbf{m}_{uv} = \psi(\mathbf{x}_u, \mathbf{x}_v)$.

additional geometric structure.

As per our discussion in Section 4.1, we consider a graph to be specified with an adjacency matrix $\mathbf{A}$ and node features $\mathbf{X}$. We will study GNN architectures that are *permutation equivariant* functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$ constructed by applying shared *permutation invariant* functions $\phi(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$ over local neighbourhoods. Under various guises, this local function ϕ can be referred to as "diffusion", "propagation", or "message passing", and the overall computation of such $\mathbf{F}$ as a "GNN layer".

The design and study of GNN layers is one of the most active areas of deep learning at the time of writing, making it a landscape that is challenging to navigate. Fortunately, we find that the vast majority of the literature may be derived from only three "flavours" of GNN layers (Figure 17), which we will present here. These flavours govern the extent to which ϕ transforms the neighbourhood features, allowing for varying degrees of complexity when modelling interactions across the graph.

In all three flavours, permutation invariance is ensured by *aggregating* features from $\mathbf{X}_{\mathcal{N}_u}$ (potentially transformed, by means of some function ψ) with some permutation-invariant function $\bigoplus$, and then *updating* the features of node u, by means of some function ϕ. Typically, ψ and ϕ are learnable, whereas $\bigoplus$ is realised as a nonparametric operation such as sum, mean, or maximum, though it can also be constructed e.g. using recurrent neural networks (Murphy et al., 2018).

Most commonly, ψ and ϕ are learnable affine transformations with activation functions; e.g. $\psi(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$; $\phi(\mathbf{x}, \mathbf{z}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{U}\mathbf{z} + \mathbf{b})$, where $\mathbf{W}, \mathbf{U}, \mathbf{b}$ are learnable parameters and σ is an activation function such as the rectified linear unit. The additional input of $\mathbf{x}_u$ to ϕ represents an optional *skip-connection*, which is often very useful.

In the **convolutional** flavour (Kipf and Welling, 2016a; Defferrard et al., 2016; Wu et al., 2019), the features of the neighbourhood nodes are directly aggregated with fixed weights,

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v\in\mathcal{N}_u} c_{uv}\psi(\mathbf{x}_v)\right). \tag{33}$$

Here, c_{uv} specifies the *importance* of node v to node u's representation. It is a constant that often directly depends on the entries in $\mathbf{A}$ representing the structure of the graph. Note that when the aggregation operator $\bigoplus$ is chosen to be the summation, it can be considered as a linear diffusion or position-dependent linear filtering, a generalisation of convolution. In particular, the spectral filters we have seen in Sections 4.4 and 4.6 fall under this category, as they amount to applying fixed local operators (e.g. the Laplacian matrix) to node-wise signals.

It is worthy to note that this flavour does not express *every* GNN layer that is convolutional (in the sense of commuting with the graph structure), but covers most such approaches proposed in practice. We will provide detailed discussion and extensions in future work.

In the **attentional** flavour (Veličković et al., 2018; Monti et al., 2017; Zhang et al., 2018), the interactions are implicit

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v\in\mathcal{N}_u} a(\mathbf{x}_u,\mathbf{x}_v)\psi(\mathbf{x}_v)\right). \tag{34}$$

Here, a is a learnable *self-attention mechanism* that computes the importance coefficients $\alpha_{uv} = a(\mathbf{x}_u,\mathbf{x}_v)$ implicitly. They are often softmax-normalised across all neighbours. When $\bigoplus$ is the summation, the aggregation is still a linear combination of the neighbourhood node features, but now the weights are feature-dependent.

Finally, the **message-passing** flavour (Gilmer et al., 2017; Battaglia et al., 2018) amounts to computing arbitrary vectors ("messages") across edges,

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v\in\mathcal{N}_u} \psi(\mathbf{x}_u,\mathbf{x}_v)\right). \tag{35}$$

Here, ψ is a learnable *message function*, computing v's vector sent to u, and the aggregation can be considered as a form of message passing on the graph.

One important thing to note is a representational containment between these approaches: *convolution* $\subseteq$ *attention* $\subseteq$ *message-passing*. Indeed, attentional GNNs can represent convolutional GNNs by an attention mechanism implemented as a look-up table $a(\mathbf{x}_u,\mathbf{x}_v) = c_{uv}$, and both convolutional and

attentional GNNs are special cases of message-passing where the messages are only the sender nodes' features: $\psi(\mathbf{x}_u, \mathbf{x}_v) = c_{uv}\psi(\mathbf{x}_v)$ for convolutional GNNs and $\psi(\mathbf{x}_u, \mathbf{x}_v) = a(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)$ for attentional GNNs.

This does not imply that message passing GNNs are always the most useful variant; as they have to compute vector-valued messages across edges, they are typically harder to train and require unwieldy amounts of memory. Further, on a wide range of naturally-occurring graphs, the graph's edges encode for downstream class similarity (i.e. an edge (u, v) implies that u and v are likely to have the same output). For such graphs (often called *homophilous*), convolutional aggregation across neighbourhoods is often a far better choice, both in terms of regularisation and scalability. Attentional GNNs offer a "middle-ground": they allow for modelling complex interactions within neighbourhoods while computing only scalar-valued quantities across the edges, making them more scalable than message-passing.

The "three flavour" categorisation presented here is provided with brevity in mind and inevitably neglects a wealth of nuances, insights, generalisations, and historical contexts to GNN models. Importantly, it excludes higher-dimensional GNN based on the Weisfeiler-Lehman hierarchy and spectral GNNs relying on the explicit computation of the graph Fourier transform.

5.4 Deep Sets, Transformers, and Latent Graph Inference

We close the discussion on GNNs by remarking on permutation-equivariant neural network architectures for learning representations of *unordered sets*. While sets have the least structure among the domains we have discussed in this text, their importance has been recently highlighted by highly-popular architectures such as Transformers (Vaswani et al., 2017) and Deep Sets (Zaheer et al., 2017). In the language of Section 4.1, we assume that we are given a matrix of node features, $\mathbf{X}$, but without any specified adjacency or ordering information between the nodes. The specific architectures will arise by deciding to what extent to model *interactions* between the nodes.

Empty edge set Unordered sets are provided without any additional structure or geometry whatsoever—hence, it could be argued that the most natural way to process them is to treat each set element entirely *independently*. This translates to a permutation equivariant function over such inputs, which

was already introduced in Section 4.1: a shared transformation applied to every node in isolation. Assuming the same notation as when describing GNNs (Section 5.3), such models can be represented as

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

where ψ is a learnable transformation. It may be observed that this is a special case of a convolutional GNN with $\mathcal{N}_u = \{u\}$—or, equivalently, $\mathbf{A} = \mathbf{I}$. Such an architecture is commonly referred to as Deep Sets, in recognition of the work of Zaheer et al. (2017) that have theoretically proved several universal-approximation properties of such architectures. It should be noted that the need to process unordered sets commonly arises in computer vision and graphics when dealing with *point clouds*; therein, such models are known as PointNets (Qi et al., 2017).

Complete edge set While assuming an empty edge set is a very efficient construct for building functions over unordered sets, often we would expect that elements of the set exhibit some form of relational structure—i.e., that there exists a *latent graph* between the nodes. Setting $\mathbf{A} = \mathbf{I}$ discards any such structure, and may yield suboptimal performance. Conversely, we could assume that, in absence of any other prior knowledge, we cannot upfront exclude *any* possible links between nodes. In this approach we assume the *complete* graph, $\mathbf{A} = \mathbf{1}\mathbf{1}^\top$; equivalently, $\mathcal{N}_u = \mathcal{V}$. As we do not assume access to any coefficients of interaction, running *convolutional*-type GNNs over such a graph would amount to:

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

where the second input, $\bigoplus_{v\in\mathcal{V}} \psi(\mathbf{x}_v)$ is *identical* for all nodes u, and as such makes the model equivalently expressive to ignoring that input altogether; i.e. the $\mathbf{A} = \mathbf{I}$ case mentioned above.

This is a direct consequence of the permutation invariance of $\bigoplus$.

This motivates the use of a more expressive GNN flavour, the *attentional*,

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v\in\mathcal{V}} a(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)\right) \tag{36}$$

which yields the *self-attention* operator, the core of the Transformer architecture (Vaswani et al., 2017). Assuming some kind of normalisation over

the attentional coefficients (e.g. softmax), we can constrain all the scalars $a(\mathbf{x}_u, \mathbf{x}_v)$ to be in the range $[0, 1]$; as such, we can think of self-attention as inferring a *soft adjacency matrix*, $a_{uv} = a(\mathbf{x}_u, \mathbf{x}_v)$, as a byproduct of gradient-based optimisation for some downstream task.

It is also appropriate to apply the message-passing flavour. While popular for physics simulations and relational reasoning (e.g. Battaglia et al. (2016); Santoro et al. (2017)), they have not been as widely used as Transformers. This is likely due to the memory issues associated with computing vector messages over a complete graph, or the fact that vector-based messages are less interpretable than the "soft adjacency" provided by self-attention.

The above perspective means that we can pose Transformers exactly as attentional GNNs over a complete graph (Joshi, 2020). However, this is in apparent conflict with Transformers being initially proposed for modelling *sequences*—the representations of $\mathbf{h}_u$ should be mindful of node u's *position* in the sequence, which complete-graph aggregation would ignore. Transformers address this issue by introducing *positional encodings*: the node features $\mathbf{x}_u$ are augmented to encode node u's position in the sequence, typically as samples from a sine wave whose frequency is dependent on u.

On graphs, where no natural ordering of nodes exists, multiple alternatives were proposed to such positional encodings. While we defer discussing these alternatives for later, we note that one promising direction involves a realisation that the positional encodings used in Transformers can be directly related to the discrete Fourier transform (DFT), and hence to the eigenvectors of the graph Laplacian of a "circular grid". Hence, Transformers' positional encodings are implicitly representing our assumption that input nodes are connected in a grid. For more general graph structures, one may simply use the Laplacian eigenvectors of the (assumed) graph—an observation exploited by Dwivedi and Bresson (2020) within their empirically powerful Graph Transformer model.

Inferred edge set Finally, one can try to learn the latent relational structure, leading to some general $\mathbf{A}$ that is neither $\mathbf{I}$ nor $\mathbf{1}\mathbf{1}^\top$. The problem of inferring a latent adjacency matrix $\mathbf{A}$ for a GNN to use (often called *latent graph inference*) is of high interest for graph representation learning. This is due to the fact that assuming $\mathbf{A} = \mathbf{I}$ may be representationally inferior, and $\mathbf{A} = \mathbf{1}\mathbf{1}^\top$ may be challenging to implement due to memory requirements and large neighbourhoods to aggregate over. Additionally, it is closest to the "true" problem: inferring an adjacency matrix $\mathbf{A}$ implies detecting useful structure between the rows of $\mathbf{X}$, which may then help formulate hypotheses such as causal relations between variables.

Unfortunately, such a framing necessarily induces a step-up in modelling complexity. Specifically, it requires properly balancing a structure learning

objective (which is *discrete*, and hence challenging for gradient-based optimisation) with any downstream task the graph is used for. This makes latent graph inference a highly challenging and intricate problem.

5.5 Equivariant Message Passing Networks

In many applications of Graph Neural Networks, node features (or parts thereof) are not just arbitrary vectors but *coordinates* of geometric entities. This is the case, for example, when dealing with molecular graphs: the nodes representing atoms may contain information about the atom type as well as its 3D spatial coordinates. It is desirable to process the latter part of the features in a manner that would transform in the same way as the molecule is transformed in space, in other words, be equivariant to the Euclidean group $\mathrm{E}(3)$ of rigid motions (rotations, translations, and reflections) in addition to the standard permutation equivariance discussed before.

To set the stage for our (slightly simplified) analysis, we will make a distinction between node *features* $\mathbf{f}_u \in \mathbb{R}^d$ and node *spatial coordinates* $\mathbf{x}_u \in \mathbb{R}^3$; the latter are endowed with Euclidean symmetry structure. In this setting, an equivariant layer explicitly transforms these two inputs separately, yielding modified node features $\mathbf{f}'_u$ and coordinates $\mathbf{x}'_u$.

We can now state our desirable equivariance property, following the Geometric Deep Learning blueprint. If the spatial component of the input is transformed by $\mathfrak{g} \in \mathrm{E}(3)$ (represented as $\rho(\mathfrak{g})\mathbf{x} = \mathbf{R}\mathbf{x} + \mathbf{b}$, where $\mathbf{R}$ is an orthogonal matrix modeling rotations and reflections, and $\mathbf{b}$ is a translation vector), the spatial component of the output transforms in the same way (as $\mathbf{x}'_u \mapsto \mathbf{R}\mathbf{x}'_u + \mathbf{b}$), whereas $\mathbf{f}'_u$ remains invariant.

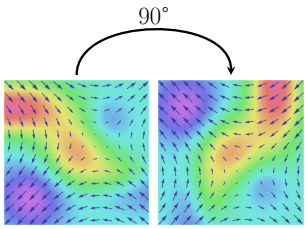
Much like the space of permutation equivariant functions we discussed before in the context of general graphs, there exists a vast amount of $\mathrm{E}(3)$-equivariant layers that would satisfy the constraints above—but not all of these layers would be geometrically stable, or easy to implement. In fact, the space of practically useful equivariant layers may well be easily described by a simple categorisation, not unlike our "three flavours" of spatial GNN layers. One elegant solution was suggested by Satorras et al. (2021) in the

form of *equivariant message passing*. Their model operates as follows:

$$\begin{aligned}\mathbf{f}'_u &= \phi\left(\mathbf{f}_u, \bigoplus_{v\in\mathcal{N}_u} \psi_{\mathrm{f}}(\mathbf{f}_u, \mathbf{f}_v, \|\mathbf{x}_u - \mathbf{x}_v\|^2)\right), \\ \mathbf{x}'_u &= \mathbf{x}_u + \sum_{v\neq u} (\mathbf{x}_u - \mathbf{x}_v)\psi_{\mathrm{c}}(\mathbf{f}_u, \mathbf{f}_v, \|\mathbf{x}_u - \mathbf{x}_v\|^2)\end{aligned}$$

where ψ_{f} and ψ_{c} are two distinct (learnable) functions. It can be shown that such an aggregation is equivariant under Euclidean transformations of the spatial coordinates. This is due to the fact that the only dependence of $\mathbf{f}'_u$ on $\mathbf{x}_u$ is through the distances $\|\mathbf{x}_u - \mathbf{x}_v\|^2$, and the action of $\mathrm{E}(3)$ necessarily leaves distances between nodes unchanged. Further, the computations of such a layer can be seen as a particular instance of the "message-passing" GNN flavour, hence they are efficient to implement.

To summarise, in contrast to ordinary GNNs, Satorras et al. (2021) enable the correct treatment of 'coordinates' for each point in the graph. They are now treated as a member of the $\mathrm{E}(3)$ group, which means the network outputs behave correctly under rotations, reflections and translations of the input. The features, $\mathbf{f}_u$, however, are treated in a channel-wise manner and still assumed to be *scalars* that do not change under these transformations. This limits the type of spatial information that can be captured within such a framework. For example, it may be desirable for some features to be encoded as *vectors*—e.g. point velocities—which *should* change direction under such transformations. Satorras et al. (2021) partially alleviate this issue by introducing the concept of velocities in one variant of their architecture. Velocities are a 3D vector property of each point which rotates appropriately. However, this is only a small subspace of the general representations that could be learned with an $\mathrm{E}(3)$ equivariant network. In general, node features may encode *tensors* of arbitrary dimensionality that would still transform according to $\mathrm{E}(3)$ in a well-defined manner.



While scalar features (heatmap) do not change under rotations, vector features (arrows) may change direction. The simple $\mathrm{E}(3)$ equivariant GNN given before does not take this into account.

Hence, while the architecture discussed above already presents an elegant equivariant solution for many practical input representations, in some cases it may be desirable to explore a broader collection of functions that satisfy the equivariance property. Existing methods dealing with such settings can be categorised into two classes: *irreducible representations* (of which the previously mentioned layer is a simplified instance) and *regular representations*. We briefly survey them here, leaving detailed discussion to future work.

Irreducible representations Irreducible representations build on the finding that all elements of the roto-translation group can be brought into an irreducible form: a vector that is rotated by a block diagonal matrix. Crucially, each of those blocks is a *Wigner D-matrix* (the aforementioned Fourier basis for Spherical CNNs). Approaches under this umbrella map from one set of irreducible representations to another using equivariant kernels. To find the full set of equivariant mappings, one can then directly solve the equivariance constraint over these kernels. The solutions form a linear combination of equivariant basis matrices derived by *Clebsch-Gordan matrices* and the spherical harmonics.

Early examples of the irreducible representations approach include Tensor Field Networks (Thomas et al., 2018) and 3D Steerable CNNs (Weiler et al., 2018), both convolutional models operating on point clouds. The SE(3)-Transformer of Fuchs et al. (2020) extends this framework to the graph domain, using an attentional layer rather than convolutional. Further, while our discussion focused on the special case solution of Satorras et al. (2021), we note that the motivation for rotation or translation equivariant predictions over graphs had historically been explored in other fields, including architectures such as Dynamic Graph CNN (Wang et al., 2019b) for point clouds and efficient message passing models for quantum chemistry, such as SchNet (Schütt et al., 2018) and DimeNet (Klicpera et al., 2020).

Regular representations While the approach of irreducible representations is attractive, it requires directly reasoning about the underlying group representations, which may be tedious, and only applicable to groups that are compact. Regular representation approaches are more general, but come with an additional computational burden – for exact equivariance they require storing copies of latent feature embeddings for *all* group elements.

This approach was, in fact, pioneered by the group convolutional neural networks we presented in previous sections.

One promising approach in this space aims to observe equivariance to *Lie groups*—through definitions of exponential and logarithmic maps—with the promise of rapid prototyping across various symmetry groups. While Lie groups are out of scope for this section, we refer the reader to two recent successful instances of this direction: the LieConv of Finzi et al. (2020), and the LieTransformer of Hutchinson et al. (2020).

The approaches covered in this section represent popular ways of processing data on geometric graphs in an way that is explicitly equivariant to the un-

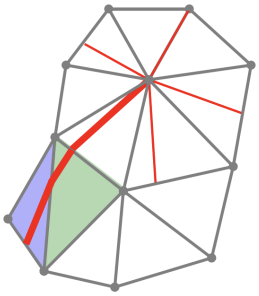
derlying geometry. As discussed in Section 4.6, *meshes* are a special instance of geometric graphs that can be understood as discretisations of continuous surfaces. We will study mesh-specific equivariant neural networks next.

5.6 Intrinsic Mesh CNNs

Meshes, in particular, triangular ones, are the 'bread and butter' of computer graphics and perhaps the most common way of modeling 3D objects. The remarkable success of deep learning in general and CNNs in computer vision in particular has lead to a keen interest in the graphics and geometry processing community around the mid-2010s to construct similar architectures for mesh data.

Examples of geodesic patches. In order for the resulting patch to be a topological disk, its radius R must be smaller than the injectivity radius.

