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)
Felix Klein and the “Erlangen Program” (1872)

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

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!
...now it’s our turn to study geometry :)

Michael Bronstein
Oxford / Twitter

Joan Bruna
NYU

Taco Cohen
Qualcomm

Petar Veličković
DeepMind / Cambridge
Geometric Deep Learning
GNNs Beyond Permutation Equivariance

Petar Veličković
Stanford University
CS224W
30 November 2021
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?
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, \( \Omega \)
  - e.g. for images, \( u \in \Omega \) are pixels; for graphs they are nodes

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

- We can then define *featurised domains* using a space of feature functions \( X(\Omega, C) \)
  - \( x \in X(\Omega, C) \) is a function s.t. \( x(u) \in C \) gives features of element \( u \in \Omega \)
  - For discrete environments we can think of \( X \) as a feature matrix \( (X \in \mathbb{R}^{|\Omega| \times k}) \)
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 $\mathcal{g}: \Omega \to \Omega$).
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 $\mathcal{G}$ along with a binary operation $\circ : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ called **composition** (for brevity, denoted by juxtaposition $g \circ h = gh$) satisfying the following axioms:

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

**Identity:** there exists a unique $e \in \mathcal{G}$ satisfying $eg = ge = g$ for all $g \in \mathcal{G}$.

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

**Closure:** The group is closed under composition, i.e., for every $g, h \in \mathcal{G}$, we have $gh \in \mathcal{G}$. 
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 $\mathcal{G}$ along with a binary operation $\circ : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ called **composition** (for brevity, denoted by juxtaposition $g \circ h = gh$) satisfying the following axioms:

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

**Identity**: there exists a unique $e \in \mathcal{G}$ satisfying $eg = ge = g$ for all $g \in \mathcal{G}$.

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

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

- We are interested in how these groups affect data
  - Group action $(g, u) \mapsto g.u$ 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:
  $$g.(\alpha x + \beta x') = \alpha(g.x) + \beta(g.x')$$

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

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

- We can largely simplify high-dimensional learning by exploiting the symmetries in \(\mathcal{G}\)!

A function \(f : \mathcal{X}(\Omega) \rightarrow \mathcal{Y}\) is \(\mathcal{G}\)-invariant if \(f(\rho(g)x) = f(x)\) for all \(g \in \mathcal{G}\), 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 \(f : \mathcal{X}(\Omega) \rightarrow \mathcal{X}(\Omega)\) is \(\mathcal{G}\)-equivariant if \(f(\rho(g)x) = \rho(g)f(x)\) for all \(g \in \mathcal{G}\), 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
3 The Blueprint of Geometric Deep Learning
The key “building blocks” of Geometric Deep Learning

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

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

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

$\mathcal{G}$-invariant layer (global pooling) $A : \mathcal{X}(\Omega, \mathcal{C}) \rightarrow \mathcal{Y}$, satisfying $A(g. x) = A(x)$ for all $g \in \mathcal{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$. 
The key “building blocks” of Geometric Deep Learning

**Equivariant local layers**

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

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

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

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

**Invariant “tail”** (if necessary)
The key “building blocks” of Geometric Deep Learning

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

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

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

$\mathcal{G}$-invariant layer (global pooling) $A : \mathcal{X}(\Omega, \mathcal{C}) \rightarrow \mathcal{Y}$, satisfying $A(g \cdot x) = A(x)$ for all $g \in \mathcal{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$. 
The key “building blocks” of Geometric Deep Learning

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

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

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

$\mathcal{G}$-invariant layer (global pooling) $A : \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{Y}$, satisfying $A(g \cdot x) = A(x)$ for all $g \in \mathcal{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$. (Not covered here in detail, but follows from scale separation!)
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**! :)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Domain $\Omega$</th>
<th>Symmetry group $\mathcal{G}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Grid</td>
<td>Translation</td>
</tr>
<tr>
<td><em>Spherical CNN</em></td>
<td>Sphere / SO(3)</td>
<td>Rotation SO(3)</td>
</tr>
<tr>
<td><em>Intrinsic / Mesh CNN</em></td>
<td>Manifold</td>
<td>Isometry $\text{Iso}(\Omega)$ / Gauge symmetry SO(2)</td>
</tr>
<tr>
<td>GNN</td>
<td>Graph</td>
<td>Permutation $\Sigma_n$</td>
</tr>
<tr>
<td><em>Deep Sets</em></td>
<td>Set</td>
<td>Permutation $\Sigma_n$</td>
</tr>
<tr>
<td><em>Transformer</em></td>
<td>Complete Graph</td>
<td>Permutation $\Sigma_n$</td>
</tr>
<tr>
<td>LSTM</td>
<td>1D Grid</td>
<td>Time warping</td>
</tr>
</tbody>
</table>
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.
The “5G” of Geometric Deep Learning

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

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
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 $x_i \in \mathbb{R}^k$ be the features of node $i$.

