GEOMETRIC DEEP LEARNING (**L65**)

Pietro Liò *University of Cambridge*
Petar Veličković *Google DeepMind / University of Cambridge*

Lent Term 2024

CST Part III / MPhil ACS / MPhil MLMI

3. HOW TO BUILD GEOMETRIC NEURAL NETWORKS

Fundamentals of learning on sets, grids and spheres

Pietro Liò

Recap from previous lectures

We have seen why it is a good idea to exploit the *geometry* in data
And how to leverage *groups, representations and invariance* to do so

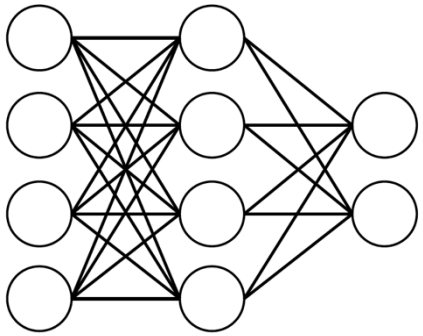
We also surveyed the significance of studying data on *graphs*

We saw several traditional methods for doing so:

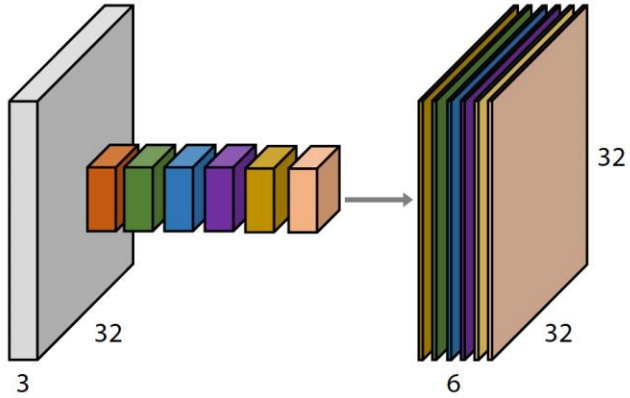
- Network science
- Spectral graph theory
- Classical graph generative models

Today, we will see how to *learn* useful geometric functions.

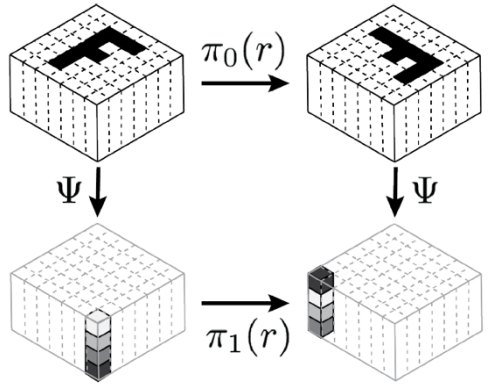
Recall: Architectures of interest



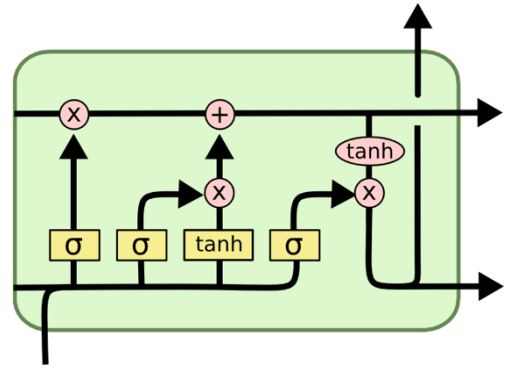
Perceptrons
Function regularity



CNNs
Translation

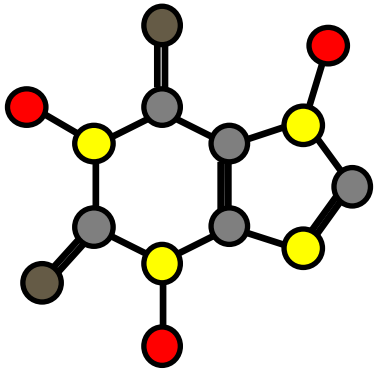


Group-CNNs
Global groups

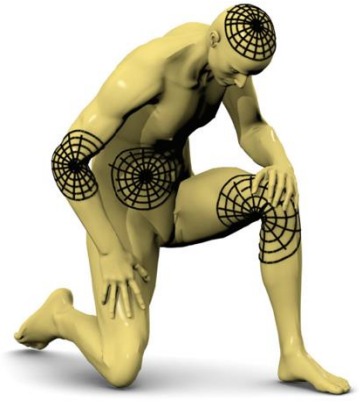


LSTMs
Time warping

Deep Sets / Transformers

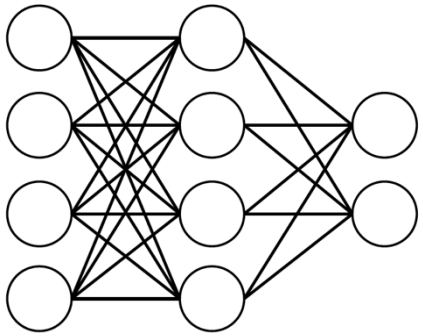


GNNs
Permutation

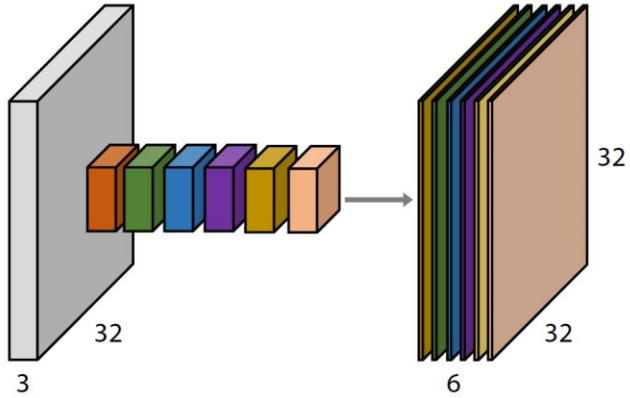


Intrinsic CNNs
Isometry / Gauge choice

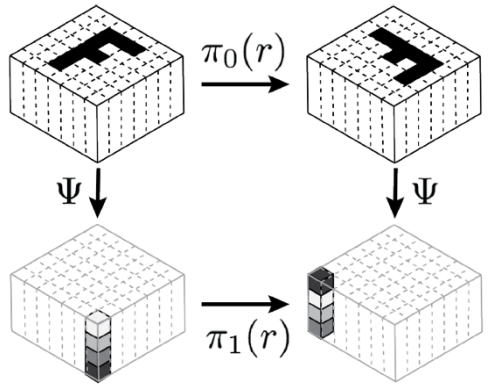
*Our targets for today (**Lecture 3** / **Lecture 4**)*



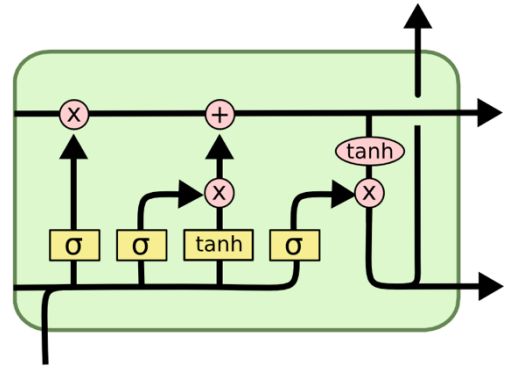
Perceptrons
Function regularity



CNNs
Translation

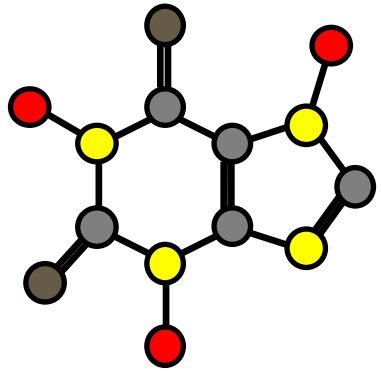


Spherical CNNs
Rotation

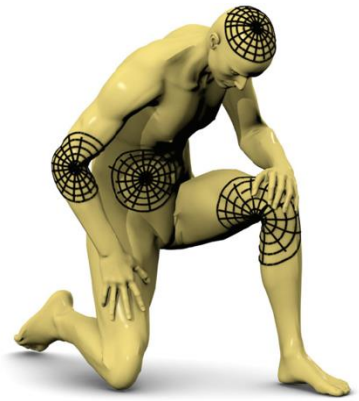


LSTMs
Time warping

Deep Sets
Permutation



GNNs
Permutation



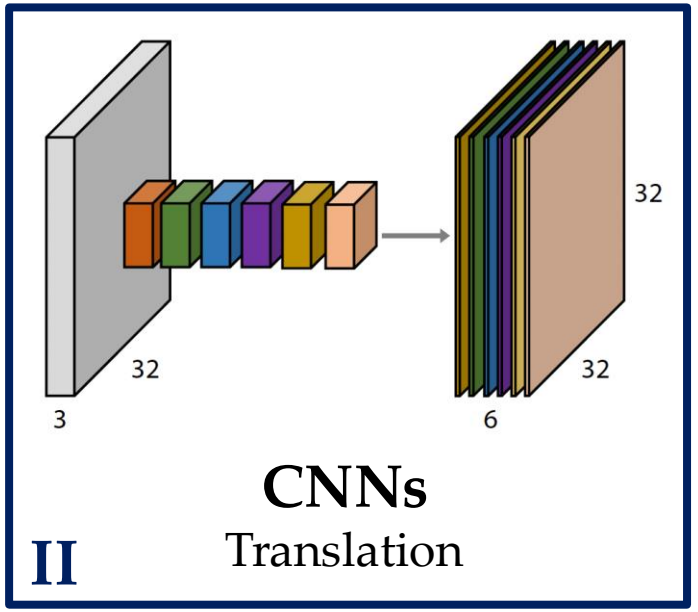
Intrinsic CNNs
Isometry / Gauge choice

Our targets for today

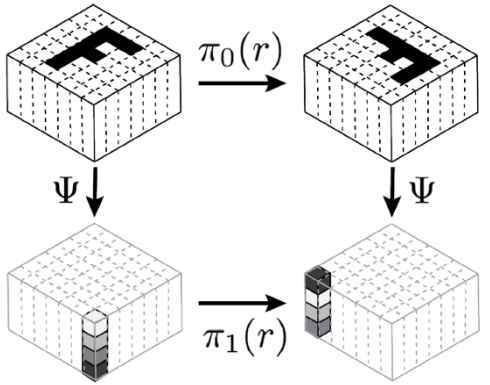
We aim to cover a *vast* landscape of *four* GDL architecture types

In **Lecture 3**, we'll start with one *simple* domain (**sets**) and one *familiar* domain (**grids**), showing immediately how models in that domain can extend to more *exotic* domains (**spheres**)

Then, **Lecture 4** will fully dive into machine learning on **graphs**

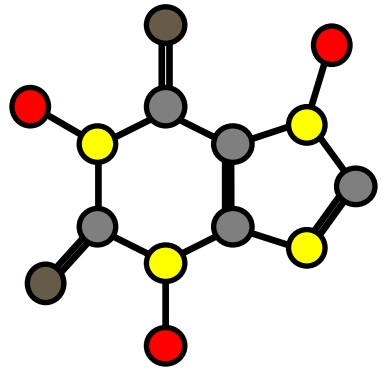


CNNs
II Translation



Spherical CNNs
III Rotation

Deep Sets
I Permutation



GNNs
IV Permutation

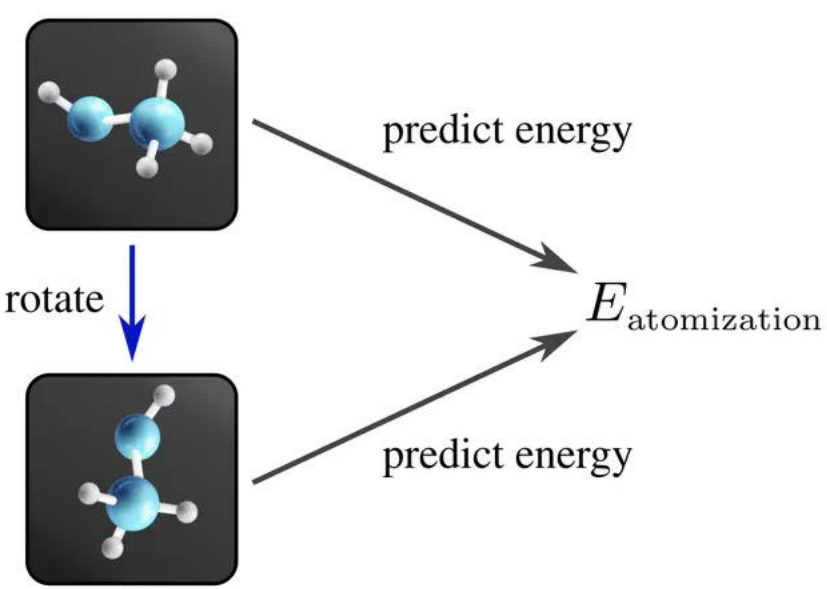
Popular architectures as instances of GDL blueprint

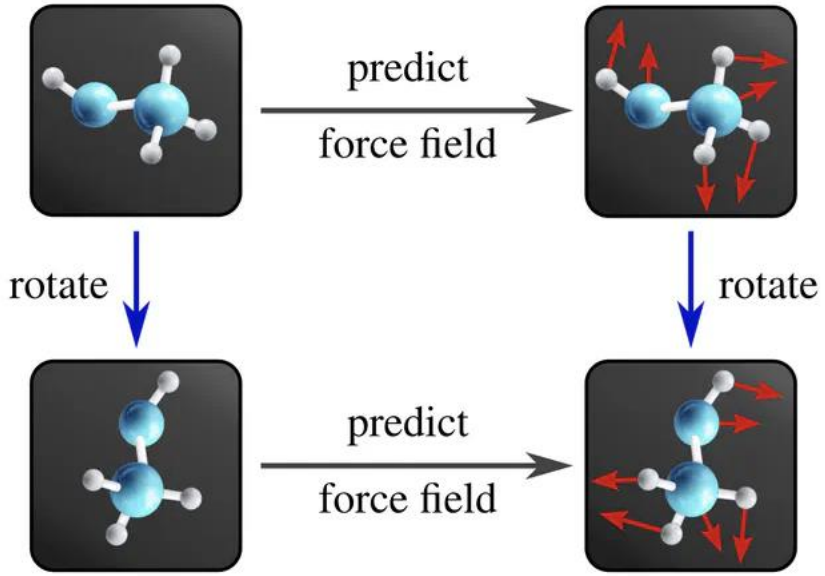
Architecture	**Domain Ω**	**Symmetry Group $\mathfrak{G}$**
CNN	Grid	Translation
Spherical CNN	Sphere / $\mathrm{SO}(3)$	Rotation $\mathrm{SO}(3)$
Mesh CNN	Manifold	Isometry $\mathrm{Iso}(\Omega)$ / Gauge Symmetry $\mathrm{SO}(2)$
GNN	Graph	Permutation Σ_n
Deep Sets	Set	Permutation Σ_n
Transformer	Complete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

Symmetries

- Symmetries are transformations that leave an object *unchanged*
- In ML we care about symmetries of an object, of the parameterization, label function and domain.
- Encoding symmetry can be a powerful way to *restrict* the functions under consideration, allowing models to be learnt more effectively.

Invariant and equivariant properties (quick remind)





An example of an *invariant* task is the prediction of a molecule's *atomization energy*, which does not depend on the molecule's *rotation*.

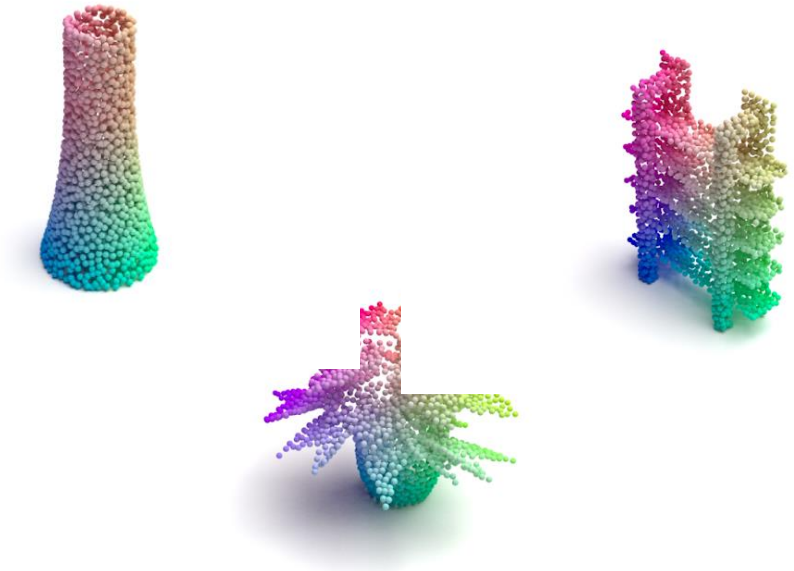
Predicting a *force field* is, on the other hand, an *equivariant* task, since the force vectors are supposed to *rotate together with the molecule*.