Geodesic patches Most of the architectures for deep learning on meshes implement convolutional filters of the form (21) by discretising or approximating the exponential map and expressing the filter in a coordinate system of the tangent plane. Shooting a geodesic $\gamma : [0, T] \to \Omega$ from a point $u = \gamma(0)$ to nearby point $v = \gamma(T)$ defines a local system of *geodesic polar coordinates* $(r(u, v), \vartheta(u, v))$ where r is the geodesic distance between u and v (length of the geodesic γ) and ϑ is the angle between $\gamma'(0)$ and some local reference direction. This allows to define a *geodesic patch* $x(u, r, \vartheta) = x(\exp_u \tilde{\omega}(r, \vartheta))$, where $\tilde{\omega}_u : [0, R] \times [0, 2\pi) \to T_u\Omega$ is the local polar frame.



Construction of discrete geodesics on a mesh.

On a surface discretised as a mesh, a geodesic is a poly-line that traverses the triangular faces. Traditionally, geodesics have been computed using the Fast Marching algorithm Kimmel and Sethian (1998), an efficient numerical approximation of a nonlinear PDE called the *eikonal equation* encountered in physical models of wave propagation in a medium. This scheme was adapted by Kokkinos et al. (2012) for the computation of local geodesic patches and later reused by Masci et al. (2015) for the construction of *Geodesic CNNs*, the first intrinsic CNN-like architectures on meshes.

Isotropic filters Importantly, in the definition of the geodesic patch we have ambiguity in the choice of the reference direction and the patch orientation. This is exactly the ambiguity of the choice of the gauge, and our local system of coordinates is defined up to arbitrary rotation (or a shift in the angular coordinate, $x(u, r, \vartheta + \vartheta_0)$), which can be different at every node. Perhaps

the most straightforward solution is to use isotropic filters of the form $\theta(r)$ that perform a direction-independent aggregation of the neighbour features,

$$(x \star \theta)(u) = \int_0^R \int_0^{2\pi} x(u, r, \vartheta)\theta(r)\mathrm{d}r\mathrm{d}\vartheta.$$

Spectral filters discussed in Sections 4.4–4.6 fall under this category: they are based on the Laplacian operator, which is isotropic. Such an approach, however, discards important directional information, and might fail to extract edge-like features.

Fixed gauge An alternative, to which we have already alluded in Section 4.4, is to *fix some gauge*. Monti et al. (2017) used the principal curvature directions: while this choice is not intrinsic and may ambiguous at flat points (where curvature vanishes) or uniform curvature (such as on a perfect sphere), the authors showed that it is reasonable for dealing with deformable human body shapes, which are approximately piecewise-rigid. Later works, e.g. Melzi et al. (2019), showed reliable intrinsic construction of gauges on meshes, computed as (intrinsic) gradients of intrinsic functions. While such tangent fields might have singularities (i.e., vanish at some points), the overall procedure is very robust to noise and remeshing.

Angular pooling Another approach, referred to as *angular max pooling*, was used by Masci et al. (2015). In this case, the filter $\theta(r, \vartheta)$ is anisotropic, but its matching with the function is performed over *all the possible rotations*, which are then aggregated:

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

Conceptually, this can be visualised as correlating geodesic patches with a rotating filter and collecting the strongest responses.

On meshes, the continuous integrals can be discretised using a construction referred to as *patch operators* (Masci et al., 2015). In a geodesic patch around node u, the neighbour nodes $\mathcal{N}_u$, represented in the local polar coordinates as (r_{uv}, ϑ_{uv}), are weighted by a set of weighting functions $w_1(r, \vartheta), \dots, w_K(r, \vartheta)$ (shown in Figure 18 and acting as 'soft pixels') and aggregated,

Typically multi-hop neighbours are used.

$$(x \star \theta)_u = \frac{\sum_{k=1}^K w_k \sum_{v \in \mathcal{N}_u} (r_{uv}, \vartheta_{uv}) x_v \, \theta_k}{\sum_{k=1}^K w_k \sum_{v \in \mathcal{N}_u} (r_{uv}, \vartheta_{uv}) \theta_k}$$

(here $\theta_1, \dots, \theta_K$ are the learnable coefficients of the filter). Multi-channel features are treated channel-wise, with a family of appropriate filters. Masci et al. (2015); Boscaini et al. (2016a) used pre-defined weighting functions w, while Monti et al. (2017) further allowed them to be learnable.

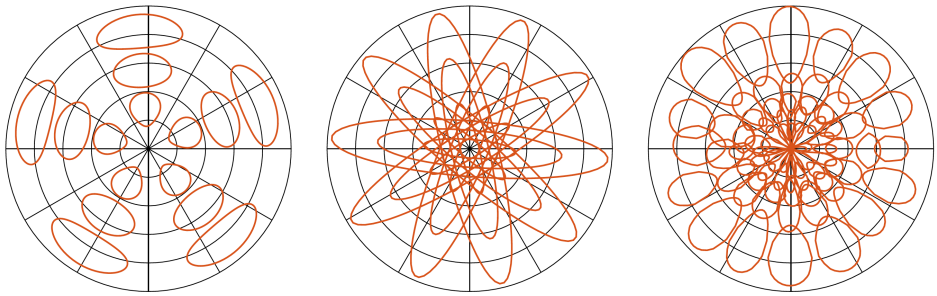


Figure 18: Left-to-right: examples of patch operators used in Geodesic CNN (Masci et al., 2015), Anisotropic CNN (Boscaini et al., 2016b) and MoNet (Monti et al., 2017), with the level sets of the weighting functions $w_k(r, \vartheta)$ shown in red.

Gauge-equivariant filters Both isotropic filters and angular max pooling lead to features that are *invariant* to gauge transformations; they transform according to the trivial representation $\rho(\mathfrak{g}) = 1$ (where $\mathfrak{g} \in \mathrm{SO}(2)$ is a rotation of the local coordinate frame). This point of view suggests another approach, proposed by Cohen et al. (2019); de Haan et al. (2020) and discussed in Section 4.5, where the features computed by the network are associated with an arbitrary representation ρ of the structure group $\mathfrak{G}$ (e.g. $\mathrm{SO}(2)$ or $\mathrm{O}(2)$ of rotations or rotations+reflections of the coordinate frame, respectively). Tangent vectors transform according to the standard representation $\rho(\mathfrak{g}) = \mathfrak{g}$. As another example, the feature vector obtained by matching n rotated copies of the same filter transforms by cyclic shifts under rotations of the gauge; this is known as the regular representation of the cyclic group C_n.

As discussed in Section 4.5, when dealing with such geometric features (associated to a non-trivial representation), we must first parallel transport them to the same vector space before applying the filter. On a mesh, this can be implemented via the following message passing mechanism described by de Haan et al. (2020). Let $\mathbf{x}_u \in \mathbb{R}^d$ be a d-dimensional input feature at mesh node u. This feature is expressed relative to an (arbitrary) choice of gauge at u, and is assumed to transform according to a representation ρ_{in} of

$\mathfrak{G} = \mathrm{SO}(2)$ under rotations of the gauge. Similarly, the output features $\mathbf{h}_u$ of the mesh convolution are d' dimensional and should transform according to ρ_{out} (which can be chosen at will by the network designer).

By analogy to Graph Neural Networks, we can implement the gauge-equivariant convolution (23) on meshes by sending messages from the neighbours $\mathcal{N}_u$ of u (and from u itself) to u:

$$\mathbf{h}_u = \mathbf{\Theta}_{\text{self}}\, \mathbf{x}_u + \sum_{v \in \mathcal{N}_u} \mathbf{\Theta}_{\text{neigh}}(\vartheta_{uv}) \rho(\mathfrak{g}_{v \to u}) \mathbf{x}_v, \tag{37}$$

where $\mathbf{\Theta}_{\text{self}}, \mathbf{\Theta}_{\text{neigh}}(\vartheta_{uv}) \in \mathbb{R}^{d' \times d}$ are learned filter matrices. The structure group element $\mathfrak{g}_{v \to u} \in \mathrm{SO}(2)$ denotes the effect of parallel transport from v to u, expressed relative to the gauges at u and v, and can be precomputed for each mesh. Its action is encoded by a *transporter matrix* $\rho(\mathfrak{g}_{v \to u}) \in \mathbb{R}^{d \times d}$. The matrix $\mathbf{\Theta}_{\text{neigh}}(\vartheta_{uv})$ depends on the angle ϑ_{uv} of the neighbour v to the reference direction (e.g. first axis of the frame) at u, so this kernel is anisotropic: different neighbours are treated differently.

Note that d is the feature dimension and is not necessarily equal to 2, the dimension of the mesh.

As explained in Section 4.5, for $\mathbf{h}(u)$ to be a well-defined geometric quantity, it should transform as $\mathbf{h}(u) \mapsto \rho_{\text{out}}(\mathfrak{g}^{-1}(u))\mathbf{h}(u)$ under gauge transformations. This will be the case when $\mathbf{\Theta}_{\text{self}} \rho_{\text{in}}(\vartheta) = \rho_{\text{out}}(\vartheta) \mathbf{\Theta}_{\text{self}}$ for all $\vartheta \in \mathrm{SO}(2)$, and $\mathbf{\Theta}_{\text{neigh}}(\vartheta_{uv} - \vartheta)\rho_{\text{in}}(\vartheta) = \rho_{\text{out}}(\vartheta)\mathbf{\Theta}_{\text{neigh}}(\vartheta_{uv})$. Since these constraints are linear, the space of matrices $\mathbf{\Theta}_{\text{self}}$ and matrix-valued functions $\mathbf{\Theta}_{\text{neigh}}$ satisfying these constraints is a linear subspace, and so we can parameterise them as a linear combination of basis kernels with learnable coefficients: $\mathbf{\Theta}_{\text{self}} = \sum_i \alpha_i \mathbf{\Theta}^i_{\text{self}}$ and $\mathbf{\Theta}_{\text{neigh}} = \sum_i \beta_i \mathbf{\Theta}^i_{\text{neigh}}$.

Here we abuse the notation, identifying 2D rotations with angles ϑ.

5.7 Recurrent Neural Networks

Our discussion has thus far always assumed the inputs to be solely *spatial* across a given domain. However, in many common use cases, the inputs can also be considered *sequential* (e.g. video, text or speech). In this case, we assume that the input consists of arbitrarily many *steps*, wherein at each step t we are provided with an input signal, which we represent as $\mathbf{X}^{(t)} \in \mathcal{X}(\Omega^{(t)})$.

While in general the domain can evolve in time together with the signals on it, it is typically assumed that the domain is kept fixed across all the t, i.e. $\Omega^{(t)} = \Omega$. Here, we will exclusively focus on this case, but note that

Whether the domain is considered static or dynamic concerns *time scales*: e.g., a road network *does* change over time (as new roads are built and old ones are demolished), but significantly slower compared to the flow of traffic. Similarly, in social networks, changes in engagement (e.g. Twitter users re-tweeting a tweet) happen at a much higher frequency than changes in the follow graph.

exceptions are common. Social networks are an example where one often has to account for the domain changing through time, as new links are regularly created as well as erased. The domain in this setting is often referred to as a *dynamic graph* (Xu et al., 2020a; Rossi et al., 2020).

Often, the individual $\mathbf{X}^{(t)}$ inputs will exhibit useful symmetries and hence may be nontrivially treated by any of our previously discussed architectures. Some common examples include: *videos* (Ω is a fixed grid, and signals are a sequence of *frames*); *fMRI scans* (Ω is a fixed *mesh* representing the geometry of the brain cortex, where different regions are activated at different times as a response to presented stimuli); and *traffic flow networks* (Ω is a fixed *graph* representing the road network, on which e.g. the average traffic speed is recorded at various nodes).

We do not lose generality in our example; equivalent analysis can be done e.g. for node-level outputs on a spatiotemporal graph; the only difference is in the choice of encoder f (which will then be a permutation equivariant GNN).

Let us assume an *encoder* function $f(\mathbf{X}^{(t)})$ providing latent representations at the level of granularity appropriate for the problem and respectful of the symmetries of the input domain. As an example, consider processing video frames: that is, at each timestep, we are given a *grid-structured input* represented as an $n \times d$ matrix $\mathbf{X}^{(t)}$, where n is the number of pixels (fixed in time) and d is the number of input channels (e.g. $d = 3$ for RGB frames). Further, we are interested in analysis at the level of entire frames, in which case it is appropriate to implement f as a translation invariant CNN, outputting a k-dimensional representation $\mathbf{z}^{(t)} = f(\mathbf{X}^{(t)})$ of the frame at time-step t.

Note that the $\mathbf{z}^{(t)}$ vectors can be seen as points on a *temporal grid*: hence, processing them with a CNN is also viable in some cases. Transformers are also increasingly popular models for processing generic sequential inputs.

We are now left with the task of appropriately *summarising* a sequence of vectors $\mathbf{z}^{(t)}$ *across all the steps.* A canonical way to *dynamically* aggregate this information in a way that respects the temporal progression of inputs and also easily allows for *online* arrival of novel data-points, is using a *Recurrent Neural Network* (RNN). What we will show here is that RNNs are an interesting geometric architecture to study in their own right, since they implement a rather unusual type of symmetry over the inputs $\mathbf{z}^{(t)}$.

SimpleRNNs At each step, the recurrent neural network computes an m-dimensional *summary* vector $\mathbf{h}^{(t)}$ of all the input steps up to and including t. This (partial) summary is computed conditional on the current step's features and the previous step's summary, through a shared *update* function, $R : \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^m$, as follows (see Figure 19 for a summary):

$$\mathbf{h}^{(t)} = R(\mathbf{z}^{(t)}, \mathbf{h}^{(t-1)}) \tag{38}$$

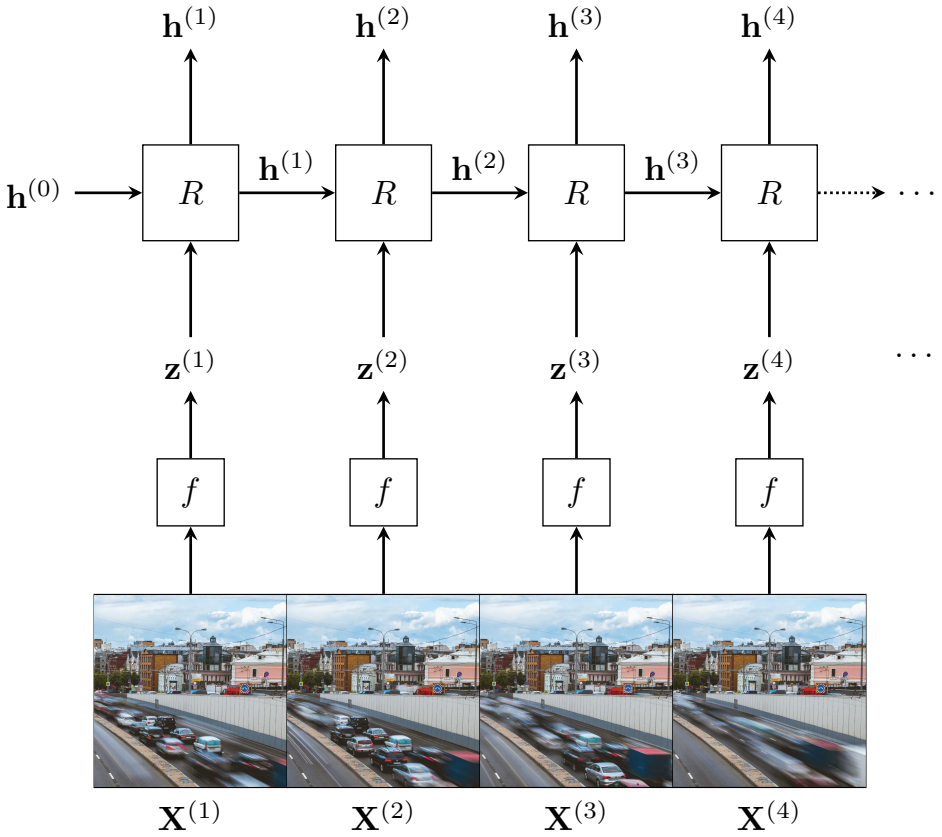


Figure 19: Illustration of processing video input with RNNs. Each input video frame $\mathbf{X}^{(t)}$ is processed using a shared function f—e.g. a translation invariant CNN—into a flat representation $\mathbf{z}^{(t)}$. Then the RNN update function R is iterated across these vectors, iteratively updating a summary vector $\mathbf{h}^{(t)}$ which summarises all the inputs up to and including $\mathbf{z}^{(t)}$. The computation is seeded with an initial summary vector $\mathbf{h}^{(0)}$, which may be either pre-determined or learnable.

and, as both $\mathbf{z}^{(t)}$ and $\mathbf{h}^{(t-1)}$ are *flat* vector representations, R may be most easily expressed as a single fully-connected neural network layer (often known as *SimpleRNN*; see Elman (1990); Jordan (1997)):

In spite of their name, SimpleRNNs are remarkably expressive. For example, it was shown by Siegelmann and Sontag (1995) that such models are *Turing-complete*, meaning that they can likely represent *any* computation we may ever be able to execute on computers.

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}\mathbf{z}^{(t)} + \mathbf{U}\mathbf{h}^{(t-1)} + \mathbf{b}) \tag{39}$$

where $\mathbf{W} \in \mathbb{R}^{k\times m}$, $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{b} \in \mathbb{R}^m$ are learnable parameters, and σ is an activation function. While this introduces *loops* in the network's computational graph, in practice the network is unrolled for an appropriate number of steps, allowing for *backpropagation through time* (Robinson and Fallside, 1987; Werbos, 1988; Mozer, 1989) to be applied.

The summary vectors may then be appropriately leveraged for the downstream task—if a prediction is required at every step of the sequence, then a shared predictor may be applied to each $\mathbf{h}^{(t)}$ individually. For classifying entire sequences, typically the final summary, $\mathbf{h}^{(T)}$, is passed to a classifier. Here, T is the length of the sequence.

Specially, the initial summary vector is usually either set to the zero-vector, i.e. $\mathbf{h}^{(0)} = \mathbf{0}$, or it is made learnable. Analysing the manner in which the initial summary vector is set also allows us to deduce an interesting form of *translation equivariance* exhibited by RNNs.

Translation equivariance in RNNs Since we interpret the individual steps t as *discrete time-steps*, the input vectors $\mathbf{z}^{(t)}$ can be seen as living on a one-dimensional *grid* of time-steps. While it might be attractive to attempt extending our translation equivariance analysis from CNNs here, it cannot be done in a trivial manner.

Note that this construction is extendable to grids in higher dimensions, allowing us to, e.g., process signals living on images in a *scanline* fashion. Such a construction powered a popular series of models, such as the PixelRNN from van den Oord et al. (2016b).

To see why, let us assume that we have produced a new sequence $\mathbf{z}'^{(t)} = \mathbf{z}^{(t+1)}$ by performing a left-shift of our sequence by one step. It might be tempting to attempt showing $\mathbf{h}'^{(t)} = \mathbf{h}^{(t+1)}$, as one expects with translation equivariance; however, this will not generally hold. Consider $t = 1$; directly applying and expanding the update function, we recover the following:

$$\mathbf{h}'^{(1)} = R(\mathbf{z}'^{(1)}, \mathbf{h}^{(0)}) = R(\mathbf{z}^{(2)}, \mathbf{h}^{(0)}) \tag{40}$$

$$\mathbf{h}^{(2)} = R(\mathbf{z}^{(2)}, \mathbf{h}^{(1)}) = R(\mathbf{z}^{(2)}, R(\mathbf{z}^{(1)}, \mathbf{h}^{(0)})) \tag{41}$$

Hence, unless we can guarantee that $\mathbf{h}^{(0)} = R(\mathbf{z}^{(1)}, \mathbf{h}^{(0)})$, we will not recover translation equivariance. Similar analysis can then be done for steps $t > 1$.