- We can stack them into a node feature matrix of shape $n \times k$:

  $$X = (x_1, \ldots, x_n)^T$$

- That is, the $i$th row of $X$ corresponds to $x_i$

- 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?

\[ f \left( x_1, x_2, x_3, x_4, x_5 \right) = y \]
What do we want?

\[ f(x_1, x_2, x_3, x_4, x_5) = y = f(x_1, x_3, x_4, x_5) \]
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 $y_1 \leftarrow x_2$, $y_2 \leftarrow x_4$, $y_3 \leftarrow x_1$, $y_4 \leftarrow x_3$.

- To stay within linear algebra, each permutation defines an $n \times n$ 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:

\[
P_{(2,4,1,3)}X = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} \begin{bmatrix}
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\end{bmatrix} = \begin{bmatrix}
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\end{bmatrix} = \begin{bmatrix}
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\end{bmatrix} = \begin{bmatrix}
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\_ & \_ & \_ & \_ \\
\end{bmatrix}
\]
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 $P$:

$$f(PX) = f(X)$$

- One very generic form is the Deep Sets model (Zaheer et al., NeurIPS’17): $f(X) = \phi \left( \sum_{i \in V} \psi(x_i) \right)$ where $\psi$ and $\phi$ 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 $P$:  
  $$f(PX) = Pf(X)$$
Learning on graphs

- Now we augment the set of nodes with \textbf{edges} between them.
  - That is, we consider general $E \subseteq V \times V$.

- We can represent these edges with an \textit{adjacency matrix}, $A$, such that:

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

- Further additions (e.g. \textit{edge features}) are possible but \textbf{ignored} for simplicity.

- Our main desiderata (\textit{permutation \{in,equi\}variance}) still hold!
What’s changed?

\[ f \left( \begin{array}{c} x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \end{array} \right) = y = f \left( \begin{array}{c} x_2 \\ x_5 \\ x_4 \\ x_1 \\ x_3 \end{array} \right) \]
What’s changed?

\[ f \left( \begin{array}{c} x_5 \\ x_4 \\ x_3 \end{array} \right) = y = f \left( \begin{array}{c} x_2 \\ x_5 \\ x_4 \end{array} \right) \]

\[ f \left( \begin{array}{c} x_5 \\ x_4 \\ x_3 \end{array} \right) = y = f \left( \begin{array}{c} x_2 \\ x_5 \\ x_4 \end{array} \right) \]
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 $PAP^T$.

- We arrive at updated definitions of suitable functions $f(X, A)$ over graphs:

  **Invariance:** $f(PX, PAP^T) = f(X, A)$
  
  **Equivariance:** $f(PX, PAP^T) = Pf(X, A)$
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:
    \[ \mathcal{N}_i = \{ j : (i, j) \in \mathcal{E} \lor (j, i) \in \mathcal{E} \} \]
    
    **N.B.** we do not explicitly consider directed edges, and often we assume $i \in \mathcal{N}_i$.

- 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: $g(x_i, \mathbf{X}_{\mathcal{N}_i})$. 
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:

\[
\begin{bmatrix}
g(x_1, X_{N_1}) \\
g(x_2, X_{N_2}) \\
\vdots \\
g(x_n, X_{N_n}) 
\end{bmatrix}
\]

- To ensure equivariance, we need \( g \) to not depend on the order of the vertices in \( X_{Ni} \)
  - Hence, \( g \) should be permutation invariant!
A recipe for **graph** neural networks, visualised

\[ X_{N_b} = \{ x_a, x_b, x_c, x_d, x_e \} \]
How to use GNNs?
How to use GNNs?

Inputs
$(X, A)$

Latents
$(H, A)$

GNN
How to use GNNs?

Node classification: $z_i = f(h_i)$
How to use GNNs?

Inputs $(X, A)$

Latents $(H, A)$

Node classification $z_i = f(h_i)$

Graph classification $z_G = f(\bigoplus_{i \in V} h_i)$
How to use GNNs?