Deep Sets

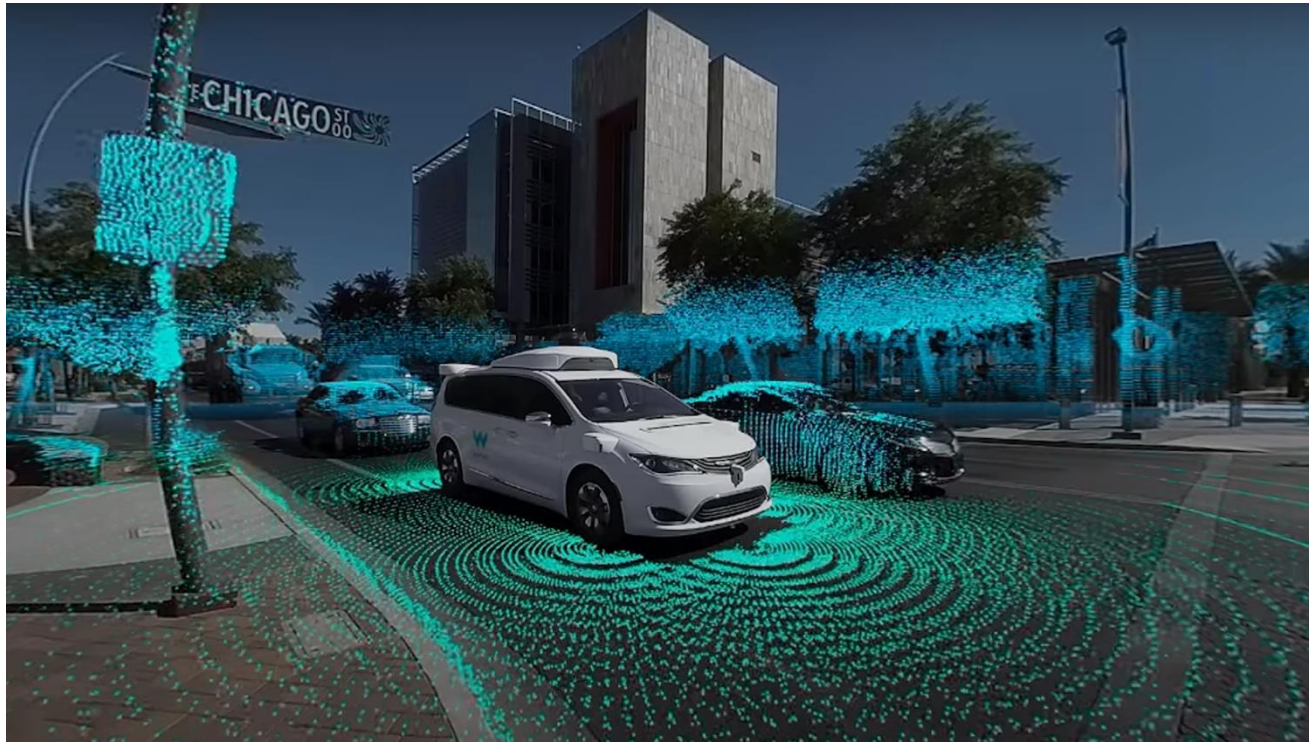
Where do we begin?

We will first look at "*graphs without connections*" (**sets**)

- Arguably the simplest domain to analyse
- Most conclusions will naturally *carry over* to graphs
- Still *very relevant*! (point clouds / LiDAR)

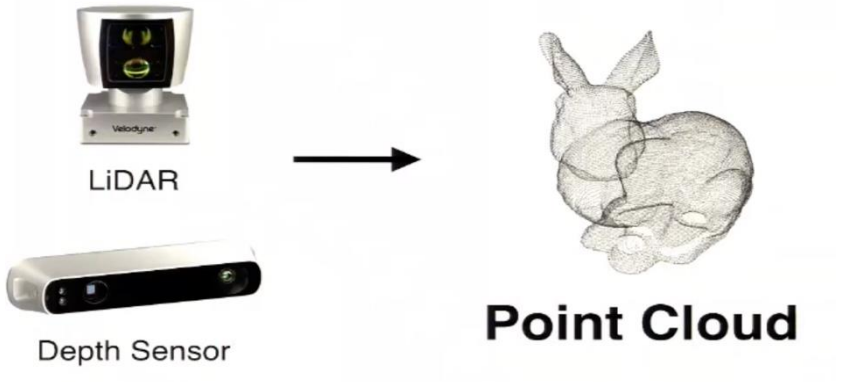


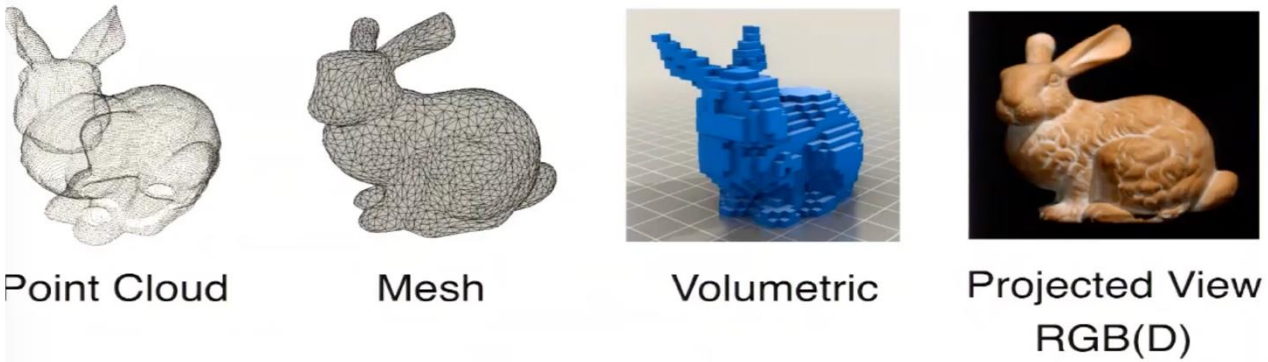
An optical sensing technology used to determine the position, velocity, of distant objects by analysis of pulsed laser light reflected from their surfaces.



What can be formulated in terms of sets

- Parsing a scene composed of a set of objects
- Making predictions from a set of points forming a 3D point cloud
- Training a set of agents in reinforcement learning
- Attention-based models perform a weighted summation of a set of features
- Concepts in explainability can be seen as set of objects
- 3D Point cloud is close to raw sensor data, and is canonical form (we can easily convert other representations into point cloud or viceversa)





Learning on sets: Setup

For now, assume graph **with no edges** ($\Omega = \mathcal{V}$, the set of **nodes**)

Let $\mathbf{x}_u \in \mathbb{R}^k$ be the features of node u.
This is a *signal* over Ω.

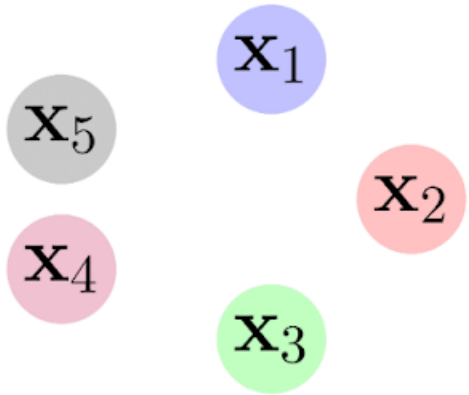
As Ω is discrete, we can stack into a *node feature matrix* of shape $|\mathcal{V}| \times k$:

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

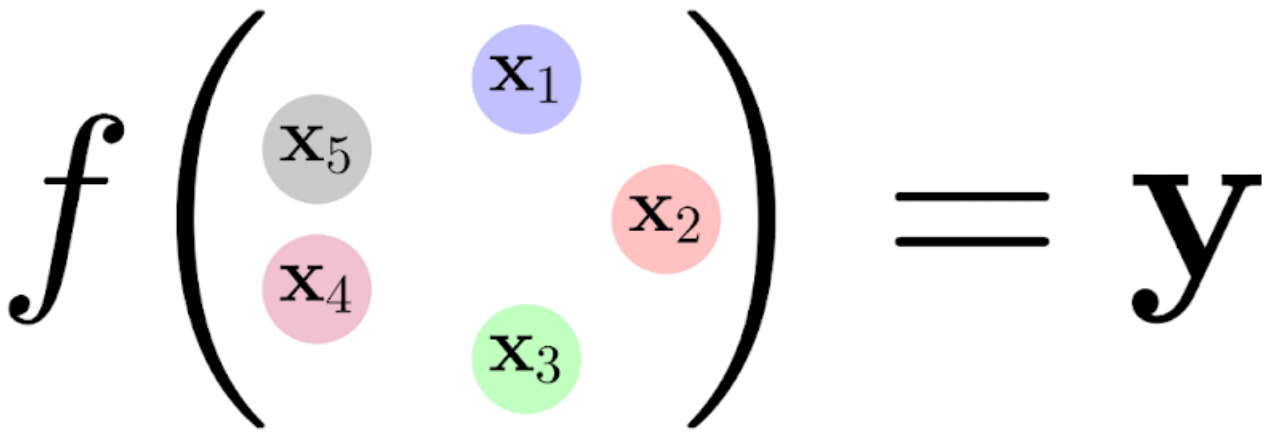
That is, the uth row of $\mathbf{X}$ corresponds to $\mathbf{x}_u$

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.

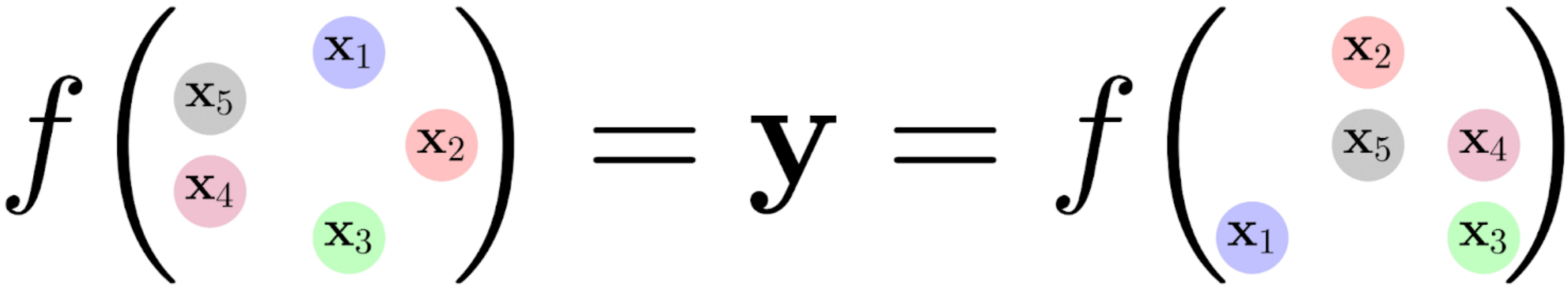
What do we want?



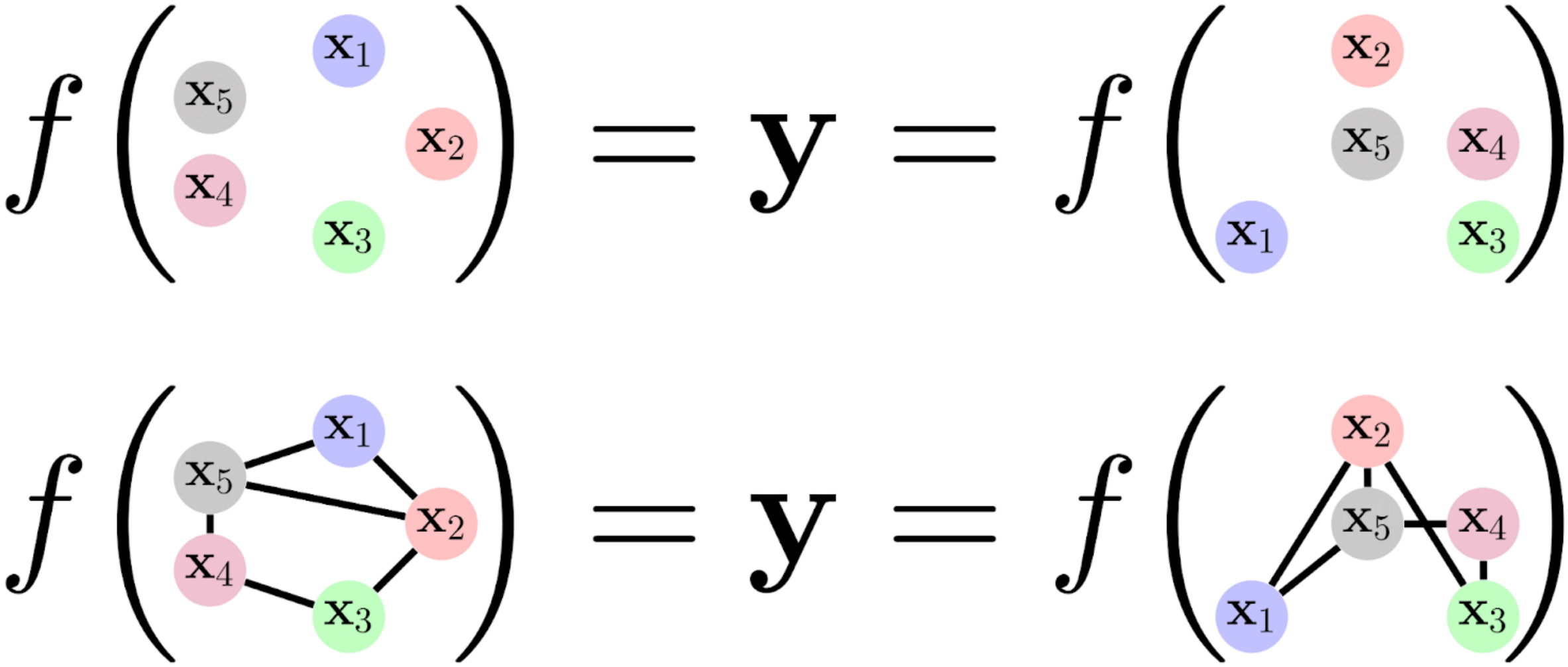
What do we want?



What do we want?



*(also useful for **later**)*

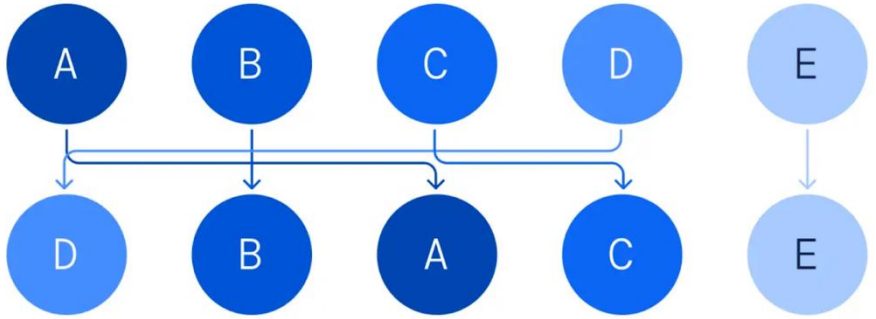


Permutations and permutation matrices

It will be useful to think about operators that **change** the node order

Such operations are known as **permutations** (there are $n!$ of them)

e.g. a permutation (2, 4, 1, 3) maps $\mathbf{x}_1 \leftarrow \mathbf{x}_2, \mathbf{x}_2 \leftarrow \mathbf{x}_4, \mathbf{x}_3 \leftarrow \mathbf{x}_1, \mathbf{x}_4 \leftarrow \mathbf{x}_3$



Permutations form a *symmetry group* $\mathfrak{G}$: n-element *permutation group* Σ_n
The group elements, $\sigma \in \Sigma_n$, are individual permutations

Permutations and permutation matrices

Within linear algebra, each permutation defines a $|\mathcal{V}| \times |\mathcal{V}|$ **matrix**

- Such matrices are called **permutation matrices**
- They have exactly one 1 in every row and column, zeroes elsewhere
- Their effect when left-multiplied is to permute rows of $\mathbf{X}$, 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 matrices are *group representations,* $\rho_X(\sigma) = \mathbf{P}_\sigma$ for $\sigma \in \Sigma_n$

Permutation invariance

Want: functions $f(\mathbf{X})$ over sets that will not depend on the order

Equivalently: applying a permutation matrix shouldn't modify result!