Fortunately, with a slight refactoring of how we represent $\mathbf{z}$, and for a suitable choice of R, it is possible to satisfy the equality above, and hence demonstrate a setting in which RNNs are equivariant to shifts. Our problem was largely one of *boundary conditions*: the equality above includes $\mathbf{z}^{(1)}$, which our left-shift operation destroyed. To abstract this problem away, we will observe how an RNN processes an appropriately *left-padded* sequence, $\bar{\mathbf{z}}^{(t)}$, defined as follows:

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

Such a sequence now allows for left-shifting by up to t' steps without destroying any of the original input elements. Further, note we do not need to handle right-shifting separately; indeed, equivariance to right shifts naturally follows from the RNN equations.

Note that equivalent analyses will arise if we use a different padding vector than $\mathbf{0}$.

We can now again analyse the operation of the RNN over a left-shifted verson of $\bar{\mathbf{z}}^{(t)}$, which we denote by $\bar{\mathbf{z}}'^{(t)} = \bar{\mathbf{z}}^{(t+1)}$, as we did in Equations 40–41:

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

where the substitution $\bar{\mathbf{z}}^{(1)} = \mathbf{0}$ holds as long as $t' \geq 1$, i.e. as long as any padding is applied. Now, we can guarantee equivariance to left-shifting by one step ($\mathbf{h}'^{(t)} = \mathbf{h}^{(t+1)}$) as long as $\mathbf{h}^{(0)} = R(\mathbf{0}, \mathbf{h}^{(0)})$.

In a very similar vein, we can derive equivariance to left-shifting by s steps as long as $t' \geq s$.

Said differently, $\mathbf{h}^{(0)}$ must be chosen to be a *fixed point* of a function $\gamma(\mathbf{h}) = R(\mathbf{0}, \mathbf{h})$. If the update function R is conveniently chosen, then not only can we guarantee existence of such fixed points, but we can even directly obtain them by iterating the application of R until convergence; e.g., as follows:

$$\mathbf{h}_0 = \mathbf{0} \qquad \mathbf{h}_{k+1} = \gamma(\mathbf{h}_k), \tag{42}$$

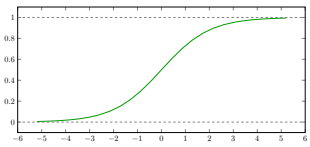
where the index k refers to the iteration of R in our computation, as opposed to the the index (t) denoting the time step of the RNN. If we choose R such that γ is a *contraction mapping*, such an iteration will indeed converge to a *unique* fixed point. Accordingly, we can then iterate Equation (42) until $\mathbf{h}_{k+1} = \mathbf{h}_k$, and we can set $\mathbf{h}^{(0)} = \mathbf{h}_k$. Note that this computation is equivalent to *left-padding* the sequence with "sufficiently many" zero-vectors.

Contractions are functions $\gamma : \mathcal{X} \to \mathcal{X}$ such that, under some norm $\|\cdot\|$ on $\mathcal{X}$, applying γ *contracts* the distances between points: for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, and some $q \in [0, 1)$, it holds that $\|\gamma(\mathbf{x}) - \gamma(\mathbf{y})\| \leq q\|\mathbf{x} - \mathbf{y}\|$. Iterating such a function then necessarily converges to a unique fixed point, as a direct consequence of *Banach's Fixed Point Theorem* (Banach, 1922).

Depth in RNNs It is also easy to stack multiple RNNs—simply use the $\mathbf{h}^{(t)}$ vectors as an input sequence for a second RNN. This kind of construction is occasionally called a "deep RNN", which is potentially misleading.

Effectively, due to the repeated application of the recurrent operation, even a single RNN "layer" has depth *equal to the number of input steps*.

This often introduces uniquely challenging learning dynamics when optimising RNNs, as each training example induces many gradient updates to the *shared* parameters of the update network. Here we will focus on perhaps the most prominent such issue—that of *vanishing* and *exploding* gradients (Bengio et al., 1994)—which is especially problematic in RNNs, given their depth and parameter sharing. Further, it has single-handedly spurred some of the most influential research on RNNs. For a more detailed overview, we refer the reader to Pascanu et al. (2013), who have studied the training dynamics of RNNs in great detail, and exposed these challenges from a variety of perspectives: analytical, geometrical, and the lens of dynamical systems.



Examples of such an activation include the *logistic function*, $\sigma(x) = \frac{1}{1+\exp(-x)}$, and the *hyperbolic tangent*, $\sigma(x) = \tanh x$. They are called sigmoidal due to the distinct S-shape of their plots.

To illustrate vanishing gradients, consider a SimpleRNN with a sigmoidal activation function σ, whose derivative magnitude $|\sigma'|$ is always between 0 and 1. Multiplying many such values results in gradients that quickly tend to zero, implying that early steps in the input sequence may not be able to have influence in updating the network parameters at all.

For example, consider the next-word prediction task (common in e.g. predictive keyboards), and the input text *"Petar is Serbian. He was born on ...* [*long paragraph*] *... Petar currently lives in* _____". Here, predicting the next word as "Serbia" may only be reasonably concluded by considering the very start of the paragraph—but gradients have likely vanished by the time they reach this input step, making learning from such examples very challenging.

Deep feedforward neural networks have also suffered from the vanishing gradient problem, until the invention of the ReLU activation (which has gradients equal to *exactly* zero or one—thus fixing the vanishing gradient problem). However, in RNNs, using ReLUs may easily lead to *exploding* gradients, as the output space of the update function is now *unbounded*, and gradient descent will update the cell once for every input step, quickly building up the scale of the updates. Historically, the vanishing gradient phenomenon was recognised early on as a significant obstacle in the use of recurrent networks. Coping with this problem motivated the development of more sophisticated RNN layers, which we describe next.

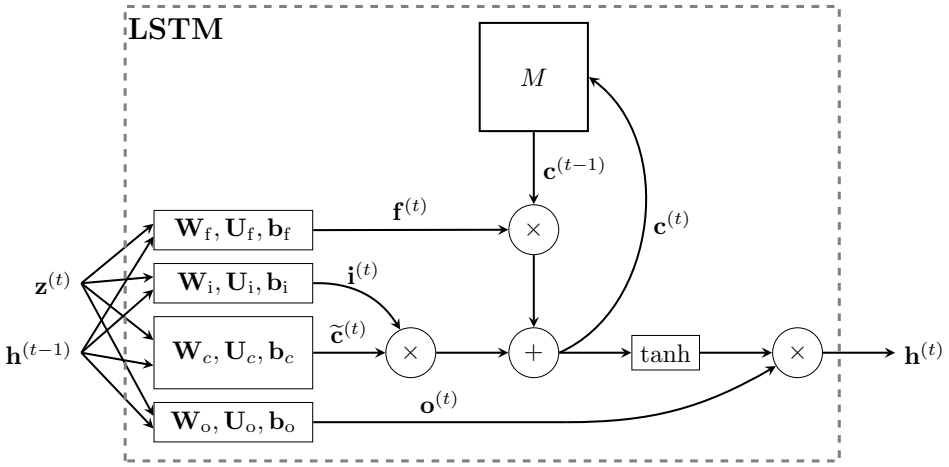


Figure 20: The dataflow of the long short-term memory (LSTM), with its components and memory cell (M) clearly highlighted. Based on the current input $\mathbf{z}^{(t)}$, previous summary $\mathbf{h}^{(t-1)}$ and previous cell state $\mathbf{c}^{(t-1)}$, the LSTM predicts the updated cell state $\mathbf{c}^{(t)}$ and summary $\mathbf{h}^{(t)}$.

5.8 Long Short-Term Memory networks

A key invention that significantly reduced the effects of vanishing gradients in RNNs is that of *gating mechanisms*, which allow the network to selectively *overwrite* information in a data-driven way. Prominent examples of these *gated RNNs* include the *Long Short-Term Memory* (LSTM; Hochreiter and Schmidhuber (1997)) and the *Gated Recurrent Unit* (GRU; Cho et al. (2014)). Here we will primarily discuss the LSTM—specifically, the variant presented by Graves (2013)—in order to illustrate the operations of such models. Concepts from LSTMs easily carry over to other gated RNNs.

Throughout this section, it will likely be useful to refer to Figure 20, which illustrates all of the LSTM operations that we will discuss in text.

The LSTM augments the recurrent computation by introducing a *memory cell*, which stores *cell state* vectors, $\mathbf{c}^{(t)} \in \mathbb{R}^m$, that are *preserved* between computational steps. The LSTM computes summary vectors, $\mathbf{h}^{(t)}$, directly based on $\mathbf{c}^{(t)}$, and $\mathbf{c}^{(t)}$ is, in turn, computed using $\mathbf{z}^{(t)}$, $\mathbf{h}^{(t-1)}$ and $\mathbf{c}^{(t-1)}$. Critically, the cell is **not** completely overwritten based on $\mathbf{z}^{(t)}$ and $\mathbf{h}^{(t-1)}$, which would expose the network to the same issues as the SimpleRNN. Instead, a certain quantity of the previous cell state may be *retained*—and

the proportion by which this occurs is explicitly *learned* from data.

Just like in SimpleRNN, we compute features by using a single fully-connected neural network layer over the current input step and previous summary:

Note that we have set the activation function to $\tanh$ here; as LSTMs are designed to ameliorate the vanishing gradient problem, it is now appropriate to use a sigmoidal activation.

$$\widetilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{W}_c\mathbf{z}^{(t)} + \mathbf{U}_c\mathbf{h}^{(t-1)} + \mathbf{b}_c) \tag{43}$$

But, as mentioned, we do not allow *all* of this vector to enter the cell—hence why we call it the vector of *candidate* features, and denote it as $\widetilde{\mathbf{c}}^{(t)}$. Instead, the LSTM directly learns *gating vectors*, which are real-valued vectors in the range $[0, 1]$, and decide how much of the signal should be allowed to enter, exit, and overwrite the memory cell.

Three such gates are computed, all based on $\mathbf{z}^{(t)}$ and $\mathbf{h}^{(t-1)}$: the *input gate* $\mathbf{i}^{(t)}$, which computes the proportion of the candidate vector allowed to enter the cell; the *forget gate* $\mathbf{f}^{(t)}$, which computes the proportion of the previous cell state to be retained, and the *output gate* $\mathbf{o}^{(t)}$, which computes the proportion of the new cell state to be used for the final summary vector. Typically all of these gates are also derived using a single fully connected layer, albeit with the *logistic sigmoid* activation $\mathrm{logistic}(x) = \frac{1}{1+\exp(-x)}$, in order to guarantee that the outputs are in the $[0, 1]$ range:

Note that the three gates are themselves *vectors*, i.e. $\mathbf{i}^{(t)}, \mathbf{f}^{(t)}, \mathbf{o}^{(t)} \in [0, 1]^m$. This allows them to control how much *each* of the m dimensions is allowed through the gate.

$$\mathbf{i}^{(t)} = \mathrm{logistic}(\mathbf{W}_\mathrm{i}\mathbf{z}^{(t)} + \mathbf{U}_\mathrm{i}\mathbf{h}^{(t-1)} + \mathbf{b}_\mathrm{i}) \tag{44}$$

$$\mathbf{f}^{(t)} = \mathrm{logistic}(\mathbf{W}_\mathrm{f}\mathbf{z}^{(t)} + \mathbf{U}_\mathrm{f}\mathbf{h}^{(t-1)} + \mathbf{b}_\mathrm{f}) \tag{45}$$

$$\mathbf{o}^{(t)} = \mathrm{logistic}(\mathbf{W}_\mathrm{o}\mathbf{z}^{(t)} + \mathbf{U}_\mathrm{o}\mathbf{h}^{(t-1)} + \mathbf{b}_\mathrm{o}) \tag{46}$$

Finally, these gates are appropriately applied to decode the *new* cell state, $\mathbf{c}^{(t)}$, which is then modulated by the output gate to produce the summary vector $\mathbf{h}^{(t)}$, as follows:

$$\mathbf{c}^{(t)} = \mathbf{i}^{(t)} \odot \widetilde{\mathbf{c}}^{(t)} + \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} \tag{47}$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)}) \tag{48}$$

where $\odot$ is element-wise vector multiplication. Applied together, Equations (43)–(48) completely specify the *update rule* for the LSTM, which now takes into account the cell vector $\mathbf{c}^{(t)}$ as well:

This is still compatible with the RNN update blueprint from Equation (38); simply consider the summary vector to be the *concatenation* of $\mathbf{h}^{(t)}$ and $\mathbf{c}^{(t)}$; sometimes denoted by $\mathbf{h}^{(t)} \| \mathbf{c}^{(t)}$.

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

Note that, as the values of $\mathbf{f}^{(t)}$ are derived from $\mathbf{z}^{(t)}$ and $\mathbf{h}^{(t-1)}$—and therefore directly *learnable* from data—the LSTM effectively learns how to appropriately forget past experiences. Indeed, the values of $\mathbf{f}^{(t)}$ directly appear in

the backpropagation update for all the LSTM parameters $(\mathbf{W}_*, \mathbf{U}_*, \mathbf{b}_*)$, allowing the network to explicitly *control*, in a data-driven way, the degree of vanishing for the gradients across the time steps.

Besides tackling the vanishing gradient issue head-on, it turns out that gated RNNs also unlock a very useful form of invariance to *time-warping* transformations, which remains out of reach of SimpleRNNs.

Time warping invariance of gated RNNs We will start by illustrating, in a *continuous-time* setting, what does it mean to *warp time*, and what is required of a recurrent model in order to achieve invariance to such transformations. Our exposition will largely follow the work of Tallec and Ollivier (2018), that initially described this phenomenon—and indeed, they were among the first to actually study RNNs from the lens of invariances.

We focus on the continuous setting as it will be easier to reason about manipulations of time there.

Let us assume a continuous time-domain signal $z(t)$, on which we would like to apply an RNN. To align the RNN's discrete-time computation of summary vectors $\mathbf{h}^{(t)}$ with an analogue in the continuous domain, $h(t)$, we will observe its linear Taylor expansion:

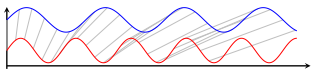
We will use $h(t)$ to denote a continuous signal at time t, and $\mathbf{h}^{(t)}$ to denote a discrete signal at time-step t.

$$h(t+\delta) \approx h(t) + \delta \frac{\mathrm{d}h(t)}{\mathrm{d}t} \tag{49}$$

and, setting $\delta = 1$, we recover a relationship between $h(t)$ and $h(t+1)$, which is exactly what the RNN update function R (Equation 38) computes. Namely, the RNN update function satisfies the following differential equation:

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = h(t+1) - h(t) = R(z(t+1), h(t)) - h(t) \tag{50}$$

We would like the RNN to be resilient to the way in which the signal is sampled (e.g. by changing the time unit of measurement), in order to account for any imperfections or irregularities therein. Formally, we denote a *time warping* operation $\tau : \mathbb{R}^+ \to \mathbb{R}^+$, as any monotonically increasing differentiable mapping between times. The notation τ is chosen because time warping represents an *automorphism* of time.



Such warping operations can be *simple*, such as time rescaling; e.g. $\tau(t) = 0.7t$ (displayed above), which, in a discrete setting, would amount to new inputs being received every ~ 1.43 steps. However, it also admits a wide spectrum of *variably-changing* sampling rates, e.g. sampling may freely accelerate or decelerate throughout the time domain.

Further, we state that a class of models is *invariant* to time warping if, for any model of the class and any such τ, there exists another (possibly the same) model from the class that processes the warped data in the same way as the original model did in the non-warped case.

This is a potentially very useful property. If we have an RNN class capable of modelling short-term dependencies well, and we can also show that this class is invariant to time warping, then we know it is possible to train such a model in a way that will usefully capture long-term dependencies as well (as they would correspond to a time dilation warping of a signal with short-term dependencies). As we will shortly see, it is no coincidence that *gated* RNN models such as the LSTM were proposed to model long-range dependencies. Achieving time warping invariance is tightly coupled with presence of gating mechanisms, such as the input/forget/output gates of LSTMs.

When time gets warped by τ, the signal observed by the RNN at time t is $z(\tau(t))$ and, to remain invariant to such warpings, it should predict an equivalently-warped summary function $h(\tau(t))$. Using Taylor expansion arguments once more, we derive a form of Equation 50 for the warped time, that the RNN update R should satisfy:

$$\frac{\mathrm{d}h(\tau(t))}{\mathrm{d}\tau(t)} = R(z(\tau(t+1)), h(\tau(t))) - h(\tau(t)) \tag{51}$$

However, the above derivative is computed with respect to the warped time $\tau(t)$, and hence does not take into account the original signal. To make our model take into account the warping transformation explicitly, we need to differentiate the warped summary function with respect to t. Applying the chain rule, this yields the following differential equation:

$$\frac{\mathrm{d}h(\tau(t))}{\mathrm{d}t} = \frac{\mathrm{d}h(\tau(t))}{\mathrm{d}\tau(t)}\frac{\mathrm{d}\tau(t)}{\mathrm{d}t} = \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}R(z(\tau(t+1)), h(\tau(t))) - \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}h(\tau(t)) \tag{52}$$

and, for our (continuous-time) RNN to remain invariant to *any* time warping $\tau(t)$, it needs to be able to explicitly represent the derivative $\frac{\mathrm{d}\tau(t)}{\mathrm{d}t}$, which is not assumed known upfront! We need to introduce a *learnable* function Γ which approximates this derivative. For example, Γ could be a neural network taking into account $z(t+1)$ and $h(t)$ and predicting scalar outputs.

Now, remark that, from the point of view of a *discrete* RNN model under time warping, its input $\mathbf{z}^{(t)}$ will correspond to $z(\tau(t))$, and its summary $\mathbf{h}^{(t)}$ will correspond to $h(\tau(t))$. To obtain the required relationship of $\mathbf{h}^{(t)}$ to $\mathbf{h}^{(t+1)}$ in order to remain invariant to time warping, we will use a one-step Taylor expansion of $h(\tau(t))$:

$$h(\tau(t+\delta)) \approx h(\tau(t)) + \delta\frac{\mathrm{d}h(\tau(t))}{\mathrm{d}t}$$

and, once again, setting $\delta = 1$ and substituting Equation 52, then discretising:

$$\begin{aligned}\mathbf{h}^{(t+1)} &= \mathbf{h}^{(t)} + \frac{\mathrm{d}\tau(t)}{\mathrm{d}t} R(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) - \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}\mathbf{h}^{(t)} \\ &= \frac{\mathrm{d}\tau(t)}{\mathrm{d}t} R(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) + \left(1 - \frac{\mathrm{d}\tau(t)}{\mathrm{d}t}\right)\mathbf{h}^{(t)}\end{aligned}$$

Finally, we swap $\frac{\mathrm{d}\tau(t)}{\mathrm{d}t}$ with the aforementioned learnable function, Γ. This gives us the required form for our time warping-invariant RNN:

$$\mathbf{h}^{(t+1)} = \Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) R(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) + (1 - \Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}))\mathbf{h}^{(t)} \tag{53}$$

We may quickly deduce that SimpleRNNs (Equation 39) are *not* time warping invariant, given that they do not feature the second term in Equation 53. Instead, they fully overwrite $\mathbf{h}^{(t)}$ with $R(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)})$, which corresponds to assuming no time warping at all; $\frac{\mathrm{d}\tau(t)}{\mathrm{d}t} = 1$, i.e. $\tau(t) = t$.

