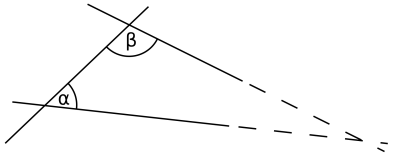
Going back in time...

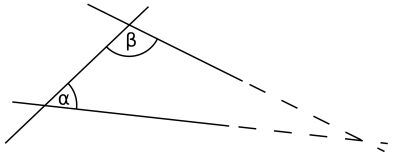
Going back in time...

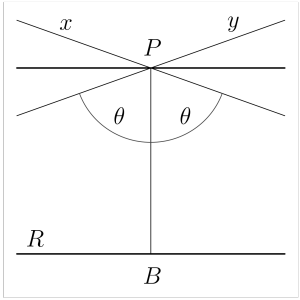


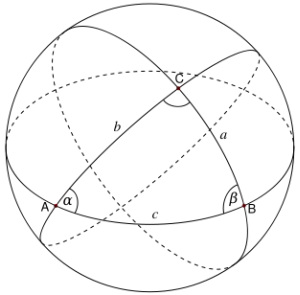
Euclid

~300 B.C.

Going back in time...







Euclid

~300 B.C.

Lobachevsky

~1829

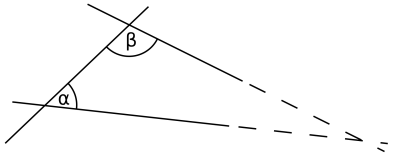
Bolyai

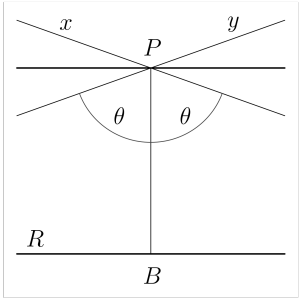
~1830

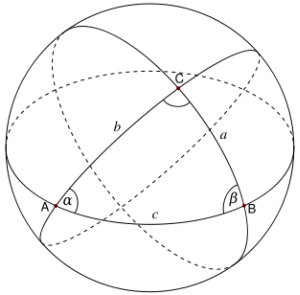
Riemann

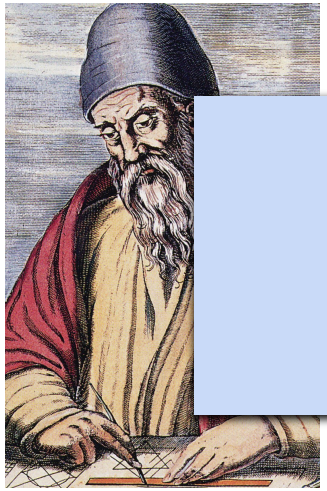
~1856

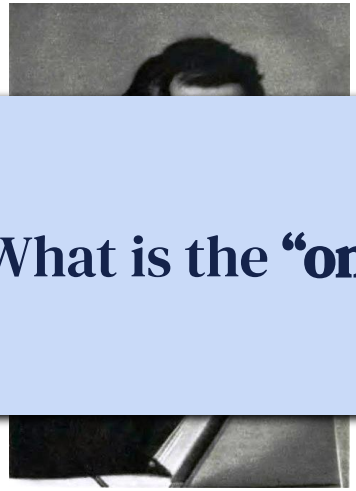
Going back in time...

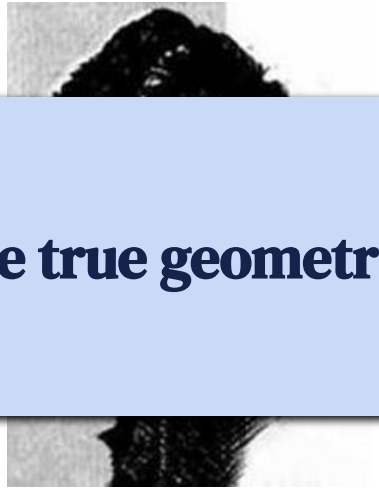


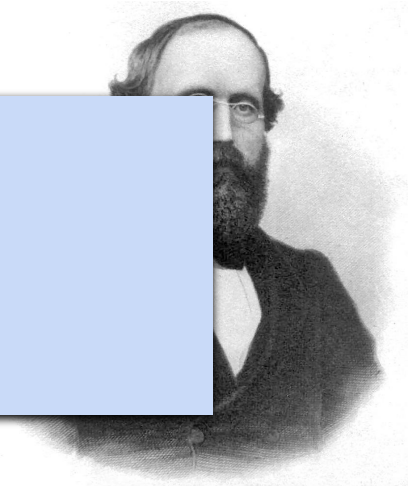












What is the **“one true geometry”**?

Euclid

~300 B.C.

Lobachevsky

~1829

Bolyai

~1830

Riemann

~1856

Felix Klein and the “Erlangen Program” (1872)

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.

Felix Klein and the “Erlangen Program” (1872)

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.

Blueprint for **unifying** geometries

Lens of **invariances** and **symmetries**

Formalised in the language of **group theory**!

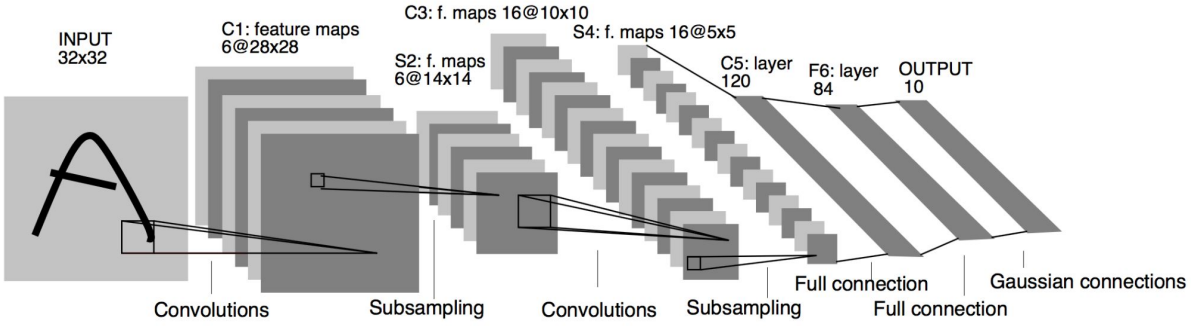
Impact of the Erlangen Program

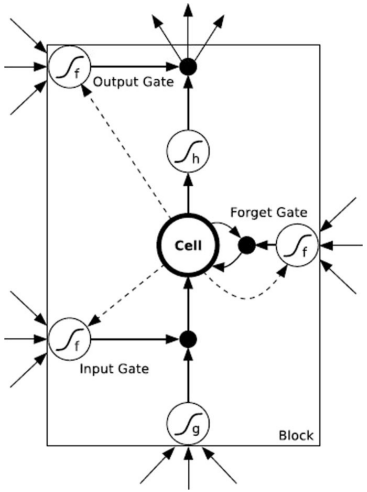
- Strong impact on **geometry**; no longer hunting for the “one true geometry”
  - Formalised by *Élie Cartan* in the 1920s

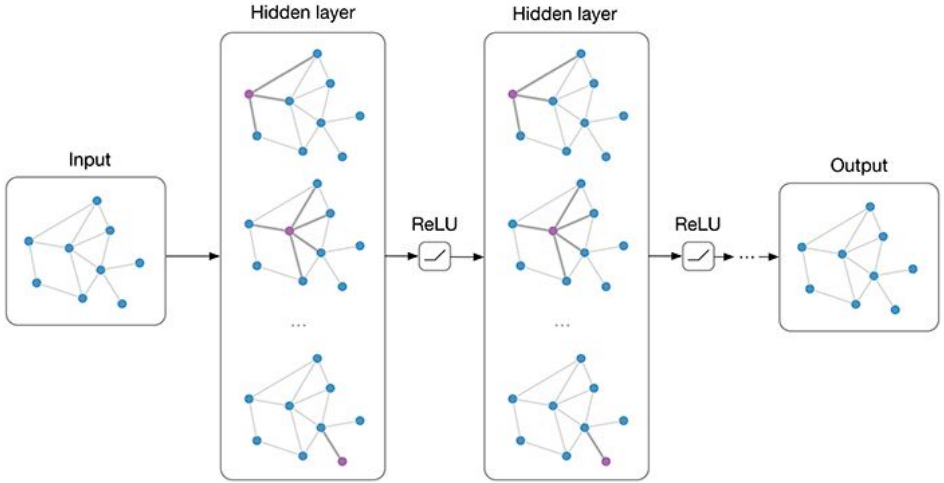
- *Physics*: **Noether’s Theorem** (all conservation laws derivable from symmetry!)
  - Even enabled the classification of elementary particles (*irreducible representations*)

- *Category Theory*
  - *“can be regarded as a continuation of the Klein Erlangen Program, in the sense that a geometrical space with its group of transformations is generalized to a category with its algebra of mappings”*; – Eilenber, Lane (creators of CT)

Deep learning, circa 2020







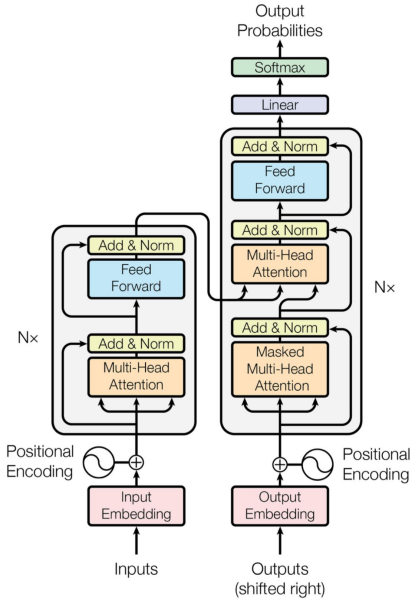
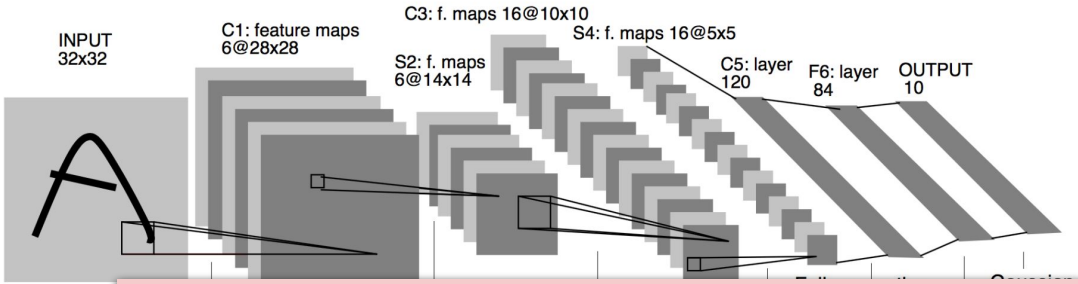
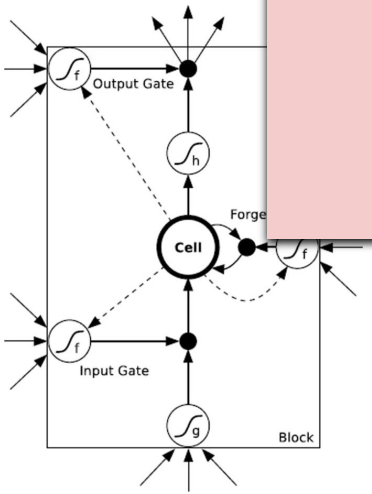
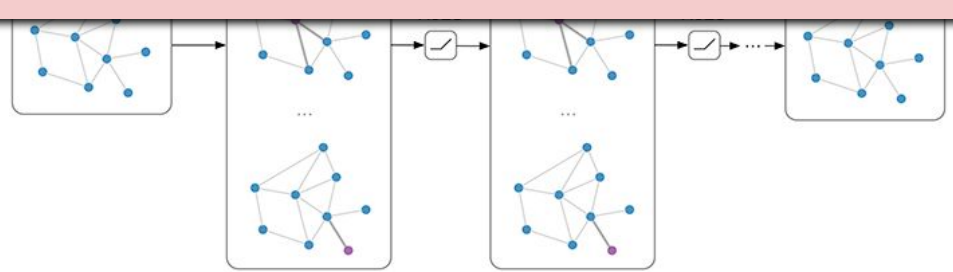


Figure 1: The Transformer - model architecture.

Deep learning, circa 2020







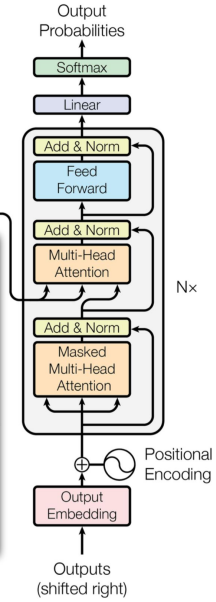
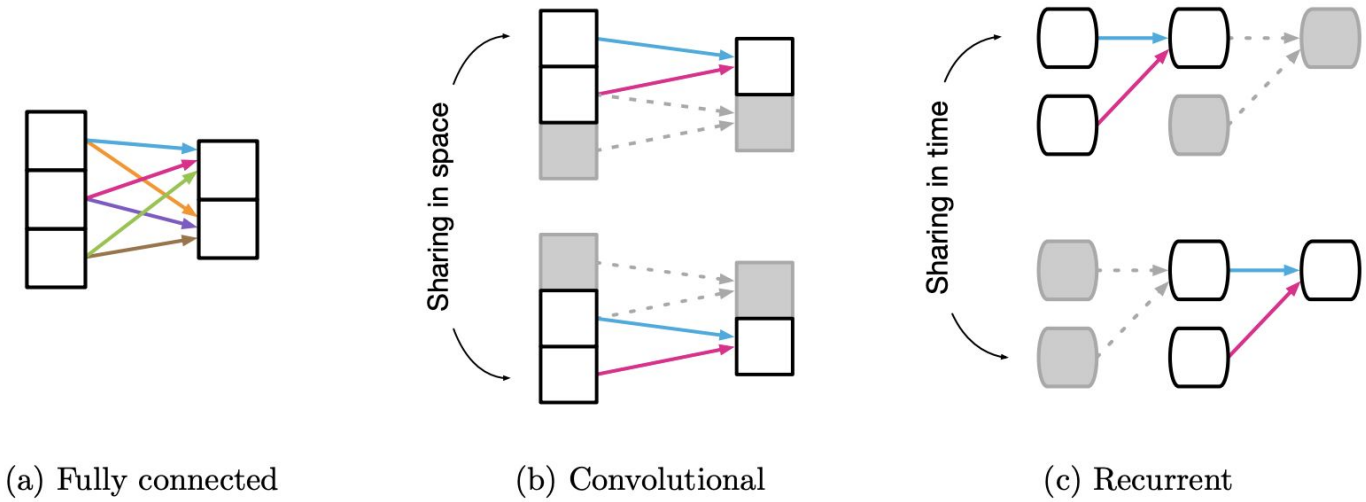


Figure 1: The Transformer - model architecture.

What is the "one true architecture"?

Could GNNs be the answer?

- "If we squint hard enough", (m)any NNs can be seen as message passing over a graph
  - Further, most data we receive from **nature** is inherently graph-structured
  - So, GNNs likely play a part in the "one true architecture" (motivating this course)



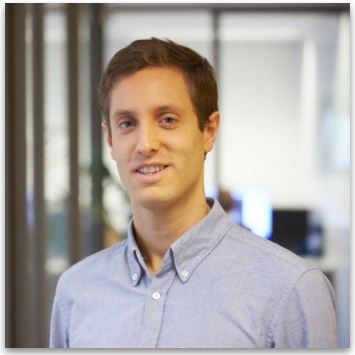
- But to formalise this, we need to understand GNNs **beyond** permutation equivariance!