We arrive at a very useful notion of **permutation invariance**.
$f(\mathbf{X})$ is permutation *invariant* if, for *all* permutation matrices $\mathbf{P}$:

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

Recall the condition for $\mathfrak{G}$-*invariance:* $f(\rho_X(\mathfrak{g})x) = f(x)$

Deep Sets

A very generic form is the *Deep Sets* model (Zaheer *et al.*, NeurIPS'17):

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

where ψ and ϕ are (learnable) functions, e.g. MLPs.

The **sum** aggregation is *critical*!
(other choices possible, e.g. **max** or **avg**)

Deep Sets

A very generic form is the *Deep Sets* model (Zaheer *et al.*, NeurIPS'17):

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

where ψ and ϕ are (learnable) functions, e.g. MLPs.

The **sum** aggregation is *critical*!
(other choices possible, e.g. **max** or **avg**)

We will use $\bigoplus$ to denote *any* permutation-invariant operator.

Permutation equivariance

Permutation invariant models are good for set-level outputs
What if we want answers at **node** level, or fine-grained intermediate representations (which would not be destroyed by invariance)?

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

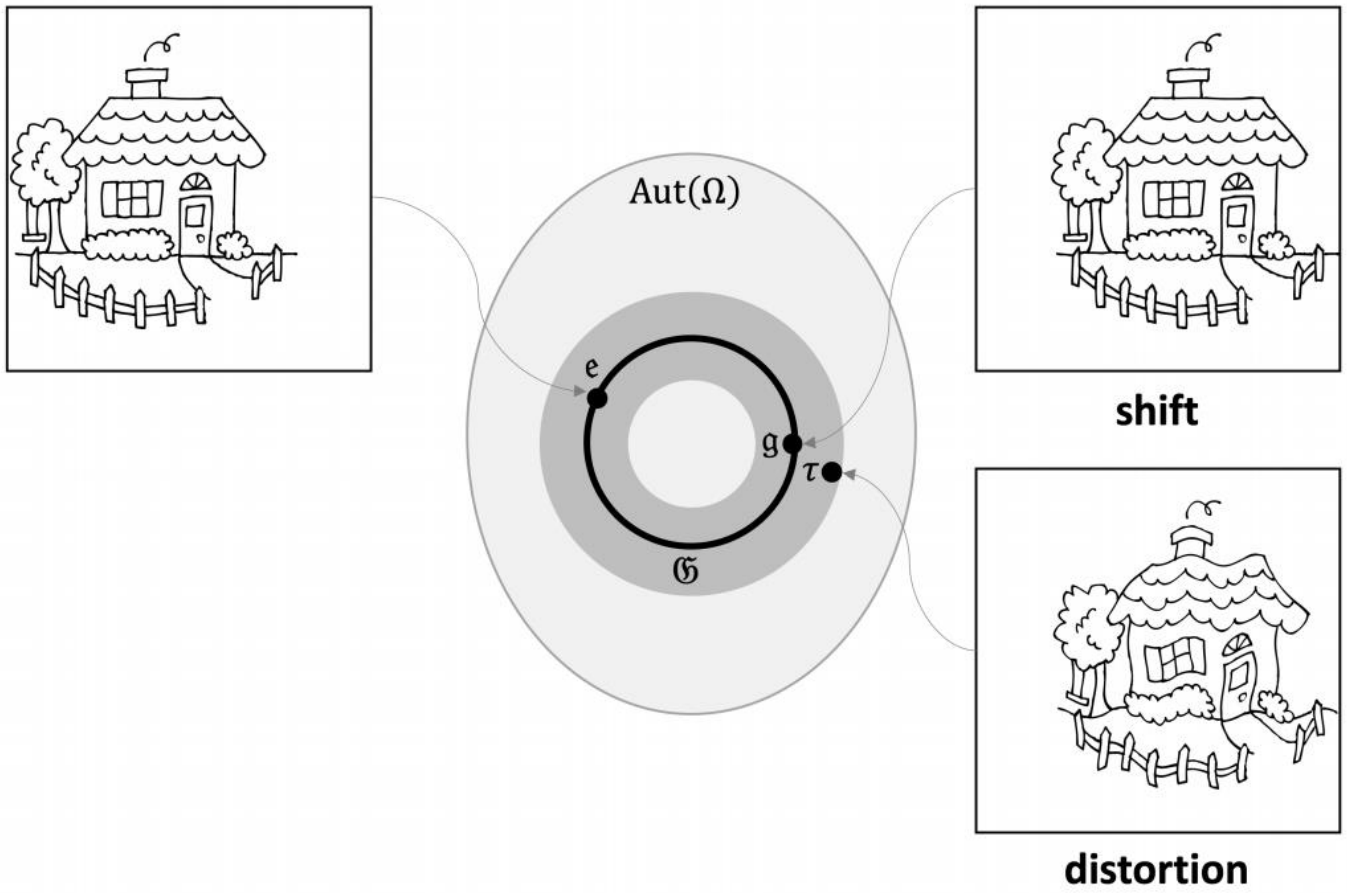
$\mathbf{F}(\mathbf{X})$ is **permutation equivariant** if, for all permutation matrices $\mathbf{P}$:

$$\mathbf{F}(\mathbf{PX}) = \mathbf{PF}(\mathbf{X})$$

Recall the condition for $\mathfrak{G}$-*equivariance:* $f(\rho_X(\mathfrak{g})x) = \rho_Z(\mathfrak{g})f(x)$
If we assume **F** does not change the structure of the vector space, we can equate $\mathcal{X} = \mathcal{Z}$

Important constraint: Locality

Equivariance is not enough; not all transformations come from $\mathfrak{G}$!
Want our signal to be **stable** under slight *deformations* of the domain



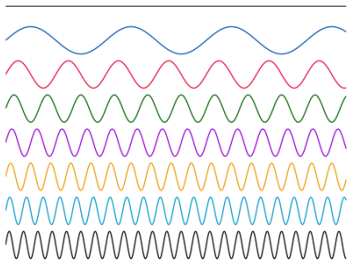
Important constraint: Locality

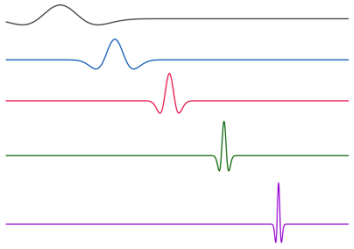
Equivariance is not enough; not all transformations come from $\mathfrak{G}$!
Want our signal to be **stable** under slight *deformations* of the domain

It is highly beneficial to compose **local** operations to model larger-scale ones, as 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





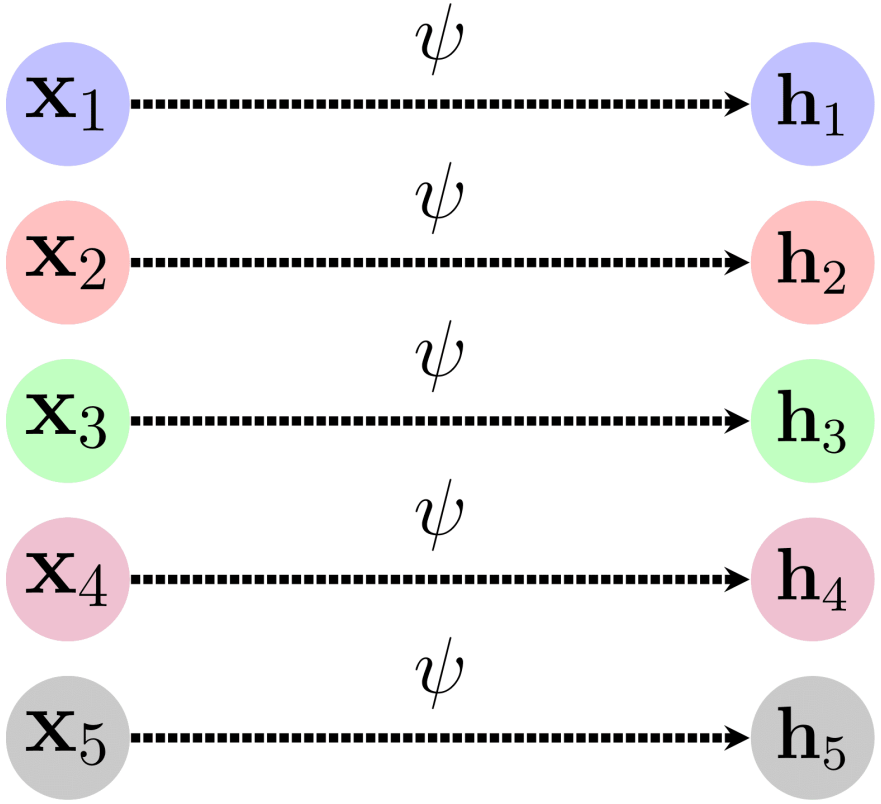
<u>What does this mean for sets?</u>

General blueprint for learning on sets

One way we can enforce **locality** in equivariant set functions is through a *shared* function ψ applied to every node in isolation:

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

Stacking $\mathbf{h}_u$ into a matrix yields $\mathbf{H} = \mathbf{F}(\mathbf{X})$



(remark: this is typically **as far** as we can get with sets, without assuming or inferring additional structure)

General blueprint for learning on sets

One way we can enforce **locality** in equivariant set functions is through a *shared* function ψ applied to every node in isolation:

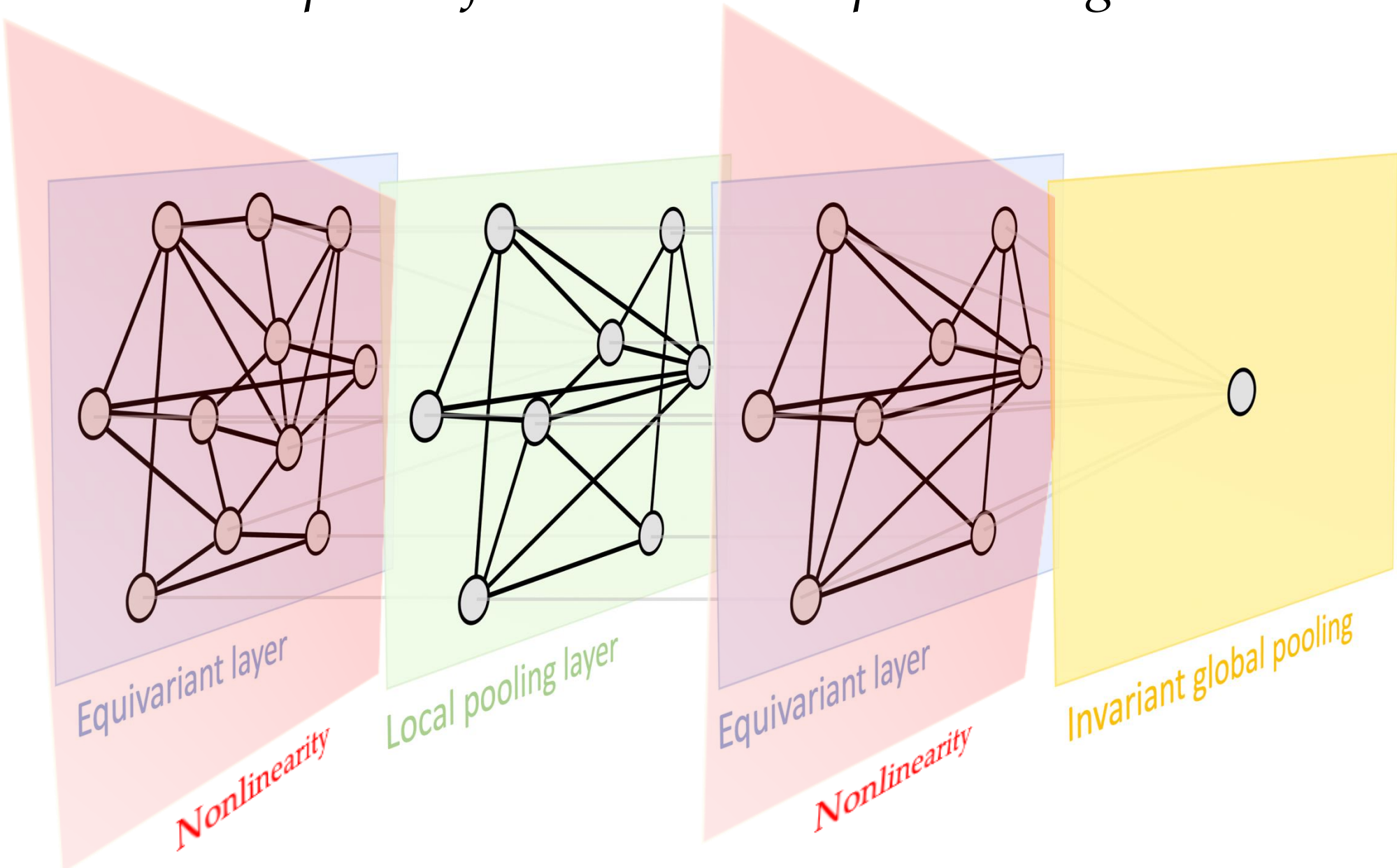
$$\mathbf{h}_u = \psi(\mathbf{x}_u)$$

Deep Sets as ***GDL blueprint***: (stacking) **equivariant** function(s), potentially with **invariant** tail---yields (m)any useful set neural nets!

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

(remark: this is typically **as far** as we can get with sets, without assuming or inferring additional structure)

Recall: The blueprint of Geometric Deep Learning



Is this really as far as we can get?

What we showed so far is:

- Useful functions on sets are *permutation equivariant / invariant*
- Deep Sets offer *one* way of achieving permutation invariance
- Is this necessarily the *only* way?

For many classes of input sets, the answer is a decisive **yes**!

That is, no matter what other permutation invariant model you propose, it will necessarily be expressible as a Deep Sets

We will cover proofs from:
Zaheer *et al.* (NeurIPS'17)
Wagstaff, Fuchs, Engelcke *et al.* (ICML'19)

Universality of Deep Sets: ***countable*** *case*

Assume that set elements come from a *countable* set of values, $\mathcal{X}$
Then, every possible set is expressible as a bit-mask, $\mathbb{B}^{\mathcal{X}}$
(where $\mathbb{B} = \{0,1\}$ is the set of bits)

Universality of Deep Sets: ***countable*** *case*

Assume that set elements come from a *countable* set of values, $\mathcal{X}$
Then, every possible set is expressible as a bit-mask, $\mathbb{B}^{\mathcal{X}}$
(where $\mathbb{B} = \{0,1\}$ is the set of bits)

Hence, *any permutation-invariant* real function $f(\mathbf{X})$ on such sets can be written as $f : \mathbb{B}^{\mathcal{X}} \to \mathbb{R}$

(if it would not be representable like this, it would depend on the **order** of the elements of the set)

Universality of Deep Sets: ***countable*** *case*

Assume that set elements come from a *countable* set of values, $\mathcal{X}$
Then, every possible set is expressible as a bit-mask, $\mathbb{B}^{\mathcal{X}}$
(where $\mathbb{B} = \{0,1\}$ is the set of bits)

Hence, *any permutation-invariant* real function $f(\mathbf{X})$ on such sets can be written as $f : \mathbb{B}^{\mathcal{X}} \to \mathbb{R}$

Since $\mathcal{X}$ is countable, there must exist an *injective* mapping $c : \mathcal{X} \to \mathbb{N}$
Using it, we may define an injective element-level function $\psi : \mathcal{X} \to \mathbb{R}$

One example is $\psi(x) = 4^{-c(x)}$ (it acts like a compressed *one-hot* representation of c(*x*))

Universality of Deep Sets: ***countable*** *case*

Assume that set elements come from a *countable* set of values, $\mathcal{X}$
Then, every possible set is expressible as a bit-mask, $\mathbb{B}^{\mathcal{X}}$
(where $\mathbb{B} = \{0,1\}$ is the set of bits)

Hence, *any permutation-invariant* real function $f(\mathbf{X})$ on such sets can be written as $f : \mathbb{B}^{\mathcal{X}} \to \mathbb{R}$

Since $\mathcal{X}$ is countable, there must exist an *injective* mapping $c : \mathcal{X} \to \mathbb{N}$
Using it, we may define an injective element-level function $\psi : \mathcal{X} \to \mathbb{R}$

Now $\Phi(\mathbf{X}) = \sum_{u \in \mathcal{V}} \psi(x_u)$ is an *injective* function of every set $\mathcal{V} \in \mathbb{B}^{\mathcal{X}}$

Universality of Deep Sets: ***countable*** *case*

Assume that set elements come from a *countable* set of values, $\mathcal{X}$
Then, every possible set is expressible as a bit-mask, $\mathbb{B}^{\mathcal{X}}$
(where $\mathbb{B} = \{0,1\}$ is the set of bits)

Hence, *any permutation-invariant* real function $f(\mathbf{X})$ on such sets can be written as $f : \mathbb{B}^{\mathcal{X}} \to \mathbb{R}$

Since $\mathcal{X}$ is countable, there must exist an *injective* mapping $c : \mathcal{X} \to \mathbb{N}$
Using it, we may define an injective element-level function $\psi : \mathcal{X} \to \mathbb{R}$

Now $\Phi(\mathbf{X}) = \sum_{u \in \mathcal{V}} \psi(x_u)$ is an *injective* function of every set $\mathcal{V} \in \mathbb{B}^{\mathcal{X}}$

This means that there **must** exist a $\phi : \mathbb{R} \to \mathbb{R}$ s.t. $f(\mathbf{X}) = \phi(\sum_{u \in \mathcal{V}} \psi(x_u))$
for all possible sets $\mathbf{X}$ (first "invert" $\Phi(\mathbf{X})$ to recover the original set, then apply f to it)

Universality of Deep Sets: ***countable*** *case*

There **must** exist a $\phi : \mathbb{R} \to \mathbb{R}$ s.t. $f(\mathbf{X}) = \phi(\sum_{u \in V} \psi(x_u))$
for all possible sets $\mathbf{X}$

This implies that **any** permutation-invariant function over sets that have countable values must be representable in the "Deep Set" model

Hence, so long as we choose ϕ and ψ as *universal approximators* (MLPs), Deep Sets can learn to model any "useful" function on such sets
(**NB**. This does not tell us anything about how *easy* it is to learn such functions with gradient descent!)