Further, our link between continuous-time RNNs and the discrete RNN based on R rested on the accuracy of the Taylor approximation, which holds only if the time-warping derivative is not too large, i.e., $\frac{\mathrm{d}\tau(t)}{\mathrm{d}t} \lesssim 1$. The intuitive explanation of this is: if our time warping operation ever *contracts time* in a way that makes time increments ($t \to t + 1$) large enough that intermediate data changes are not sampled, the model can never hope to process time-warped inputs in the same way as original ones—it simply would not have access to the same information. Conversely, time *dilations* of any form (which, in discrete terms, correspond to interspersing the input time-series with zeroes) are perfectly allowed within our framework.

Combined with our requirement of monotonically increasing τ ($\frac{\mathrm{d}\tau(t)}{\mathrm{d}t} > 0$), we can bound the output space of Γ as $0 < \Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) < 1$, which motivates the use of the logistic sigmoid activation for Γ, e.g.:

$$\Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) = \mathrm{logistic}(\mathbf{W}_\Gamma \mathbf{z}^{(t+1)} + \mathbf{U}_\Gamma \mathbf{h}^{(t)} + \mathbf{b}_\Gamma)$$

exactly matching the LSTM gating equations (e.g. Equation 44). The main difference is that LSTMs compute gating *vectors*, whereas Equation 53 implies Γ should output a scalar. Vectorised gates (Hochreiter, 1991) allow to fit a *different* warping derivative in every dimension of $\mathbf{h}^{(t)}$, allowing for reasoning over *multiple* time horizons simultaneously.

It is worth taking a pause here to summarise what we have done. By requiring that our RNN class is invariant to (non-destructive) time warping, we have

derived the necessary form that it must have (Equation 53), and showed that it exactly corresponds to the class of *gated* RNNs. The gates' primary role under this perspective is to accurately fit the *derivative* $\frac{\mathrm{d}\tau(t)}{\mathrm{d}t}$ of the warping transformation.

The notion of *class invariance* is somewhat distinct from the invariances we studied previously. Namely, once we train a gated RNN on a time-warped input with $\tau_1(t)$, we typically cannot zero-shot transfer it to a signal warped by a different $\tau_2(t)$. Rather, class invariance only guarantees that gated RNNs are powerful enough to fit both of these signals in the same manner, but potentially with vastly different model parameters. That being said, the realisation that effective gating mechanisms are tightly related to fitting the warping derivative can yield useful prescriptions for gated RNN optimisation, as we now briefly demonstrate.

One case where zero-shot transfer is possible is when the second time warping is assumed to be a *time rescaling* of the first one ($\tau_2(t) = \alpha\tau_1(t)$). Transferring a gated RNN pre-trained on τ_1 to a signal warped by τ_2 merely requires *rescaling the gates*: $\Gamma_2(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)}) = \alpha\Gamma_1(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)})$. R can retain its parameters ($R_1 = R_2$).

For example, we can often assume that the range of the dependencies we are interested in tracking within our signal will be in the range $[T_l, T_h]$ time-steps.

By analysing the analytic solutions to Equation 52, it can be shown that the characteristic *forgetting time* of $\mathbf{h}^{(t)}$ by our gated RNN is proportional to $\frac{1}{\Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)})}$. Hence, we would like our gating values to lie between $\left[\frac{1}{T_h}, \frac{1}{T_m}\right]$ in order to effectively remember information within the assumed range.

Further, if we assume that $\mathbf{z}^{(t)}$ and $\mathbf{h}^{(t)}$ are roughly *zero-centered*—which is a common by-product of applying transformations such as layer normalisation (Ba et al., 2016)—we can assume that $\mathbb{E}[\Gamma(\mathbf{z}^{(t+1)}, \mathbf{h}^{(t)})] \approx \mathrm{logistic}(\mathbf{b}_\Gamma)$. Controlling the *bias* vector of the gating mechanism is hence a very powerful way of controlling the effective gate value.

This insight was already spotted by Gers and Schmidhuber (2000); Jozefowicz et al. (2015), who empirically recommended initialising the forget-gate bias of LSTMs to a constant positive vector, such as $\mathbf{1}$.

Combining the two observations, we conclude that an appropriate range of gating values can be obtained by initialising $\mathbf{b}_\Gamma \sim -\log(\mathcal{U}(T_l, T_h) - 1)$, where $\mathcal{U}$ is the uniform real distribution. Such a recommendation was dubbed *chrono initialisation* by Tallec and Ollivier (2018), and has been empirically shown to improve the long-range dependency modelling of gated RNNs.

Sequence-to-sequence learning with RNNs One prominent historical example of using RNN-backed computation are *sequence-to-sequence* translation tasks, such as *machine translation* of natural languages. The pioneering *seq2seq* work by Sutskever et al. (2014) achieved this by passing the summary vector,

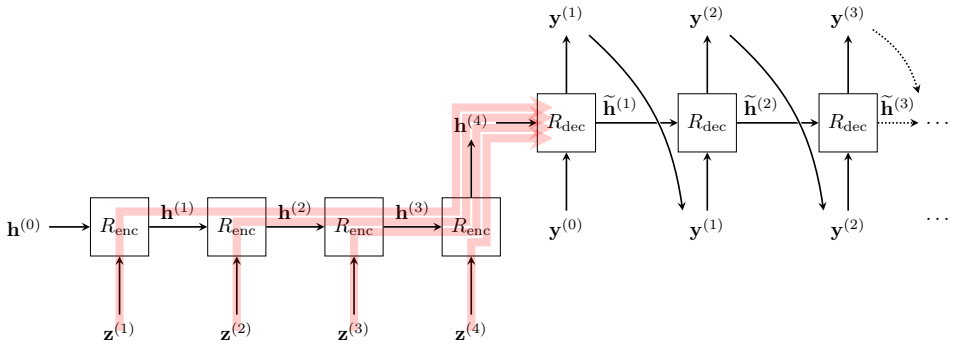


Figure 21: One typical example of a seq2seq architecture with an RNN encoder R_{enc} and RNN decoder R_{dec}. The decoder is seeded with the final summary vector $\mathbf{h}^{(T)}$ coming out of the encoder, and then proceeds in an *autoregressive* fashion: at each step, the predicted output from the previous step is fed back as input to R_{dec}. The bottleneck problem is also illustrated with the red lines: the summary vector $\mathbf{h}^{(T)}$ is pressured to store *all* relevant information for translating the input sequence, which becomes increasingly challenging as the input length grows.

$\mathbf{h}^{(T)}$ as an initial input for a *decoder* RNN, with outputs of RNN blocks being given as inputs for the next step.

This placed substantial representational pressure on the summary vector, $\mathbf{h}^{(T)}$. Within the context of deep learning, $\mathbf{h}^{(T)}$ is sometimes referred to as a *bottleneck*. Its fixed capacity must be sufficient for representing the content of the entire input sequence, in a manner that is conducive to generating a corresponding sequence, while also supporting input sequences of substantially different lengths (Figure 21).

The bottleneck effect has recently received substantial attention in the graph representation learning community (Alon and Yahav, 2020), as well as neural algorithmic reasoning (Cappart et al., 2021).

In reality, different steps of the output may wish to focus (*attend*) on different parts of the input, and all such choices are difficult to represent via a bottleneck vector. Following from this observation, the popular *recurrent attention* model was proposed by Bahdanau et al. (2014). At every step of processing, a *query vector* is generated by an RNN; this query vector then interacts with the representation of *every* time-step $\mathbf{h}^{(t)}$, primarily by computing a weighted sum over them. This model pioneered neural content-based attention and predates the success of the Transformer model.

Lastly, while attending offers a *soft* way to dynamically focus on parts of

the input content, substantial work also learnt more *explicit* ways to direct attention to the input. A powerful algorithmically grounded way of doing so is the *pointer network* of Vinyals et al. (2015), which proposes a simple modification of recurrent attention to allow for pointing over elements of *variable-sized* inputs. These findings have then been generalised to the *set2set* architecture (Vinyals et al., 2016), which generalises seq2seq models to unordered sets, supported by pointer network-backed LSTMs.

6 Problems and Applications

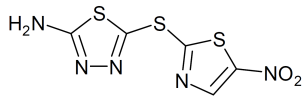
Invariances and symmetries arise all too commonly across data originating in the real world. Hence, it should come as no surprise that some of the most popular applications of machine learning in the 21st century have come about as a direct byproduct of Geometric Deep Learning, perhaps sometimes without fully realising this fact. We would like to provide readers with an overview—by no means comprehensive—of influential works in Geometric Deep Learning and exciting and promising new applications. Our motivation is twofold: to demonstrate specific instances of scientific and industrial problems where the five geometric domains commonly arise, and to serve additional motivation for further study of Geometric Deep Learning principles and architectures.

Chemistry and Drug Design One of the most promising applications of representation learning on graphs is in computational chemistry and *drug development*. Traditional drugs are small molecules that are designed to chemically attach ('bind') to some target molecule, typically a protein, in order to activate or disrupt some disease-related chemical process. Unfortunately, drug development is an extremely long and expensive process: at the time of writing, bringing a new drug to the market typically takes more than a decade and costs more than a billion dollars. One of the reasons is the cost of testing where many drugs fail at different stages – less than 5% of candidates make it to the last stage (see e.g. Gaudelet et al. (2020)).

Many drugs are not designed but discovered, often serendipitously. The historic source of a number of drugs from the plant kingdom is reflected in their names: e.g., the acetylsalicylic acid, commonly known as *aspirin*, is contained in the bark of the willow tree (*Salix alba*), whose medicinal properties are known since antiquity.

Since the space of chemically synthesisable molecules is very large (estimated around 10^{60}), the search for candidate molecules with the right combination of properties such as target binding affinity, low toxicity, solubility, etc. cannot be done experimentally, and *virtual* or *in silico screening* (i.e., the use

of computational techniques to identify promising molecules), is employed. Machine learning techniques play an increasingly more prominent role in this task. A prominent example of the use of Geometric Deep Learning for virtual drug screening was recently shown by Stokes et al. (2020) using a graph neural network trained to predict whether or not candidate molecules inhibit growth in the model bacterium *Escherichia coli*, they were able to effectively discover that *Halicin*, a molecule originally indicated for treating diabetes, is a highly potent antibiotic, even against bacteria strains with known antibiotic resistance. This discovery was widely covered in both scientific and popular press.



Molecular graph of Halicin.

Speaking more broadly, the application of graph neural networks to molecules modeled as graphs has been a very active field, with multiple specialised architectures proposed recently that are inspired by physics and e.g. incorporate equivariance to rotations and translations (see e.g. Thomas et al. (2018); Anderson et al. (2019); Fuchs et al. (2020); Satorras et al. (2021)). Further, Bapst et al. (2020) have successfully demonstrated the utility of GNNs for predictively modelling the dynamics of glass, in a manner that outperformed the previously available physics-based models. Historically, many works in computational chemistry were precursors of modern graph neural network architectures sharing many common traits with them.

Drug Repositioning While generating entirely novel drug candidates is a potentially viable approach, a faster and cheaper avenue for developing new therapies is *drug repositioning*, which seeks to evaluate already-approved drugs (either alone or in combinations) for a novel purpose. This often significantly decreases the amount of clinical evaluation that is necessary to release the drug to the market. At some level of abstraction, the action of drugs on the body biochemistry and their interactions between each other and other biomolecules can be modeled as a graph, giving rise to the concept of 'network medicine' coined by the prominent network scientist Albert-László Barabási and advocating the use of biological networks (such as protein-protein interactions and metabolic pathways) to develop new therapies (Barabási et al., 2011).

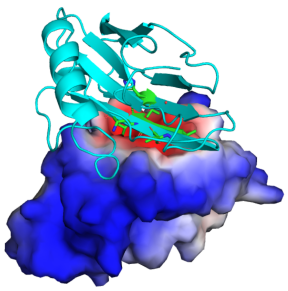
Geometric Deep Learning offers a modern take on this class of approaches. A prominent early example is the work of Zitnik et al. (2018), who used graph neural networks to predict side effects in a form of drug repositioning known as *combinatorial therapy* or *polypharmacy*, formulated as edge prediction in

a drug-drug interaction graph. The novel coronavirus pandemic, which is largely ongoing at the time of writing this text, has sparked a particular interest in attempting to apply such approaches against COVID-19 (Gysi et al., 2020). Finally, we should note that drug repositioning is not necessarily limited to synthetic molecules: Veselkov et al. (2019) applied similar approaches to drug-like molecules contained in food (since, as we mentioned, many plant-based foods contain biological analogues of compounds used in oncological therapy). One of the authors of this text is involved in a collaboration adding a creative twist to this research, by partnering with a molecular chef that designs exciting recipes based on the 'hyperfood' ingredients rich in such drug-like molecules.

Protein biology Since we have already mentioned proteins as drug targets, lets us spend a few more moments on this topic. Proteins are arguably among the most important biomolecules that have myriads of functions in our body, including protection against pathogens (antibodies), giving structure to our skin (collagen), transporting oxygen to cells (haemoglobin), catalysing chemical reactions (enzymes), and signaling (many hormones are proteins). Chemically speaking, a protein is a biopolymer, or a chain of small building blocks called *aminoacids* that under the influence of electrostatic forces fold into a complex 3D structure. It is this structure that endows the protein with its functions, and hence it is crucial to the understanding of how proteins work and what they do. Since proteins are common targets for drug therapies, the pharmaceutical industry has a keen interest in this field.

A common metaphor, dating back to the chemistry Nobel laureate Emil Fischer is the *Schlüssel-Schloss-Prinzip* ('key-lock principle', 1894): two proteins often only interact if they have geometrically and chemically complementary structures.

A typical hierarchy of problems in protein bioinformatics is going from protein *sequence* (a 1D string over an alphabet of of 20 different amino acids) to 3D *structure* (a problem known as 'protein folding') to *function* ('protein function prediction'). Recent approaches such as DeepMind's AlphaFold by Senior et al. (2020) used *contact graphs* to represent the protein structure. Gligorijevic et al. (2020) showed that applying graph neural networks on such graphs allows to achieve better function prediction than using purely sequence-based methods.

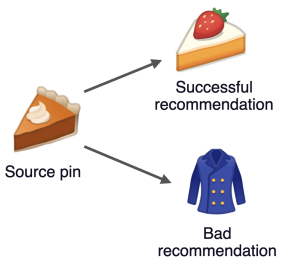


Oncologial target PD-L1 protein surface (heat map indicated the predicted binding site) and the designed binder (shown as ribbon diagram).

Gainza et al. (2020) developed a Geometric Deep Learning pipeline called MaSIF predicting interactions between proteins from their 3D structure. MaSIF models the protein as a molecular surface discretised as a mesh, arguing that this representation is advantageous when dealing with interactions as it allows to abstract the internal fold structure. The architecture was based

on mesh convolutional neural network operating on pre-computed chemical and geometric features in small local geodesic patches. The network was trained using a few thousand co-crystal protein 3D structures from the Protein Data Bank to address multiple tasks, including interface prediction, ligand classification, and docking, and allowed to do *de novo* ('from scratch') design of proteins that could in principle act as biological immunotherapy drug against cancer – such proteins are designed to inhibit protein-protein interactions (PPI) between parts of the programmed cell death protein complex (PD-1/PD-L1) and give the immune system the ability to attack the tumor cells.

Recommender Systems and Social Networks The first popularised large-scale applications of graph representation learning have occurred within *social networks*, primarily in the context of *recommender systems*. Recommenders are tasked with deciding which content to serve to users, potentially depending on their previous history of interactions on the service. This is typically realised through a *link prediction* objective: supervise the embeddings of various nodes (pieces of content) such that they are kept close together if they are deemed *related* (e.g. commonly viewed together). Then the *proximity* of two embeddings (e.g. their inner product) can be interpreted as the probability that they are linked by an edge in the content graph, and hence for any content queried by users, one approach could serve its k nearest neighbours in the embedding space.

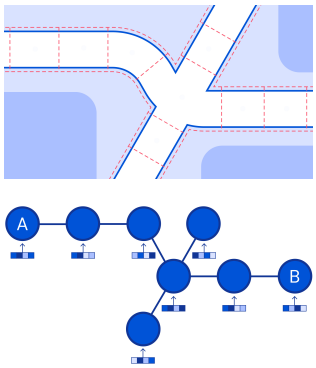


Pinterest had also presented follow-up work, PinnerSage (Pal et al., 2020), which effectively integrates user-specific contextual information into the recommender.

Among the pioneers of this methodology is the American image sharing and social media company Pinterest: besides presenting one of the first successful deployments of GNNs in production, their method, PinSage, successfully made graph representation learning *scalable* to graphs of millions of nodes and billions of edges (Ying et al., 2018). Related applications, particularly in the space of product recommendations, soon followed. Popular GNN-backed recommenders that are currently deployed in production include Alibaba's Aligraph (Zhu et al., 2019) and Amazon's P-Companion (Hao et al., 2020). In this way, graph deep learning is influencing millions of people on a daily level.

Within the context of content analysis on social networks, another noteworthy effort is Fabula AI, which is among the first GNN-based startups to be acquired (in 2019, by Twitter). The startup, founded by one of the authors of the text and his team, developed novel technology for detecting

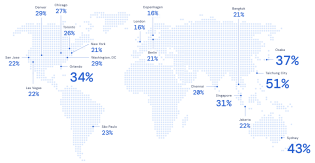
misinformation on social networks (Monti et al., 2019). Fabula's solution consists of modelling the spread of a particular news item by the network of users who shared it. The users are connected if one of them re-shared the information from the other, but also if they follow each other on the social network. This graph is then fed into a graph neural network, which classifies the entire graph as either 'true' or 'fake' content – with labels based on agreement between fact-checking bodies. Besides demonstrating strong predictive power which stabilises quickly (often within a few hours of the news spreading), analysing the embeddings of individual user nodes revealed clear clustering of users who tend to share incorrect information, exemplifying the well-known *'echo chamber'* effect.



A road network (top) with its corresponding graph representation (bottom).

Traffic forecasting Transportation networks are another area where Geometric Deep Learning techniques are already making an actionable impact over billions of users worldwide. For example, on road networks, we can observe intersections as nodes, and road segments as edges connecting them—these edges can then be featurised by the road length, current or historical speeds along their segment, and the like.