DeepMind

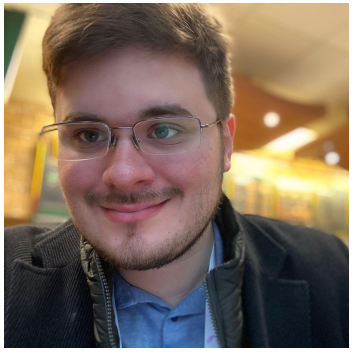
...now it's our turn to study geometry :)

Michael Bronstein
Oxford / Twitter

Joan Bruna
NYU



Taco Cohen
Qualcomm



Petar Veličković
DeepMind / Cambridge

DeepMind

Geometric Deep Learning

GNNs Beyond Permutation Equivariance

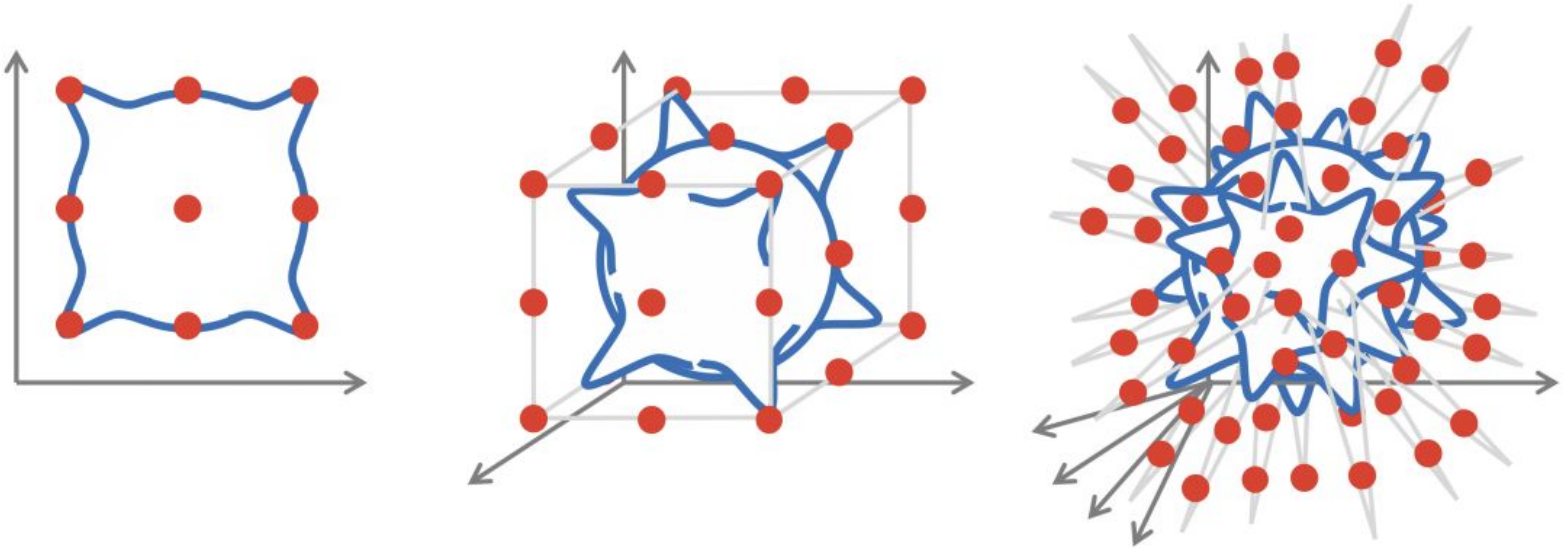
Petar Veličković

Stanford University
CS224W
30 November 2021

DeepMind

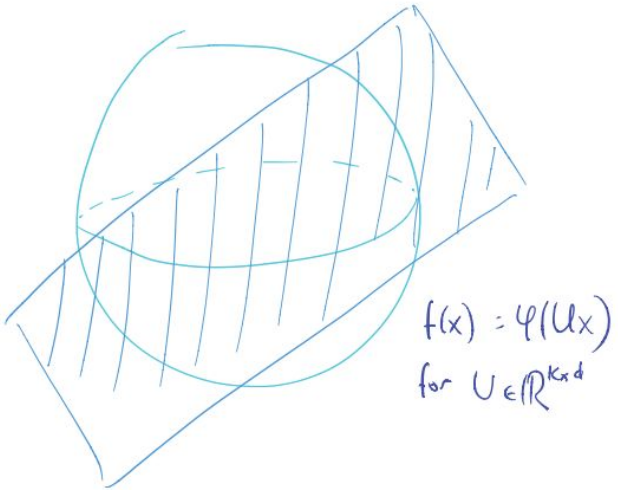
1 Learning in high dimensions is **hard**

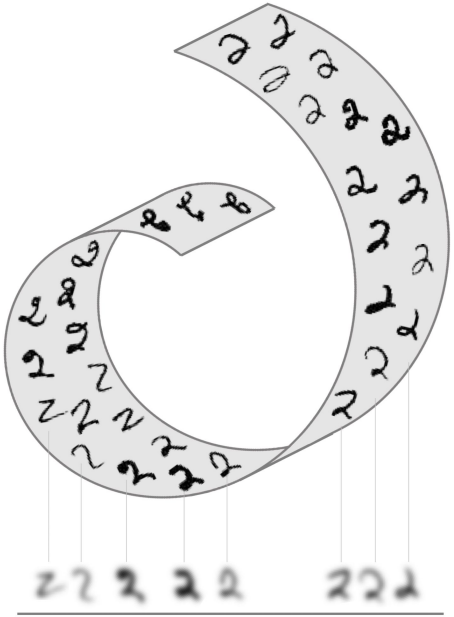
The Curse of Dimensionality



Even “simple” target functions (e.g. 1-Lipschitz) require **exponential** samples in nb. dimensions.

Low-dimensional projections don't necessarily help!





Shallow MLPs can lose a lot of the **fidelity** of their inputs.

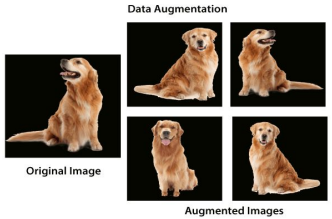
What can we do?

DeepMind

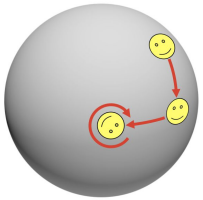
2 Symmetries, Groups and Invariances

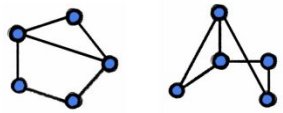
Geometry to the rescue!

- We can inject further assumptions about the **geometry** of through *inductive biases*
  - Restrict the functions in our hypothesis space to ones that *respect* the geometry.
  - This can make the high-dimensional problem more tractable!



- Some popular examples:
  - **Image** data should be processed independently of **shifts**
  - **Spherical** data should be processed independently of **rotations**
  - **Graph** data should be processed independently of **isomorphism**





- We will now attempt to **formalise** this!

Key elements!

- We assume data lives on a ***domain***, $\mathbf{\Omega}$
  - e.g. for images, $u \in \mathbf{\Omega}$ are *pixels*; for graphs they are *nodes*

- We assume a ***feature space***, $\mathbf{C}$, to be stored in elements of a domain
  - For our purposes, $\mathbf{C} = \mathbb{R}^k$

- We can then define *featurised domains* using a space of **feature functions** $\mathbf{X}(\mathbf{\Omega}, \mathbf{C})$
  - $x \in \mathbf{X}(\mathbf{\Omega}, \mathbf{C})$ is a function s.t. $x(u) \in \mathbf{C}$ gives features of element $u \in \mathbf{\Omega}$
  - For discrete environments we can think of $\mathbf{X}$ as a *feature matrix* ($\mathbf{X} \in \mathbb{R}^{|\mathbf{\Omega}| \times k}$)

Principle 1: Symmetry groups

- Symmetry is a transformation that leaves an object **invariant** (i.e. *unchanged*)
  - Hence they must be **composable**, **invertible**, contain **identity**…

- In fact, they can be reasoned about using a very elegant mathematical object: the **group**

- Elements of these groups are domain **transformations** (e.g. some functions $\mathfrak{g} : \mathbf{\Omega} \to \mathbf{\Omega}$).

Principle 1: Symmetry groups

- Symmetry is a transformation that leaves an object **invariant** (i.e. *unchanged*)
  - Hence they must be **composable**, **invertible**, contain **identity**…

- In fact, they can be reasoned about using a very elegant mathematical object: the **group**

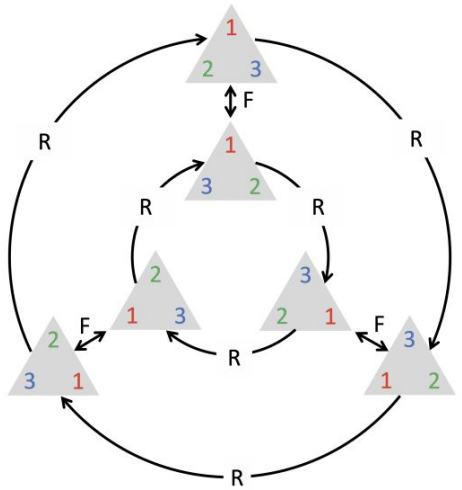
A *group* is a set $\mathfrak{G}$ along with a binary operation $\circ : \mathfrak{G} \times \mathfrak{G} \to \mathfrak{G}$ called *composition* (for brevity, denoted by juxtaposition $\mathfrak{g} \circ \mathfrak{h} = \mathfrak{gh}$) satisfying the following axioms:

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

Identity: there exists a unique $\mathfrak{e} \in \mathfrak{G}$ satisfying $\mathfrak{eg} = \mathfrak{ge} = \mathfrak{g}$ 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 $\mathfrak{gg}^{-1} = \mathfrak{g}^{-1}\mathfrak{g} = \mathfrak{e}$.

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



Principle 1: Symmetry groups

- Symmetry is a transformation that leaves an object **invariant** (i.e. *unchanged*)
  - Hence they must be **composable**, **invertible**, contain **identity**…

- In fact, they can be reasoned about using a very elegant mathematical object: the **group**

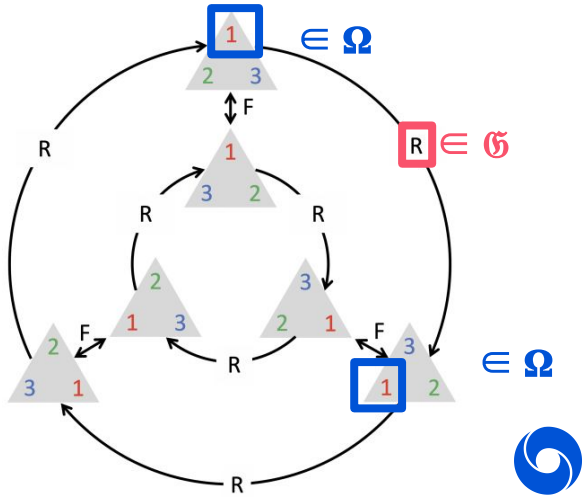
A *group* is a set $\mathfrak{G}$ along with a binary operation $\circ : \mathfrak{G} \times \mathfrak{G} \to \mathfrak{G}$ called *composition* (for brevity, denoted by juxtaposition $\mathfrak{g} \circ \mathfrak{h} = \mathfrak{gh}$) satisfying the following axioms:

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

Identity: there exists a unique $\mathfrak{e} \in \mathfrak{G}$ satisfying $\mathfrak{eg} = \mathfrak{ge} = \mathfrak{g}$ 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 $\mathfrak{gg}^{-1} = \mathfrak{g}^{-1}\mathfrak{g} = \mathfrak{e}$.

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



Group actions

- We are interested in how these groups affect data
  - Group action $(\mathfrak{g}, u) \mapsto \mathfrak{g}.u$ for a group element *g*, and a domain element *u*
  - E.g. translating or rotating an image, or permuting a set

- We will be interested in ***linear*** group actions: $\mathfrak{g}.(\alpha x + \beta x') = \alpha(\mathfrak{g}.x) + \beta(\mathfrak{g}.x')$

- This also allows us to represent group actions using **linear algebra**; $\rho : \mathfrak{G} \to \mathbb{R}^{n \times n}$

> 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 $\rho(\mathfrak{g}\mathfrak{h}) = \rho(\mathfrak{g})\rho(\mathfrak{h})$ 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 $\mathfrak{g} \in \mathfrak{G}$.

Invariance and equivariance

- We can largely simplify **high-dimensional learning** by *exploiting* the symmetries in $\mathfrak{G}$!

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

e.g. **image classification**: output class (*likely?*) won't depend on image **shifts**

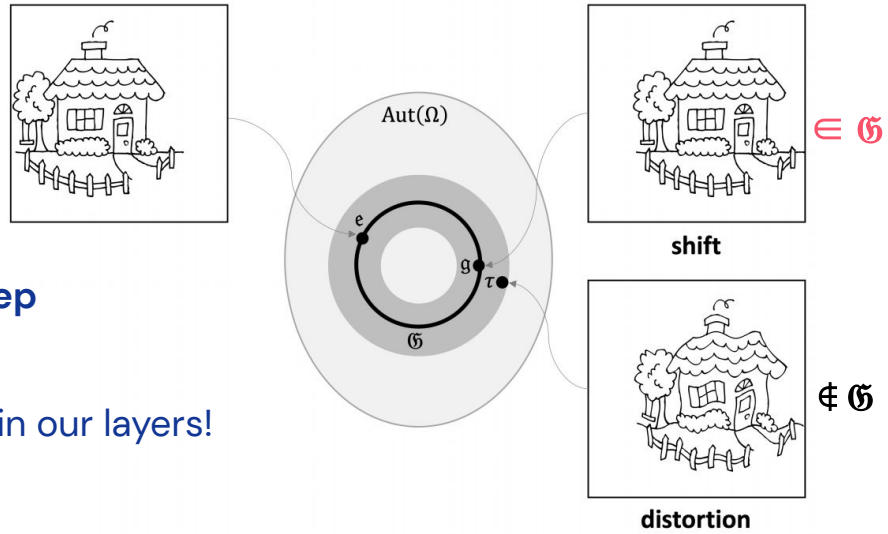
A function $f : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega)$ is $\mathfrak{G}$-*equivariant* if $f(\rho(\mathfrak{g})x) = \rho(\mathfrak{g})f(x)$ for all $\mathfrak{g} \in \mathfrak{G}$, i.e., group action on the input affects the output in the same way.

e.g. **image segmentation**: segmentation mask must **follow** any shifts in the input

Principle 2: Scale separation

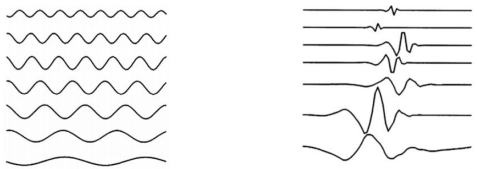
- Want signal to be **stable** under slight *deformations* of the domain

- We derive: highly beneficial to compose **local** operations to model larger-scale ones
  - local ops won't globally propagate errors
  - e.g. CNNs with 3 x 3 kernels, but **very deep**



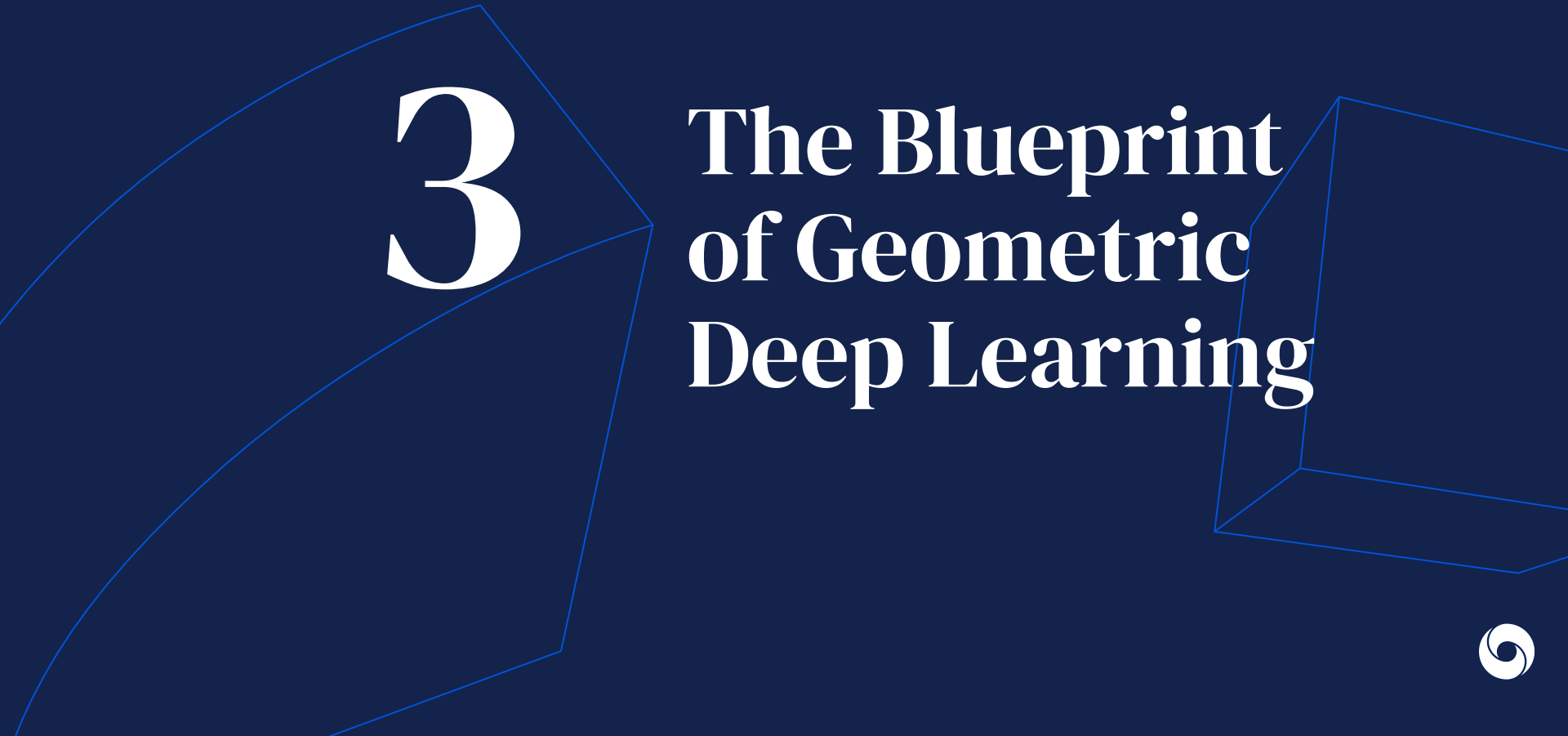
- Accordingly, we would like to support ***locality*** in our layers!