Inputs $(X, A)$

GNN

Latents $(H, A)$

Node classification $z_i = f(h_i)$

Graph classification $z_G = f(\oplus_{i \in V} h_i)$

Link prediction $z_{ij} = f(h_i, h_j, e_{ij})$
The three “flavours” of GNN layers

- Convolutional: \( h_i = \phi \left( x_i, \bigoplus_{j \in N_i} c_{ij} \psi(x_j) \right) \)
  - e.g. GraphSAGE, GCN, SGC

- Attentional: \( h_i = \phi \left( x_i, \bigoplus_{j \in N_i} a(x_i, x_j) \psi(x_j) \right) \)
  - e.g. MoNet, GAT, GATv2

- Message-passing: \( h_i = \phi \left( x_i, \bigoplus_{j \in N_i} \psi(x_i, x_j) \right) \)
  - 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
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)

\[ \mathcal{N}_i = \{i\} \]

\[ h_i = \psi(x_i) \]
Option 2: Assume \textit{all} edges

\[ \mathcal{N}_i = \mathcal{V} \]

Let the GNN decide which edges matter!

Using conv-GNNs no longer makes sense.

If we use \textit{attentional} GNNs we recover:

\[ h_i = \phi \left( x_i, \bigoplus_{j \in \mathcal{V}} a(x_i, x_j) \psi(x_j) \right) \]

Does this look familiar?

Interaction Nets (Battaglia \textit{et al.}, NeurIPS’16)
Relational Nets (Santoro \textit{et al.}, NeurIPS’17)

\textbf{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
Look to the Fourier transform

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

\[(x * y)(\xi) = \hat{x}(\xi) \cdot \hat{y}(\xi)\]

\[\hat{x}(\xi) = \int_{-\infty}^{+\infty} x(u)e^{-i\xi u} du\]

“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. $H = f(X) = CX$

$$f(X) = \begin{bmatrix} b & c & a \\ a & b & c \\ \vdots & \vdots & \vdots \\ c & a & b \end{bmatrix} \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ x_0 \\ \vdots \\ \vdots \\ x_{n-2} \\ \vdots \\ x_{n-1} \\ \vdots \end{bmatrix}$$
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**

\[
\phi_{\ell} = \frac{1}{\sqrt{n}} \left( 1, e^{\frac{2\pi i \ell}{n}}, e^{\frac{4\pi i \ell}{n}}, \ldots, e^{\frac{2\pi i (n-1)\ell}{n}} \right)^\top, \quad \ell = 0, 1, \ldots, n-1
\]

● This can be easily computed by studying $C([0, 1, 0, 0, 0, \ldots])$, which is the **shift** matrix.
The DFT and the convolution theorem

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

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

- The convolution theorem naturally follows:

\[
f(X) = C(\theta)X = \Phi \Lambda \Phi^* X = \Phi \begin{bmatrix} \hat{\theta}_0 \\ \vdots \\ \hat{\theta}_{n-1} \end{bmatrix} \Phi^* X = \Phi(\hat{\theta} \circ \hat{X})
\]

- Now, as long as we know \( \Phi \), we can express our convolution using \( \hat{\theta} \) rather than \( \theta \)
What we have covered so far

Key idea: we don’t need to know the circulant if we know its eigenvalues!
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”, $\Phi$, 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, $\Phi$ was always the same ($n$-way DFT).
  - Now every graph will have its own $\Phi$!

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

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

\[ L = 
\begin{bmatrix}
2 & -1 & 0 & 0 & -1 & 0 \\
-1 & 3 & -1 & 0 & -1 & 0 \\
0 & -1 & 2 & -1 & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & -1 \\
-1 & -1 & 0 & -1 & 3 & 0 \\
0 & 0 & 0 & -1 & 0 & 1 \\
\end{bmatrix} \]
Graph Fourier Transform

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

- This allows us to re-express $L = \Phi \Lambda \Phi^*$, as before.
  - Changing the eigenvalues in $\Lambda$ 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):

$$f(X) = \Phi \begin{bmatrix} \hat{\theta}_0 & \cdots & \hat{\theta}_{n-1} \end{bmatrix} \Phi^* X$$
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 $\Lambda$, the eigenvalues of $L$
  - Commonly by a degree-$k$ polynomial function, $p_k$
  - Yielding $f(x) = \Phi p_k(\Lambda) \Phi^* x = p_k(L)x$
  - 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 $c_{ij} = (p_k(L))_{ij}$
  - 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

\[
PE_{\text{pos},2i} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)
\]

\[
PE_{\text{pos},2i+1} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)
\]