This analysis naturally extends to vector-valued $\mathbf{x}_i$
What about the *uncountable* case?

Universality of Deep Sets: ***uncountable*** *case*

Do we even need to consider the uncountable case?
After all, computers can only ever represent $\mathbb{Q}$…

However, we are fitting *neural networks* (MLPs) to certain ψ and ϕ

And universal approximation results of MLPs (Cybenko, 1989) rely on the target function being *real* and *continuous*!

What's "continuous in $\mathbb{Q}$" need not be continuous in $\mathbb{R}$!!!

The importance of continuity for universal approximation

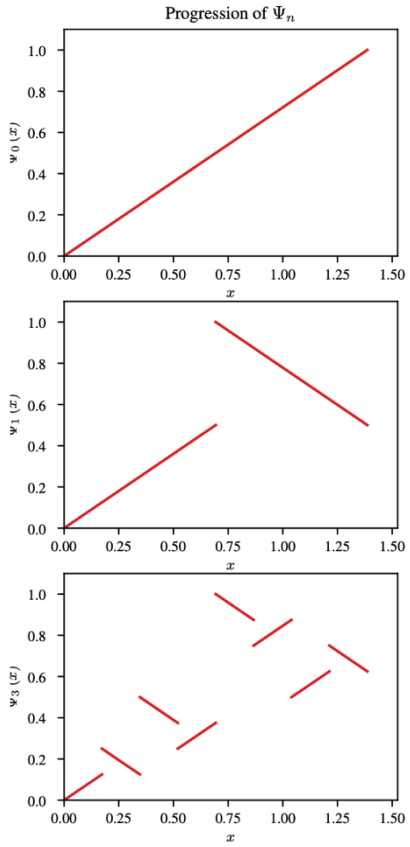
A function is *continuous* if, at each domain point, the *variation* of the *output* can be made *arbitrarily small*, by limiting the *input variation*

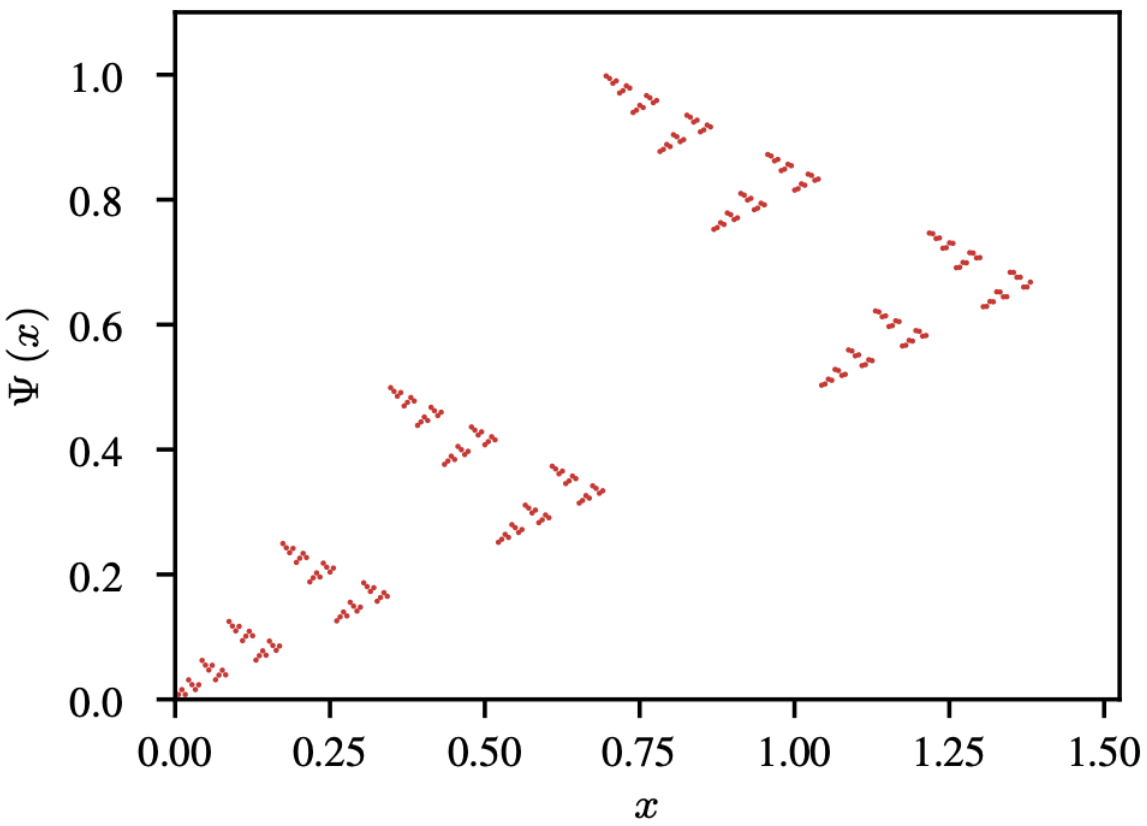
The *universal approximation theorem* says only that *any* ***continuous*** *function* can be approximated by a neural network.

A similar statement is true for other approximators, e.g. *Gaussian processes*. The notion of continuity is that on *compact subsets* of $\mathbb{R}^N$

If we wish to work mathematically with continuity in a way that closely matches our intuitions, we must consider *uncountable domains*

Universality of Deep Sets: ***uncountable*** *case*





Ψ is continuous at *every rational point* in $[0, \log 4]$. This is because all *jumps* occur at *irrational values*.

It illustrates the fact that continuity on $\mathbb{Q}$ is a much weaker property than continuity on $\mathbb{R}$.

The latter is required to satisfy the universal approximation theorem for neural networks.

This motivates us to study sets defined over $\mathbb{R}$.

Universality of Deep Sets: ***uncountable*** *case*

When our underlying element domain is $\mathbb{R}$, the proofs become substantially more involved, but the high-level idea remains similar:

- Let f be any permutation-invariant function over real-valued sets
- Find an element-level mapping, $\psi : \mathbb{R} \to \mathbb{R}^N$
- Show that the aggregated map, $\Phi(\mathbf{X}) = \sum_{u \in \mathcal{V}} \psi(x_u)$ is injective
- Show that Φ has an inverse (Φ^{-1})
- Let $\phi = f \circ \Phi^{-1}$, completing the proof.

Universality of Deep Sets: ***uncountable*** *case*

When our underlying element domain is $\mathbb{R}$, the proofs become substantially more involved, but the high-level idea remains similar:

- Let f be any permutation-invariant function over real-valued sets
- Find an element-level mapping, $\psi : \mathbb{R} \to \mathbb{R}^N$
- Show that the aggregated map, $\Phi(\mathbf{X}) = \sum_{u \in V} \psi(x_u)$ is injective
- Show that Φ has an inverse (Φ^{-1})
- Let $\phi = f \circ \Phi^{-1}$, completing the proof.

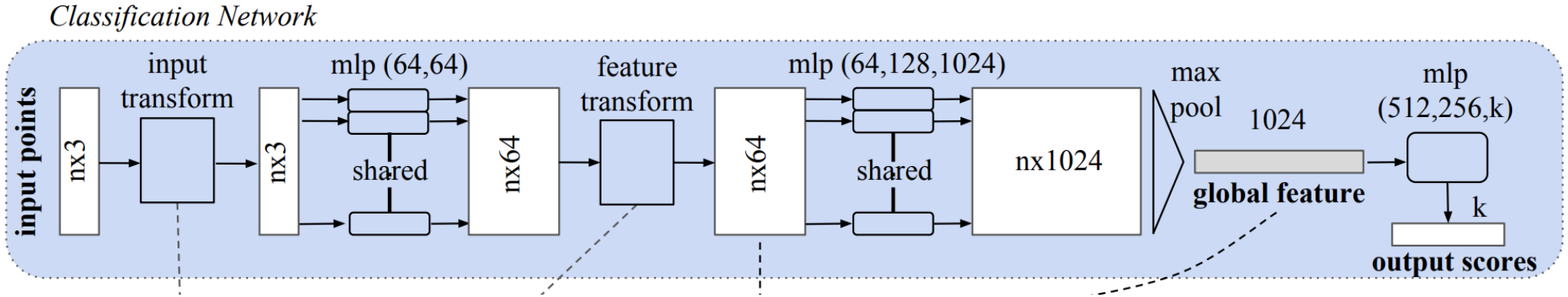
It turns out that finding a suitable ψ cannot always be done!

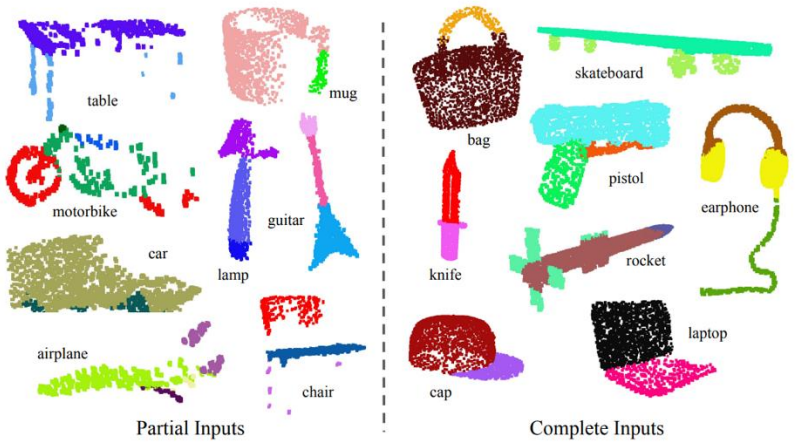
According to Wagstaff, Fuchs, Engelcke *et al.*, this construction can **only** be done when $N \geq M$, where M is the maximal input set size.

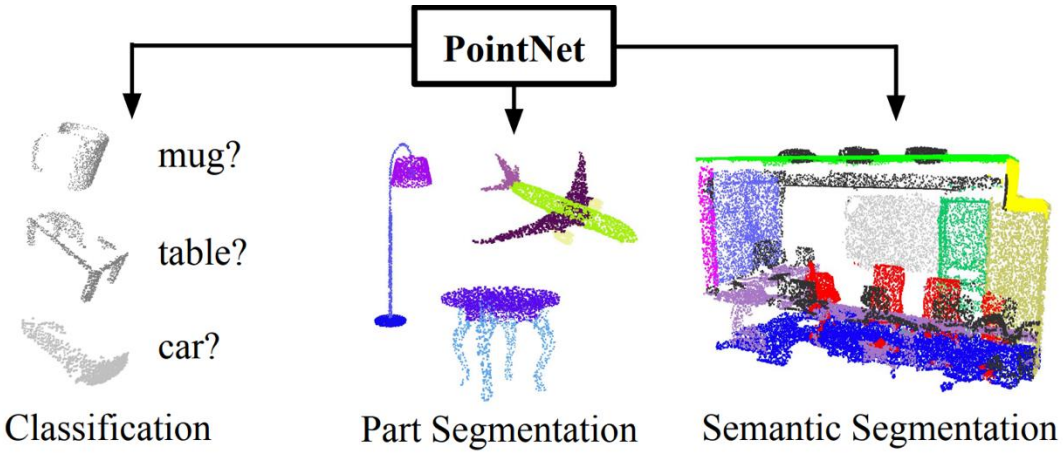
(from a deep learning perspective, N can be interpreted as an MLP-specific hyperparameter)
(keep this limitation in mind – it will come in handy when we study GNN expressive power!)

Applicability of Deep Sets: PointNet

The *PointNet* model of Qi, Su *et al.* (CVPR'17) empirically demonstrates that Deep Sets-style ideas work well on relevant *point cloud* tasks







Applicability of Deep Sets: PointNet

The *PointNet* model of Qi, Su *et al.* (CVPR'17) empirically demonstrates that Deep Sets-style ideas work well on relevant *point cloud* tasks

It applies point-level MLPs and (max)-pooling, just like Deep Sets

Uses a very similar proof strategy to show their network is universal

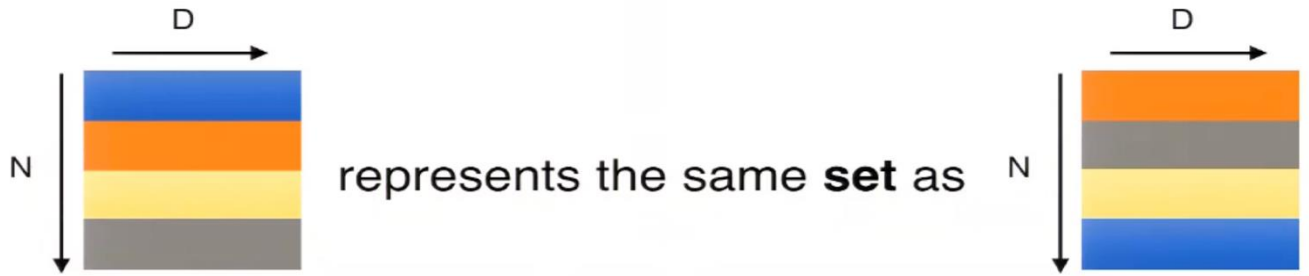
PointNets also feature a *coordinate transformation* which is mindful of the points' geometry in 3D space (so the set is not "purely unordered")

We won't study such transformations more deeply at this time
Keep it in mind – it will be useful for geometry-aware GNNs

PointNet: achieving feature learning directly on point cloud

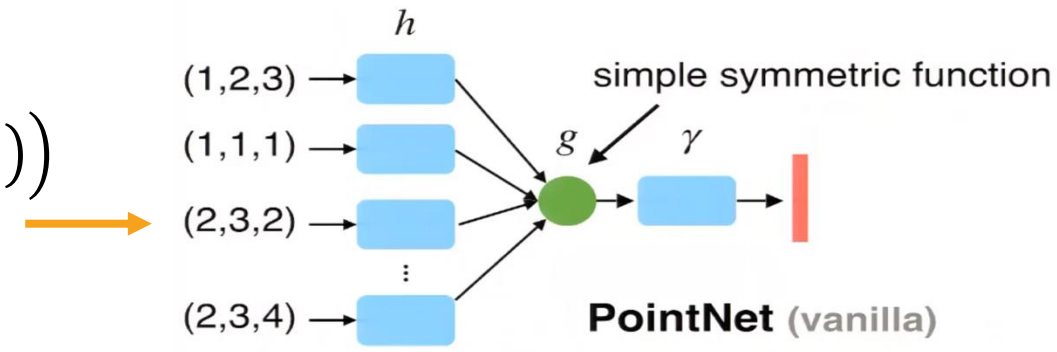
Point cloud: N orderless points each represented by a d-dim. vector

- Model needs to be invariant to $N!$ permutations
- Permutation invariance: symmetry function