- cf. Fourier Transform vs. Wavelets



DeepMind

3 The Blueprint of Geometric Deep Learning



The key "building blocks" of Geometric Deep Learning

Linear $\mathfrak{G}$-equivariant layer $B : \mathcal{X}(\Omega, \mathcal{C}) \rightarrow \mathcal{X}(\Omega', \mathcal{C}')$,
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 $\sigma : \mathcal{C} \rightarrow \mathcal{C}'$ applied element-wise as $\big(\boldsymbol{\sigma}(x)\big)(u) = \sigma(x(u))$.

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

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

The key "building blocks" of Geometric Deep Learning

Equivariant local layers

Linear $\mathfrak{G}$-equivariant layer $B : \mathcal{X}(\Omega, \mathcal{C}) \rightarrow \mathcal{X}(\Omega', \mathcal{C}')$,
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 $\sigma : \mathcal{C} \rightarrow \mathcal{C}'$ applied element-wise as $\big(\boldsymbol{\sigma}(x)\big)(u) = \sigma(x(u))$.

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

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

Invariant "tail" (if necessary)

The key "building blocks" of Geometric Deep Learning

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 $x \in \mathcal{X}(\Omega, \mathcal{C})$.

Activation function

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

(necessary for deep learning! :))

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 $x \in \mathcal{X}(\Omega, \mathcal{C})$.

The key "building blocks" of Geometric Deep Learning

Linear $\mathfrak{G}$-equivariant layer $B : \mathcal{X}(\Omega, \mathcal{C}) \rightarrow \mathcal{X}(\Omega', \mathcal{C}')$,
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 $\sigma : \mathcal{C} \rightarrow \mathcal{C}'$ applied element-wise as $\big(\boldsymbol{\sigma}(x)\big)(u) = \sigma(x(u))$.

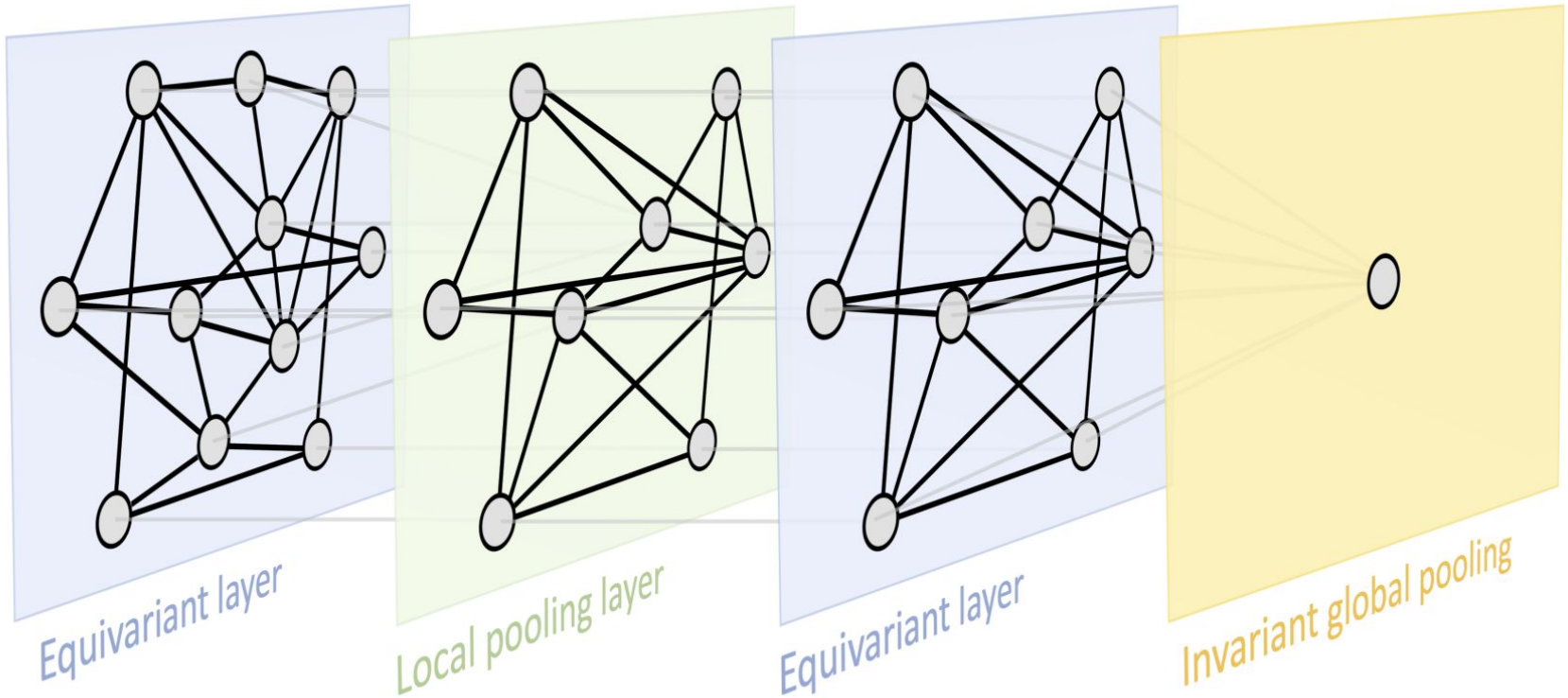
Coarsening layer

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

(Not covered here in detail, but follows from scale separation!)

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

All you need to build the architectures that are all you need :)



All the fan-favourites are easily derivable

...with many (potentially) unexpected and useful **extras**! :)

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

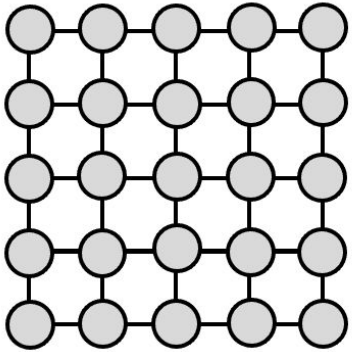
DeepMind

4

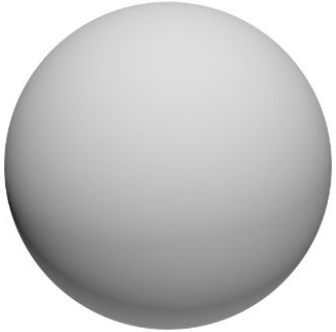
The “5G” of Geometric Deep Learning

The “5G” of Geometric Deep Learning

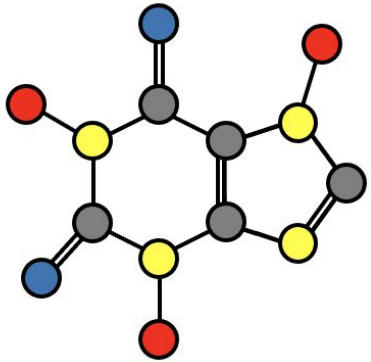
We will use the remainder of this lecture to study a few interesting **instances** of this blueprint



Grids



Groups

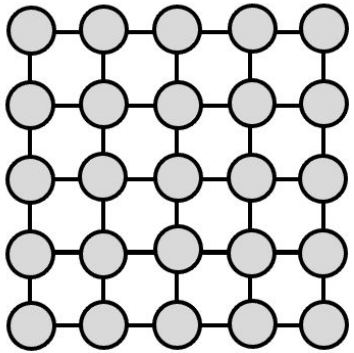


Graphs

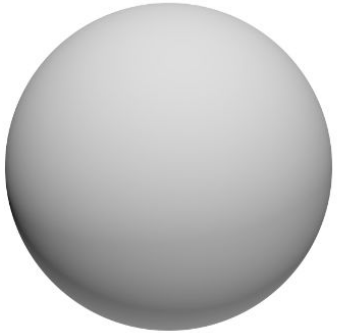
Geodesics & Gauges

The “5G” of Geometric Deep Learning

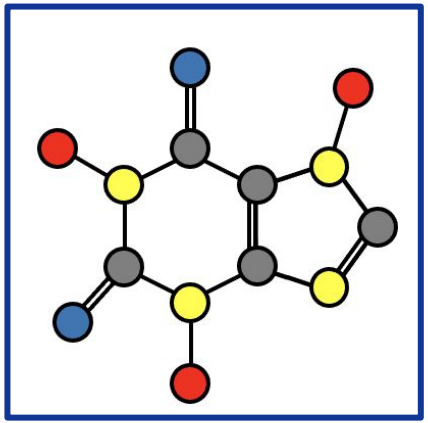
We will use the remainder of this lecture to study a few interesting **instances** of this blueprint



Images & Sequences



Homogeneous spaces



Graphs & Sets

Manifolds, Meshes & Geometric graphs

Throughout CS224W, the focus was primarily on the domain of **graphs**.

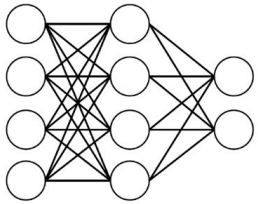
Our strategy for the rest of the lecture

- We start by seeing how **GNNs** fit in this paradigm
  - This will involve re-deriving / re-introducing some concepts you've seen in the course

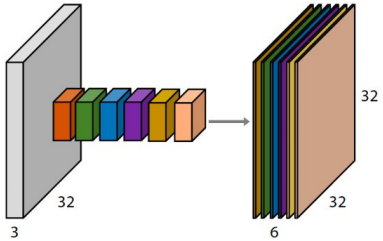
- Then, we will see how we can use the blueprint to **expand** GNNs into other domains
  - Also, it will give us an insight into "the world beyond"
  - GNNs **beyond** permutation equivariance!

- Our discussion will span many architectures you (may) know of :)
  - Deep Sets
  - Transformers
  - CNNs
  - Spherical CNNs
  - Mesh CNNs

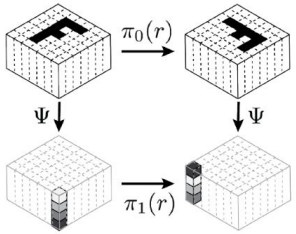
Architectures of interest



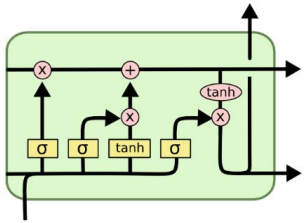
Perceptrons
Function regularity



CNNs
Translation

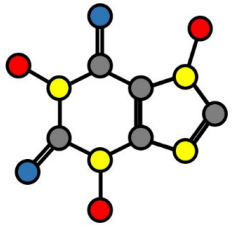


Group-CNNs
Translation+Rotation,
Global groups



LSTMs
Time warping

DeepSets / Transformers
Permutation



GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

DeepMind

5 Geometric DL Perspective on Graph Neural Networks

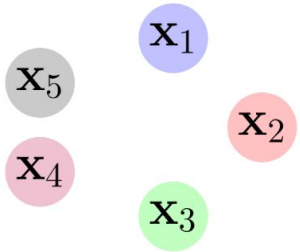
Learning on sets: Setup

- For now, assume the graph **has no edges** (e.g. our domain is just the *set* of nodes, V).
- Let $\mathbf{x}_i \in \mathbb{R}^k$ be the features of node *i*.
- We can stack them into a node feature matrix of shape *n* x *k*:

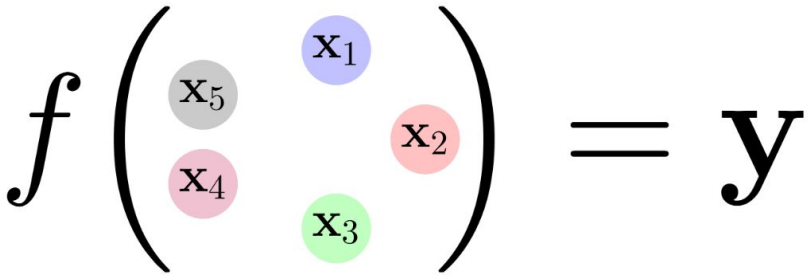
$$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$$

- That is, the *i*th row of **X** corresponds to $\mathbf{x}_i$
- Note that, by doing so, we have specified a **node ordering**!
  - We would like the result of any neural networks to not depend on this.

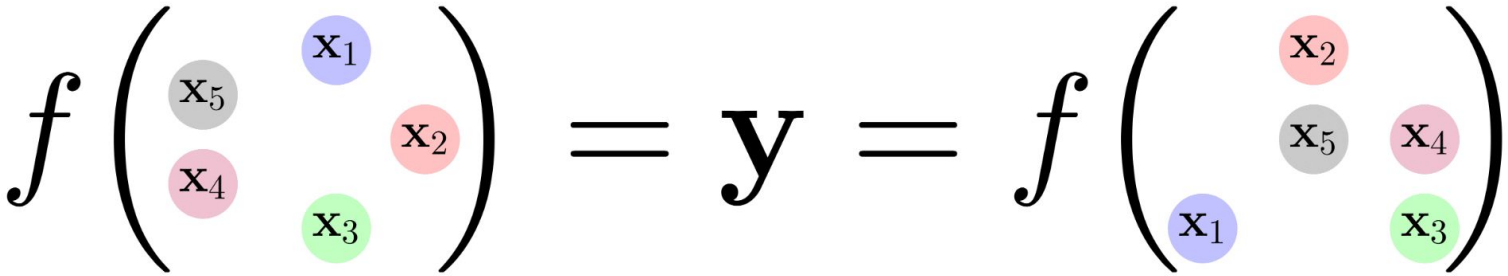
What do we want?



What do we want?



What do we want?



Permutations and permutation matrices

- It will be useful to think about the operations that **change** the node order
  - Such operations are known as **permutations** (there are **n!** of them)
  - e.g. a permutation (2, 4, 1, 3) means $\mathbf{y}_1 \leftarrow \mathbf{x}_2$, $\mathbf{y}_2 \leftarrow \mathbf{x}_4$, $\mathbf{y}_3 \leftarrow \mathbf{x}_1$, $\mathbf{y}_4 \leftarrow \mathbf{x}_3$.

- To stay within linear algebra, each permutation defines an *n* x *n* **matrix (group action!)**
  - Such matrices are called **permutation matrices**
  - They have exactly one 1 in every row and column, and zeros everywhere else
  - Their effect when left-multiplied is to permute the rows of **X**, like so:

$$\mathbf{P}_{(2,4,1,3)}\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} — & \mathbf{x}_1 & — \\ — & \mathbf{x}_2 & — \\ — & \mathbf{x}_3 & — \\ — & \mathbf{x}_4 & — \end{bmatrix} = \begin{bmatrix} — & \mathbf{x}_2 & — \\ — & \mathbf{x}_4 & — \\ — & \mathbf{x}_1 & — \\ — & \mathbf{x}_3 & — \end{bmatrix}$$

Permutation *invariance*

- We want to design functions f(**X**) over sets that will not depend on the order
- Equivalently, applying a permutation matrix shouldn't modify the result!
- We arrive at a useful notion of **permutation invariance**. We say that f(**X**) is permutation *invariant* if, for *all* permutation matrices **P**:

$$f(\mathbf{PX}) = f(\mathbf{X})$$

- One very generic form is the *Deep Sets* model (Zaheer et al., NeurIPS'17): $f(\mathbf{X}) = \phi\left(\sum_{i \in \mathcal{V}} \psi\left(\mathbf{x}_i\right)\right)$ where ψ and ϕ are (learnable) functions, e.g. MLPs.
  - The **sum** aggregation is *critical*! (other choices possible, e.g. **max** or **avg**)

Permutation *equivariance*

- Permutation-*invariant* models are a good way to obtain set-level outputs

- What if we would like answers at the **node** level?
  - We want to still be able to **identify** node outputs, which a permutation-invariant aggregator would destroy!

- We may instead seek functions that don't **change** the node order
  - i.e. if we permute the nodes, it doesn't matter if we do it **before** or **after** the function!

- Accordingly, we say that f(**X**) is **permutation equivariant** if, for all permutation matrices **P**:

$$f(\mathbf{PX}) = \mathbf{P}f(\mathbf{X})$$

Learning on graphs

- Now we augment the set of nodes with **edges** between them.
  - That is, we consider general E ⊆ V x V.