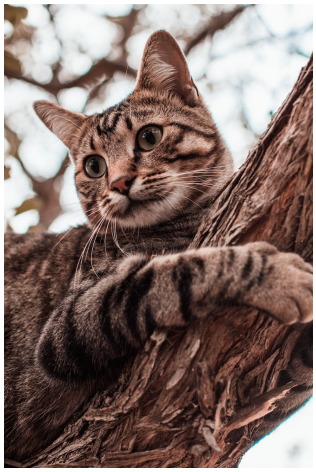
One standard prediction problem in this space is predicting the *estimated time of arrival* (ETA): for a given candidate route, providing the expected travel time necessary to traverse it. Such a problem is essential in this space, not only for user-facing traffic recommendation apps, but also for enterprises (such as food delivery or ride-sharing services) that leverage these predictions within their own operations.



Several of the metropolitan areas where GNNs are serving queries within Google Maps, with indicated relative improvements in prediction quality (40+% in cities like Sydney).

Graph neural networks have shown immense promise in this space as well: they can, for example, be used to directly predict the ETA for a relevant subgraph of the road network (effectively, a *graph regression* task). Such an approach was successfully leveraged by DeepMind, yielding a GNN-based ETA predictor which is now deployed in production at Google Maps (Derrow-Pinion et al., 2021), serving ETA queries in several major metropolitan areas worldwide. Similar returns have been observed by the Baidu Maps team, where travel time predictions are currently served by the ConSTGAT model, which is itself based on a spatio-temporal variant of the graph attention network model (Fang et al., 2020).

Object recognition A principal benchmark for machine learning techniques in computer vision is the ability to *classify* a central object within a provided image. The ImageNet large scale visual recognition challenge (Russakovsky et al., 2015, ILSVRC) was an annual object classification challenge that propelled much of the early development in Geometric Deep Learning. ImageNet requires models to classify realistic images scraped from the Web into one of 1000 categories: such categories are at the same time diverse (covering both animate and inanimate objects), and specific (with many classes focused on distinguishing various cat and dog breeds). Hence, good performance on ImageNet often implies a solid level of feature extraction from general photographs, which formed a foundation for various *transfer learning* setups from pre-trained ImageNet models.



One example input image, the likes of which can be found in ImageNet, representing the *"tabby cat"* class.

Interestingly, the VGG-16 architecture has sixteen convolutional layers and is denoted as "very deep" by the authors. Subsequent developments quickly scaled up such models to hundreds or even thousands of layers.

The success of convolutional neural networks on ImageNet—particularly the AlexNet model of Krizhevsky et al. (2012), which swept ILSVRC 2012 by a large margin—has in a large way spearheaded the adoption of deep learning as a whole, both in academia and in industry. Since then, CNNs have consistently ranked on top of the ILSVRC, spawning many popular architectures such as VGG-16 (Simonyan and Zisserman, 2014), Inception (Szegedy et al., 2015) and ResNets (He et al., 2016), which have successfully surpassed human-level performance on this task. The design decisions and regularisation techniques employed by these architectures (such as rectified linear activations (Nair and Hinton, 2010), dropout (Srivastava et al., 2014), skip connections (He et al., 2016) and batch normalisation (Ioffe and Szegedy, 2015)) form the backbone of many of the effective CNN models in use today.

Concurrently with object classification, significant progress had been made on object *detection*; that is, isolating all objects of interest within an image, and tagging them with certain classes. Such a task is relevant in a variety of downstream problems, from image captioning all the way to autonomous vehicles. It necessitates a more fine-grained approach, as the predictions need to be *localised*; as such, often, translation equivariant models have proven their worth in this domain. One impactful example in this space includes the R-CNN family of models (Girshick et al., 2014; Girshick, 2015; Ren et al., 2015; He et al., 2017) whereas, in the related field of *semantic segmentation*, the SegNet model of Badrinarayanan et al. (2017) proved influential, with its encoder-decoder architecture relying on the VGG-16 backbone.

Game playing Convolutional neural networks also play a prominent role as translation-invariant feature extractors in *reinforcement learning* (RL) environments, whenever the observed state can be represented in a grid domain; e.g. this is the case when learning to play video games from pixels. In this case, the CNN is responsible for reducing the input to a flat vector representation, which is then used for deriving *policy* or *value functions* that drive the RL agent's behaviour. While the specifics of reinforcement learning are not the focus of this section, we do note that some of the most impactful results of deep learning in the past decade have come about through CNN-backed reinforcement learning.

One particular example that is certainly worth mentioning here is DeepMind's *AlphaGo* (Silver et al., 2016). It encodes the current state within a game of Go by applying a CNN to the 19×19 grid representing the current positions of the placed stones. Then, through a combination of learning from previous expert moves, Monte Carlo tree search, and self-play, it had successfully reached a level of Go mastery that was sufficient to outperform Lee Sedol, one of the strongest Go players of all time, in a five-round challenge match that was widely publicised worldwide.

The game of Go is played on a 19×19 board, with two players placing white and black *stones* on empty fields. The number of legal states has been estimated at $\approx 2 \times 10^{170}$ (Tromp and Farnebäck, 2006), vastly outnumbering the number of atoms in the universe.

While this already represented a significant milestone for broader artificial intelligence—with Go having a substantially more complex state-space than, say, chess—the development of AlphaGo did not stop there. The authors gradually removed more and more Go-specific biases from the architecture, with *AlphaGo Zero* removing human biases, optimising purely through self-play (Silver et al., 2017), *AlphaZero* expands this algorithm to related two-player games, such as Chess and Shogi; lastly, *MuZero* (Schrittwieser et al., 2020) incorporates a model that enables *learning* the rules of the game on-the-fly, which allows reaching strong performance in the Atari 2600 console, as well as Go, Chess and Shogi, without any upfront knowledge of the rules. Throughout all of these developments, CNNs remained the backbone behind these models' representation of the input.

While several high-performing RL agents were proposed for the Atari 2600 platform over the years (Mnih et al., 2015, 2016; Schulman et al., 2017), for a long time they were unable to reach human-level performance on *all* of the 57 games provided therein. This barrier was finally broken with Agent57 (Badia et al., 2020), which used a parametric family of policies, ranging from strongly exploratory to purely exploitative, and prioritising them in different ways during different stages of training. It, too, powers most of its

computations by a CNN applied to the video game's framebuffer.

Text and speech synthesis Besides images (which naturally map to a *two*-dimensional grid), several of (geometric) deep learning's strongest successes have happened on one-dimensional grids. Natural examples of this are *text* and *speech*, folding the Geometric Deep Learning blueprint within diverse areas such as natural language processing and digital signal processing.

Some of the most widely applied and publicised works in this space focus on *synthesis*: being able to generate speech or text, either unconditionally or conditioned on a particular *prompt*. Such a setup can support a plethora of useful tasks, such as *text-to-speech* (TTS), predictive text completion, and machine translation. Various neural architectures for text and speech generation have been proposed over the past decade, initially mostly based on *recurrent* neural networks (e.g. the aforementioned seq2seq model (Sutskever et al., 2014) or recurrent attention (Bahdanau et al., 2014)). However, in recent times, they have been gradually replaced by convolutional neural networks and Transformer-based architectures.

One particular limitation of simple 1D convolutions in this setting is their linearly growing *receptive field*, requiring many layers in order to cover the sequence generated so far. *Dilated* convolutions, instead, offer an *exponentially* growing receptive field with an equivalent number of parameters. Owing to this, they proved a very strong alternative, eventually becoming competitive with RNNs on machine translation (Kalchbrenner et al., 2016), while drastically reducing the computational complexity, owing to their parallelisability across all input positions. The most well-known application of dilated convolutions is the *WaveNet* model from van den Oord et al. (2016a). WaveNets demonstrated that, using dilations, it is possible to synthesise speech at the level of *raw waveform* (typically 16,000 samples per second or more), producing speech samples that were significantly more "human-like" than the best previous text-to-speech (TTS) systems. Subsequently, it was further demonstrated that the computations of WaveNets can be distilled in a much simpler model, the *WaveRNN* (Kalchbrenner et al., 2018)—and this model enabled effectively deploying this technology at an industrial scale. This allowed not only its deployment for large-scale speech generation for services such as the Google Assistant, but also allowing for efficient on-device computations; e.g. for Google Duo, which uses end-to-end encryption.

Dilated convolution is also referred to as *à trous* convolution (literally *"holed"* in French).

Such techniques have also outperformed RNNs on problems as diverse as protein-protein interaction (Deac et al., 2019).

Besides this, the WaveNet model proved capable of generating piano pieces.

Transformers (Vaswani et al., 2017) have managed to surpass the limitations of both recurrent and convolutional architectures, showing that *self-attention* is sufficient for achieving state-of-the-art performance in machine translation. Subsequently, they have revolutionised natural language processing. Through the pre-trained embeddings provided by models such as BERT (Devlin et al., 2018), Transformer computations have become enabled for a large amount of downstream applications of natural language processing—for example, Google uses BERT embeddings to power its search engine.

Arguably the most widely publicised application of Transformers in the past years is text generation, spurred primarily by the *Generative Pre-trained Transformer* (GPT, Radford et al. (2018, 2019); Brown et al. (2020)) family of models from OpenAI. In particular, GPT-3 (Brown et al., 2020) successfully scaled language model learning to 175 billion learnable parameters, trained on next-word prediction on web-scale amounts of scraped textual corpora. This allowed it not only to become a highly-potent few-shot learner on a variety of language-based tasks, but also a text generator with capability to produce coherent and human-sounding pieces of text. This capability not only implied a large amount of downstream applications, but also induced a vast media coverage.

Healthcare Applications in the medical domain are another promising area for Geometric Deep Learning. There are multiple ways in which these methods are being used. First, more traditional architectures such as CNNs have been applied to grid-structured data, for example, for the prediction of length of stay in Intensive Care Units (Rocheteau et al., 2020), or diagnosis of sight-threatening diseases from retinal scans (De Fauw et al., 2018). Winkels and Cohen (2019) showed that using 3D roto-translation group convolutional networks improves the accuracy of pulmonary nodule detection compared to conventional CNNs.

Second, modelling organs as geometric surfaces, mesh convolutional neural networks were shown to be able to address a diverse range of tasks, from reconstructing facial structure from genetics-related information (Mahdi et al., 2020) to brain cortex parcellation (Cucurull et al., 2018) to regressing demographic properties from cortical surface structures (Besson et al., 2020). The latter examples represent an increasing trend in neuroscience to consider the brain as a surface with complex folds giving rise to highly non-Euclidean structures.

Such structure of the brain cortex are called *sulci* and *gyri* in anatomical literature.

At the same time, neuroscientists often try construct and analyse *functional networks* of the brain representing the various regions of the brain that are activated together when performing some cognitive function; these networks are often constructed using functional magnetic resonance imaging (fMRI) that shows in real time which areas of the brain consume more blood. These functional networks can reveal patient demographics (e.g., telling apart males from females, Arslan et al. (2018)), as well as used for neuropathology diagnosis, which is the third area of application of Geometric Deep Learning in medicine we would like to highlight here. In this context, Ktena et al. (2017) pioneered the use of graph neural networks for the prediction of neurological conditions such as Autism Spectrum Disorder. The geometric and functional structure of the brain appears to be intimately related, and recently Itani and Thanou (2021) pointed to the benefits of exploiting them jointly in neurological disease analysis.

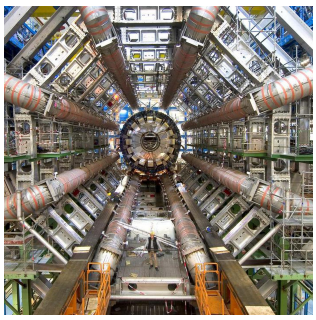
Typically, Blood Oxygen-Level Dependent (BOLD) contrast imaging is used.

Fourth, *patient networks* are becoming more prominent in ML-based medical diagnosis. The rationale behind these methods is that the information of patient demographic, genotypic, and phenotypic similarity could improve predicting their disease. Parisot et al. (2018) applied graph neural networks on networks of patients created from demographic features for neurological disease diagnosis, showing that the use of the graph improves prediction results. Cosmo et al. (2020) showed the benefits of latent graph learning (by which the network *learns* an unknown patient graph) in this setting. The latter work used data from the UK Biobank, a large-scale collection of medical data including brain imaging (Miller et al., 2016).

A wealth of data about hospital patients may be found in *electronic health records* (EHRs). Besides giving a comprehensive view of the patient's progression, EHR analysis allows for *relating* similar patients together. This aligns with the *pattern recognition method*, which is commonly used in diagnostics. Therein, the clinician uses *experience* to recognise a pattern of clinical characteristics, and it may be the primary method used when the clinician's experience may enable them to diagnose the condition quickly. Along these lines, several works attempt to construct a patient graph based on EHR data, either by analysing the embeddings of their doctor's notes (Malone et al., 2018), diagnosis similarity on admission (Rocheteau et al., 2021), or even assuming a fully-connected graph (Zhu and Razavian, 2019). In all cases, promising results have been shown in favour of using graph representation learning for processing EHRs.

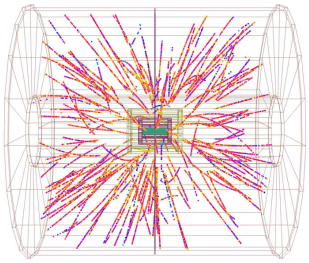
Publicly available anonymised critical-care EHR datasets include MIMIC-III (Johnson et al., 2016) and eICU (Pollard et al., 2018).

Particle physics and astrophysics High energy physicists were perhaps among the first domain experts in the field of natural sciences to embrace the new shiny tool, graph neural networks. In a recent review paper, Shlomi et al. (2020) note that machine learning has historically been heavily used in particle physics experiments, either to learn complicated inverse functions allowing to infer the underlying physics process from the information measured in the detector, or to perform classification and regression tasks. For the latter, it was often necessary to force the data into an unnatural representation such as grid, in order to be able to used standard deep learning architectures such as CNN. Yet, many problems in physics involve data in the form of unordered sets with rich relations and interactions, which can be naturally represented as graphs.



Part of the Large Hadron Collider detectors.

One important application in high-energy physics is the reconstruction and classification of *particle jets* – sprays of stable particles arising from multiple successive interaction and decays of particles originating from a single initial event. In the Large Hardon Collider, the largest and best-known particle accelerator built at CERN, such jet are the result of collisions of protons at nearly the speed of light. These collisions produce massive particles, such as the long though-for Higgs boson or the top quark. The identification and classification of collision events is of crucial importance, as it might provide experimental evidence to the existence of new particles.

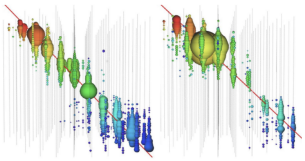


Example of a particle jet.

Multiple Geometric Deep Learning approaches have recently been proposed for particle jet classification task, e.g. by Komiske et al. (2019) and Qu and Gouskos (2019), based on DeepSet and Dynamic Graph CNN architectures, respectively. More recently, there has also been interest in developing specialsed architectures derived from physics consideration and incorporating inductive biases consistent with Hamiltonian or Lagrangian mechanics (see e.g. Sanchez-Gonzalez et al. (2019); Cranmer et al. (2020)), equivariant to the Lorentz group (a fundamental symmetry of space and time in physics) (Bogatskiy et al., 2020), or even incorporating symbolic reasoning (Cranmer et al., 2019) and capable of learning physical laws from data. Such approaches are more interpretable (and thus considered more 'trustworthy' by domain experts) and also offer better generalisation.

Besides particle accelerators, particle detectors are now being used by astrophysicist for *multi-messenger astronomy* – a new way of coordinated observation of disparate signals, such as electromagnetic radiation, gravitational waves, and neutrinos, coming from the same source. Neutrino astronomy is

of particular interest, since neutrinos interact only very rarely with matter, and thus travel enormous distances practically unaffected. Detecting neutrinos allows to observe objects inaccessible to optical telescopes, but requires enormously-sized detectors – the IceCube neutrino observatory uses a cubic kilometer of Antarctic ice shelf on the South Pole as its detector. Detecting high-energy neutrinos can possibly shed lights on some of the most mysterious objects in the Universe, such as blazars and black holes. Choma et al. (2018) used a Geometric neural network to model the irregular geometry of the IceCube neutrino detector, showing significantly better performance in detecting neutrinos coming from astrophysical sources and separating them from background events.



The characteristic pattern of light deposition in IceCube detector from background events (muon bundles, left) and astrophysical neutrinos (high-energy single muon, right). Choma et al. (2018)

While neutrino astronomy offers a big promise in the study of the Cosmos, traditional optical and radio telescopes are still the 'battle horses' of astronomers. With these traditional instruments, Geometric Deep Learning can still offer new methodologies for data analysis. For example, Scaife and Porter (2021) used rotationally-equivariant CNNs for the classification of radio galaxies, and McEwen et al. (2021) used spherical CNNs for the analysis of cosmic microwave background radiation, a relic from the Big Bang that might shed light on the formation of the primordial Universe. As we already mentioned, such signals are naturally represented on the sphere and equivariant neural networks are an appropriate tool to study them.

Virtual and Augmented Reality Another field of applications which served as the motivation for the development of a large class of Geometric Deep Learning methods is computer vision and graphics, in particular, dealing with 3D body models for virtual and augmented reality. Motion capture technology used to produce special effects in movies like Avatar often operates in two stages: first, the input from a 3D scanner capturing the motions of the body or the face of the actor is put into correspondence with some canonical shape, typically modelled as a discrete manifold or a mesh (this problem is often called 'analysis'). Second, a new shape is generated to repeat the motion of the input ('synthesis'). Initial works on Geometric Deep Learning in computer graphics and vision (Masci et al., 2015; Boscaini et al., 2016a; Monti et al., 2017) developed mesh convolutional neural networks to address the analysis problem, or more specifically, deformable shape correspondence.

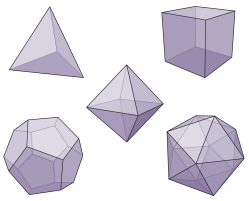
First geometric autoencoder architectures for 3D shape synthesis were proposed independently by Litany et al. (2018) and Ranjan et al. (2018). In

these architectures, a canonical mesh (of the body, face, or hand) was assumed to be known and the synthesis task consisted of regressing the 3D coordinates of the nodes (the embedding of the surface, using the jargon of differential geometry). Kulon et al. (2020) showed a hybrid pipeline for 3D hand pose estimation with an image CNN-based encoder and a geometric decoder. A demo of this system, developed in collaboration with a British startup company Ariel AI and presented at CVPR 2020, allowed to create realistic body avatars with fully articulated hands from video input on a mobile phone faster than real-time. Ariel AI was acquired by Snap in 2020, and at the time of writing its technology is used in Snap's augmented reality products.

Examples of complex 3D hand poses reconstructed from 2D images in the wild (Kulon et al., 2020).