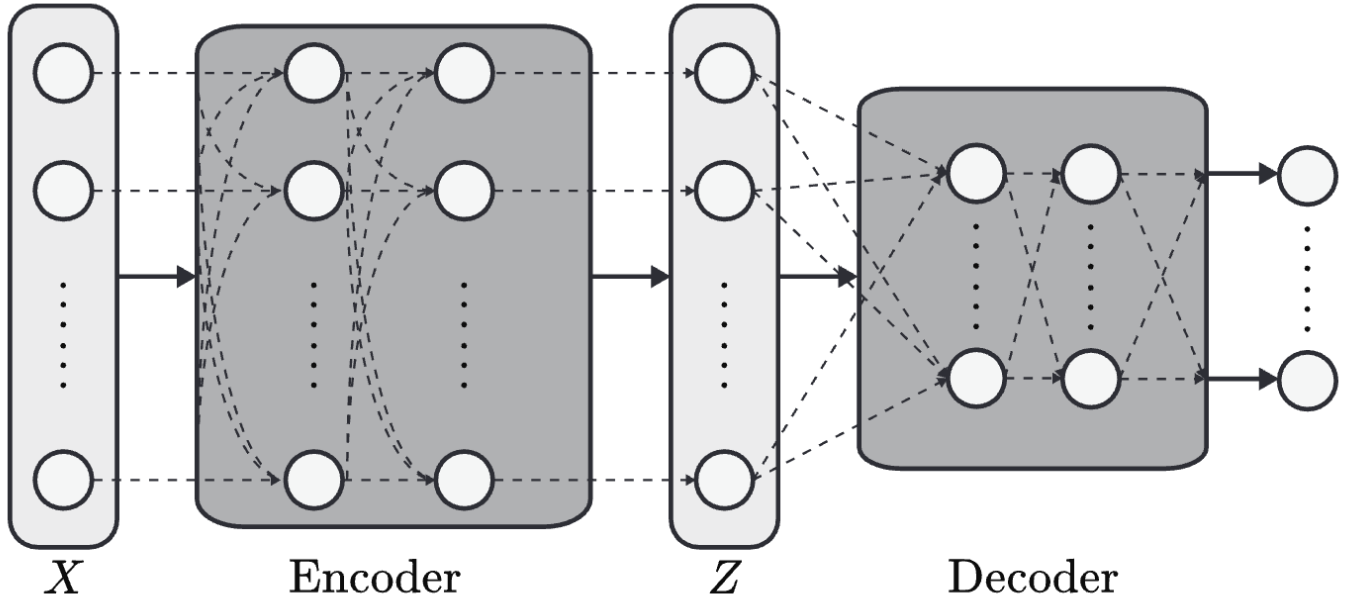
$$f(x_1, x_2, \dots, x_n) = \gamma\left(g\left(h(x_1), h(x_2), \dots, h(x_n)\right)\right)$$

is symmetric if g is symmetric



A remark on "Set Transformers"

The *Set Transformer* model (Lee *et al.*, ICML'19) builds up on Deep Sets
Use Transformer-style encoders instead of the pointwise one

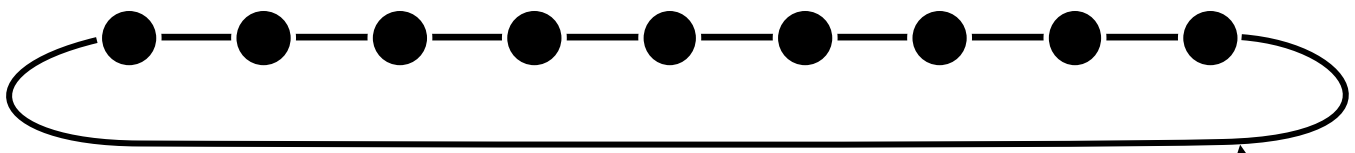


More powerful than Deep Sets, but *not a set model* under our theory.
Recall, we assume *no edges exist* in Ω, but self-attention performs explicit pairwise interactions between elements

Transformers will receive a thorough treatment later in the course

Grids

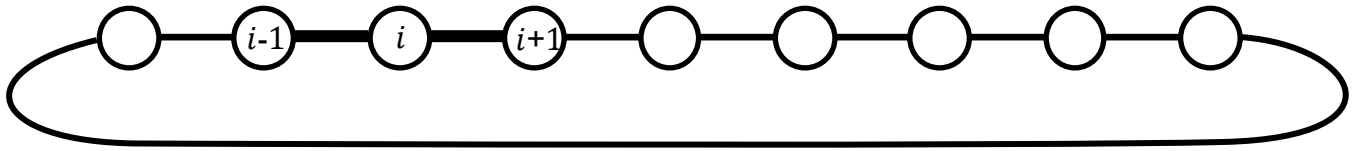
Grids vs Graphs



grid = ring graph

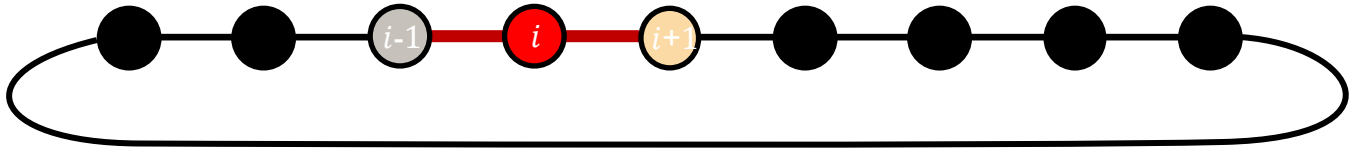
periodic b.c.

Grids vs Graphs



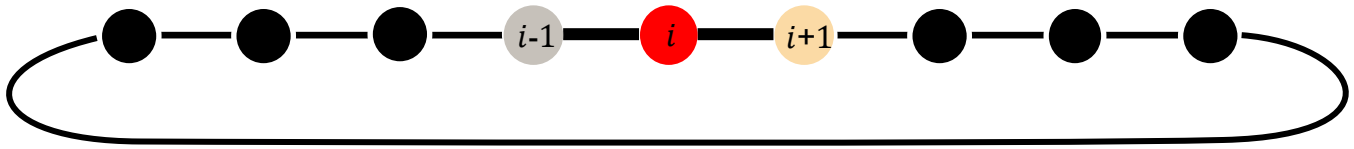
fixed neighbourhood structure

Grids vs Graphs



fixed neighbourhood structure

Grids vs Graphs

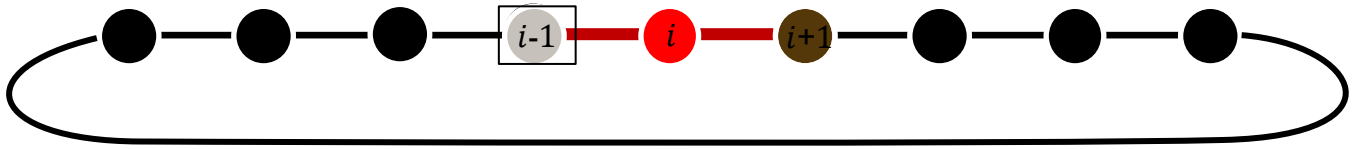


local aggregation function

$$f(\mathbf{x}_i) = \phi(\mathbf{x}_i, \{\mathbf{x}_{i-1}, \mathbf{x}_{i+1}\})$$

Each node has two neighbors, the order of the neighbors is *fixed*
So in the case of grids, we have a *prescribed order* of the nodes
so we are not in the case of permutation invariance anymore

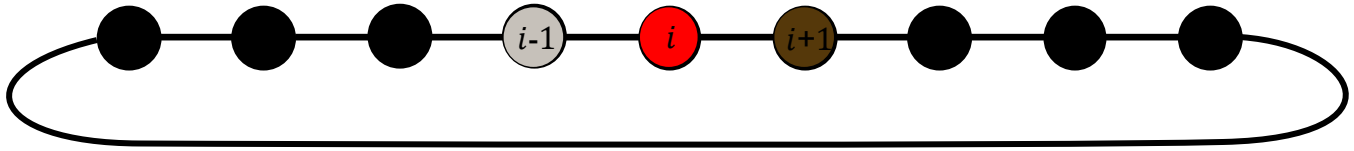
Grids vs Graphs



local aggregation function

$$f(\mathbf{x}_i) = \phi(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1})$$

Grids vs Graphs



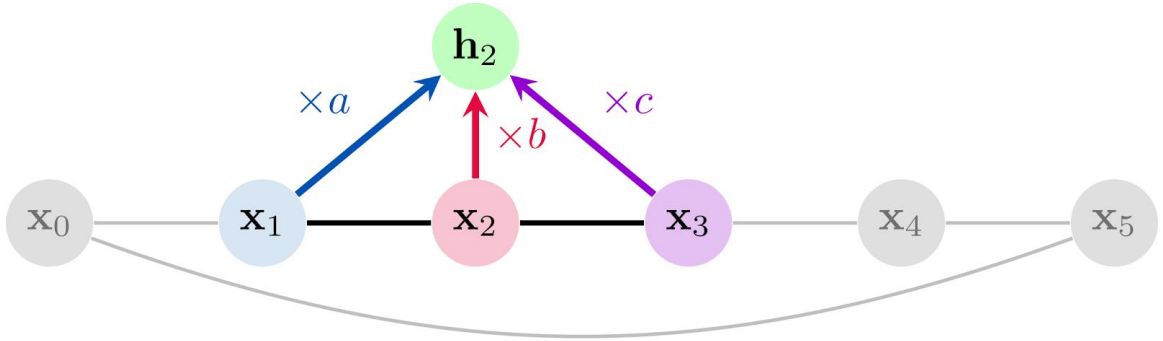
linear local aggregation function

$$f(\mathbf{x}_i) = a\mathbf{x}_{i-1} + b\mathbf{x}_i + c\mathbf{x}_{i+1}$$

If we aggregate as a sum, we get a *convolution*

Rethinking the convolution on sequences

If we further plug in the constraint of *linear* equivariance



We recover exactly the familiar convolutional layer!
NB: This defines a circulant matrix $\mathbf{C}([b, c, 0, \ldots, 0, a])$:

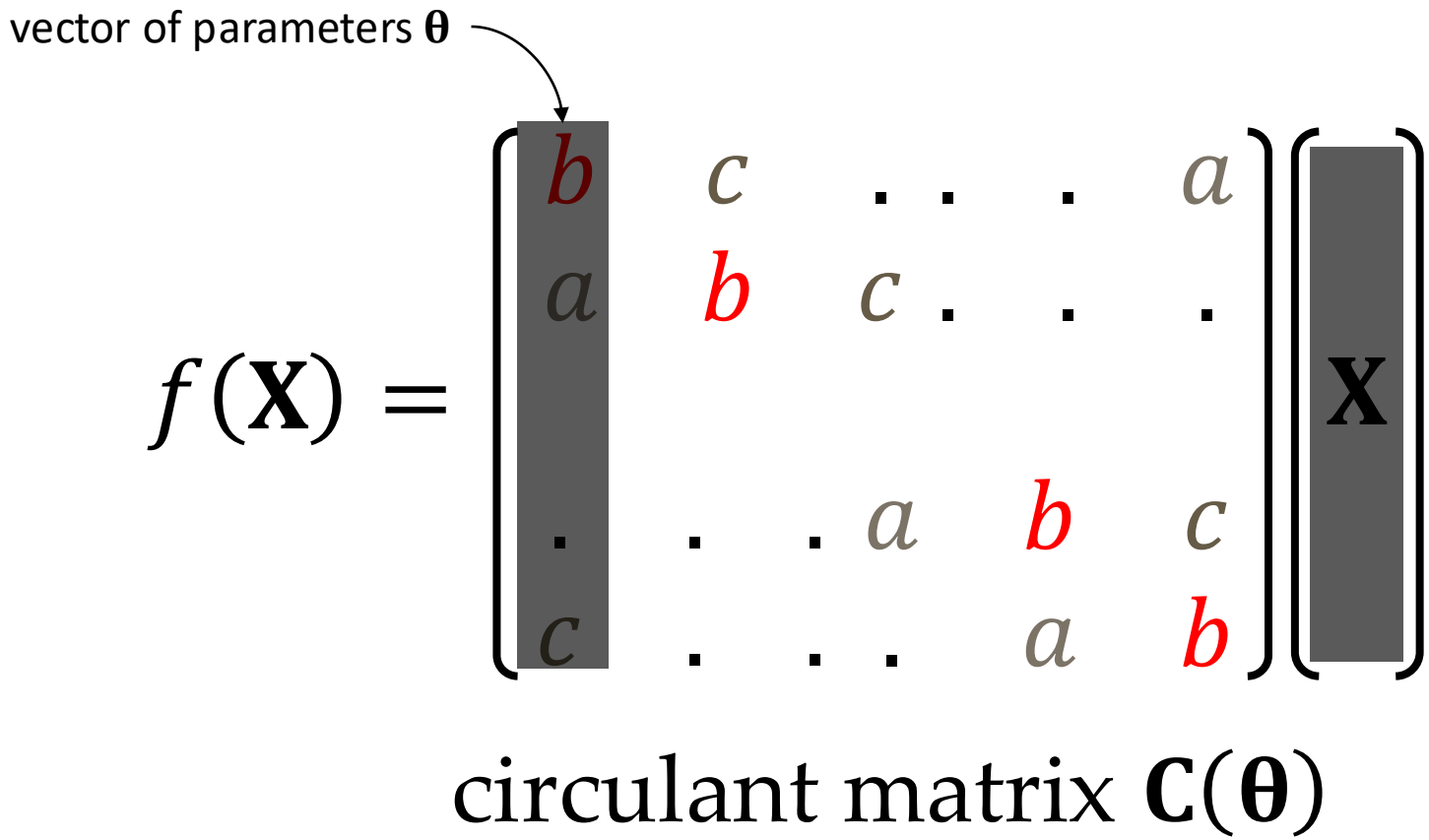
$$f(\mathbf{X}) = \begin{bmatrix} b & c & & & a \\ a & b & c & & \\ & \ddots & \ddots & \ddots & \\ & & a & b & c \\ c & & & a & b \end{bmatrix} \begin{bmatrix} - & \mathbf{x}_0 & - \\ - & \mathbf{x}_1 & - \\ & \vdots & \\ - & \mathbf{x}_{n-2} & - \\ - & \mathbf{x}_{n-1} & - \end{bmatrix}$$

Convolution

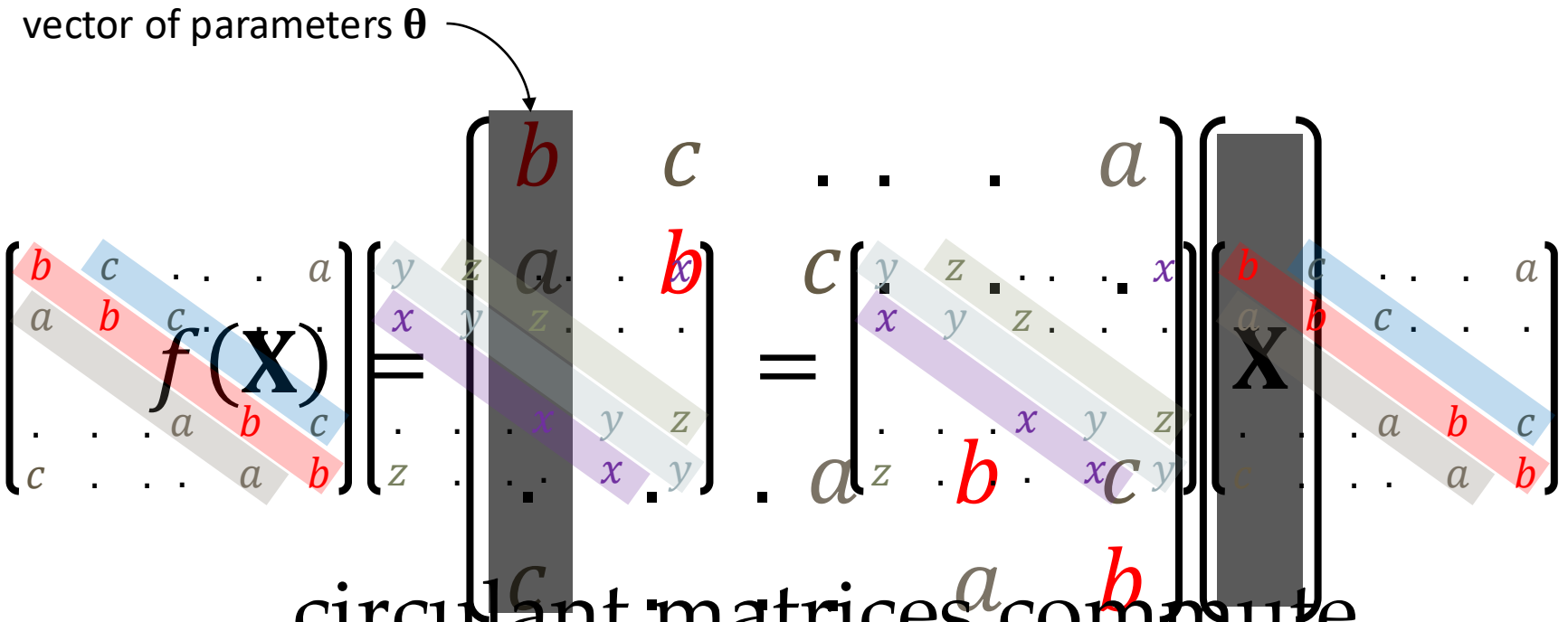
$$f(\mathbf{X}) = \begin{bmatrix} b & c & \cdots & a \\ a & b & c & \cdots \\ & & & \\ \cdots & a & b & c \\ c & \cdots & a & b \end{bmatrix} \begin{bmatrix} \\ \mathbf{X} \\ \\ \end{bmatrix}$$