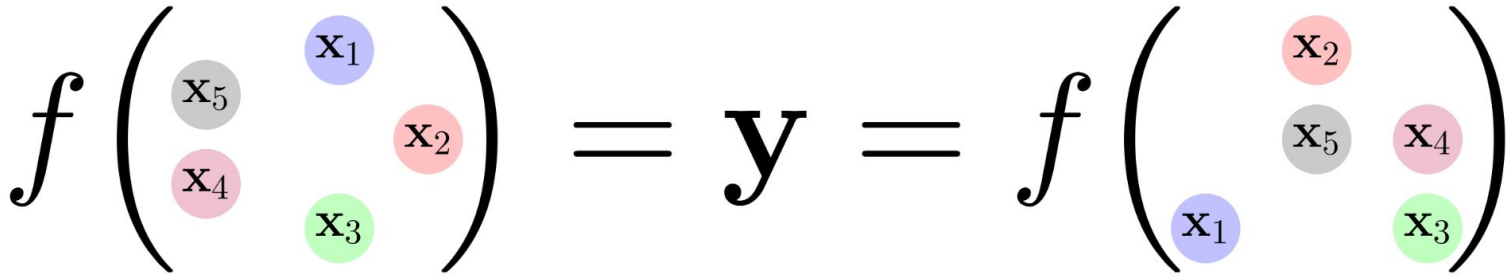
- We can represent these edges with an **adjacency matrix**, **A**, such that:

$$a_{ij} = \begin{cases} 1 & (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

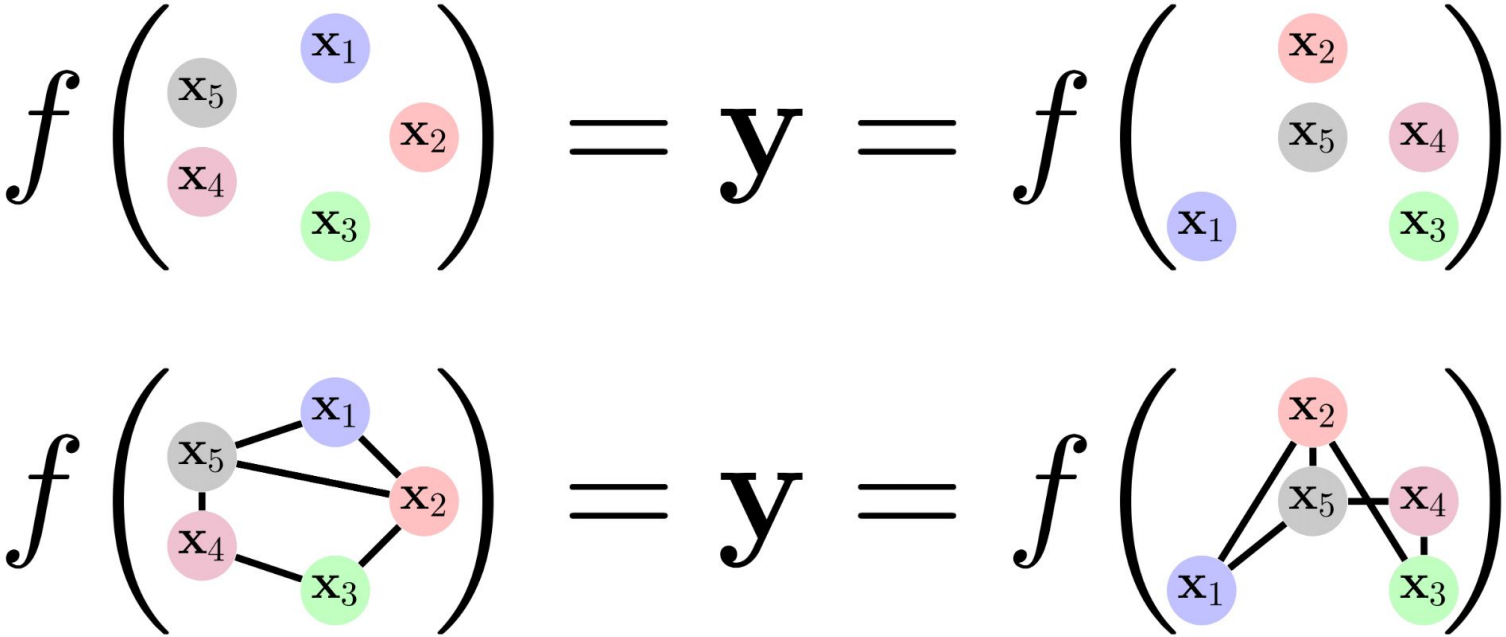
- Further additions (e.g. *edge features*) are possible but **ignored** for simplicity.

- Our main desiderata (*permutation {in,equi}variance*) still hold!

What's changed?



What's changed?



Permutation invariance and equivariance on **graphs**

- The main difference: node permutations now also accordingly act on the **edges**
- We need to appropriately permute both **rows** and **columns** of **A**
  - When applying a permutation matrix **P**, this amounts to $\mathbf{PAP}^\intercal$
- We arrive at updated definitions of suitable functions f(**X, A**) over graphs:

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

Equivariance: $f(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$

Locality on graphs: **neighbourhoods**

- Recall: it is also highly beneficial to design *geometrically stable* (**local**) equivariant layers

- **Graphs** give us a context for locality: a node's **neighbourhood**
  - For a node *i*, its (1-hop) neighbourhood is commonly defined as follows:

$$\mathcal{N}_i = \{j : (i,j) \in \mathcal{E} \vee (j,i) \in \mathcal{E}\}$$

N.B. we do not explicitly consider *directed* edges, and often we assume $i \in N_i$

- Accordingly, we can extract the *multiset* of **features** in the neighbourhood

$$\mathbf{X}_{\mathcal{N}_i} = \{\{\mathbf{x}_j : j \in \mathcal{N}_i\}\}$$

and define a *local* function, g, as operating over this multiset: $g(\mathbf{x}_i, \mathbf{X}_{Ni})$.

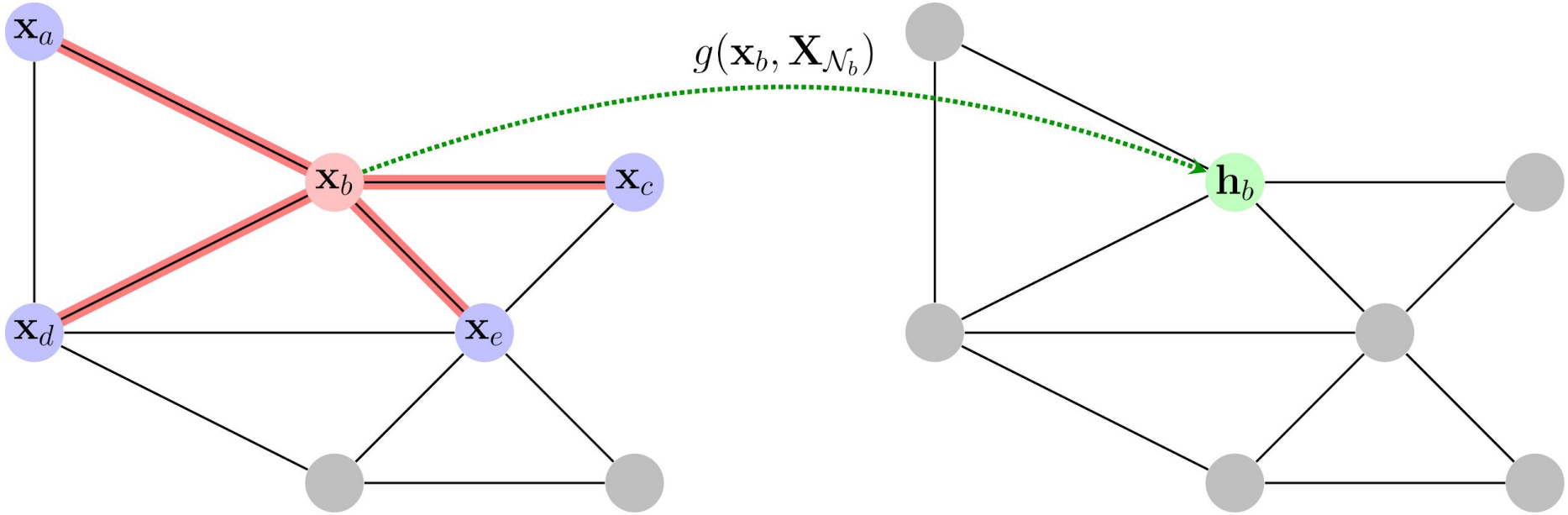
A recipe for **graph** neural networks

- Now we can construct permutation equivariant functions, f(**X, A**), by appropriately applying the local function, g, over *all* neighbourhoods:

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

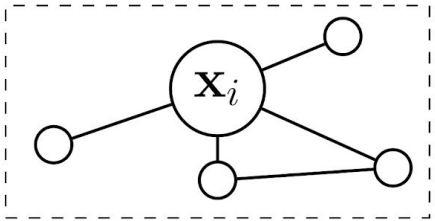
- To ensure equivariance, we need g to not depend on the **order** of the vertices in $\mathbf{X}_{\text{Ni}}$
  - Hence, g should be permutation **invariant**!

A recipe for **graph** neural networks, visualised



$$\mathbf{X}_{\mathcal{N}_b} = \{\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}\}$$

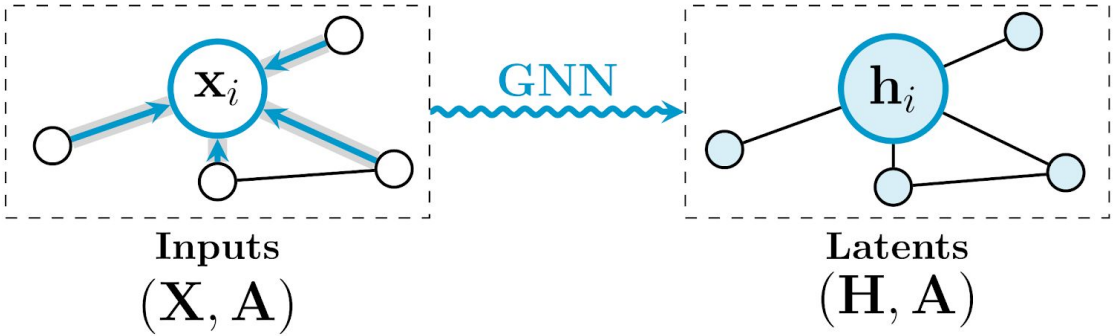
How to use GNNs?



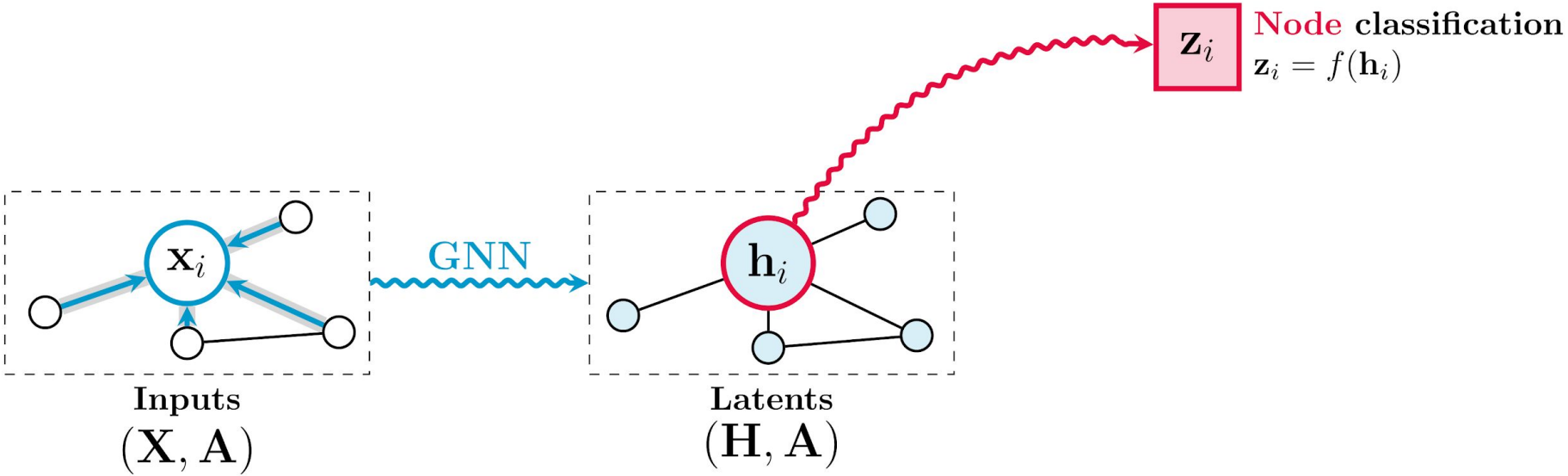
Inputs

$(\mathbf{X}, \mathbf{A})$

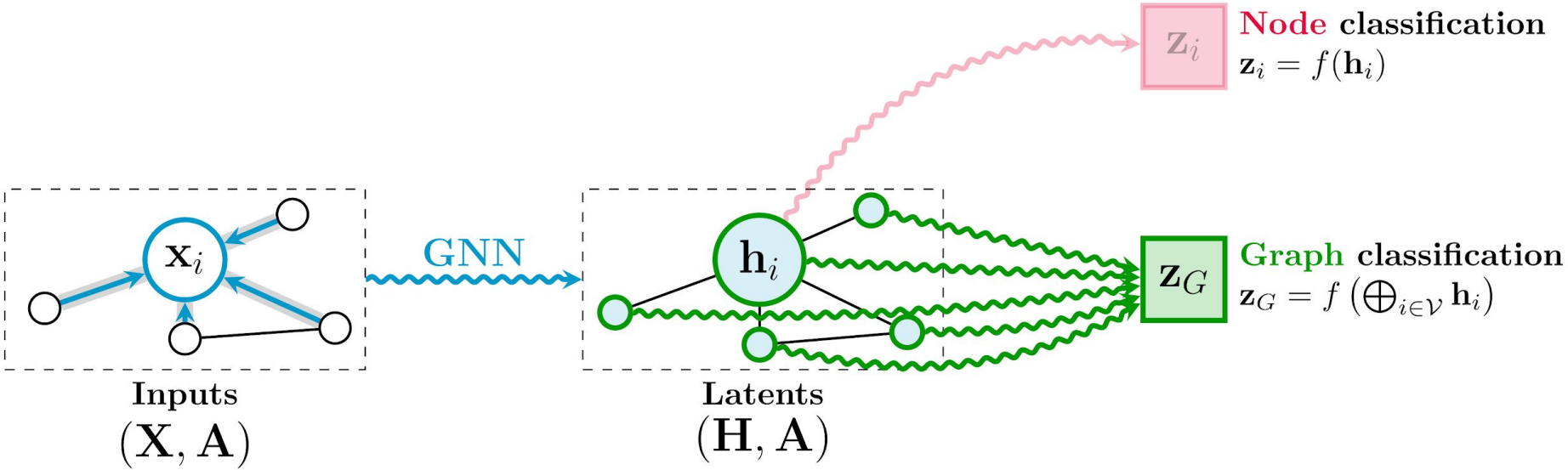
How to use GNNs?



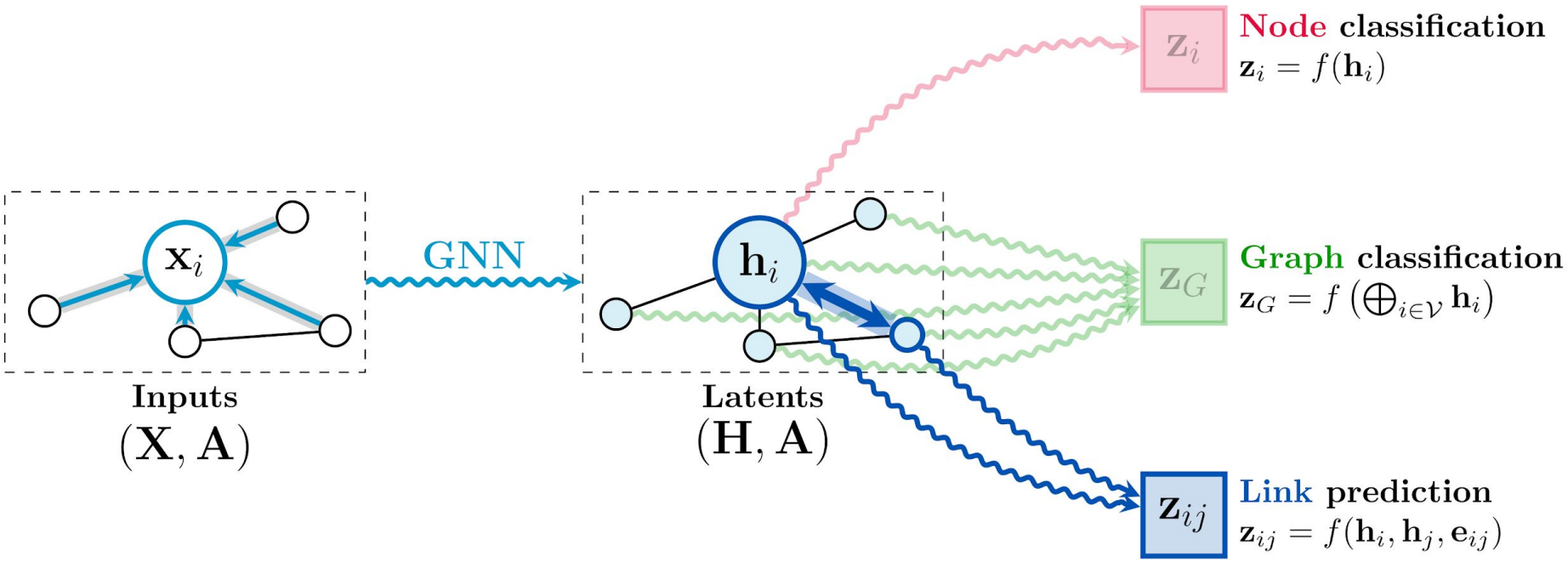
How to use GNNs?