7 Historic Perspective

"Symmetry, as wide or as narrow as you may define its meaning, is one idea by which man through the ages has tried to comprehend and create order, beauty, and perfection." This somewhat poetic definition of symmetry is given in the eponymous book of the great mathematician Hermann Weyl (2015), his *Schwanengesang* on the eve of retirement from the Institute for Advanced Study in Princeton. Weyl traces the special place symmetry has occupied in science and art to the ancient times, from Sumerian symmetric designs to the Pythagoreans who believed the circle to be perfect due to its rotational symmetry. Plato considered the five regular polyhedra bearing his name today so fundamental that they must be the basic building blocks shaping the material world. Yet, though Plato is credited with coining the term συμμετρία, which literally translates as 'same measure', he used it only vaguely to convey the beauty of proportion in art and harmony in music. It was the astronomer and mathematician Johannes Kepler to attempt the first rigorous analysis of the symmetric shape of water crystals. In his treatise ('On the Six-Cornered Snowflake'), he attributed the six-fold dihedral structure of snowflakes to hexagonal packing of particles – an idea that though preceded the clear understanding of how matter is formed, still holds today as the basis of crystallography (Ball, 2011).



The tetrahedron, cube, octahedron, dodecahedron, and icosahedron are called *Platonic solids*.

Fully titled *Strena, Seu De Nive Sexangula* ('New Year's gift, or on the Six-Cornered Snowflake') was, as suggested by the title, a small booklet sent by Kepler in 1611 as a Christmas gift to his patron and friend Johannes Matthäus Wackher von Wackenfels.

Symmetry in Mathematics and Physics In modern mathematics, symmetry is almost univocally expressed in the language of group theory. The origins of this theory are usually attributed to Évariste Galois, who coined

the term and used it to study solvability of polynomial equations in the 1830s. Two other names associated with group theory are those of Sophus Lie and Felix Klein, who met and worked fruitfully together for a period of time (Tobies, 2019). The former would develop the theory of continuous symmetries that today bears his name; the latter proclaimed group theory to be the organising principle of geometry in his Erlangen Program, which we mentioned in the beginning of this text. Riemannian geometry was explicitly excluded from Klein's unified geometric picture, and it took another fifty years before it was integrated, largely thanks to the work of Élie Cartan in the 1920s.

Emmy Noether, Klein's colleague in Göttingen, proved that every differentiable symmetry of the action of a physical system has a corresponding conservation law (Noether, 1918). In physics, it was a stunning result: beforehand, meticulous experimental observation was required to discover fundamental laws such as the conservation of energy, and even then, it was an empirical result not coming from anywhere. Noether's Theorem — "a guiding star to 20th and 21st century physics", in the words of the Nobel laureate Frank Wilczek — showed that the conservation of energy emerges from the translational symmetry of time, a rather intuitive idea that the results of an experiment should not depend on whether it is conducted today or tomorrow.

The symmetry associated with charge conservation is the global *gauge invariance* of the electromagnetic field, first appearing in Maxwell's formulation of electrodynamics (Maxwell, 1865); however, its importance initially remained unnoticed. The same Hermann Weyl who wrote so dithyrambically about symmetry is the one who first introduced the concept of gauge invariance in physics in the early 20th century, emphasizing its role as a principle from which electromagnetism can be *derived*. It took several decades until this fundamental principle — in its generalised form developed by Yang and Mills (1954) — proved successful in providing a unified framework to describe the quantum-mechanical behavior of electromagnetism and the weak and strong forces, finally culminating in the Standard Model that captures all the fundamental forces of nature but gravity. We can thus join another Nobel-winning physicist, Philip Anderson (1972), in concluding that "it is only slightly overstating the case to say that physics is the study of symmetry."

Weyl first conjectured (incorrectly) in 1919 that invariance under the change of scale or "gauge" was a local symmetry of electromagnetism. The term *gauge*, or *Eich* in German, was chosen by analogy to the various track gauges of railroads. After the development of quantum mechanics, Weyl (1929) modified the gauge choice by replacing the scale factor with a change of wave phase. See Straumann (1996).

Early Use of Symmetry in Machine Learning In machine learning and its applications to pattern recognition and computer vision, the importance of symmetry has long been recognised. Early work on designing equivariant feature detectors for pattern recognition was done by Amari (1978), Kanatani (2012), and Lenz (1990). In the neural networks literature, the famous Group Invariance Theorem for Perceptrons by Minsky and Papert (2017) puts fundamental limitations on the capabilities of (single-layer) perceptrons to learn invariants. This was one of the primary motivations for studying multi-layer architectures (Sejnowski et al., 1986; Shawe-Taylor, 1989, 1993), which ultimately led to deep learning.

Shun'ichi Amari is credited as the creator of the field of *information geometry* that applies Riemannian geometry models to probability. The main object studied by information geometry is a *statistical manifold*, where each point corresponds to a probability distribution.

In the neural network community, *Neocognitron* (Fukushima and Miyake, 1982) is credited as the first implementation of shift invariance in a neural network for "pattern recognition unaffected by shift in position". His solution came in the form of hierarchical neural network with local connectivity, drawing inspiration from the receptive fields discovered in the visual cortex by the neuroscientists David Hubel and Torsten Wiesel two decades earlier (Hubel and Wiesel, 1959). These ideas culminated in Convolutional Neural Networks in the seminal work of Yann LeCun and co-authors (LeCun et al., 1998). The first work to take a representation-theoretical view on invariant and equivariant neural networks was performed by Wood and Shawe-Taylor (1996), unfortunately rarely cited. More recent incarnations of these ideas include the works of Makadia et al. (2007); Esteves et al. (2020) and one of the authors of this text (Cohen and Welling, 2016).

This classical work was recognised by the Nobel Prize in Medicine in 1981, which Hubel and Wiesel shared with Roger Sperry.

Graph Neural Networks It is difficult to pinpoint exactly when the concept of Graph Neural Networks began to emerge—partly due to the fact that most of the early work did not place graphs as a first-class citizen, partly since GNNs became practical only in the late 2010s, and partly because this field emerged from the confluence of several research areas. That being said, early forms of graph neural networks can be traced back at least to the 1990s, with examples including Alessandro Sperduti's Labeling RAAM (Sperduti, 1994), the "backpropagation through structure" of Goller and Kuchler (1996), and adaptive processing of data structures (Sperduti and Starita, 1997; Frasconi et al., 1998). While these works were primarily concerned with operating over "structures" (often trees or directed acyclic graphs), many of the invariances preserved in their architectures are reminiscent of the GNNs more commonly in use today.

The first proper treatment of the processing of generic graph structures (and the coining of the term *"graph neural network"*) happened after the turn of the 21st century. Within the Artificial Intelligence lab at the Università degli Studi di Siena (Italy), papers led by Marco Gori and Franco Scarselli have proposed the first "GNN" (Gori et al., 2005; Scarselli et al., 2008). They relied on recurrent mechanisms, required the neural network parameters to specify *contraction mappings*, and thus computing node representations by searching for a fixed point—this in itself necessitated a special form of backpropagation (Almeida, 1990; Pineda, 1988) and did not depend on node features at all. All of the above issues were rectified by the Gated GNN (GGNN) model of Li et al. (2015). GGNNs brought many benefits of modern RNNs, such as gating mechanisms (Cho et al., 2014) and backpropagation through time, to the GNN model, and remain popular today.

Concurrently, Alessio Micheli had proposed the *neural network for graphs* (NN4G) model, which focused on a *feedforward* rather than recurrent paradigm (Micheli, 2009).

Computational chemistry It is also very important to note an independent and concurrent line of development for GNNs: one that was entirely driven by the needs of computational chemistry, where molecules are most naturally expressed as graphs of atoms (nodes) connected by chemical bonds (edges). This invited computational techniques for molecular property prediction that operate directly over such a graph structure, which had become present in machine learning in the 1990s: this includes the ChemNet model of Kireev (1995) and the work of Baskin et al. (1997). Strikingly, the "molecular graph networks" of Merkwirth and Lengauer (2005) explicitly proposed many of the elements commonly found in contemporary GNNs—such as edge type-conditioned weights or global pooling—as early as 2005. The chemical motivation continued to drive GNN development into the 2010s, with two significant GNN advancements centered around improving molecular fingerprinting (Duvenaud et al., 2015) and predicting quantum-chemical properties (Gilmer et al., 2017) from small molecules. At the time of writing this text, molecular property prediction is one of the most successful applications of GNNs, with impactful results in virtual screening of new antibiotic drugs (Stokes et al., 2020).

Node embeddings Some of the earliest success stories of deep learning on graphs involve learning representations of nodes in an unsupervised fashion, based on the graph structure. Given their structural inspiration, this direction also provides one of the most direct links between graph representation learning and network science communities. The key early

approaches in this space relied on *random walk*-based embeddings: learning node representations in a way that brings them closer together if the nodes co-occur in a short random walk. Representative methods in this space include DeepWalk (Perozzi et al., 2014), node2vec (Grover and Leskovec, 2016) and LINE (Tang et al., 2015), which are all purely self-supervised. Planetoid (Yang et al., 2016) was the first in this space to incorporate supervision label information, when it is available.

Recently, a theoretical framework was developed by Srinivasan and Ribeiro (2019) in which the equivalence of structural and positional representations was demonstrated. Additionally, Qiu et al. (2018) have demonstrated that all random-walk based embedding techniques are equivalent to an appropriately-posed matrix factorisation task.

Unifying random walk objectives with GNN encoders was attempted on several occasions, with representative approaches including Variational Graph Autoencoder (VGAE, Kipf and Welling (2016b)), embedding propagation (García-Durán and Niepert, 2017), and unsupervised variants of GraphSAGE (Hamilton et al., 2017). However, this was met with mixed results, and it was shortly discovered that pushing neighbouring node representations together is already a key part of GNNs' inductive bias. Indeed, it was shown that an *untrained* GNN was already showing performance that is competitive with DeepWalk, in settings where node features are available (Veličković et al., 2019; Wu et al., 2019). This launched a direction that moves away from combining random walk objectives with GNNs and shifting towards *contrastive* approaches inspired by mutual information maximisation and aligning to successful methods in the image domain. Prominent examples of this direction include Deep Graph Informax (DGI, Veličković et al. (2019)), GRACE (Zhu et al., 2020), BERT-like objectives (Hu et al., 2020) and BGRL (Thakoor et al., 2021).

Probabilistic graphical models Graph neural networks have also, concurrently, resurged through embedding the computations of *probabilistic graphical models* (PGMs, Wainwright and Jordan (2008)). PGMs are a powerful tool for processing graphical data, and their utility arises from their probabilistic perspective on the graph's edges: namely, the nodes are treated as *random variables*, while the graph structure encodes *conditional independence* assumptions, allowing for significantly simplifying the calculation and sampling from the joint distribution. Indeed, many algorithms for (exactly or approximately) supporting learning and inference on PGMs rely on forms of passing messages over their edges (Pearl, 2014), with examples including variational mean-field inference and loopy belief propagation (Yedidia et al., 2001; Murphy et al., 2013).

This connection between PGMs and message passing was subsequently

developed into GNN architectures, with early theoretical links established by the authors of structure2vec (Dai et al., 2016). Namely, by posing a graph representation learning setting as a Markov random field (of nodes corresponding to input features and latent representations), the authors directly align the computation of both mean-field inference and loopy belief propagation to a model not unlike the GNNs commonly in use today.

The key "trick" which allowed for relating the latent representations of a GNN to probability distributions maintained by a PGM was the usage of *Hilbert-space embeddings* of distributions (Smola et al., 2007). Given ϕ, an appropriately chosen embedding function for features $\mathbf{x}$, it is possible to embed their probability distribution $p(\mathbf{x})$ as the *expected* embedding $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\phi(\mathbf{x})$. Such a correspondence allows us to perform GNN-like computations, knowing that the representations computed by the GNN will always correspond to an embedding of *some* probability distribution over the node features.

The structure2vec model itself is, ultimately, a GNN architecture which easily sits within our framework, but its setup has inspired a series of GNN architectures which more directly incorporate computations found in PGMs. Emerging examples have successfully combined GNNs with conditional random fields (Gao et al., 2019; Spalević et al., 2020), relational Markov networks (Qu et al., 2019) and Markov logic networks (Zhang et al., 2020).

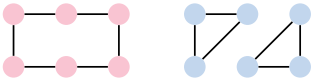
The Weisfeiler-Lehman formalism The resurgence of graph neural networks was followed closely by a drive to understand their fundamental limitations, especially in terms of expressive power. While it was becoming evident that GNNs are a strong modelling tool of graph-structured data, it was also clear that they wouldn't be able to solve *any* task specified on a graph perfectly. A canonical illustrative example of this is deciding *graph isomorphism*: is our GNN able to attach different representations to two given non-isomorphic graphs? This is a useful framework for two reasons. If the GNN is unable to do this, then it will be hopeless on any task requiring the discrimination of these two graphs. Further, it is currently not known if deciding graph isomorphism is in P, the complexity class in which all GNN computations typically reside.

Due to their permutation invariance, GNNs will attach identical representations to two isomorphic graphs, so this case is trivially solved.

The best currently known algorithm for deciding graph isomorphism is due to Babai and Luks (1983), though a recent (not fully reviewed) proposal by Babai (2016) implies a quasi-polynomial time solution.

The key framework which binds GNNs to graph isomorphism is the *Weisfeiler-Lehman* (WL) graph isomorphism test (Weisfeiler and Leman, 1968). This test generates a graph representation by iteratively passing node features

along the edges of the graph, then *randomly hashing* their sums across neighbourhoods. Connections to *randomly-initialised* convolutional GNNs are apparent, and have been observed early on: for example, within the GCN model of Kipf and Welling (2016a). Aside from this connection, the WL iteration was previously introduced in the domain of *graph kernels* by Shervashidze et al. (2011), and it still presents a strong baseline for unsupervised learning of whole-graph representations.



One simple example: the WL test cannot distinguish a *6-cycle* from *two triangles*.

While the WL test is conceptually simple, and there are many simple examples of non-isomorphic graphs it cannot distinguish, its expressive power is ultimately strongly tied to GNNs. Analyses by Morris et al. (2019) and Xu et al. (2018) have both reached a striking conclusion: *any* GNN conforming to one of the three flavours we outlined in Section 5.3 cannot be more powerful than the WL test!

In order to exactly reach this level of representational power, certain constraints must exist on the GNN update rule. Xu et al. (2018) have shown that, in the discrete-feature domain, the aggregation function the GNN uses must be *injective*, with *summation* being a key representative. Based on the outcome of their analysis, Xu et al. (2018) propose the Graph Isomorphism Network (GIN), which is a simple but powerful example of a maximally-expressive GNN under this framework. It is also expressible under the convolutional GNN flavour we propose.

Popular aggregators such as maximisation and averaging fall short in this regard, because they would not be able to distinguish e.g. the neighbour multisets $\{\!\{\mathbf{a}, \mathbf{b}\}\!\}$ and $\{\!\{\mathbf{a}, \mathbf{a}, \mathbf{b}, \mathbf{b}\}\!\}$.

Lastly, it is worth noting that these findings do not generalise to *continuous* node feature spaces. In fact, using the Borsuk-Ulam theorem (Borsuk, 1933), Corso et al. (2020) have demonstrated that, assuming real-valued node features, obtaining injective aggregation functions requires *multiple* aggregators (specifically, equal to the *degree* of the receiver node). Their findings have driven the Principal Neighbourhood Aggregation (PNA) architecture, which proposes a multiple-aggregator GNN that is empirically powerful and stable.

One example of such aggregators are the *moments* of the multiset of neighbours.

Higher-order methods The findings of the previous paragraphs do not contradict the practical utility of GNNs. Indeed, in many real-world applications the input features are sufficiently *rich* to support useful discriminative computations over the graph structure, despite of the above limitations.

Which, in contrast, almost always consider *featureless* or *categorically-featured* graphs.

However, one key corollary is that GNNs are relatively quite weak at detecting some rudimentary *structures* within a graph. Guided by the specific

limitations or failure cases of the WL test, several works have provided *stronger* variants of GNNs that are *provably* more powerful than the WL test, and hence likely to be useful on tasks that require such structural detection.

One prominent example is computational chemistry, wherein a molecule's chemical function can be strongly influenced by the presence of aromatic *rings* in its molecular graph.

Perhaps the most direct place to hunt for more expressive GNNs is the WL test itself. Indeed, the strength of the original WL test can be enhanced by considering a *hierarchy* of WL tests, such that k-WL tests attach representations to k-*tuples* of nodes (Morris et al., 2017). The k-WL test has been directly translated into a *higher-order* k-GNN architecture by Morris et al. (2019), which is provably more powerful than the GNN flavours we considered before. However, its requirement to maintain tuple representations implies that, in practice, it is hard to scale beyond $k = 3$.

There have been efforts, such as the δ-k-LGNN (Morris et al., 2020), to sparsify the computation of the k-GNN.

Concurrently, Maron et al. (2018, 2019) have studied the characterisation of invariant and equivariant graph networks over k-tuples of nodes. Besides demonstrating the surprising result of *any* invariant or equivariant graph network being expressible as a linear combination of a finite number of generators—the amount of which only depends on k—the authors showed that the expressive power of such layers is equivalent to the k-WL test, and proposed an empirically scalable variant which is provably 3-WL powerful.

Besides generalising the domain over which representations are computed, significant effort had also went into analysing specific failure cases of 1-WL and augmenting GNN *inputs* to help them distinguish such cases. One common example is attaching *identifying features* to the nodes, which can help detecting structure. Proposals to do this include *one-hot* representations (Murphy et al., 2019), as well as purely *random* features (Sato et al., 2020).

For example, if a node sees its own identifier k hops away, it is a direct indicator that it is within a k-cycle.

More broadly, there have been many efforts to incorporate *structural* information within the message passing process, either by modulating the message function or the graph that the computations are carried over. Several interesting lines of work here involve sampling *anchor node sets* (You et al., 2019), aggregating based on *Laplacian eigenvectors* (Stachenfeld et al., 2020; Beaini et al., 2020; Dwivedi and Bresson, 2020), or performing *topological data analysis*, either for positional embeddings (Bouritsas et al., 2020) or driving message passing (Bodnar et al., 2021).

In the computational chemistry domain, it is often assumed that molecular function is driven by substructures (the *functional groups*), which have directly inspired the modelling of molecules at a *motif* level. For references, consider Jin et al. (2018, 2020); Fey et al. (2020).

Signal processing and Harmonic analysis Since the early successes of Convolutional Neural Networks, researchers have resorted to tools from harmonic analysis, image processing, and computational neuroscience trying to

provide a theoretical framework that explains their efficiency. M-theory is a framework inspired by the visual cortex, pioneered by Tomaso Poggio and collaborators (Riesenhuber and Poggio, 1999; Serre et al., 2007), based on the notion of templates that can be manipulated under certain symmetry groups. Another notable model arising from computational neuroscience were *steerable pyramids*, a form of multiscale wavelet decompositions with favorable properties against certain input transformations, developed by Simoncelli and Freeman (1995). They were a central element in early generative models for textures (Portilla and Simoncelli, 2000), which were subsequently improved by replacing steerable wavelet features with deep CNN features Gatys et al. (2015). Finally, Scattering transforms, introduced by Stéphane Mallat (2012) and developed by Bruna and Mallat (2013), provided a framework to understand CNNs by replacing trainable filters with multiscale wavelet decompositions, also showcasing the deformation stability and the role of depth in the architecture.