circulant matrix = convolution

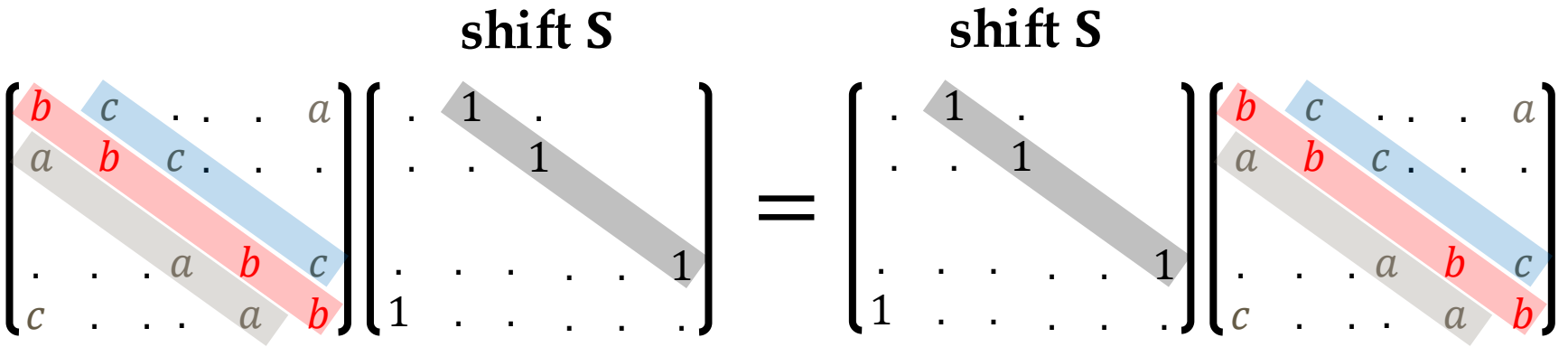
Convolution



Deriving Convolution from Symmetry

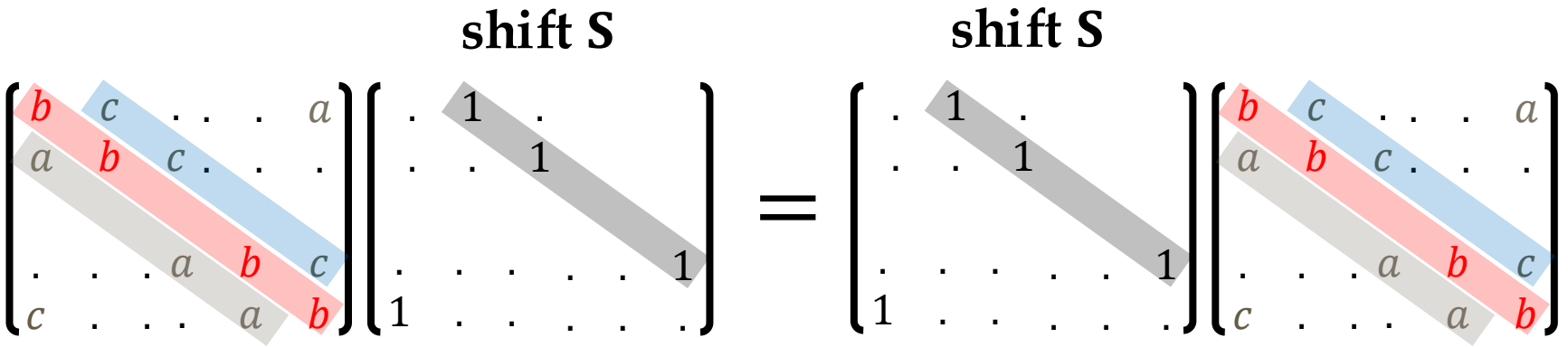


Deriving Convolution from Symmetry



circulant matrix $\Longrightarrow$ commutes with shift

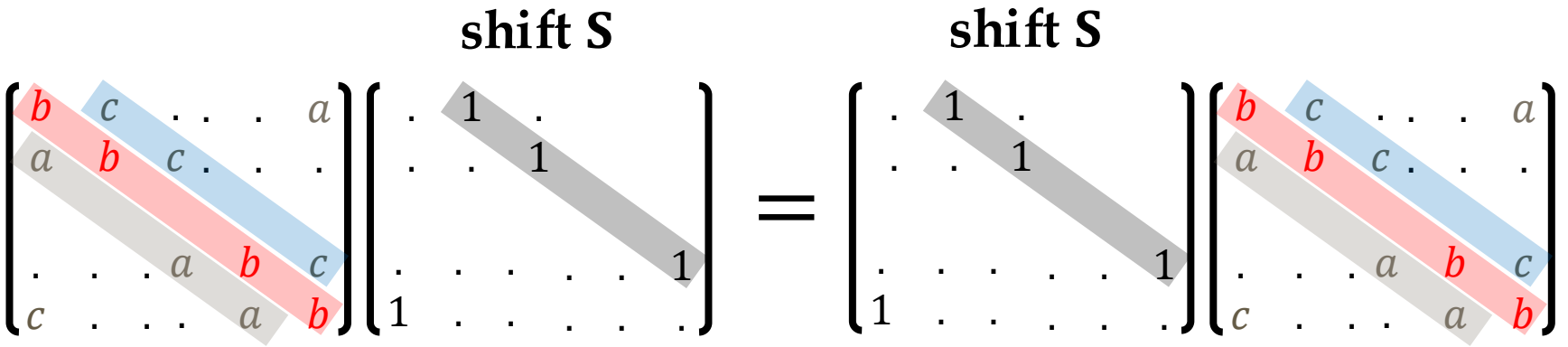
Deriving Convolution from Symmetry



convolution $\Longrightarrow$ shift-equivariant

$$\mathbf{CS = SC}$$

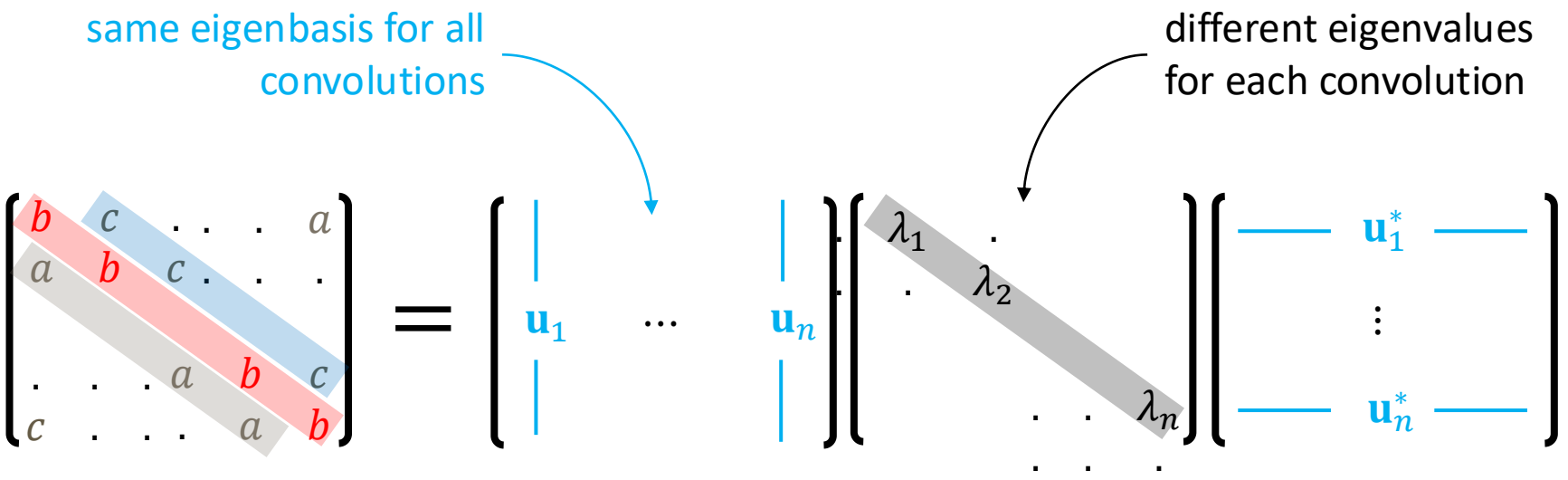
Deriving Convolution from Symmetry



convolution $\Leftrightarrow$ shift-equivariant

convolution **emerges** from translation symmetry

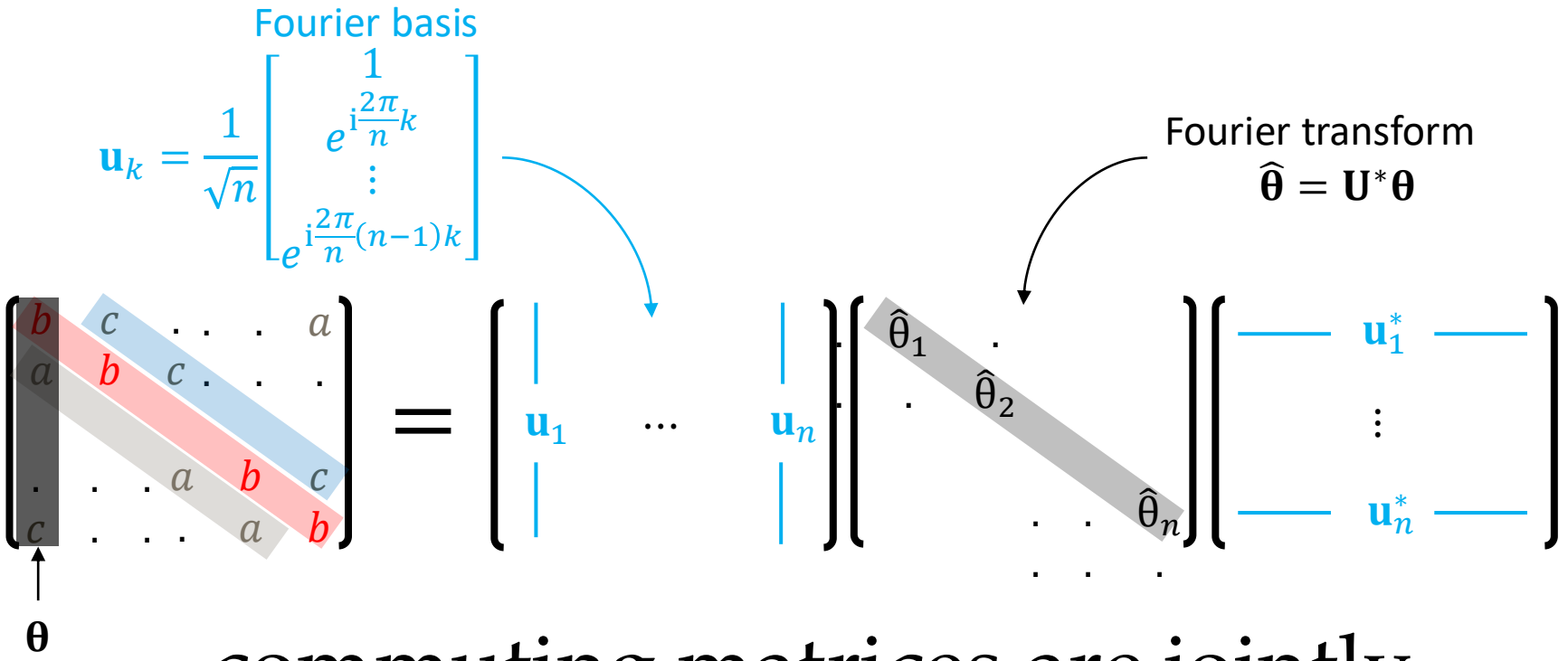
Deriving Fourier Transform from Symmetry