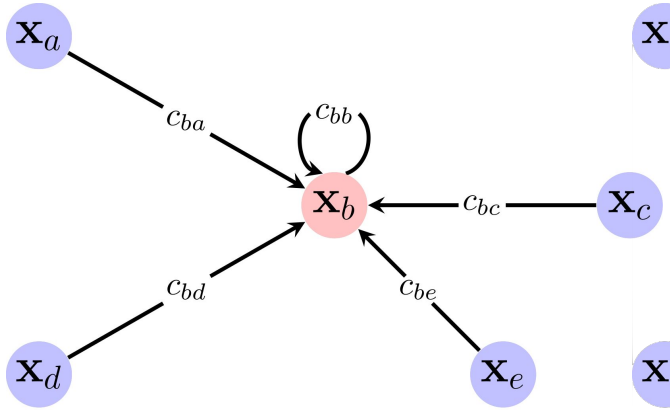
How to use GNNs?



How to use GNNs?



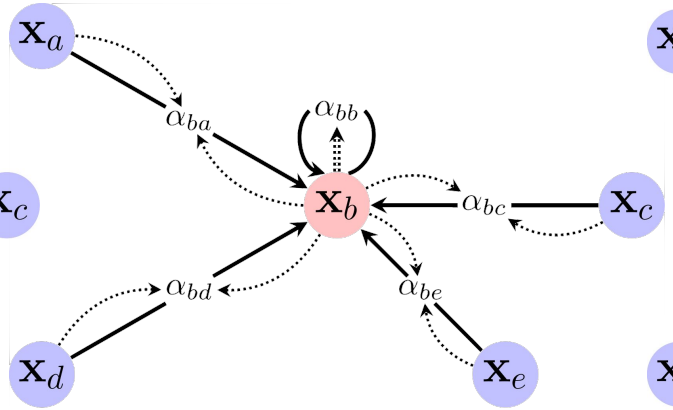
The three “flavours” of GNN layers



Convolutional

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$

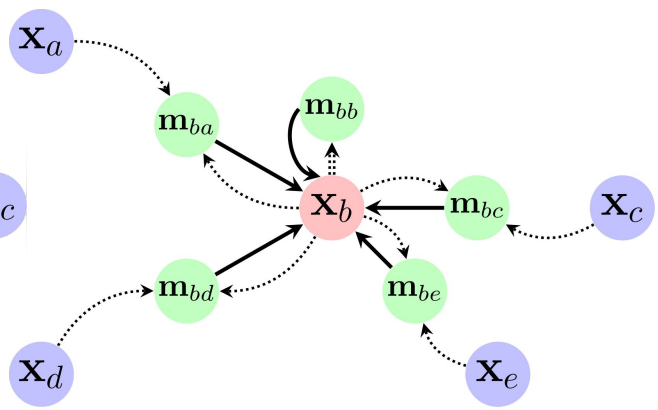
e.g. GraphSAGE, GCN, SGC



Attentional

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

e.g. MoNet, GAT, GATv2

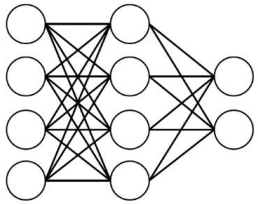


Message-passing

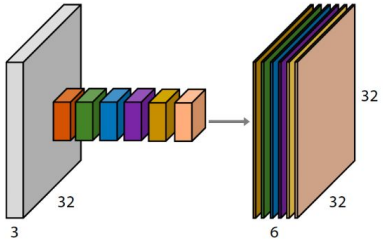
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

e.g. IN, MPNN, GraphNet

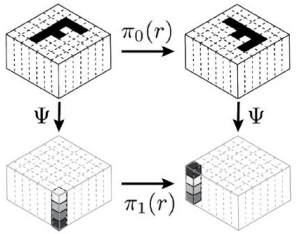
Architectures of interest



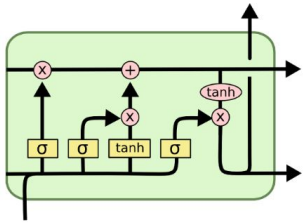
Perceptrons
Function regularity



CNNs
Translation

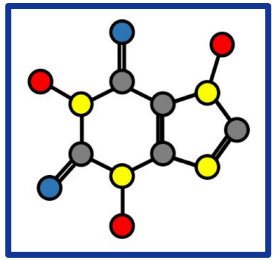


Group-CNNs
Translation+Rotation,
Global groups



LSTMs
Time warping

DeepSets / Transformers
Permutation



GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

DeepMind

6

Permutation equivariant NNs are GNNs

What to do when there is **no** graph?

- Before we go beyond permutation equivariance, a brief note on how we can use the GDL blueprint to immediately relate permutation equivariant NNs as **special cases** of GNNs

- So far, we've assumed something (seemingly) very innocent: that *the **graph** is given to us!*

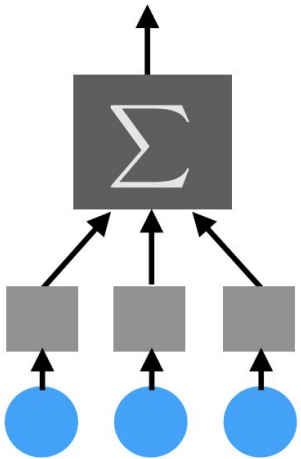
What to do when there is no graph?

- Before we go beyond permutation equivariance, a brief note on how we can use the GDL blueprint to immediately relate permutation equivariant NNs as **special cases** of GNNs
- So far, we've assumed something (seemingly) very innocent: that *the **graph** is given to us!*
- In practice, the given graph may often be *suboptimal* for the task we're solving
  - For various connectivity querying on graphs, maintaining the right set of edges can make a difference between **linear-time** and (amortised) **constant-time** complexity!
- Taken to the extreme: *what to do when there is **no** graph?*
  - Assume we're given a node feature matrix, but no adjacency
- Let's briefly cover two "special case" solutions...

Option 1: Assume *no* edges

Deep Sets (Zaheer *et al.*, NeurIPS'17)

$$\mathcal{N}_i = \{i\}$$



$$\mathbf{h}_i = \psi(\mathbf{x}_i)$$

No edges (*set*)

Option 2: Assume *all* edges

$$\mathcal{N}_i = \mathcal{V}$$

Interaction Nets (Battaglia *et al.*, NeurIPS'16)

Relational Nets (Santoro *et al.*, NeurIPS'17)

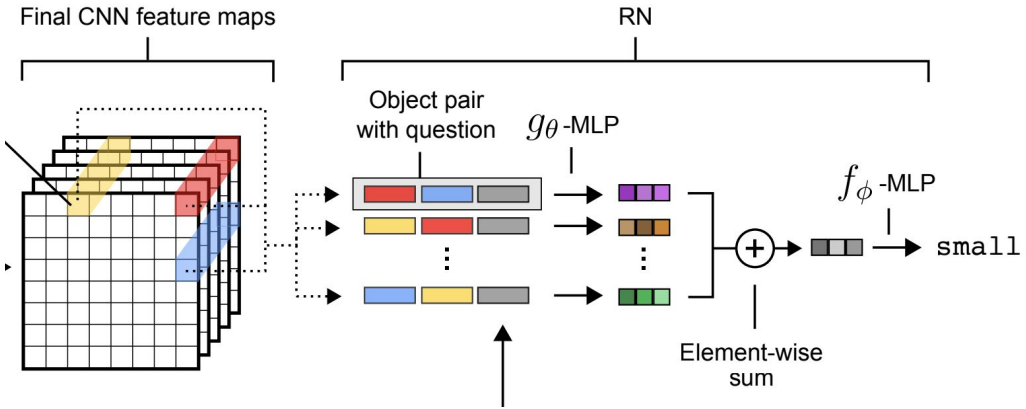
Let the GNN decide which edges matter!

Using conv-GNNs no longer makes sense.

If we use **attentional** GNNs we recover:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{V}} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

Does this look *familiar*?



All edges (*fully-connected graph*)

A note on Transformers

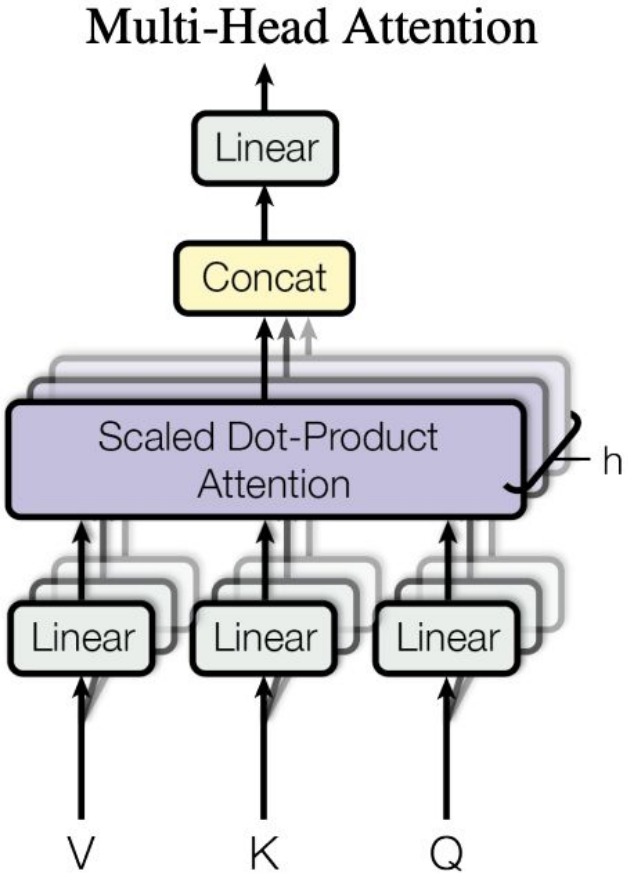
Transformers **are** Graph Neural Networks!

- Fully-connected graph
- Attentional flavour

The sequential structural information is injected through the **positional embeddings**. Dropping them yields a fully-connected GAT model.

Attention can be seen as inferring **soft adjacency**.

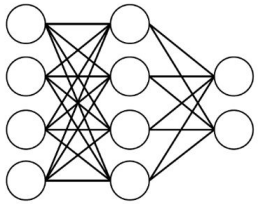
See Joshi (The Gradient; 2020).



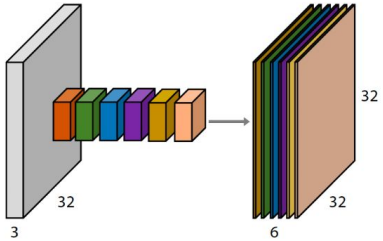
Remark: the "truth" likely lies *in between*

- Empty graph ignores a potential **wealth** of information
- Full graph can be hard to scale (*quadratic* complexity, large neighbourhoods)
- Ideally, we want to **infer** the adjacency matrix **A**, then use it as **edges** for a GNN!
  - Commonly termed *"latent graph inference"*.
  - Choosing edges is a **discrete decision** -- inherently hard to backpropagate!
- Out of scope for this lecture. Some interesting pointers include:
  - Neural Relational Inference (Kipf, Fetaya *et al.*, ICML'18)
  - Dynamic Graph CNN (Wang *et al.*, ACM TOG'18)
  - Differentiable Graph Module (Kazi *et al.*, MICCAI'20)
  - Pointer Graph Networks (Veličković *et al.*, NeurIPS'20)

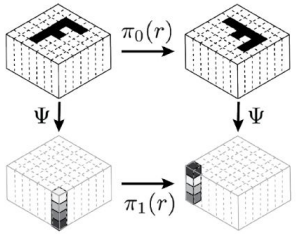
Architectures of interest



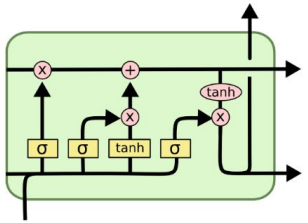
Perceptrons
Function regularity



CNNs
Translation

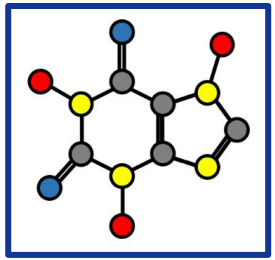


Group-CNNs
Translation+Rotation,
Global groups



LSTMs
Time warping

DeepSets / Transformers
Permutation



GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

DeepMind

7

The Fourier connection: CNNs and Spectral GNNs

Look to the Fourier transform

- The **convolution theorem** defines a very attractive identity:

$$(x \star y)(\xi) = \hat{x}(\xi) \cdot \hat{y}(\xi) \qquad \hat{x}(\xi) = \int_{-\infty}^{+\infty} x(u) e^{-\mathrm{i}\xi u} du$$

"convolution in the time domain is multiplication in the frequency domain"

- This could give us a 'detour' to defining convolutions on graphs
  - Pointwise multiplication is **easy**!
  - But what are the 'domains' in this case?

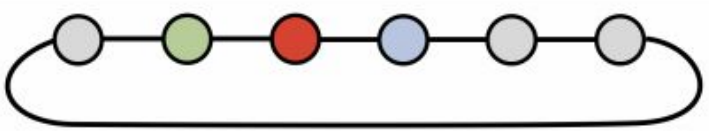
- We will first see how graphs arise in ***discrete*** sequences (grids).

What's changed?

- Grids (e.g. images, text, speech) can still be seen as a **graph**
  - Pixels connected to immediate neighbours in the grid

- What would a graph neural network look like in this case?

- Simplifying assumption: **cyclical** grids



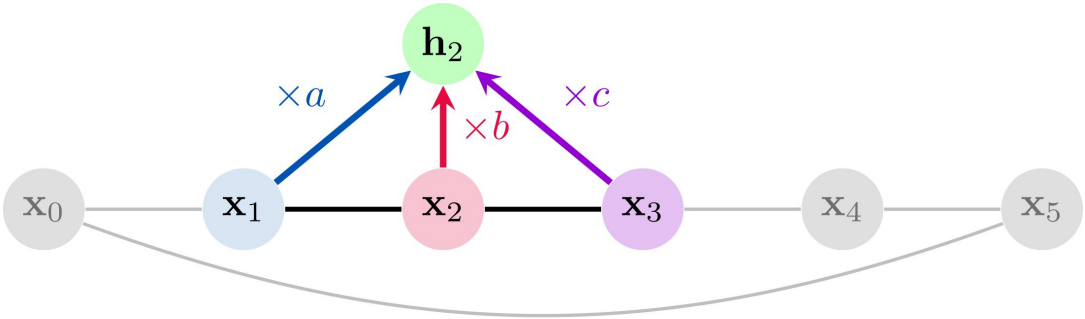
- Now every node has *identical* structure and degree
  - Allows us to more directly define a *convolution* over a neighbourhood

- Also, **coordinates matter**: we get a much stronger ***translation*** equivariance requirement!

Rethinking the convolution on *sequences*

*for easier handling of boundary conditions

- We can imagine a sequence as a *cyclical **grid** graph*, and a **convolution** over it:



- NB this defines a **circulant** matrix **C**([*b*, *c*, 0, 0, ..., 0, *a*]) s.t. **H** = f(**X**) = **CX**

$$f(\mathbf{X}) = \begin{bmatrix} b & c & & & a \\ a & b & c & & \\ & \ddots & \ddots & \ddots & \\ & & a & b & c \\ c & & & a & b \end{bmatrix} \begin{bmatrix} — & \mathbf{x}_0 & — \\ — & \mathbf{x}_1 & — \\ & \vdots & \\ — & \mathbf{x}_{n-2} & — \\ — & \mathbf{x}_{n-1} & — \end{bmatrix}$$

Properties of circulants, and their eigenvectors

- Circulant matrices **commute**!
  - That is, **C**(**v**)**C**(**w**) = **C**(**w**)**C**(**v**), for *any* parameter vectors **v**, **w**.

- Matrices that commute are **jointly diagonalisable**.
  - That is, the eigenvectors of one are eigenvectors of *all* of them!

- Conveniently, the eigenvectors of circulants are the *discrete Fourier basis*

$$\boldsymbol{\phi}_\ell = \frac{1}{\sqrt{n}} \left(1, e^{\frac{2\pi \mathrm{i} \ell}{n}}, e^{\frac{4\pi \mathrm{i} \ell}{n}}, \ldots, e^{\frac{2\pi \mathrm{i}(n-1)\ell}{n}}\right)^\top, \qquad \ell = 0, 1, \ldots, n-1$$

- This can be easily computed by studying **C**([0, 1, 0, 0, 0, ...]), which is the **shift** matrix.