- **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 \( \Phi \))
  - 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
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:

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

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

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

● Now we can see how to generalise the convolution to more general
Group convolutions

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

\[ (x \ast \psi)(u) = \langle x, T_u \psi \rangle = \int_{\mathbb{R}} x(v)\psi(u + v)dv \]  

\[ (x \ast \psi)(g) = \langle x, \rho(g)\psi \rangle = \int_{\Omega} x(u)\psi(g^{-1}u)du \]  

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

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

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

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

- We want to be **rotation equivariant**
  - This means using the rotation group $\mathfrak{g} = \text{SO}(3)$

- We can represent the points on a sphere with 3-dim unit vectors, $u$
  - The **group action** (rotation) transforms it with a $3 \times 3$ orthogonal matrix, $R$

- We recover our spherical convolution:

$$ (x * \psi)(R) = \int_{S^2} x(u)\psi(R^{-1}u)du $$
Example: SO(3) convolution

- 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 $\mathfrak{g} = \mathbb{O} = \text{SO}(3)$.

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

- The corresponding group action can be defined as follows: $(\rho(g)x)(h) = x(g^{-1}h)$.

- 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

- SO(3) group actions defined by 3x3 rotation matrices $\mathbf{R}$

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

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

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

\[
((x \ast \psi) \ast \phi)(\mathbf{R}) = \int_{\text{SO}(3)} (x \ast \psi)(\mathbf{Q}) \phi(\mathbf{R}^{-1}\mathbf{Q}) d\mathbf{Q}
\]

- 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 \textit{discretised} and the filter $\psi$ is \textit{local}
  ○ The non-zero elements of $\psi$ are the neighbourhood of point $u$
  ○ The integration is akin to message passing! (as many points will have zero product)

\[
(x \ast \psi)(R) = \int_{S^2} x(u)\psi(R^{-1}u)du
\]

● Further, the $G$-convolution described here works over \textbf{any} global symmetry group $G$
Caveat 1: Tractability

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

- 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 $\Sigma_n$ 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 homogenous

- For any two points $u, v \in \Omega$, there exists some $\varphi \in \mathcal{G}$ s.t. $\varphi(u) = v$

- In a sense, “all points look the same”
  - This does not hold for graphs!

- This allowed ‘sliding’ filters across $\Omega$ 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
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, \( f_u \) and coordinates \( x_u \in \mathbb{R}^3 \)

- An equivariant message passing layer transforms them separately
  - Yielding updated features \( f'_u \) and coordinates \( x'_u \)

- We can now express a group of symmetries \( \mathbb{G} \) 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 \( \rho(\mathbb{G})x = Rx + b \)
  - 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):

\[
\begin{align*}
\mathbf{f}'_u &= \phi \left( \mathbf{f}_u, \bigoplus_{v \in \mathcal{N}_u} \psi_f(\mathbf{f}_u, \mathbf{f}_v, \|\mathbf{x}_u - \mathbf{x}_v\|^2) \right), \\
\mathbf{x}'_u &= \mathbf{x}_u + \sum_{v \neq u} (\mathbf{x}_u - \mathbf{x}_v) \psi_c(\mathbf{f}_u, \mathbf{f}_v, \|\mathbf{x}_u - \mathbf{x}_v\|^2)
\end{align*}
\]

- The actions of E(n) do not change distance between nodes (they are isometric).
  - Hence if \( \mathbf{x}_u \leftarrow R\mathbf{x}_u + \mathbf{b} \ldots \)
  - \( \mathbf{f}'_u \) does not change, and \( \mathbf{x}'_u \leftarrow R\mathbf{x}'_u + \mathbf{b} \), 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 \( (f_u) \) 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 ($f_u$) 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:
  \[
  \begin{align*}
  \mathbf{v}_i^{l+1} &= \phi_v (\mathbf{h}_i^l) \mathbf{v}_i^l + C \sum_{j \neq i} (\mathbf{x}_i^l - \mathbf{x}_j^l) \phi_x (\mathbf{m}_{ij}) \\
  \mathbf{x}_i^{l+1} &= \mathbf{x}_i^l + \mathbf{v}_i^{l+1}
  \end{align*}
  \]

- 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 $l$!

$$\rho(g) = Q^\top \left( \bigoplus_{\ell} D_{\ell}(g) \right) Q$$

where $D_\ell$ are the Wigner D-matrices, and $Q$ is a $g$-specific change-of-basis matrix.