commuting matrices are jointly diagonalisable

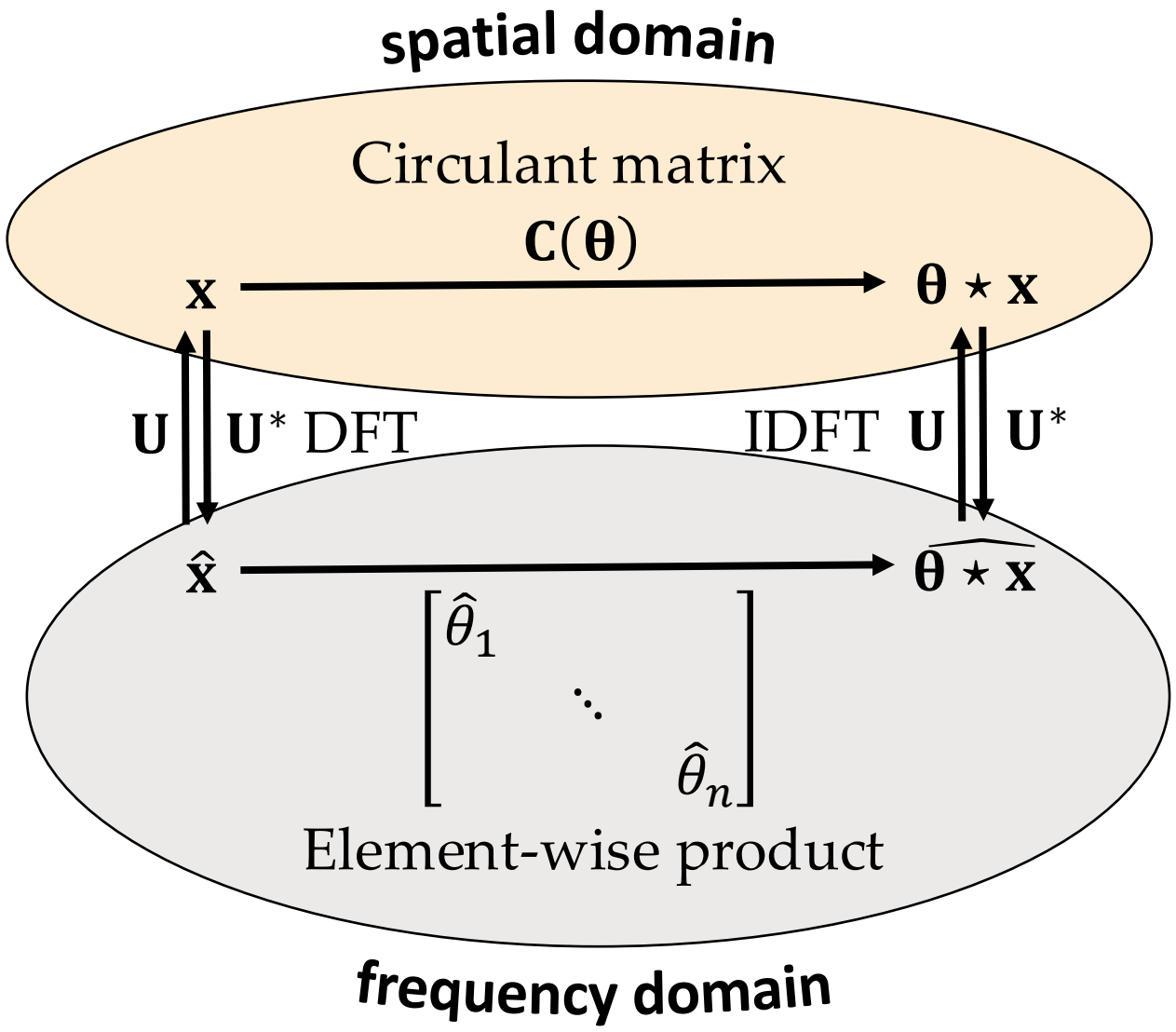
by eigenvectors of **S**

Deriving Fourier Transform from Symmetry



commuting matrices are jointly diagonalisable by Fourier Transform

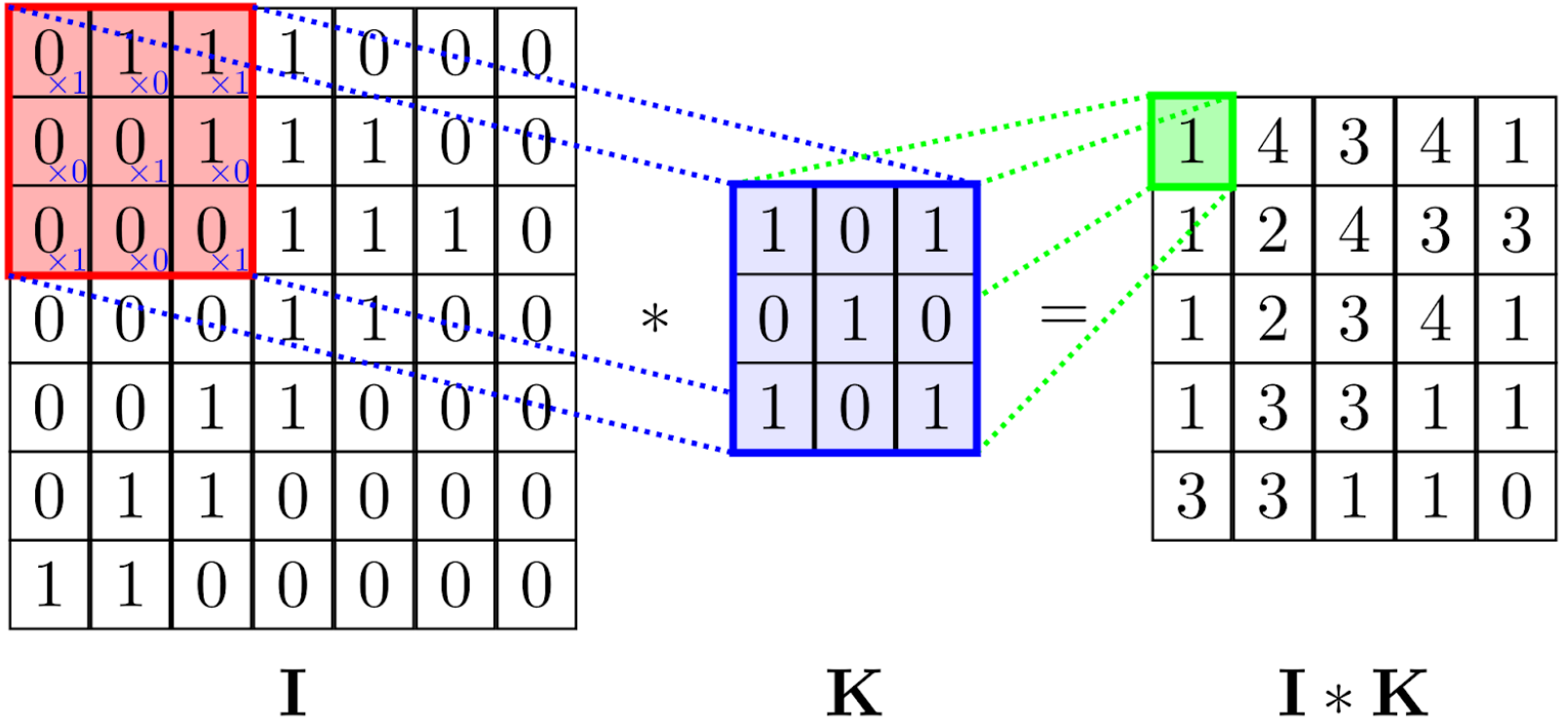
The Convolution Theorem in signal processing



CNNs

2D convolutions

This naturally generalises to higher dimensions (from the *1D cyclical shifts* to the general *translation* group in n dimensions)
For example, in 2D, we recover the familiar *image CNNs*
(Fukushima and Miyake, PR'82) (LeCun *et al.*, Proc. IEEE'98)



CNN has useful functions over images

The workhorse of neural networks on images is the *convolution*

We can see convolution as a sliding window pattern matching operator

Useful functions over images

Convolutions have some very nice properties:

- The output changes *predictably* when the input is *shifted*
- Errors in one input pixel *do not propagate* to the entire output
- The computation is distributed across *neighbourhoods*

In the next lecture: We see how to formalise these concepts over graphs

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:

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

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

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

Now we will *generalise* the convolution to more general Ω and $\mathfrak{G}$

Group convolutions

Recall that we can define **inner products** over a domain Ω. Hence,

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

Grids

convolution= matching shifted filter

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

Groups

convolution= matching transformed filter

NB: group convolution gives a function over elements of $\mathfrak{G}$!!!
Conveniently, for grids, $\mathfrak{G} = \Omega = \mathbb{R}$ (scalar **shifts**!)

Group convolutions

Grids

convolution = matching shifted filter

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

shift vector — shift operator

domain Ω = symmetry group $\mathfrak{G}$

Groups

convolution = matching transformed filter

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

group element — group representation

Useful functions over images

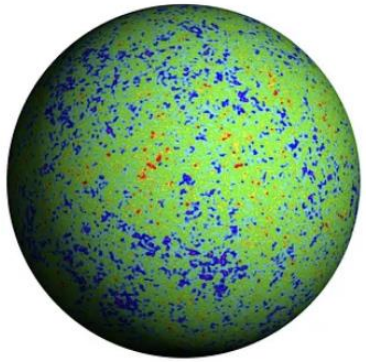
For 2D images, *translational symmetry* can be encoded efficiently by applying a stack of *convolutional filters*, translated across the image.

Because same convolutional filters are being applied across all locations, the operation is *translationally equivariant*.

This means that, regardless of *where* a feature is located, it stimulates activations in the corresponding location in an *identical* manner.

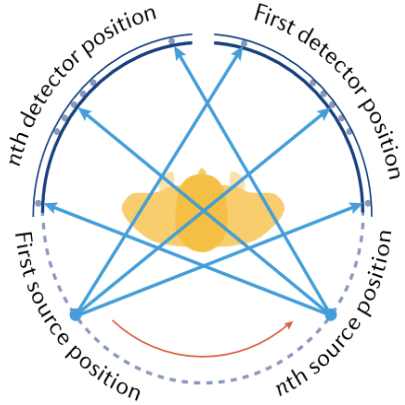
Spherical CNNs

Example: ***Spherical*** *convolutions*

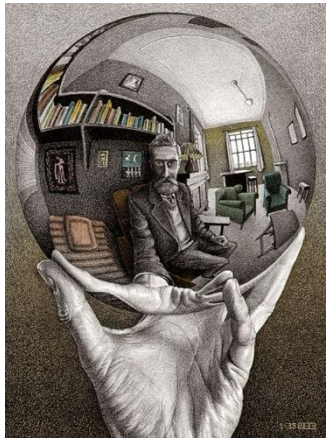


Cosmic microwave background

360° virtual reality



Medicine



Art

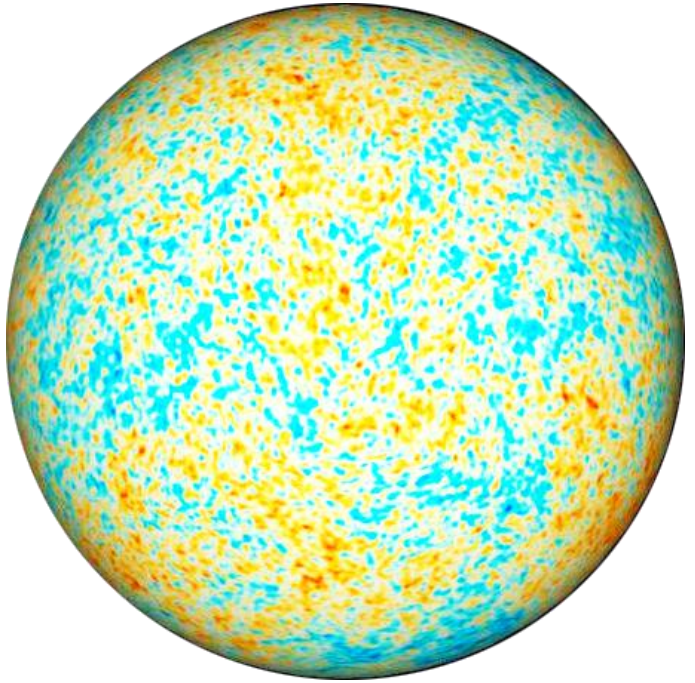
Many fields involve data that live inherently on the *sphere*

Spherical data can arise when observations are made at each point on a spherical surface, such as a topographic map of the *Earth*.

However, it also arises when observations are made over *directions*, such as for the *cosmic microwave background* (CMB) in cosmology or for 360° imagery in VR and computer vision, 3D scans in *medical imaging*, and *meshed surfaces* in graphics

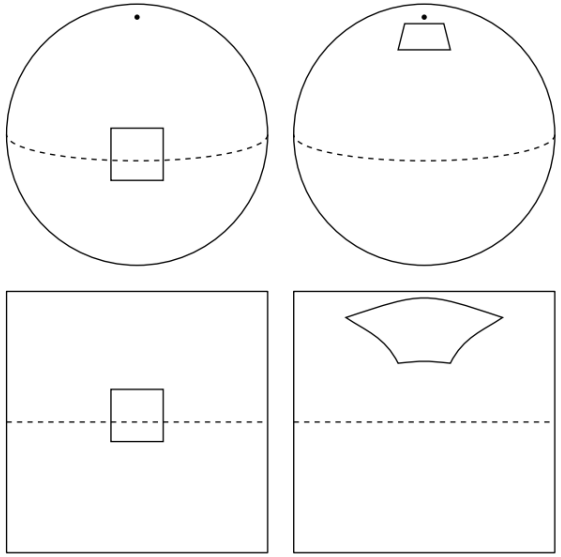
SO(3) = Special 3D orthogonal group

Example: ***Spherical*** *convolutions*



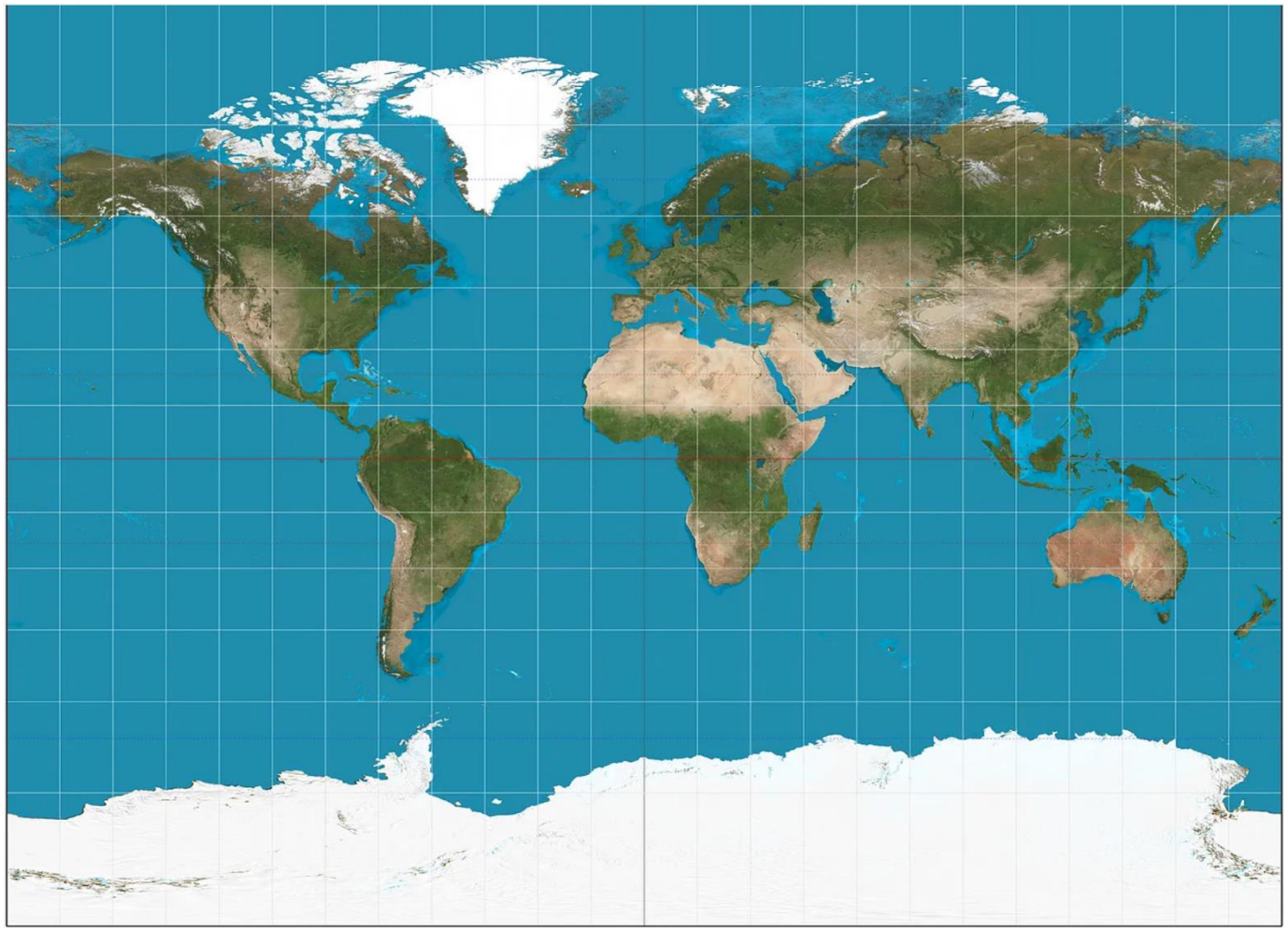
Consider a signal defined over a **sphere**, $\Omega = \mathbb{S}^2$
Very relevant, e.g., for Earth maps, astrophysics…

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



Any planar projection of a spherical signal will result in *distortions*. Rotation of a spherical signal *cannot* be emulated by translation of its planar projection.

*Example: **Spherical** convolutions*

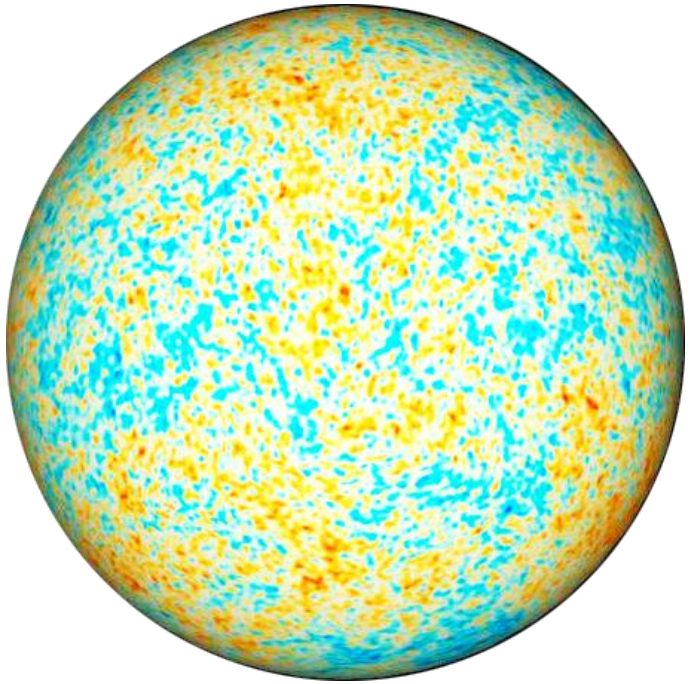


Projections of the sphere to the plane introduce *distortions* that are *unavoidable*, irrespective of the projection method used.

For this reason, Greenland commonly appears to be a similar size to Africa on maps of the Earth, whereas it is actually *less than one tenth* of the size.