The DFT and the convolution theorem

- If we stack these Fourier basis vectors into a matrix: $\mathbf{\Phi} = (\phi_0, \ldots, \phi_{n-1})$
  - We recover the *discrete Fourier transform* (DFT), as multiplication by $\mathbf{\Phi}^*$. (conjugate transpose)

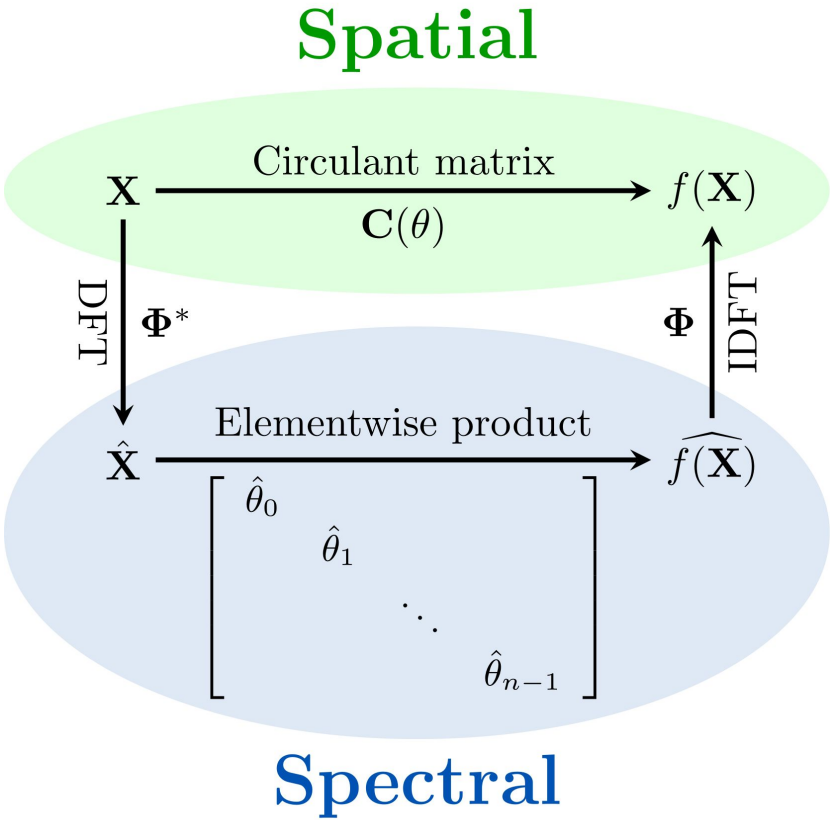
- We can now eigendecompose *any* circulant as $\mathbf{C}(\boldsymbol{\theta}) = \mathbf{\Phi\Lambda\Phi}^*$
  - Where $\mathbf{\Lambda}$ is a diagonal matrix of its eigenvalues, $\hat{\boldsymbol{\theta}}$

- The convolution theorem naturally follows:

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

- Now, as long as we know $\mathbf{\Phi}$, we can express our convolution using $\hat{\boldsymbol{\theta}}$ rather than $\boldsymbol{\theta}$

What we have covered so far



Key idea: we don't **need** to know the circulant if we know its eigenvalues!

Credits to **Michael Bronstein**

What about *graphs*?

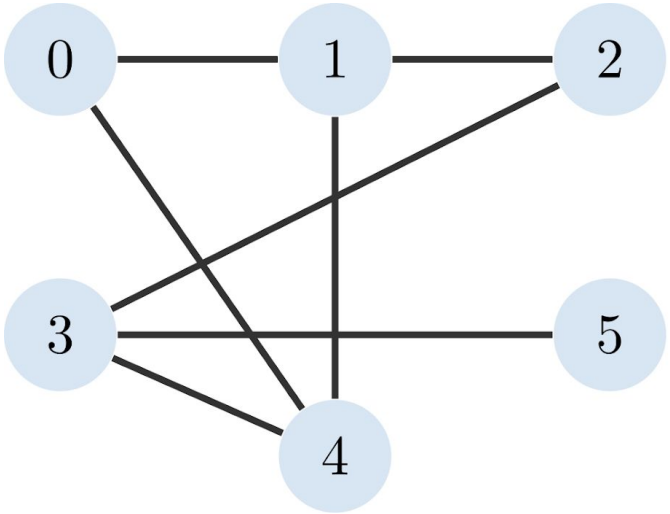
- On graphs, convolutions of interest need to be more generic than circulants.
  - But we can still use the concept of **joint eigenbases**!
  - If we know a "graph Fourier basis", $\mathbf{\Phi}$, we can only focus on learning the eigenvalues.

- For grids, we wanted our convolutions to commute with *shifts*.
  - We can think of the shift matrix as an **adjacency matrix** of the grid
  - This generalises to non-grids!
  - For the grid convolution on n nodes, $\mathbf{\Phi}$ was always the same (n-way DFT).
  - Now **every graph** will have its own $\mathbf{\Phi}$!

- Want our convolution to commute with **A**, but we cannot always eigendecompose **A**!

- Instead, use the **graph Laplacian matrix**, $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where **D** is the degree matrix.
  - Captures all adjacency properties in mathematically convenient way!

Example Laplacian



$$\mathbf{L} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Graph Fourier Transform

- Assuming undirected graphs, **L** is:
  - **Symmetric** ($\mathbf{L}^T = \mathbf{L}$)
  - **Positive semi-definite** ($\mathbf{x}^T\mathbf{L}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^{|V|}$)
  - This means we will be able to *eigendecompose* it!

- This allows us to re-express $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^*$, as before.
  - Changing the eigenvalues in $\mathbf{\Lambda}$ expresses ***any*** operation that commutes with **L**.
  - Commonly referred to as the **graph Fourier transform** (Bruna *et al.*, ICLR'14)

- Now, to convolve with some feature matrix **X**, do as follows (the diagonal can be ***learnable***):

$$f(\mathbf{X}) = \mathbf{\Phi} \begin{bmatrix} \hat{\theta}_0 & & \\ & \ddots & \\ & & \hat{\theta}_{n-1} \end{bmatrix} \mathbf{\Phi}^*\mathbf{X}$$

Spectral GNNs in practice

- However, directly learning the eigenvalues is typically inappropriate:
  - Not **localised**, doesn't **generalise** to other graphs, computationally **expensive**, etc.

- Instead, a common solution is to make the eigenvalues related to $\mathbf{\Lambda}$, the eigenvalues of **L**
  - Commonly by a degree-*k* polynomial function, p_k
  - Yielding $f(\mathbf{x}) = \mathbf{\Phi} p_k(\mathbf{\Lambda}) \mathbf{\Phi}^* \mathbf{x} = p_k(\boldsymbol{L})\mathbf{x}$
  - Popular choices include:
    - *Cubic splines* (Bruna *et al.*, ICLR'14)
    - *Chebyshev polynomials* (Defferrard *et al.*, NeurIPS'16)
    - *Cayley polynomials* (Levie *et al.*, Trans. Sig. Proc.'18)

- **NB** by using a polynomial in **L**, we have defined a **conv-GNN**!
  - With coefficients defined by $c_{ij} = (p_k(\mathbf{L}))_{ij}$
  - Most efficient spectral approaches *"spatialise"* themselves in similar ways
  - The "spatial-spectral" divide is often ***not really a divide***!

The Transformer positional encodings and beyond

- Lastly, another look at Transformers.

- Transformers signal that the input is a **sequence** of words by using *positional embeddings*
  - Sines/cosines sampled depending on position

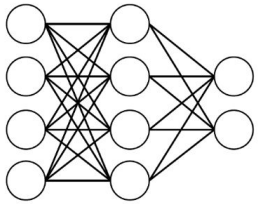
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- **Very** similar to the DFT eigenvectors!

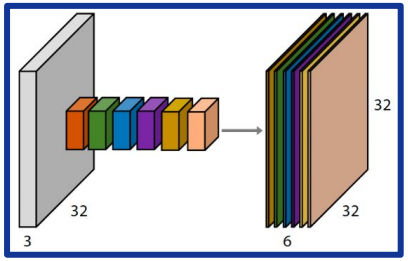
- Positional embeddings could hence be interpreted as eigenvectors of the grid graph
  - Which is the only assumed 'underlying' connectivity between the words

- We can use this idea to run Transformers over *general* graph structures!
  - Just feed some eigenvectors of the graph Laplacian (columns of $\mathbf{\Phi}$)
  - See the ***Graph Transformer*** from Dwivedi & Bresson (2021)
  - Rapidly emerging area of research!

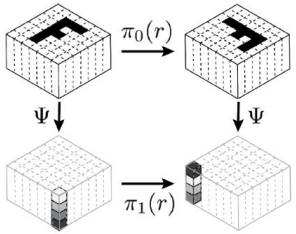
Architectures of interest



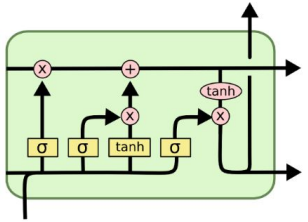
Perceptrons
Function regularity



CNNs
Translation

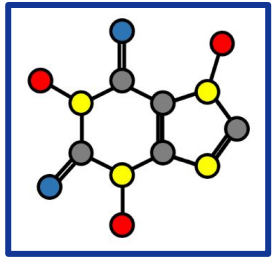


Group-CNNs
Translation+Rotation,
Global groups



LSTMs
Time warping

DeepSets / Transformers
Permutation



GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

DeepMind

8 The Group connection: Spherical CNNs

Convolutions in the Euclidean domain

*NB all our findings will be applicable to discretisations

- Now consider the *continuous* Euclidean domain $\Omega = \mathbb{R}$, where convolution takes the form:

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

$$(T_v x)(u) = x(u-v)$$

- The output of the **grid** convolution is another function on the grid
  - But this is **not** the case for **every** domain!

- Now we can see how to generalise the convolution to more general $\mathfrak{G}$

Group convolutions

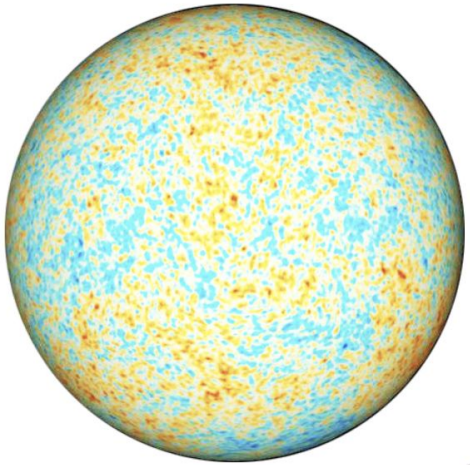
- We can define **inner products** on general domains **Ω**: $\langle x, \psi \rangle = \int_{\Omega} x(u)\psi(u)\mathrm{d}u$

$$(x \star \psi)(u) = \langle x, T_u \psi \rangle = \int_{\mathbb{R}} x(v)\psi(u+v)\mathrm{d}v$$ **Grids**

$$(x \star \psi)(\mathfrak{g}) = \langle x, \rho(\mathfrak{g})\psi \rangle = \int_{\Omega} x(u)\psi(\mathfrak{g}^{-1}u)\mathrm{d}u$$ **Groups**

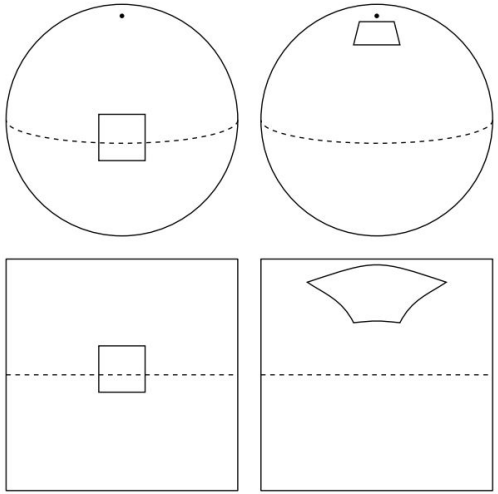
- **NB**: the group convolution is, generally, a function over elements of $\mathfrak{G}$!!!
  - It was also a real-input function under the translation group…
  - …because the group itself is also $\mathfrak{G} = \mathbb{R}$ (scalar **shifts**!!!)

Example: **Spherical** convolution



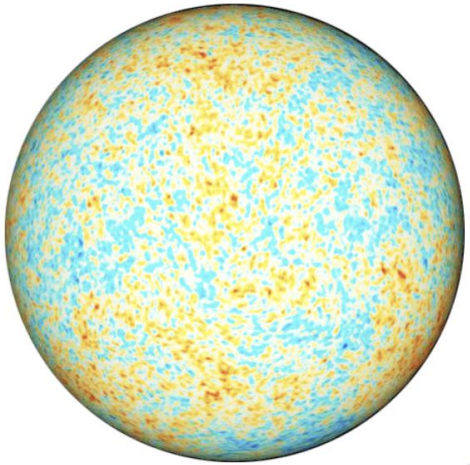
- Consider signal defined on a **sphere**, $\mathbf{\Omega} = S^2$
  - Very relevant, e.g. for earth maps, astrophysics, …

- We want to be **rotation equivariant**
  - This means using the rotation group $\mathfrak{G} = SO(3)$
  - Image CNNs cannot support this!



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

Example: **Spherical** convolution



- Consider signal defined on a **sphere**, $\mathbf{\Omega} = S^2$
  - Very relevant, e.g. for earth maps, astrophysics, …
- We want to be **rotation equivariant**
  - This means using the rotation group $\mathfrak{G} = SO(3)$
- We can represent the points on a sphere with 3-dim unit vectors, **u**
  - The **group action** (rotation) transforms it with a 3 x 3 orthogonal matrix, **R**
- We recover our spherical convolution:

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

Example: SO(3) convolution

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

- The output of a spherical convolution is a function over **all 3D rotations** in SO(3)
  - So, to "stack more layers", we need to define a **convolution over $\mathfrak{Omega}$ = $\mathfrak{G}$ = SO(3)**
- Luckily, our blueprint still works: we can define a $\mathfrak{G}$-action over elements of $\mathfrak{G}$ by function composition: $(\mathfrak{g}, \mathfrak{h}) \mapsto \mathfrak{g}\mathfrak{h}$
- The corresponding group action can be defined as follows: $(\rho(\mathfrak{g})x)(\mathfrak{h}) = x(\mathfrak{g}^{-1}\mathfrak{h})$
- Consequently, we can build $\mathfrak{G}$-equivariant layers over $\mathfrak{G}$!
- Start with spherical convolution at the input layer, then **stack** SO(3)-convolutions!

Example: SO(3) convolution, *cont'd*

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

- SO(3) group actions defined by 3x3 rotation matrices **R**
- Hence, group action $(\rho(\mathfrak{g})x)(\mathfrak{h}) = x(\mathfrak{g}^{-1}\mathfrak{h})$ is expressible as x(**R**$^{-1}$**Q**)
  - for some two rotation matrices **R** and **Q**.
- Recall the expression for the group convolution: $(x \star \psi)(\mathfrak{g}) = \langle x, \rho(\mathfrak{g})\psi \rangle = \int_{\Omega} x(u)\psi(\mathfrak{g}^{-1}u)\mathrm{d}u$
- Putting it all together, we obtain the following two-layer convolution over spheres:

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

- This forms the essence of **Spherical CNNs** (Cohen *et al.*, ICLR'18)

Relationship to GNNs

- The spherical CNN can still be related to GNNs
  - In practice, the sphere is *discretised* and the filter ψ is *local*
  - The non-zero elements of ψ are the neighbourhood of point **u**
  - The integration is akin to message passing! (as many points will have zero product)

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

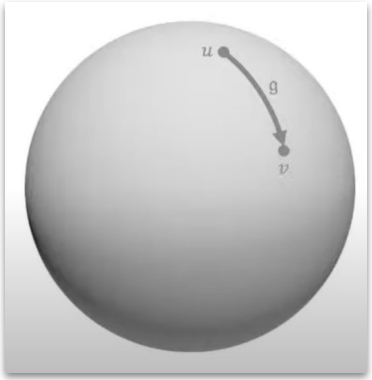
- Further, the $\mathfrak{G}$-convolution described here works over **any** global symmetry group $\mathfrak{G}$

Caveat 1: Tractability

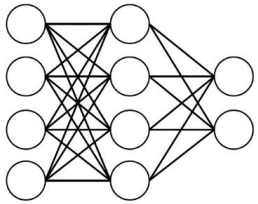
- The $\mathfrak{G}$-convolution described here works over **any** symmetry group $\mathfrak{G}$
- But it is only **tractable** for very small groups
  - e.g. SO(3) was describable with a 3x3 orthogonal matrices
- It may be tempting to apply this idea to graphs
  - But the permutation group Σ_n has **n!** entries to maintain
- This hints at existence of “graph convolutions” not captured by our ‘flavours’
  - (though it likely captures the most tractable ones :))
- Some of these convolutions may tradeoff expressivity for stability / complexity
  - Suggests a “way out” of the Weisfeiler-Lehman expressivity limit…

Caveat 2: Homogeneity

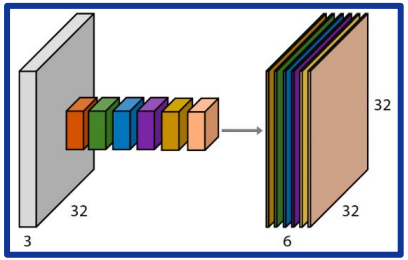
- Both the spherical and the circular grid domains are **homogeneous**
- For any two points $u, v \in \Omega$, there exists some $g \in \mathfrak{G}$ s.t. $g.u = v$
- In a sense, “all points look the same”
  - This does **not** hold for graphs!
- This allowed ‘sliding’ filters across Ω to build convolutions
- What can we do in the more general case?