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

$$f_{out, i}^\ell = \sum_{k \geq 0} \int_{k \rightarrow \ell \text{ convolution}} W_{\ell k}(x' - x_i) f_{in}^{\ell k}(x') \, dx' = \sum_{k \geq 0} \sum_{j=1}^{n} W_{\ell k}(x_j - x_i) f_{in, j}^{\ell k},$$

where $W_{\ell k}(x) = \sum_{J = |k - \ell|}^{k+\ell} \varphi_{\ell k}^{\overline{J}}(\|x\|) W_{\ell k}^{J}(x)$, and $W_{\ell k}^{J}(x) = \sum_{m=-J}^{J} Y_{Jm}(x/\|x\|) Q_{\ell k}^{Jm}$

where the matrices $W_J$ 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.

$$f_{\text{out}, i}^\ell = \underbrace{W_{V}^{\ell \ell} f_{\text{in}, i}^\ell}_{\text{3: self-interaction}} + \sum_{k \geq 0} \sum_{j \in N_i \setminus i} \alpha_{i,j} \underbrace{W_{V}^{\ell k}(x_j - x_i)f_{\text{in}, j}^k}_{\text{1: attention}} \underbrace{W_{V}^{\ell k}(x_j - x_i)f_{\text{in}, j}^k}_{\text{2: value message}}.$$

$$\alpha_{i,j} = \frac{\exp(q_i^\top k_{ij})}{\sum_{j' \in N_i \setminus i} \exp(q_i^\top k_{ij'})}, \quad q_i = \bigoplus_{\ell \geq 0} \sum_{k \geq 0} W_Q^{\ell k} f_{\text{in}, i}^k, \quad k_{ij} = \bigoplus_{\ell \geq 0} \sum_{k \geq 0} W_K^{\ell k}(x_j - x_i)f_{\text{in}, j}^k.$$

Gives all possible equivariant (attentional) GNNs, but we still need to pre-compute $W_j$. 
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

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

Step 4: Compute attention and aggregate

\[ \alpha_{ij} = \frac{\exp(q_i^T k_{ij})}{\sum_j \exp(q_i^T k_{ij})} \]

\[ f'_{out,i} = \sum_{j \in N \setminus i} \alpha_{ij} v_{ij} \]
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  \((\text{Masci et al., CVPR'15})\)
- Gauge-equivariant Mesh CNN  \((\text{de Haan et al., ICLR'21})\)
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)

\[ h_u = \Theta_{\text{self}} x_u + \sum_{v \in N_u} \Theta_{\text{neigh}}(\theta_{uv}) \rho(\varphi_{v\rightarrow u}) x_v \]
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
156-page proto book on arXiv

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

Michael M. Bronstein\textsuperscript{1}, Joan Bruna\textsuperscript{2}, Taco Cohen\textsuperscript{3}, Petar Veličković\textsuperscript{4}

May 4, 2021
# African Master’s in Machine Intelligence course (12h)

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Title</th>
<th>Instructor</th>
<th>Recording</th>
<th>Slides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture 1</td>
<td>Introduction</td>
<td>Michael M. Bronstein</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 2</td>
<td>High-Dimensional Learning</td>
<td>Joan Bruna</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 3</td>
<td>Geometric Priors I</td>
<td>Taco Cohen</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 4</td>
<td>Geometric Priors II</td>
<td>Joan Bruna</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 5</td>
<td>Graphs &amp; Sets I</td>
<td>Petar Veličković</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 6</td>
<td>Graphs &amp; Sets II</td>
<td>Petar Veličković</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 7</td>
<td>Grids</td>
<td>Joan Bruna</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 8</td>
<td>Groups</td>
<td>Taco Cohen</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 9</td>
<td>Geodesics &amp; Manifolds</td>
<td>Michael M. Bronstein</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 10</td>
<td>Gauges</td>
<td>Taco Cohen</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 11</td>
<td>Sequences &amp; Time Warping</td>
<td>Petar Veličković</td>
<td>Recording</td>
<td>Slides</td>
</tr>
<tr>
<td>Lecture 12</td>
<td>Conclusions</td>
<td>Michael M. Bronstein</td>
<td>Recording</td>
<td>Slides</td>
</tr>
</tbody>
</table>
Thank you!

Questions / Feedback?

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

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