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

*Example: **Spherical** convolutions*



Consider a signal defined over a **sphere**, $\Omega = \mathbb{S}^2$
Very relevant, e.g., for Earth maps, astrophysics…

We want to be **rotation equivariant**
This means using the rotation group, $\mathfrak{G} = \mathrm{SO}(3)$

We can represent the points on a sphere with three-dimensional unit vectors, $\mathbf{u}$
The **group representation** is a 3x3 *rotation matrix*, $\mathbf{R}$

We recover our spherical convolution:

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

Convolution, revisited

convolution = matching shifted filter

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

shift vector

shift operator

domain Ω = symmetry group $\mathfrak{G}$

Group Convolution

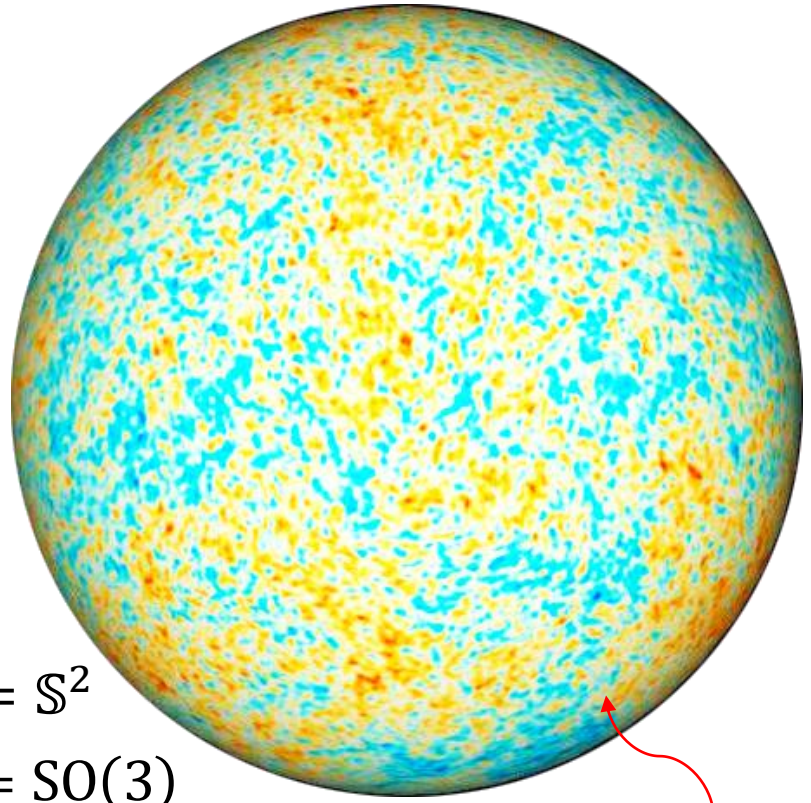
convolution = matching transformed filter

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

group element

group representation

Convolution on the Sphere



sphere $\Omega = \mathbb{S}^2$

rotation group $\mathfrak{G} = \mathrm{SO}(3)$

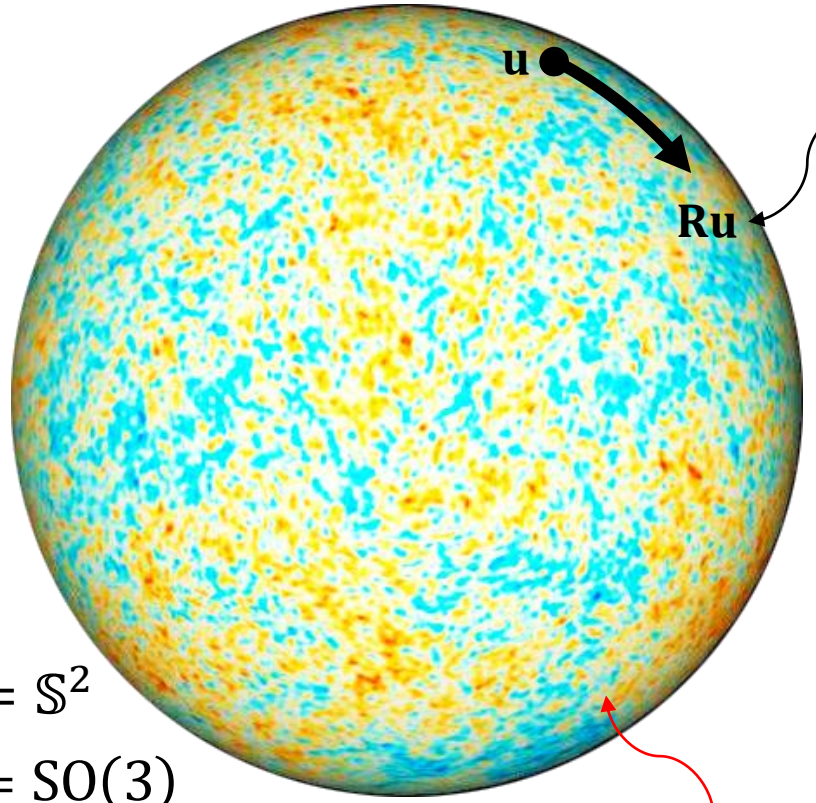
spherical signal x

Cohen, Welling 2016

Convolution on the Sphere

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

signal on $\mathrm{SO}(3)$

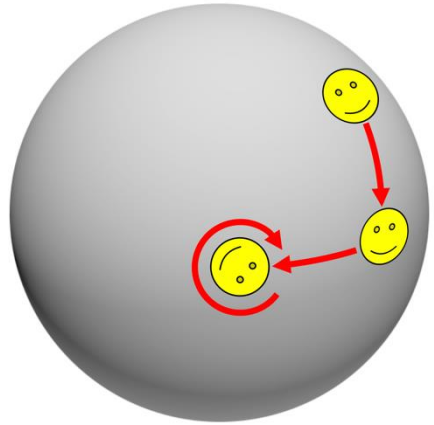


rotation transformation that preserves orientation

sphere $\Omega = \mathbb{S}^2$

rotation group $\mathfrak{G} = \mathrm{SO}(3)$

spherical signal x



Cohen, Welling 2016

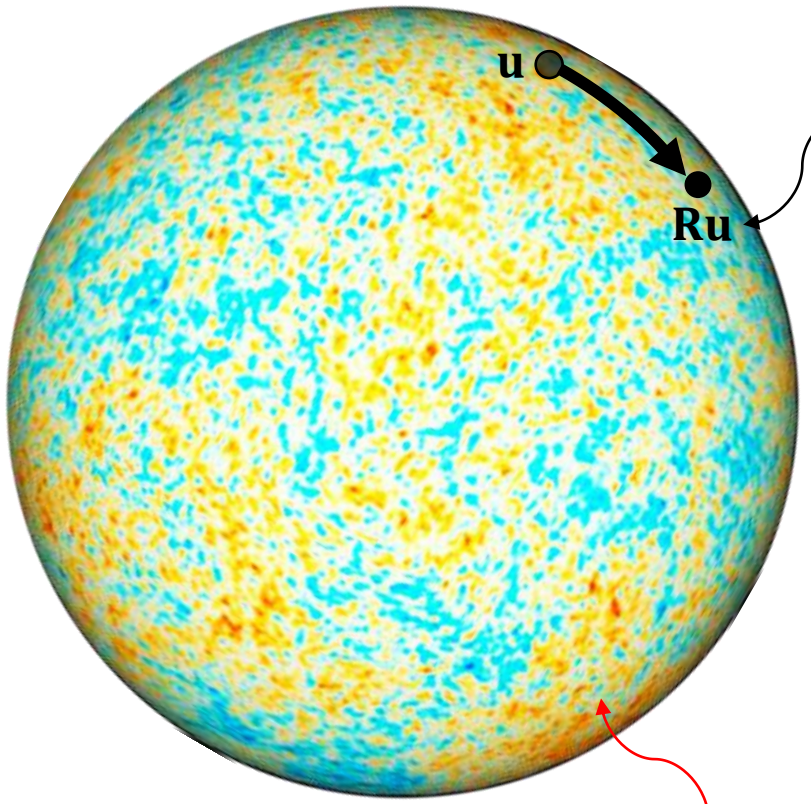
Convolution on the Sphere

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

The input and output domain structure is *very different*:

The **sphere** is a *2D manifold*: the two degrees of freedom are *longitude* and *latitude*.

There are *three directions* to rotate the sphere, therefore **SO(3)** is a *3D manifold*.



Convolution on rotation SO(3) transformation

$$\Omega = \mathfrak{G} = \mathrm{SO}(3)$$

spherical signal x

Cohen, Welling 2016

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

Example: SO(3) *convolution (recap)*

Spherical convolutions return a function over **all 3D rotations** in SO(3)
So, to *"stack more layers"*, we need to convolve over $\Omega = \mathfrak{G} = \mathrm{SO}(3)$

Luckily, our blueprint still works: we can define a $\mathfrak{G}$-action over elements of $\mathfrak{G}$ by function composition: $(\mathfrak{g}, \mathfrak{h}) \to \mathfrak{g}\mathfrak{h}$

The corresponding group action over signals can be defined as follows:

$$(\rho(\mathfrak{g})x)(\mathfrak{h}) = x(\mathfrak{g}^{-1}\mathfrak{h})$$

This construction is often referred to as the *regular representation*

Consequently, we can build $\mathfrak{G}$-equivariant layers over $\mathfrak{G}$!

Start with spherical conv at the input, then **stack** SO(3)-convolutions!

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

Example: SO(3) *convolution, (recap)*

SO(3) group representations defined by 3x3 rotation matrices **R**

Hence, group action $(\rho(\mathfrak{g})x)(\mathfrak{h}) = x(\mathfrak{g}^{-1}\mathfrak{h})$ is expressible as $x(\mathbf{R}^{-1}\mathbf{Q})$ for some two rotation matrices **R** and **Q**

Recall the expression for the group conv: $(x \star \psi)(\mathfrak{g}) = \langle x, \rho(\mathfrak{g})\psi \rangle = \int_\Omega x(u)\psi(\mathfrak{g}^{-1}u)\, du$

Putting it together, we obtain the two-layer convolution over spheres:

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

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

Caveat 1: Tractability

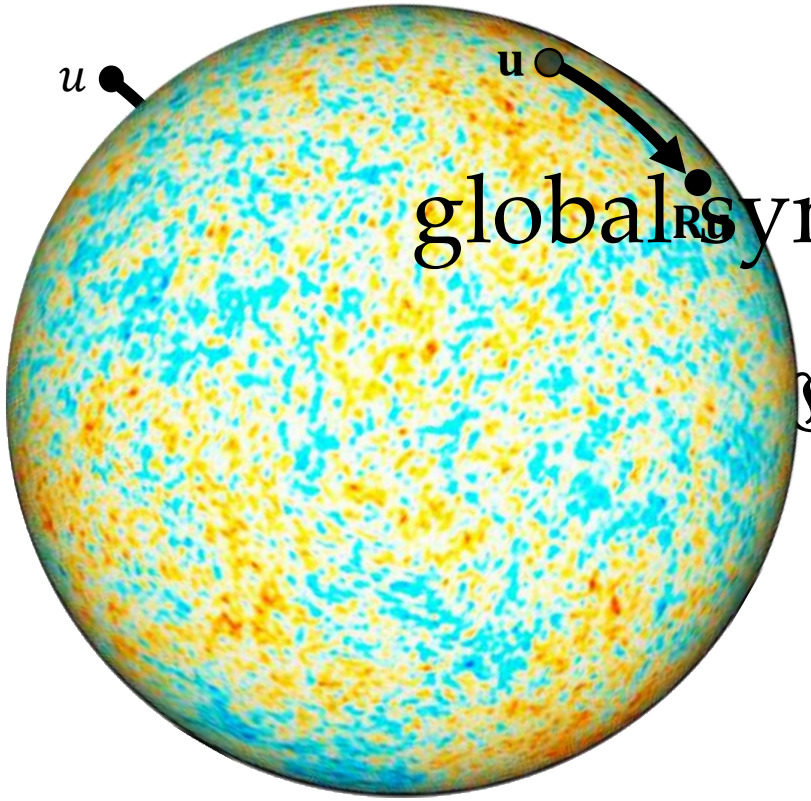
The $\mathfrak{G}$-convolution described here works over **any** symmetry group $\mathfrak{G}$

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 sets or graphs
But the permutation group Σ_n has $n!$ entries to maintain

Hints at existence of a *rich family* of equivariant convolutions
Some of these convolutions may tradeoff *expressivity* and *stability*

Caveat 2: Homogeneous Spaces



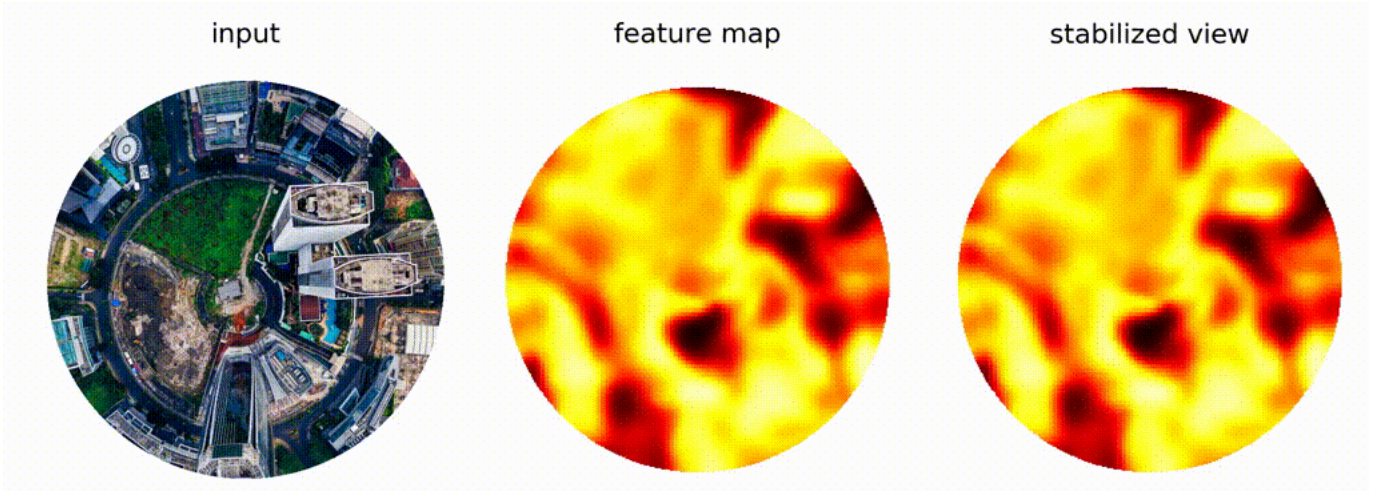
global symmetry group

$\mathfrak{G}$ s.t. $gu = v$

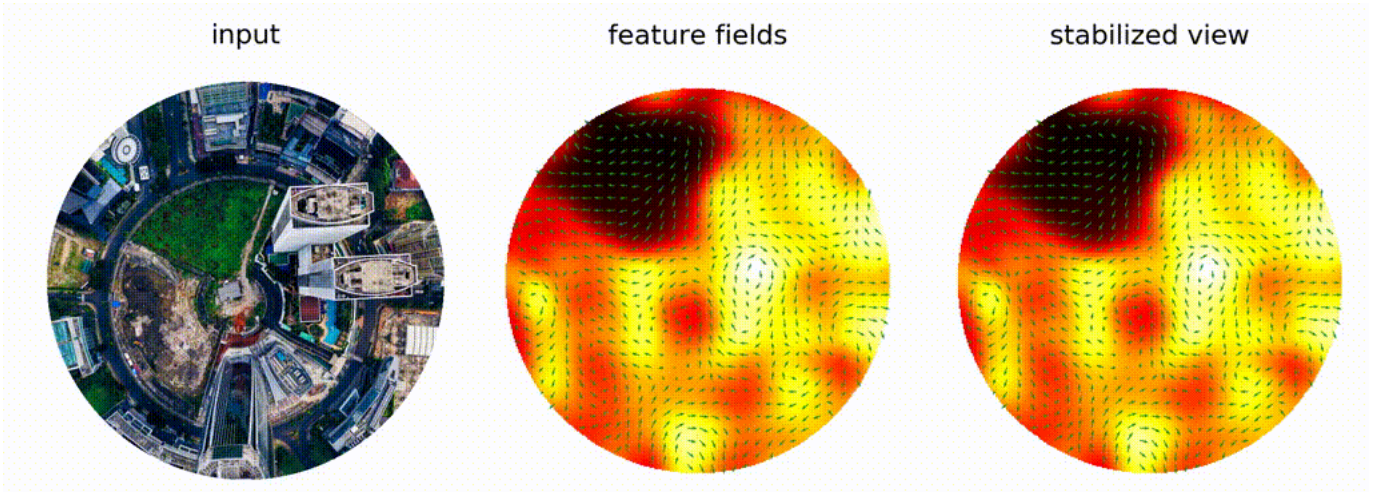
Rotation Equivariance in CNNs

A standard CNN is not rotational equivariant

CNN



G-CNN



Gabriele Cesa/Taco Cohen

Points to discuss

Augmentation and Manifolds
Fourier transform for rotational dynamics

What's next?

We've seen how to build:

Permutation *equivariant* networks over **sets** → Deep Sets
Translation *equivariant* networks over **grids** → CNNs
Rotation *equivariant* networks over **spheres** → Spherical CNNs

These were all domains with either very *limited* geometry
(*"no extra structure at all"* in the case of sets)
or very *rigid* geometry
(*"every point looks the same"* in the case of circular grids and spheres)

We will dedicate all of Lecture 4 to learning on **graphs**
Graphs have *very rich* structure which can be *highly irregular*
But *beautiful* to analyse ☺

Recap: Popular architectures as instances of GDL blueprint

Architecture	**Domain Ω**	**Symmetry Group $\mathfrak{G}$**
CNN	Grid	Translation
Spherical CNN	Sphere / $\mathrm{SO}(3)$	Rotation $\mathrm{SO}(3)$
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
E(3) GNN	Geometric Graph	Permutation Σ_n $\times$ Euclidean $E(3)$
LSTM	1D Grid	Time warping