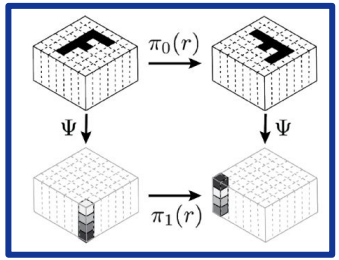
Architectures of interest



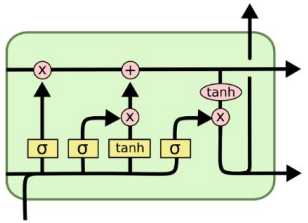
Perceptrons
Function regularity



CNNs
Translation

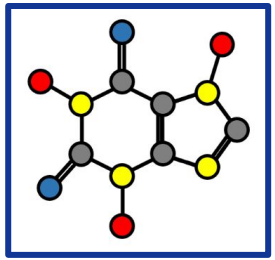


Group-CNNs
Translation+Rotation,
Global groups



LSTMs
Time warping

DeepSets / Transformers
Permutation



GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

DeepMind

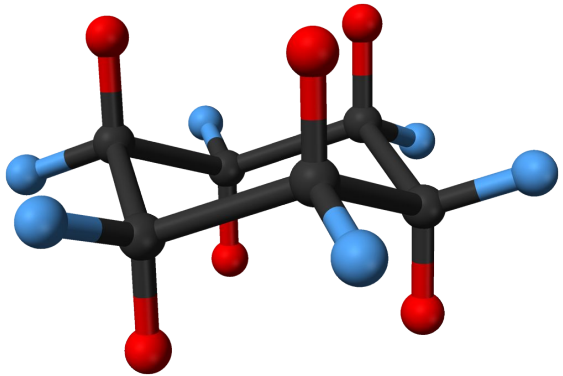
9 Geometric Graphs, Geodesics and Gauges

Geometric graphs

- Thus far, we have assumed our graphs to be a discrete, unordered, set of nodes and edges

- In many cases, this is not the entire story!
  - The graph, in fact, may often be endowed with some spatial geometry
  - It will be useful for us to **exploit** this geometry!

- Molecules are a classical case (with their three-dimensional *conformers*)



Learning over geometric graphs

- Simplified setup: nodes endowed with **features**, $\mathbf{f}_u$ and **coordinates** $\mathbf{x}_u \in \mathbb{R}^3$
- An **equivariant message passing** layer transforms them separately
  - Yielding updated features $\mathbf{f'}_u$ and coordinates $\mathbf{x'}_u$
- We can now express a group of symmetries $\mathfrak{G}$ we would like to be resistant to
  - In the case of molecules, a standard group is the Euclidean group, E(3)
  - Roto-translations and reflections
- For any 3D orthogonal matrix **R** and translation vector **b**, we define a group action $\mathfrak{g} \in$ E(3):

$$\rho(\mathfrak{g})\mathbf{x} = \mathbf{R}\mathbf{x} + \mathbf{b}$$

and applying them to coordinates typically should not affect how features are processed!

E(n)-equivariant GNNs

- As for permutation equivariance, there exist **many** GNNs that obey E(n)-equivariance

- One elegant solution was exposed by Satorras *et al.* (ICML'21):

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

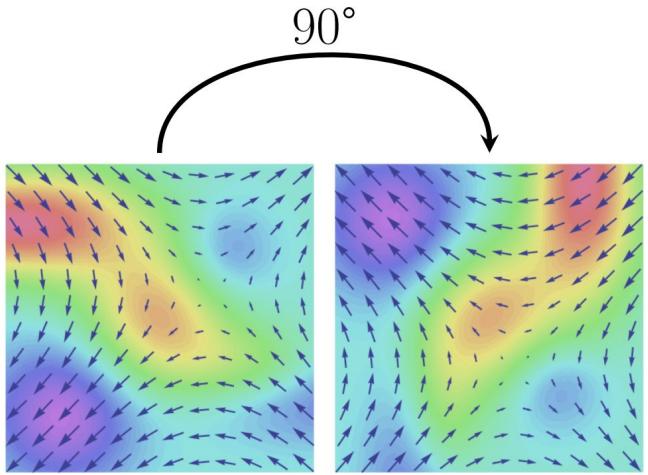
- The actions of E(n) do not change **distance** between nodes (they are *isometric*)
  - Hence if $\mathbf{x}_u \leftarrow \mathbf{R}\mathbf{x}_u + \mathbf{b}$...
  - $\mathbf{f}'_u$ does not change, and $\mathbf{x}'_u \leftarrow \mathbf{R}\mathbf{x}'_u + \mathbf{b}$, as expected!

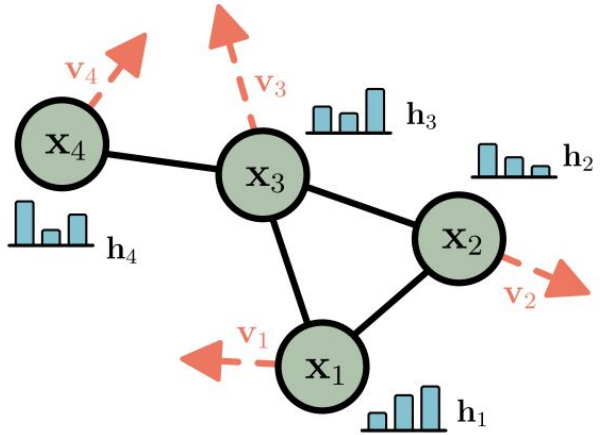
- Hence, this layer is E(n)-equivariant over scalar features $\mathbf{f}_u$

Vector-structured features

- However, what if some of the node features ($\mathbf{f}_u$) depend on the geometry?
  - e.g. they could be vector **forces**
  - Rotating the molecule should rotate these vectors too!

- The model on the previous slide does **not** take this into account!





Vector-structured features

- However, what if some of the node features ($\mathbf{f}_u$) depend on the geometry?
  - e.g. they could be vector **forces**
  - Rotating the molecule should rotate these vectors too!

- The model on the previous slide does **not** take this into account!

- Satorras *et al.* do propose a variant of their model that works with vectors:

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

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{v}_i^{l+1}$$

- But the issue will keep re-appearing as we “tensor up” our features!
  - Is there a way to talk about the general “set of solutions” for E(n)-equivariance?

Towards a general solution: Tensor Field Networks

- We actually can factorise the group action of SE(3) on inputs of *any* tensor order *l*!

$$\rho(g) = \mathbf{Q}^\top \left[\bigoplus_\ell \mathbf{D}_\ell(g) \right] \mathbf{Q}$$

where $\mathbf{D}_l$ are the Wigner D-matrices, and $\mathbf{Q}$ is a $\mathfrak{g}$-specific change-of-basis matrix.

- This can be exploited to write a generic SE(3)-equivariant model (the TFN)

$$\mathbf{f}_{\text{out},i}^\ell = \sum_{k\geq 0} \underbrace{\int \mathbf{W}^{\ell k}(\mathbf{x}' - \mathbf{x}_i)\mathbf{f}_{\text{in}}^k(\mathbf{x}')\,\mathrm{d}\mathbf{x}'}_{k\to\ell \text{ convolution}} = \sum_{k\geq 0}\sum_{j=1}^{n} \underbrace{\mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^k}_{\text{node } j \,\to \text{ node } i \text{ message}},$$

$$\mathbf{W}^{\ell k}(\mathbf{x}) = \sum_{J=|k-\ell|}^{k+\ell} \varphi_J^{\ell k}(\|\mathbf{x}\|)\mathbf{W}_J^{\ell k}(\mathbf{x}), \qquad \text{where } \mathbf{W}_J^{\ell k}(\mathbf{x}) = \sum_{m=-J}^{J} Y_{Jm}(\mathbf{x}/\|\mathbf{x}\|)\mathbf{Q}_{Jm}^{\ell k}$$

where the matrices $\mathbf{W}_J$ are precomputed based on the above parametrisation

SE(3)-Transformers (Fuchs, Worrall, *et al.*, NeurIPS'20)

Restrict TFNs to only act over neighbourhood N_i. Also adds Transformer attention.

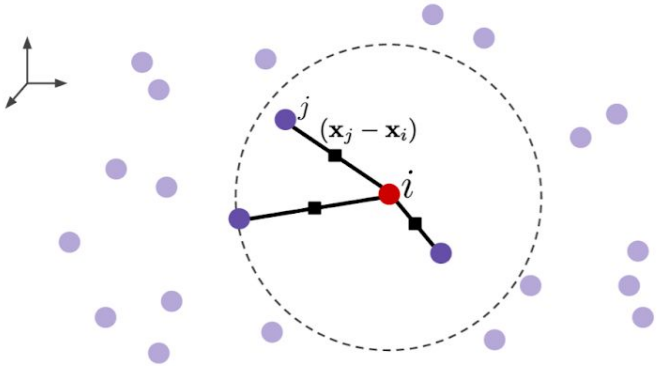
$$\mathbf{f}_{\text{out},i}^{\ell} = \underbrace{\mathbf{W}_V^{\ell\ell}\mathbf{f}_{\text{in},i}^{\ell}}_{\text{③ self-interaction}} + \sum_{k\geq 0}\sum_{j\in\mathcal{N}_i\setminus i} \underbrace{\alpha_{ij}}_{\text{① attention}} \underbrace{\mathbf{W}_V^{\ell k}(\mathbf{x}_j-\mathbf{x}_i)\mathbf{f}_{\text{in},j}^{k}}_{\text{② value message}}.$$

$$\alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_{ij})}{\sum_{j'\in\mathcal{N}_i\setminus i}\exp(\mathbf{q}_i^\top \mathbf{k}_{ij'})}, \quad \mathbf{q}_i = \bigoplus_{\ell\geq 0}\sum_{k\geq 0}\mathbf{W}_Q^{\ell k}\mathbf{f}_{\text{in},i}^{k}, \quad \mathbf{k}_{ij} = \bigoplus_{\ell\geq 0}\sum_{k\geq 0}\mathbf{W}_K^{\ell k}(\mathbf{x}_j-\mathbf{x}_i)\mathbf{f}_{\text{in},j}^{k}.$$

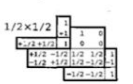
Gives all possible equivariant (attentional) GNNs, but we still need to pre-compute $\mathbf{W}_J$

SE(3)-Transformers (Fuchs, Worrall, *et al.*, NeurIPS'20)

Step 1: Get nearest neighbours and relative positions



Step 2: Get SO(3)-equivariant weight matrices



Clebsch-Gordon Coeff.

Radial Neural Network

Spherical Harmonics

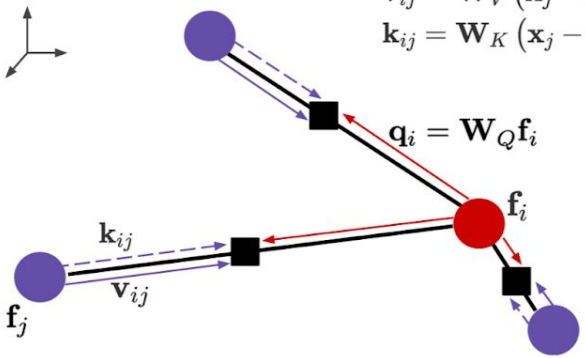
$$\underbrace{\mathbf{Q}_{Jm}^{\ell k} \quad \varphi_J^{\ell k}(||x||) \quad Y_{Jm}\left(\frac{x}{||x||}\right)}$$

Matrix W consists of blocks mapping between degrees

$$\mathbf{W}(x) = \mathbf{W}\left(\left\{\mathbf{Q}_{Jm}^{\ell k},\, \varphi_J^{\ell k}(||x||),\, Y_{Jm}\left(\frac{x}{||x||}\right)\right\}_{J,m,\ell,k}\right)$$

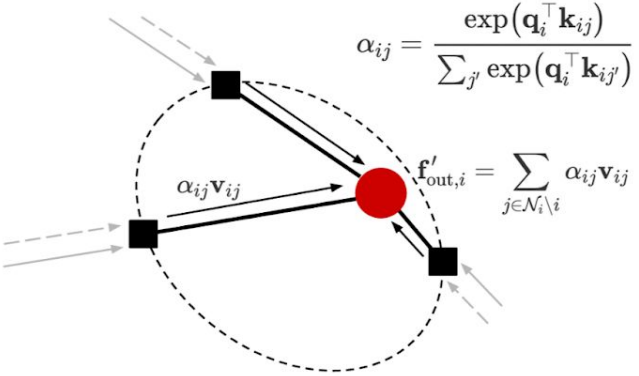
Step 3: Propagate queries, keys, and values to edges

$$\mathbf{v}_{ij} = \mathbf{W}_V\left(\mathbf{x}_j - \mathbf{x}_i\right)\mathbf{f}_j$$
$$\mathbf{k}_{ij} = \mathbf{W}_K\left(\mathbf{x}_j - \mathbf{x}_i\right)\mathbf{f}_j$$



Step 4: Compute attention and aggregate

$$\alpha_{ij} = \frac{\exp\left(\mathbf{q}_i^\top \mathbf{k}_{ij}\right)}{\sum_{j'} \exp\left(\mathbf{q}_i^\top \mathbf{k}_{ij'}\right)}$$

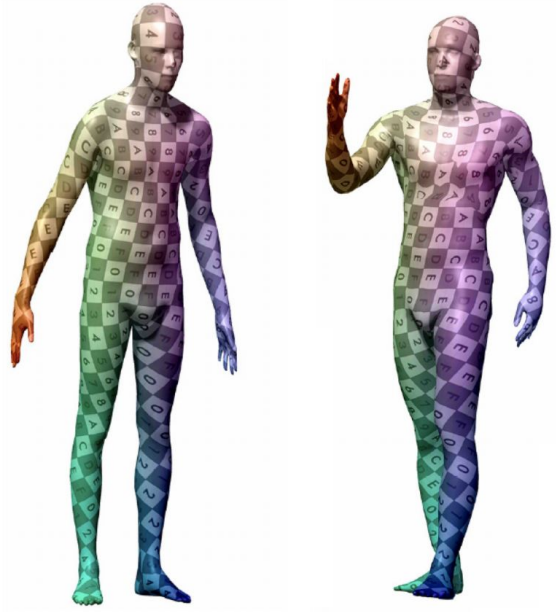


Manifolds

- More generally, we can define **manifold** domains
  - Highly relevant e.g. for computer graphics, protein design, fMRI processing

- In fact, manifold-oriented models end up not at all dissimilar to equivariant message passing!

- Detailed exposition and extended theory are deferred to the draft book

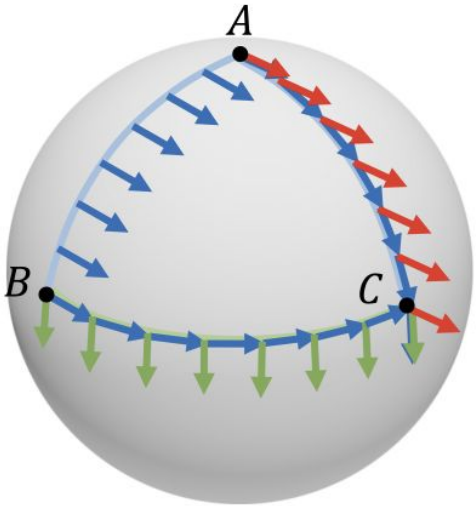


Manifolds

- More generally, we can define **manifold** domains
  - Highly relevant e.g. for computer graphics, protein design, fMRI processing

- In fact, manifold-oriented models end up not at all dissimilar to equivariant message passing!