Signal Processing on Graph and Meshes Another important class of graph neural networks, often referred to as *spectral*, has emerged from the work of one of the authors of this text (Bruna et al., 2013), using the notion of the *Graph Fourier transform*. The roots of this construction are in the signal processing and computational harmonic analysis communities, where dealing with non-Euclidean signals has become prominent in the late 2000s and early 2010s. Influential papers from the groups of Pierre Vandergheynst (Shuman et al., 2013) and José Moura (Sandryhaila and Moura, 2013) popularised the notion of "Graph Signal Processing" (GSP) and the generalisation of Fourier transforms based on the eigenvectors of graph adjacency and Laplacian matrices. The graph convolutional neural networks relying on spectral filters by Defferrard et al. (2016) and Kipf and Welling (2016a) are among the most cited in the field and can likely be credited) as ones reigniting the interest in machine learning on graphs in recent years.

It is worth noting that, in the field of computer graphics and geometry processing, non-Euclidean harmonic analysis predates Graph Signal Processing by at least a decade. We can trace spectral filters on manifolds and meshes to the works of Taubin et al. (1996). These methods became mainstream in the 2000s following the influential papers of Karni and Gotsman (2000) on spectral geometry compression and of Lévy (2006) on using the Laplacian eigenvectors as a non-Euclidean Fourier basis. Spectral methods have been used for a range of applications, most prominent of which is the construction

Learnable shape descriptors similar to spectral graph CNNs were proposed by Roee Litman and Alex Bronstein (2013), the latter being a twin brother of the author of this text.

of shape descriptors (Sun et al., 2009) and functional maps (Ovsjanikov et al., 2012); these methods are still broadly used in computer graphics at the time of writing.

Computer Graphics and Geometry Processing Models for shape analysis based on intrinsic metric invariants were introduced by various authors in the field of computer graphics and geometry processing (Elad and Kimmel, 2003; Mémoli and Sapiro, 2005; Bronstein et al., 2006), and are discussed in depth by one of the authors in his earlier book (Bronstein et al., 2008). The notions of intrinsic symmetries were also explored in the same field Raviv et al. (2007); Ovsjanikov et al. (2008). The first architecture for deep learning on meshes, Geodesic CNNs, was developed in the team of one of the authors of the text (Masci et al., 2015). This model used local filters with shared weights, applied to geodesic radial patches. It was a particular setting of gauge-equivariant CNNs developed later by another author of the text (Cohen et al., 2019). A generalisation of Geodesic CNNs with learnable aggregation operations, MoNet, proposed by Federico Monti et al. (2017) from the same team, used an attention-like mechanism over the local structural features of the mesh, that was demonstrated to work on general graphs as well. The graph attention network (GAT), which technically speaking can be considered a particular instance of MoNet, was introduced by another author of this text (Veličković et al., 2018). GATs generalise MoNet's attention mechanism to also incorporate node feature information, breaking away from the purely structure-derived relevance of prior work. It is one of the most popular GNN architectures currently in use.

In the context of computer graphics, it is also worthwhile to mention that the idea of learning on sets (Zaheer et al., 2017) was concurrently developed in the group of Leo Guibas at Stanford under the name PointNet (Qi et al., 2017) for the analysis of 3D point clouds. This architecture has lead to multiple follow-up works, including one by an author of this text called Dynamic Graph CNN (DGCNN, Wang et al. (2019b)). DGCNN used a nearest-neighbour graph to capture the local structure of the point cloud to allow exchange of information across the nodes; the key characteristic of this architecture was that the graph was constructed on-the-fly and updated between the layers of the neural network in relation to the downstream task. This latter property made DGCNN one of the first incarnations of 'latent graph learning', which in its turn has had significant follow up. Extensions to DGCNN's k-nearest neighbour graph proposal include more explicit control

over these graphs' edges, either through bilevel optimisation (Franceschi et al., 2019), reinforcement learning (Kazi et al., 2020) or direct supervision (Veličković et al., 2020). Independently, a variational direction (which probabilistically samples edges from a computed *posterior* distribution) has emerged through the NRI model (Kipf et al., 2018). While it still relies on quadratic computation in the number of nodes, it allows for explicitly encoding uncertainty about the chosen edges.

Another very popular direction in learning on graphs without a provided graph relies on performing GNN-style computations over a *complete* graph, letting the network infer its own way to exploit the connectivity. The need for this arisen particularly in natural language processing, where various words in a sentence interact in highly nontrivial and non-sequential ways. Operating over a complete graph of words brought about the first incarnation of the Transformer model (Vaswani et al., 2017), which de-throned both recurrent and convolutional models as state-of-the-art in neural machine translation, and kicked off an avalanche of related work, transcending the boundaries between NLP and other fields. Fully-connected GNN computation has also concurrently emerged on simulation (Battaglia et al., 2016), reasoning (Santoro et al., 2017), and multi-agent (Hoshen, 2017) applications, and still represents a popular choice when the number of nodes is reasonably small.

Algorithmic reasoning For most of the discussion we posed in this section, we have given examples of *spatially* induced geometries, which in turn shape the underlying domain, and its invariances and symmetries. However, plentiful examples of invariances and symmetries also arise in a *computational* setting. One critical difference to many common settings of Geometric Deep Learning is that links no longer need to encode for any kind of similarity, proximity, or types of relations—they merely specify the "recipe" for the dataflow between data points they connect.

Instead, the computations of the neural network mimic the reasoning process of an *algorithm* (Cormen et al., 2009), with additional invariances induced by the algorithm's control flow and intermediate results. In the space of algorithms the assumed input invariants are often referred to as *preconditions*, while the invariants preserved by the algorithm are known as *postconditions*.

For example, one invariant of the Bellman-Ford pathfinding algorithm (Bellman, 1958) is that, after k steps, it will always compute the shortest paths to the source node that use no more than k edges.

Eponymously, the research direction of *algorithmic reasoning* (Cappart et al., 2021, Section 3.3.) seeks to produce neural network architectures that ap-

propriately preserve algorithmic invariants. The area has investigated the construction of general-purpose neural computers, e.g., the *neural Turing machine* (Graves et al., 2014) and the *differentiable neural computer* (Graves et al., 2016). While such architectures have all the hallmarks of general computation, they introduced several components at once, making them often challenging to optimise, and in practice, they are almost always outperformed by simple relational reasoners, such as the ones proposed by Santoro et al. (2017, 2018).

As modelling complex postconditions is challenging, plentiful work on inductive biases for learning to execute (Zaremba and Sutskever, 2014) has focused on primitive algorithms (e.g. simple arithmetic). Prominent examples in this space include the *neural GPU* (Kaiser and Sutskever, 2015), *neural RAM* (Kurach et al., 2015), *neural programmer-interpreters* (Reed and De Freitas, 2015), *neural arithmetic-logic units* (Trask et al., 2018; Madsen and Johansen, 2020) and *neural execution engines* (Yan et al., 2020).

Emulating combinatorial algorithms of *superlinear* complexity was made possible with the rapid development of GNN architectures. The *algorithmic alignment* framework pioneered by Xu et al. (2019) demonstrated, theoretically, that GNNs *align* with dynamic programming (Bellman, 1966), which is a language in which most algorithms can be expressed. It was concurrently empirically shown, by one of the authors of this text, that it is possible to design and train GNNs that align with algorithmic invariants in practice (Veličković et al., 2019). Onwards, alignment was achieved with *iterative algorithms* (Tang et al., 2020), *linearithmic algorithms* (Freivalds et al., 2019), *data structures* (Veličković et al., 2020) and *persistent memory* (Strathmann et al., 2021). Such models have also seen practical use in *implicit planners* (Deac et al., 2020), breaking into the space of *reinforcement learning* algorithms.

Concurrently, significant progress has been made on using GNNs for *physics simulations* (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020). This direction yielded much of the same recommendations for the design of generalising GNNs. Such a correspondence is to be expected: given that algorithms can be phrased as discrete-time simulations, and simulations are typically implemented as step-wise algorithms, both directions will need to preserve similar kinds of invariants.

Tightly bound with the study of algorithmic reasoning are measures of *extrapolation*. This is a notorious pain-point for neural networks, given that most of their success stories are obtained when generalising *in-distribution*;

i.e. when the patterns found in the training data properly anticipate the ones found in the test data. However, algorithmic invariants must be preserved irrespective of, e.g., the size or generative distribution of the input, meaning that the training set will likely not cover any possible scenario encountered in practice. Xu et al. (2020b) have proposed a geometric argument for what is required of an extrapolating GNN backed by rectifier activations: its components and featurisation would need to be designed so as to make its constituent modules (e.g. message function) learn only *linear* target functions. Bevilacqua et al. (2021) propose observing extrapolation under the lens of *causal reasoning*, yielding *environment-invariant* representations of graphs.

Geometric Deep Learning Our final historical remarks regard the very name of this text. The term 'Geometric Deep Learning' was first introduced by one of the authors of this text in his ERC grant in 2015 and popularised in the eponymous IEEE Signal Processing Magazine paper (Bronstein et al., 2017). This paper proclaimed, albeit "with some caution", the signs of "a new field being born." Given the recent popularity of graph neural networks, the increasing use of ideas of invariance and equivariance in a broad range of machine learning applications, and the very fact of us writing this text, it is probably right to consider this prophecy at least partially fulfilled. The name "4G: Grids, Graphs, Groups, and Gauges" was coined by Max Welling for the ELLIS Program on Geometric Deep Learning, co-directed by two authors of the text. Admittedly, the last 'G' is somewhat of a stretch, since the underlying structures are manifolds and bundles rather than gauges. For this text, we added another 'G', Geodesics, in reference to metric invariants and intrinsic symmetries of manifolds.

Acknowledgements

This text represents a humble attempt to summarise and synthesise decades of existing knowledge in deep learning architectures, through the geometric lens of invariance and symmetry. We hope that our perspective will make it easier both for newcomers and practitioners to navigate the field, and for researchers to synthesise novel architectures, as instances of our blueprint. In a way, we hope to have presented *"all you need to build the architectures that are all you need"*—a play on words inspired by Vaswani et al. (2017).

The bulk of the text was written during late 2020 and early 2021. As it often happens, we had thousands of doubts whether the whole picture makes sense, and used opportunities provided by our colleagues to help us break our "stage fright" and present early versions of our work, which saw the light of day in Petar's talk at Cambridge (courtesy of Pietro Liò) and Michael's talks at Oxford (courtesy of Xiaowen Dong) and Imperial College (hosted by Michael Huth and Daniel Rueckert). Petar was also able to present our work at Friedrich-Alexander-Universität Erlangen-Nürnberg—the birthplace of the Erlangen Program!—owing to a kind invitation from Andreas Maier. The feedback we received for these talks was enormously invaluable to keeping our spirits high, as well as polishing the work further. Last, but certainly not least, we thank the organising committee of ICLR 2021, where our work will be featured in a keynote talk, delivered by Michael.

We should note that reconciling such a vast quantity of research is seldom enabled by the expertise of only four people. Accordingly, we would like to give due credit to all of the researchers who have carefully studied aspects of our text as it evolved, and provided us with careful comments and references: Yoshua Bengio, Charles Blundell, Andreea Deac, Fabian Fuchs, Francesco di Giovanni, Marco Gori, Raia Hadsell, Will Hamilton, Maksym Korablyov, Christian Merkwirth, Razvan Pascanu, Bruno Ribeiro, Anna Scaife, Jürgen Schmidhuber, Marwin Segler, Corentin Tallec, Ngân Vũ, Peter Wirnsberger and David Wong. Their expert feedback was invaluable to solidifying our unification efforts and making them more useful to various niches. Though, of course, any irregularities within this text are our responsibility alone. It is currently very much a work-in-progress, and we are very happy to receive comments at any stage. Please contact us if you spot any errors or omissions.

Bibliography

Yonathan Aflalo and Ron Kimmel. Spectral multidimensional scaling. *PNAS*, 110(45):18052–18057, 2013.

Yonathan Aflalo, Haim Brezis, and Ron Kimmel. On the optimality of shape and data representation in the spectral domain. *SIAM J. Imaging Sciences*, 8(2):1141–1160, 2015.

Luis B Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks: concept learning*, pages 102–111. 1990.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv:2006.05205*, 2020.

Sl Amari. Feature spaces which admit and detect invariant signal transformations. In *Joint Conference on Pattern Recognition*, 1978.

Brandon Anderson, Truong-Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. *arXiv:1906.04015*, 2019.

Philip W Anderson. More is different. *Science*, 177(4047):393–396, 1972.

Mathieu Andreux, Emanuele Rodola, Mathieu Aubry, and Daniel Cremers. Anisotropic Laplace-Beltrami operators for shape analysis. In *ECCV*, 2014.

Salim Arslan, Sofia Ira Ktena, Ben Glocker, and Daniel 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*, pages 3–13. 2018.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

László Babai. Graph isomorphism in quasipolynomial time. In *ACM Symposium on Theory of Computing*, 2016.

László Babai and Eugene M Luks. Canonical labeling of graphs. In *ACM Symposium on Theory of computing*, 1983.

Francis Bach. Breaking the curse of dimensionality with convex neural networks. *JMLR*, 18(1):629–681, 2017.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *ICML*, 2020.

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *Trans. PAMI*, 39(12):2481–2495, 2017.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

Philip Ball. In retrospect: On the six-cornered snowflake. *Nature*, 480(7378): 455–455, 2011.

Bassam Bamieh. Discovering transforms: A tutorial on circulant matrices, circular convolution, and the discrete fourier transform. *arXiv:1805.05533*, 2018.

Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3(1):133–181, 1922.

Victor Bapst, Thomas Keck, A Grabska-Barwińska, Craig Donner, Ekin Dogus Cubuk, Samuel S Schoenholz, Annette Obika, Alexander WR Nelson, Trevor Back, Demis Hassabis, et al. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16(4):448–454, 2020.

Albert-László Barabási, Natali Gulbahce, and Joseph Loscalzo. Network medicine: a network-based approach to human disease. *Nature Reviews Genetics*, 12(1):56–68, 2011.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Information Theory*, 39(3):930–945, 1993.

Igor I Baskin, Vladimir A Palyulin, and Nikolai S Zefirov. A neural device for searching direct correlations between structures and properties of chemical compounds. *J. Chemical Information and Computer Sciences*, 37(4): 715–721, 1997.

Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv:1612.00222*, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

Dominique Beaini, Saro Passaro, Vincent Létourneau, William L Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. *arXiv:2010.02863*, 2020.

Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16 (1):87–90, 1958.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166, 1994.

Marcel Berger. *A panoramic view of Riemannian geometry*. Springer, 2012.

Pierre Besson, Todd Parrish, Aggelos K Katsaggelos, and S Kathleen Bandt. Geometric deep learning on brain shape predicts sex and age. *BioRxiv:177543*, 2020.

Beatrice Bevilacqua, Yangze Zhou, and Bruno Ribeiro. Size-invariant graph representations for graph classification extrapolations. *arXiv:2103.05045*, 2021.

Guy Blanc, Neha Gupta, Gregory Valiant, and Paul Valiant. Implicit regularization for deep neural networks driven by an ornstein-uhlenbeck like process. In *COLT*, 2020.

Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. *arXiv:2103.03212*, 2021.

Alexander Bogatskiy, Brandon Anderson, Jan Offermann, Marwah Roussi, David Miller, and Risi Kondor. Lorentz group equivariant neural network for particle physics. In *ICML*, 2020.

Karol Borsuk. Drei sätze über die n-dimensionale euklidische sphäre. *Fundamenta Mathematicae*, 20(1):177–190, 1933.

Davide Boscaini, Davide Eynard, Drosos Kourounis, and Michael M Bronstein. Shape-from-operator: Recovering shapes from intrinsic operators. *Computer Graphics Forum*, 34(2):265–274, 2015.

Davide Boscaini, Jonathan Masci, Emanuele Rodoià, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS*, 2016a.

Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Michael M Bronstein, and Daniel Cremers. Anisotropic diffusion descriptors. *Computer Graphics Forum*, 35(2):431–441, 2016b.

Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzere, and Mario Vento. A quadratic assignment formulation of the graph edit distance. *arXiv:1512.07494*, 2015.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv:2006.09252*, 2020.

Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching. *PNAS*, 103(5):1168–1172, 2006.

Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. *Numerical geometry of non-rigid shapes*. Springer, 2008.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv:2005.14165*, 2020.

Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2013.

Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv:2102.09544*, 2021.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv:1806.07366*, 2018.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

Albert Chern, Felix Knöppel, Ulrich Pinkall, and Peter Schröder. Shape from metric. *ACM Trans. Graphics*, 37(4):1–17, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.

Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael M Bronstein, Spencer R Klein, and Joan Bruna. Graph neural networks for icecube signal classification. In *ICMLA*, 2018.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *ICML*, 2016.

Taco Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral CNN. In *ICML*, 2019.

Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *arXiv:1801.10130*, 2018.

Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv:1603.09025*, 2016.

Etienne Corman, Justin Solomon, Mirela Ben-Chen, Leonidas Guibas, and Maks Ovsjanikov. Functional characterization of intrinsic and extrinsic geometry. *ACM Trans. Graphics*, 36(2):1–17, 2017.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv:2004.05718*, 2020.

Luca Cosmo, Anees Kazi, Seyed-Ahmad Ahmadi, Nassir Navab, and Michael Bronstein. Latent-graph learning for disease prediction. In *MICCAI*, 2020.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv:2003.04630*, 2020.

Miles D Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks. *arXiv:1909.05862*, 2019.

Guillem Cucurull, Konrad Wagstyl, Arantxa Casanova, Petar Veličković, Estrid Jakobsen, Michal Drozdzal, Adriana Romero, Alan Evans, and Yoshua Bengio. Convolutional neural networks for mesh-based parcellation of the cerebral cortex. 2018.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.

Jeffrey De Fauw, Joseph R Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, et al. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature Medicine*, 24 (9):1342–1350, 2018.

Pim de Haan, Maurice Weiler, Taco Cohen, and Max Welling. Gauge equivariant mesh CNNs: Anisotropic convolutions on geometric graphs. In *NeurIPS*, 2020.

Andreea Deac, Petar Veličković, and Pietro Sormanni. Attentive cross-modal paratope prediction. *Journal of Computational Biology*, 26(6):536–545, 2019.

Andreea Deac, Petar Veličković, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolić. Xlvin: executed latent value iteration nets. *arXiv:2010.13146*, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *NIPS*, 2016.

Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Peter W Battaglia, Vishal Gupta, Ang Li, Zhongwen Xu, Alvaro Sanchez-Gonzalez, Yujia Li, and Petar Veličković. Traffic Prediction with Graph Neural Networks in Google Maps. 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *NIPS*, 2015.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv:2012.09699*, 2020.

Asi Elad and Ron Kimmel. On bending invariant signatures for surfaces. *Trans. PAMI*, 25(10):1285–1295, 2003.

Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

Carlos Esteves, Ameesh Makadia, and Kostas Daniilidis. Spin-weighted spherical CNNs. *arXiv:2006.10731*, 2020.

Xiaomin Fang, Jizhou Huang, Fan Wang, Lingke Zeng, Haijin Liang, and Haifeng Wang. ConSTGAT: Contextual spatial-temporal graph attention network for travel time estimation at baidu maps. In *KDD*, 2020.

Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs. *arXiv:2006.12179*, 2020.

Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *ICML*, 2020.

Jon Folkman. Regular line-symmetric graphs. *Journal of Combinatorial Theory*, 3(3):215–232, 1967.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *ICML*, 2019.

Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE Trans. Neural Networks*, 9 (5):768–786, 1998.

Kārlis Freivalds, Emīls Ozoliņš, and Agris Šostaks. Neural shuffle-exchange networks–sequence processing in o (n log n) time. *arXiv:1907.07897*, 2019.

Fabian B Fuchs, Daniel E Worrall, Volker Fischer, and Max Welling. SE(3)-transformers: 3D roto-translation equivariant attention networks. *arXiv:2006.10503*, 2020.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*, pages 267–285. Springer, 1982.

Pablo Gainza, Freyr Sverrisson, Frederico Monti, Emanuele Rodola, D Boscaini, MM Bronstein, and BE Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192, 2020.

Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Diffusion scattering transforms on graphs. In *ICLR*, 2019.

Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Trans. Signal Processing*, 68:5680–5695, 2020.

Hongchang Gao, Jian Pei, and Heng Huang. Conditional random field enhanced graph convolutional neural networks. In *KDD*, 2019.

Alberto García-Durán and Mathias Niepert. Learning graph representations with embedding propagation. *arXiv:1710.03059*, 2017.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. *arXiv preprint arXiv:1505.07376*, 2015.

Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. Utilising graph machine learning within drug discovery and development. *arXiv:2012.05716*, 2020.

Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *IJCNN*, 2000.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv:1704.01212*, 2017.

Ross Girshick. Fast R-CNN. In *CVPR*, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

Vladimir Gligorijevic, P Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn C Taylor, Ian M Fisk, Hera Vlamakis, et al. Structure-based function prediction using graph convolutional networks. *bioRxiv:786236*, 2020.

Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *ICNN*, 1996.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv:1406.2661*, 2014.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IJCNN*, 2005.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv:2006.07733*, 2020.

Mikhael Gromov. *Structures métriques pour les variétés riemanniennes*. Cedic, 1981.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *NIPS*, 2017.

Deisy Morselli Gysi, Ítalo Do Valle, Marinka Zitnik, Asher Ameli, Xiao Gan, Onur Varol, Helia Sanchez, Rebecca Marlene Baron, Dina Ghiassian, Joseph Loscalzo, et al. Network medicine framework for identifying drug repurposing opportunities for COVID-19. *arXiv:2004.07229*, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Junheng Hao, Tong Zhao, Jin Li, Xin Luna Dong, Christos Faloutsos, Yizhou Sun, and Wei Wang. P-companion: A principled framework for diversified complementary product recommendation. In *Information & Knowledge Management*, 2020.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv:1611.04231*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *CVPR*, 2017.

Claude Adrien Helvétius. *De l'esprit*. Durand, 1759.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019.

Sepp Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. PhD thesis, Technische Universität München, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. *arXiv:1706.06122*, 2017.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *J. Physiology*, 148(3):574–591, 1959.

Michael Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim. LieTransformer: Equivariant self-attention for Lie groups. *arXiv:2012.10885*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Haris Iqbal. Harisiqbal88/plotneuralnet v1.0.0, December 2018. URL https://doi.org/10.5281/zenodo.2526396.

Sarah Itani and Dorina Thanou. Combining anatomical and functional networks for neuropathology identification: A case study on autism spectrum disorder. *Medical Image Analysis*, 69:101986, 2021.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *ICML*, 2018.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *ICML*, 2020.

Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific Data*, 3(1):1–9, 2016.

Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in Psychology*, volume 121, pages 471–495. 1997.

Chaitanya Joshi. Transformers are graph neural networks. *The Gradient*, 2020.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *ICML*, 2015.

Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. *arXiv:1511.08228*, 2015.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv:1610.10099*, 2016.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *ICML*, 2018.

Ken-Ichi Kanatani. *Group-theoretical methods in image understanding*. Springer, 2012.

Zachi Karni and Craig Gotsman. Spectral compression of mesh geometry. In *Proc. Computer Graphics and Interactive Techniques*, 2000.

Anees Kazi, Luca Cosmo, Nassir Navab, and Michael Bronstein. Differentiable graph module (DGM) graph convolutional networks. *arXiv:2002.04999*, 2020.

Henry Kenlay, Dorina Thanou, and Xiaowen Dong. Interpretable stability bounds for spectral graph filters. *arXiv:2102.09587*, 2021.

Ron Kimmel and James A Sethian. Computing geodesic paths on manifolds. *PNAS*, 95(15):8431–8435, 1998.

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

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

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

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

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv:1611.07308*, 2016b.

Dmitry B Kireev. Chemnet: a novel neural network based method for graph/property mapping. *J. Chemical Information and Computer Sciences*, 35(2):175–180, 1995.

Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *arXiv:2003.03123*, 2020.

Iasonas Kokkinos, Michael M Bronstein, Roee Litman, and Alex M Bronstein. Intrinsic shape context descriptors for deformable shapes. In *CVPR*, 2012.

Patrick T Komiske, Eric M Metodiev, and Jesse Thaler. Energy flow networks: deep sets for particle jets. *Journal of High Energy Physics*, 2019(1):121, 2019.

Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. Surface networks. In *CVPR*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Rueckert. Distance metric learning using graph convolutional networks: Application to functional brain networks. In *MICCAI*, 2017.

Dominik Kulon, Riza Alp Guler, Iasonas Kokkinos, Michael M Bronstein, and Stefanos Zafeiriou. Weakly-supervised mesh-convolutional hand reconstruction in the wild. In *CVPR*, 2020.

Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. *arXiv:1511.06392*, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

Reiner Lenz. *Group theoretical methods in image processing*. Springer, 1990.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Processing*, 67(1):97–109, 2018.

Ron Levie, Elvin Isufi, and Gitta Kutyniok. On the transferability of spectral graph filters. In *Sampling Theory and Applications*, 2019.

Bruno Lévy. Laplace-Beltrami eigenfunctions towards an algorithm that "understands" geometry. In *Proc. Shape Modeling and Applications*, 2006.

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

Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. In *CVPR*, 2018.

Roee Litman and Alexander M Bronstein. Learning spectral descriptors for deformable shape correspondence. *Trans. PAMI*, 36(1):171–180, 2013.

Hsueh-Ti Derek Liu, Alec Jacobson, and Keenan Crane. A Dirac operator for extrinsic shape analysis. *Computer Graphics Forum*, 36(5):139–149, 2017.

Siwei Lyu and Eero P Simoncelli. Nonlinear image representation using divisive normalization. In *CVPR*, 2008.

Richard H MacNeal. *The solution of partial differential equations by means of electrical networks*. PhD thesis, California Institute of Technology, 1949.

Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. *arXiv:2001.05016*, 2020.

Soha Sadat Mahdi, Nele Nauwelaers, Philip Joris, Giorgos Bouritsas, Shunwang Gong, Sergiy Bokhnyak, Susan Walsh, Mark Shriver, Michael Bronstein, and Peter Claes. 3d facial matching by spiral convolutional metric learning and a biometric fusion-net of demographic properties. *arXiv:2009.04746*, 2020.

VE Maiorov. On best approximation by ridge functions. *Journal of Approximation Theory*, 99(1):68–94, 1999.

Ameesh Makadia, Christopher Geyer, and Kostas Daniilidis. Correspondence-free structure from motion. *IJCV*, 75(3):311–327, 2007.

Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999.

Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

Brandon Malone, Alberto Garcia-Duran, and Mathias Niepert. Learning representations of missing data for predicting patient outcomes. *arXiv:1811.04752*, 2018.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv:1812.09902*, 2018.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv:1905.11136*, 2019.

Jean-Pierre Marquis. Category theory and klein's erlangen program. In *From a Geometrical Point of View*, pages 9–40. Springer, 2009.

Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *CVPR Workshops*, 2015.

James Clerk Maxwell. A dynamical theory of the electromagnetic field. *Philosophical Transactions of the Royal Society of London*, (155):459–512, 1865.

Jason D McEwen, Christopher GR Wallis, and Augustine N Mavor-Parker. Scattering networks on the sphere for scalable and rotationally equivariant spherical cnns. *arXiv:2102.02828*, 2021.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Learning with invariances in random features and kernel models. *arXiv:2102.13219*, 2021.

Simone Melzi, Riccardo Spezialetti, Federico Tombari, Michael M Bronstein, Luigi Di Stefano, and Emanuele Rodolà. Gframes: Gradient-based local reference frame for 3d shape matching. In *CVPR*, 2019.

Facundo Mémoli and Guillermo Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Foundations of Computational Mathematics*, 5(3):313–347, 2005.

Christian Merkwirth and Thomas Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *J. Chemical Information and Modeling*, 45(5):1159–1168, 2005.

Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, pages 35–57. 2003.

Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Trans. Neural Networks*, 20(3):498–511, 2009.

Karla L Miller, Fidel Alfaro-Almagro, Neal K Bangerter, David L Thomas, Essa Yacoub, Junqian Xu, Andreas J Bartsch, Saad Jbabdi, Stamatios N Sotiropoulos, Jesper LR Andersson, et al. Multimodal population brain imaging in the uk biobank prospective epidemiological study. *Nature Neuroscience*, 19(11):1523–1536, 2016.

Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT Press, 2017.

Jovana Mitrovic, Brian McWilliams, Jacob Walker, Lars Buesing, and Charles Blundell. Representation learning via invariant causal mechanisms. *arXiv:2010.07922*, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*, 2017.

Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv:1902.06673*, 2019.

Christopher Morris, Kristian Kersting, and Petra Mutzel. Glocalized Weisfeiler-Lehman graph kernels: Global-local feature maps of graphs. In *ICDM*, 2017.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. In *NeurIPS*, 2020.

Michael C Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex Systems*, 3(4):349–381, 1989.

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

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *ICML*, 2019.

Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv:1811.01900*, 2018.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

John Nash. The imbedding problem for Riemannian manifolds. *Annals of Mathematics*, 63(1):20—63, 1956.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *COLT*, 2015.

Emmy Noether. Invariante variationsprobleme. In *König Gesellsch. d. Wiss. zu Göttingen, Math-Phys. Klassc*, pages 235–257. 1918.

Maks Ovsjanikov, Jian Sun, and Leonidas Guibas. Global intrinsic symmetries of shapes. *Computer Graphics Forum*, 27(5):1341–1348, 2008.

Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: a flexible representation of maps between shapes. *ACM Trans. Graphics*, 31(4):1–11, 2012.

Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *KDD*, 2020.

Sarah Parisot, Sofia Ira Ktena, Enzo Ferrante, Matthew Lee, Ricardo Guerrero, Ben Glocker, and Daniel Rueckert. Disease prediction using graph convolutional networks: application to autism spectrum disorder and alzheimer's disease. *Medical Image Analysis*, 48:117–130, 2018.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

Giuseppe Patanè. Fourier-based and rational graph filters for spectral processing. *arXiv:2011.04055*, 2020.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

Roger Penrose. *The road to reality: A complete guide to the laws of the universe*. Random House, 2005.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv:2010.03409*, 2020.

Fernando J Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *NIPS*, 1988.

Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

Tom J Pollard, Alistair EW Johnson, Jesse D Raffa, Leo A Celi, Roger G Mark, and Omar Badawi. The eicu collaborative research database, a freely available multi-center database for critical care research. *Scientific Data*, 5 (1):1–13, 2018.

Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*, 2018.

H Qu and L Gouskos. Particlenet: jet tagging via particle clouds. *arXiv:1902.08570*, 2019.

Meng Qu, Yoshua Bengio, and Jian Tang. GMNN: Graph Markov neural networks. In *ICML*, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J Black. Generating 3D faces using convolutional mesh autoencoders. In *ECCV*, 2018.

Dan Raviv, Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. Symmetries of non-rigid shapes. In *ICCV*, 2007.

Noam Razin and Nadav Cohen. Implicit regularization in deep learning may not be explainable by norms. *arXiv:2005.06398*, 2020.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv:1511.06279*, 2015.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv:1506.01497*, 2015.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.

Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

AJ Robinson and Frank Fallside. *The utility driven dynamic error propagation network*. University of Cambridge, 1987.

Emma Rocheteau, Pietro Liò, and Stephanie Hyland. Temporal pointwise convolutional networks for length of stay prediction in the intensive care unit. *arXiv:2007.09483*, 2020.

Emma Rocheteau, Catherine Tong, Petar Veličković, Nicholas Lane, and Pietro Liò. Predicting patient outcomes with graph representation learning. *arXiv:2101.03940*, 2021.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv:2006.10637*, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115 (3):211–252, 2015.

Raif M Rustamov, Maks Ovsjanikov, Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Leonidas Guibas. Map-based exploration of intrinsic shape differences and variability. *ACM Trans. Graphics*, 32(4):1–12, 2013.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv:1602.07868*, 2016.

Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ODE integrators. *arXiv:1909.12790*, 2019.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *ICML*, 2020.

Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE Trans. Signal Processing*, 61(7):1644–1656, 2013.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *NIPS*, 2017.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. *arXiv:1806.01822*, 2018.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *arXiv:1805.11604*, 2018.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv:2002.03155*, 2020.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. *arXiv:2102.09844*, 2021.

Anna MM Scaife and Fiona Porter. Fanaroff-Riley classification of radio galaxies using group-equivariant convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2008.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

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

Terrence J Sejnowski, Paul K Kienker, and Geoffrey E Hinton. Learning symmetry groups with hidden units: Beyond the perceptron. *Physica D: Nonlinear Phenomena*, 22(1-3):260–275, 1986.

Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

Thomas Serre, Aude Oliva, and Tomaso Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the national academy of sciences*, 104(15):6424–6429, 2007.

Ohad Shamir and Gal Vardi. Implicit regularization in relu networks with the square loss. *arXiv:2012.05156*, 2020.

John Shawe-Taylor. Building symmetries into feedforward networks. In *ICANN*, 1989.

John Shawe-Taylor. Symmetries and discriminability in feedforward network architectures. *IEEE Trans. Neural Networks*, 4(5):816–826, 1993.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12(9), 2011.

Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2 (2):021001, 2020.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Eero P Simoncelli and William T Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *Proceedings., International Conference on Image Processing*, volume 3, pages 444–447. IEEE, 1995.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A Hilbert space embedding for distributions. In *ALT*, 2007.

Stefan Spalević, Petar Veličković, Jovana Kovačević, and Mladen Nikolić. Hierachial protein function prediction with tail-GNNs. *arXiv:2007.12804*, 2020.

Alessandro Sperduti. Encoding labeled graphs by labeling RAAM. In *NIPS*, 1994.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, 8(3):714–735, 1997.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv:1412.6806*, 2014.

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

Kimberly Stachenfeld, Jonathan Godwin, and Peter Battaglia. Graph networks with spectral message passing. *arXiv:2101.00079*, 2020.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackerman, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

Heiko Strathmann, Mohammadamin Barekatain, Charles Blundell, and Petar Veličković. Persistent message passing. *arXiv:2103.01043*, 2021.

Norbert Straumann. Early history of gauge theories and weak interactions. *hep-ph/9609230*, 1996.

Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv:1409.3215*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? *arXiv:1804.11188*, 2018.

Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. In *NeurIPS*, 2020.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

Gabriel Taubin, Tong Zhang, and Gene Golub. Optimal surface smoothing as filter design. In *ECCV*, 1996.

Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Bootstrapped representation learning on graphs. *arXiv:2102.06514*, 2021.

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

Renate Tobies. Felix Klein—-mathematician, academic organizer, educational reformer. In *The Legacy of Felix Klein*, pages 5–21. Springer, 2019.

Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. *arXiv:1808.00508*, 2018.

John Tromp and Gunnar Farnebäck. Combinatorics of go. In *International Conference on Computers and Games*, 2006.

Alexandre B Tsybakov. *Introduction to nonparametric estimation*. Springer, 2008.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016.

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

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *ICLR*, 2018.

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *arXiv:1910.10593*, 2019.

Petar Veličković, Lars Buesing, Matthew C Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. *arXiv:2006.06380*, 2020.

Petar Veličković, Wiliam Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *ICLR*, 2019.

Kirill Veselkov, Guadalupe Gonzalez, Shahad Aljifri, Dieter Galea, Reza Mirnezami, Jozef Youssef, Michael Bronstein, and Ivan Laponogov. Hyperfoods: Machine intelligent mapping of cancer-beating molecules in foods. *Scientific Reports*, 9(1):1–12, 2019.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv:1506.03134*, 2015.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

Ulrike von Luxburg and Olivier Bousquet. Distance-based classification with lipschitz functions. *JMLR*, 5:669–695, 2004.

Martin J Wainwright and Michael Irwin Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

Yu Wang and Justin Solomon. Intrinsic and extrinsic operators for shape analysis. In *Handbook of Numerical Analysis*, volume 20, pages 41–115. Elsevier, 2019.

Yu Wang, Mirela Ben-Chen, Iosif Polterovich, and Justin Solomon. Steklov spectral geometry for extrinsic shape analysis. *ACM Trans. Graphics*, 38(1): 1–21, 2018.

Yu Wang, Vladimir Kim, Michael Bronstein, and Justin Solomon. Learning geometric operators on meshes. In *ICLR Workshops*, 2019a.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graphics*, 38(5):1–12, 2019b.

Max Wardetzky. Convergence of the cotangent formula: An overview. *Discrete Differential Geometry*, pages 275–286, 2008.

Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete Laplace operators: no free lunch. In *Symposium on Geometry Processing*, 2007.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *arXiv:1807.02547*, 2018.

Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI Series*, 2(9):12–16, 1968.

Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

Hermann Weyl. Elektron und gravitation. i. *Zeitschrift für Physik*, 56(5-6): 330–352, 1929.

Hermann Weyl. *Symmetry*. Princeton University Press, 2015.

Marysia Winkels and Taco S Cohen. Pulmonary nodule detection in ct scans with equivariant cnns. *Medical Image Analysis*, 55:15–26, 2019.

Jeffrey Wood and John Shawe-Taylor. Representation theory and invariant neural networks. *Discrete Applied Mathematics*, 69(1-2):33–60, 1996.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv:2002.07962*, 2020a.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv:1810.00826*, 2018.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv:1905.13211*, 2019.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv:2009.11848*, 2020b.

Yujun Yan, Kevin Swersky, Danai Koutra, Parthasarathy Ranganathan, and Milad Heshemi. Neural execution engines: Learning to execute subroutines. *arXiv:2006.08084*, 2020.

Chen-Ning Yang and Robert L Mills. Conservation of isotopic spin and isotopic gauge invariance. *Physical Review*, 96(1):191, 1954.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Jonathan S Yedidia, William T Freeman, and Yair Weiss. Bethe free energy, kikuchi approximations, and belief propagation algorithms. *NIPS*, 2001.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *ICML*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *NIPS*, 2017.

Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv:1410.4615*, 2014.

Wei Zeng, Ren Guo, Feng Luo, and Xianfeng Gu. Discrete heat kernel determines discrete riemannian metric. *Graphical Models*, 74(4):121–129, 2012.

Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv:1803.07294*, 2018.

Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Efficient probabilistic logic reasoning with graph neural networks. *arXiv:2001.11850*, 2020.

Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: A comprehensive graph neural network platform. *arXiv:1902.08730*, 2019.

Weicheng Zhu and Narges Razavian. Variationally regularized graph-based representation learning for electronic health records. *arXiv:1912.03761*, 2019.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv:2006.04131*, 2020.

Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34 (13):i457–i466, 2018.