- Key concept: ***parallel transport***, which allows us to transport tangent vectors along a curve to preserve **direction**
  - NB. It is **path-dependent** (e.g. BC ≠ ABC)



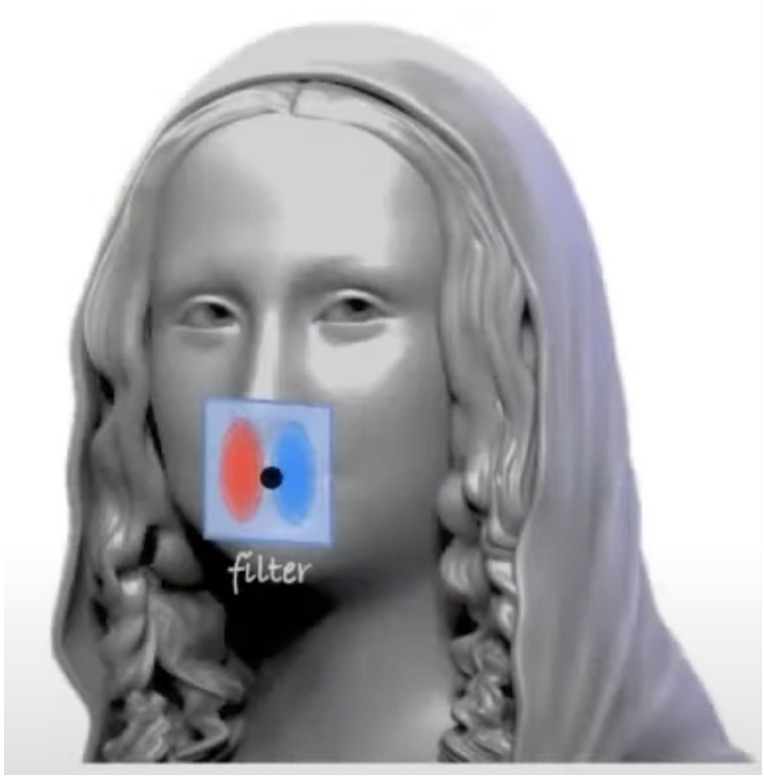
Manifolds

- More generally, we can define **manifold** domains
  - Highly relevant e.g. for computer graphics, protein design, fMRI processing

- In fact, manifold-oriented models end up not at all dissimilar to equivariant message passing!

- Key concept: ***parallel transport***, which allows us to transport tangent vectors along a curve to preserve **direction**
  - NB. It is **path-dependent**

- Now we can define (with some caveats) “sliding” a filter along more general surfaces



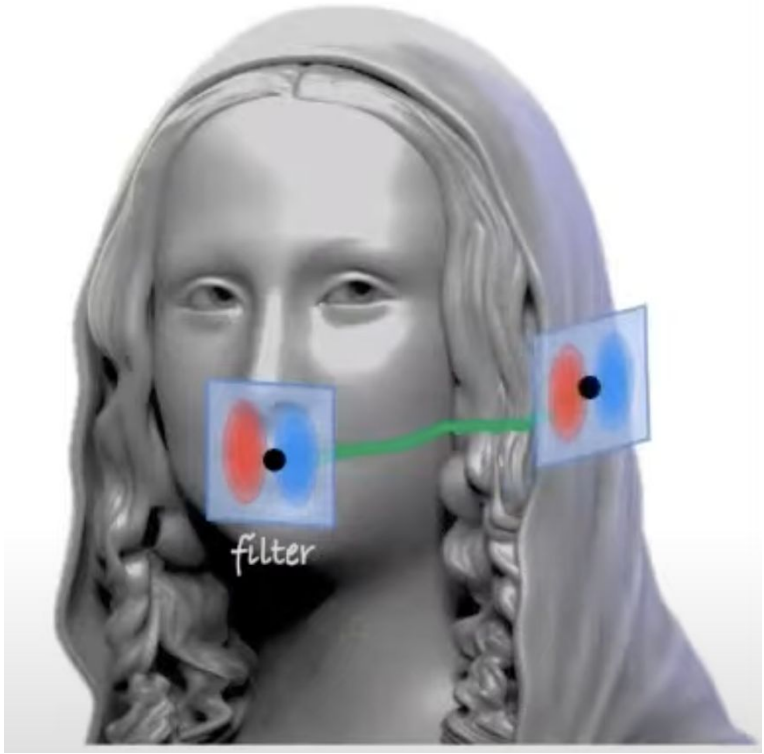
Manifolds

- More generally, we can define **manifold** domains
  - Highly relevant e.g. for computer graphics, protein design, fMRI processing

- In fact, manifold-oriented models end up not at all dissimilar to equivariant message passing!

- Key concept: ***parallel transport***, which allows us to transport tangent vectors along a curve to preserve **direction**
  - NB. It is **path-dependent**

- Now we can define (with some caveats) “sliding” a filter along more general surfaces



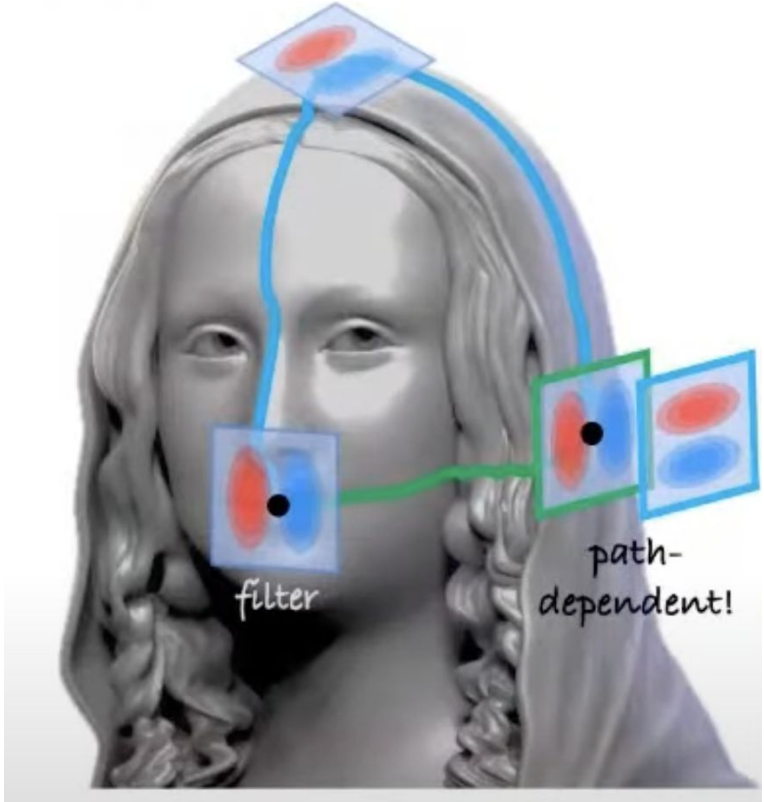
Manifolds

- More generally, we can define **manifold** domains
  - Highly relevant e.g. for computer graphics, protein design, fMRI processing

- In fact, manifold-oriented models end up not at all dissimilar to equivariant message passing!

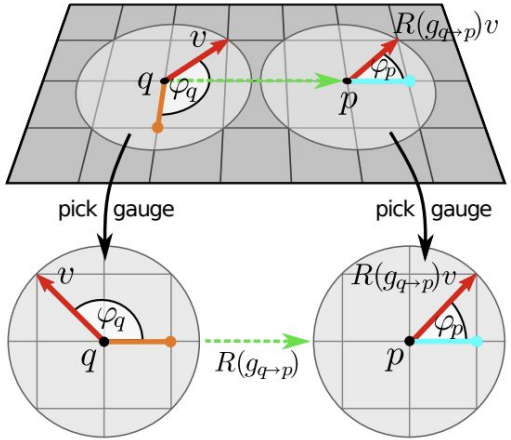
- Key concept: ***parallel transport***, which allows us to transport tangent vectors along a curve to preserve **direction**
  - NB. It is **path-dependent**

- Now we can define (with some caveats) “sliding” a filter along more general surfaces

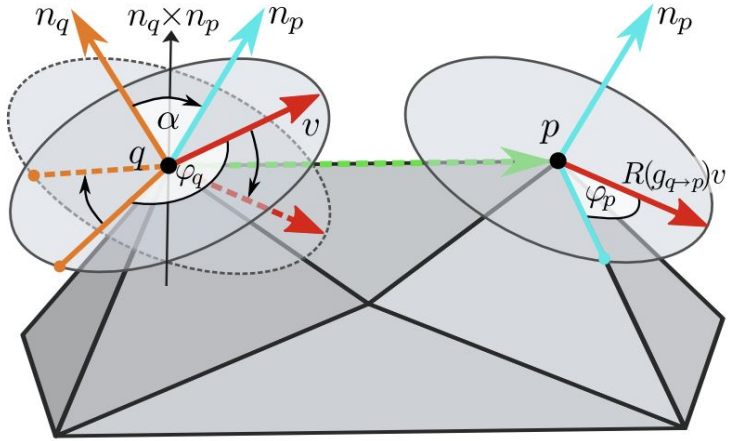


Some architectures recovered from blueprint

- Geodesic CNN (Masci *et al.*, CVPR'15)
- Gauge-equivariant Mesh CNN (de Haan *et al.*, ICLR'21)



(a) Parallel transport on a flat mesh.



(b) Parallel transport along an edge of a general mesh.

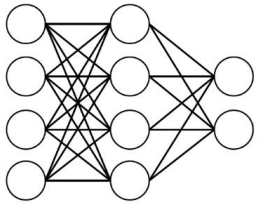
Some architectures recovered from blueprint

- Geodesic CNN (Masci *et al.*, CVPR'15)

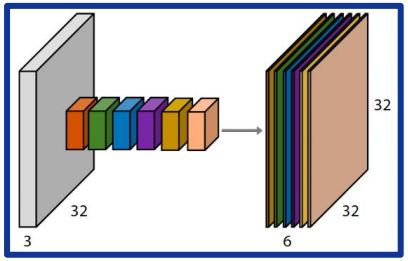
- Gauge-equivariant Mesh CNN (de Haan *et al.*, ICLR'21)
  - GEM-CNNs feel a lot like **message passing**! (with *precomputed* transport matrices)

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

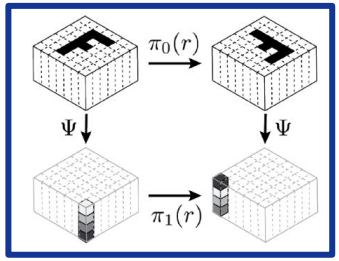
Architectures of interest



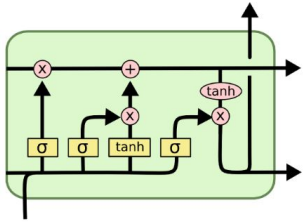
Perceptrons
Function regularity



CNNs
Translation

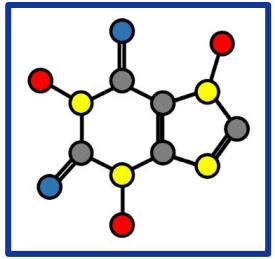


Group-CNNs
Translation+Rotation,
Global groups



LSTMs
Time warping

DeepSets / Transformers
Permutation



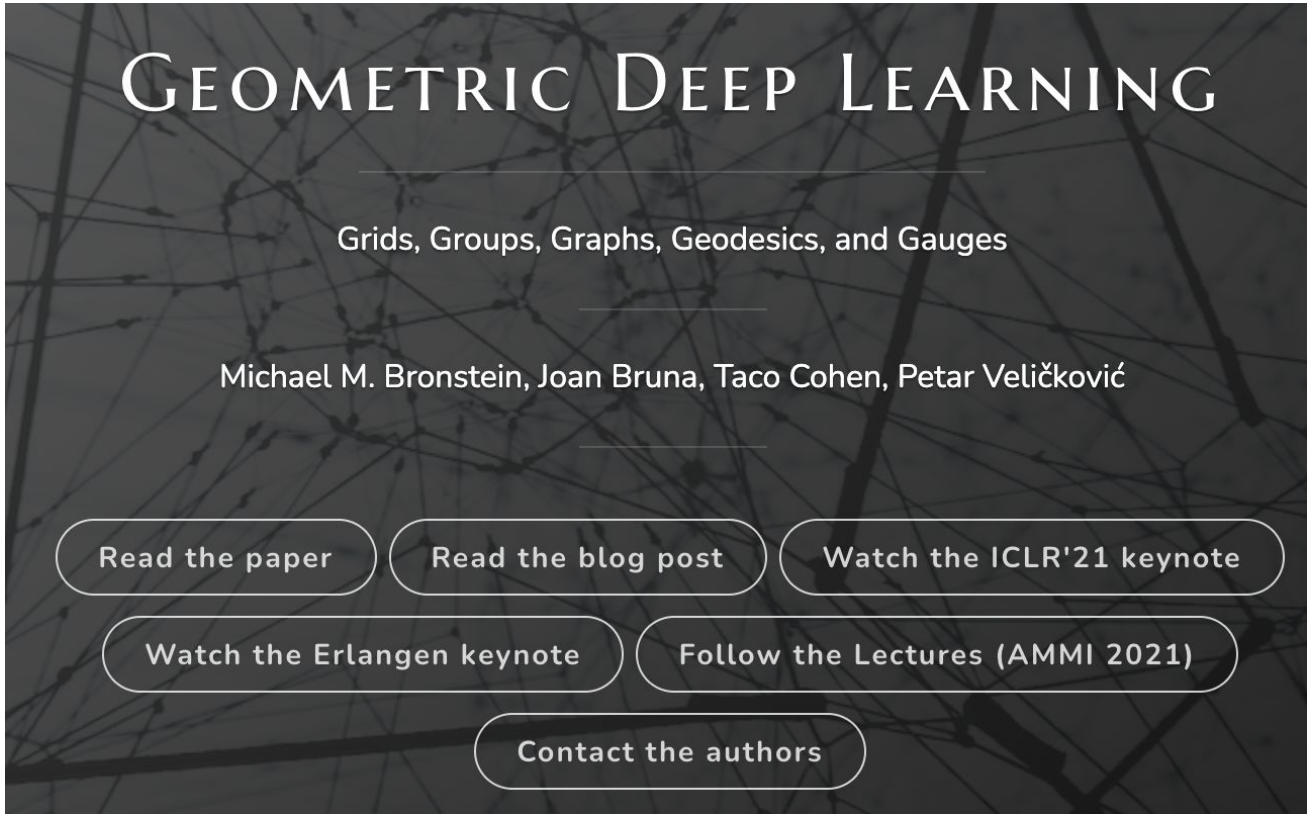
GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

DeepMind

A Further resources

geometricdeeplearning.com



156-page proto book on arXiv

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

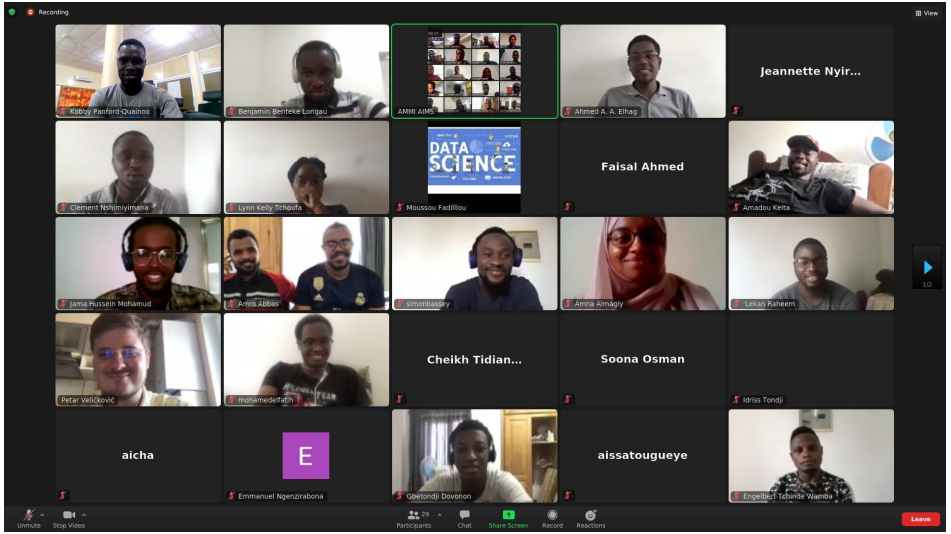
Michael M. Bronstein[1], Joan Bruna[2], Taco Cohen[3], Petar Veličković[4]

May 4, 2021

African Master's in Machine Intelligence course (12h)

All lecture recordings

Lecture 1: *Introduction*	Michael M. Bronstein	Recording	Slides
Lecture 2: *High-Dimensional Learning*	Joan Bruna	Recording	Slides
Lecture 3: *Geometric Priors I*	Taco Cohen	Recording	Slides
Lecture 4: *Geometric Priors II*	Joan Bruna	Recording	Slides
Lecture 5: *Graphs & Sets I*	Petar Veličković	Recording	Slides
Lecture 6: *Graphs & Sets II*	Petar Veličković	Recording	Slides
Lecture 7: *Grids*	Joan Bruna	Recording	Slides
Lecture 8: *Groups*	Taco Cohen	Recording	Slides
Lecture 9: *Geodesics & Manifolds*	Michael M. Bronstein	Recording	Slides
Lecture 10: *Gauges*	Taco Cohen	Recording	Slides
Lecture 11: *Sequences & Time Warping*	Petar Veličković	Recording	Slides
Lecture 12: *Conclusions*	Michael M. Bronstein	Recording	Slides



DeepMind

Thank you!

Questions / Feedback?

petarv@deepmind.com | https://petar-v.com

With many thanks to Joan Bruna, Michael Bronstein and Taco Cohen