Geometric Deep Learning: An Introduction

Benjamin Fattori

April 6, 2022

Outline

- Goal of Geometric Deep Learning
- Why Geometric Deep Learning?
- Groups and Symmetry
- Geometric Priors - Symmetry
- Geometric Priors - Scale Separation
- Geometric Deep Learning Blueprint
- Graph Neural Networks
- My Review of the "Proto Book"

Introduction

- **Geometric Deep Learning** (GDL) is a term for the techniques used to extend deep learning methods to non-euclidean structures such as graphs, meshes, and manifolds by incorporating the symmetries of the underlying domain.

Introduction

- **Geometric Deep Learning** (GDL) is a term for the techniques used to extend deep learning methods to non-euclidean structures such as graphs, meshes, and manifolds by incorporating the symmetries of the underlying domain.
- Ideas from CNNs such as shift invariance can generalize to different types of data, hinting at a more general framework for Deep Learning.

Introduction

- **Geometric Deep Learning** (GDL) is a term for the techniques used to extend deep learning methods to non-euclidean structures such as graphs, meshes, and manifolds by incorporating the symmetries of the underlying domain.
- Ideas from CNNs such as shift invariance can generalize to different types of data, hinting at a more general framework for Deep Learning.
- First occurrence of GDL is in the 2016 paper, "*Geometric deep learning: going beyond Euclidean data*" by Bronstein et al.

Why Geometric Deep Learning?

1. High-dimensional learning is very difficult - exploiting symmetries/prior knowledge of our data can help by reducing the size of our hypothesis space to functions that respect these symmetries.

Why Geometric Deep Learning?

1. High-dimensional learning is very difficult - exploiting symmetries/prior knowledge of our data can help by reducing the size of our hypothesis space to functions that respect these symmetries.
2. GDL defines a blueprint/procedure for constructing new deep learning models on non-euclidean inputs by incorporating their natural symmetries into the model design process.

Why Geometric Deep Learning?

1. High-dimensional learning is very difficult - exploiting symmetries/prior knowledge of our data can help by reducing the size of our hypothesis space to functions that respect these symmetries.
2. GDL defines a blueprint/procedure for constructing new deep learning models on non-euclidean inputs by incorporating their natural symmetries into the model design process.
3. GDL provides a common mathematical framework (Group Theory) from which to study and compare different DL architectures such as CNNs, RNNs, and Transformers.

Why Geometric Deep Learning?

1. High-dimensional learning is very difficult - exploiting symmetries/prior knowledge of our data can help by reducing the size of our hypothesis space to functions that respect these symmetries.
2. GDL defines a blueprint/procedure for constructing new deep learning models on non-euclidean inputs by incorporating their natural symmetries into the model design process.
3. GDL provides a common mathematical framework (Group Theory) from which to study and compare different DL architectures such as CNNs, RNNs, and Transformers.
4. Natural symmetries are commonplace in many branches of science including physics and chemistry (Conservation of Energy and Angular Momentum, Molecular Symmetry) and as such is a natural idea to bring into deep learning.

Preliminary Definitions - Domain and Signal

- Ω is our domain which is a set with (possible) additional structure.
- On Ω, we can define $\mathcal{X}(\Omega, \mathcal{C})$ to be the space of all $\mathcal{C}$-valued signals on Ω.
- Example: $\Omega = n \times n$ grid, then $v \in \mathcal{X}(\Omega, \mathbb{R}^3)$ is an RGB image.

Preliminary Definitions - Groups and Symmetry

- Informally a *symmetry* of an object is a transformation to that object that leaves certain properties unchanged. The mathematical formalism of a symmetry is called a **Group**. A Group has four defining properties:

Preliminary Definitions - Groups and Symmetry

- Informally a *symmetry* of an object is a transformation to that object that leaves certain properties unchanged. The mathematical formalism of a symmetry is called a **Group**. A Group has four defining properties:
  - **Identity**: The 'do nothing' transformation is a symmetry

Preliminary Definitions - Groups and Symmetry

- Informally a *symmetry* of an object is a transformation to that object that leaves certain properties unchanged. The mathematical formalism of a symmetry is called a **Group**. A Group has four defining properties:
  - **Identity**: The 'do nothing' transformation is a symmetry
  - **Inverse**: Every transformation can be reversed by applying the inverse transformation

Preliminary Definitions - Groups and Symmetry

- Informally a *symmetry* of an object is a transformation to that object that leaves certain properties unchanged. The mathematical formalism of a symmetry is called a **Group**. A Group has four defining properties:
  - **Identity**: The 'do nothing' transformation is a symmetry
  - **Inverse**: Every transformation can be reversed by applying the inverse transformation
  - **Associativity**: Rearranging the parentheses in an expression will not change the result

Preliminary Definitions - Groups and Symmetry

- Informally a *symmetry* of an object is a transformation to that object that leaves certain properties unchanged. The mathematical formalism of a symmetry is called a **Group**. A Group has four defining properties:
  - **Identity**: The 'do nothing' transformation is a symmetry
  - **Inverse**: Every transformation can be reversed by applying the inverse transformation
  - **Associativity**: Rearranging the parentheses in an expression will not change the result
  - **Closure**: The composition of any two symmetries is itself a symmetry.

Preliminary Definitions - Groups and Symmetry

- Informally a *symmetry* of an object is a transformation to that object that leaves certain properties unchanged. The mathematical formalism of a symmetry is called a **Group**. A Group has four defining properties:
  - **Identity**: The 'do nothing' transformation is a symmetry
  - **Inverse**: Every transformation can be reversed by applying the inverse transformation
  - **Associativity**: Rearranging the parentheses in an expression will not change the result
  - **Closure**: The composition of any two symmetries is itself a symmetry.
- **Examples**: The symmetry group of the circle is all rotations, the symmetry group of a set is all permutations of the elements.

Preliminary Definitions - Groups and Symmetry

- Informally a *symmetry* of an object is a transformation to that object that leaves certain properties unchanged. The mathematical formalism of a symmetry is called a **Group**. A Group has four defining properties:
  - **Identity**: The 'do nothing' transformation is a symmetry
  - **Inverse**: Every transformation can be reversed by applying the inverse transformation
  - **Associativity**: Rearranging the parentheses in an expression will not change the result
  - **Closure**: The composition of any two symmetries is itself a symmetry.
- **Examples**: The symmetry group of the circle is all rotations, the symmetry group of a set is all permutations of the elements.
- We can extend these ideas to our domains and signals by introducing the concept of a **Group Action** which gives a rule telling us how our domain and symmetry group interact.

Examples

The following are all examples of problems that GDL can (and has) been applied to

- **Classifying an image** - The class of objects in an image are *invariant* under shifts (*Euclidean Group*)[1]

Examples

The following are all examples of problems that GDL can (and has) been applied to

- **Classifying an image** - The class of objects in an image are *invariant* under shifts (*Euclidean Group*)[1]
- **Graph Labelling** - Predicting the label of some node in a graph should be invariant under all permutations of the nodes (*Symmetric Group*) [2]

Examples

The following are all examples of problems that GDL can (and has) been applied to

- **Classifying an image** - The class of objects in an image are *invariant* under shifts (*Euclidean Group*)[1]
- **Graph Labelling** - Predicting the label of some node in a graph should be invariant under all permutations of the nodes (*Symmetric Group*) [2]
- **Predicting chemical properties from molecules** - Chemical properties are independent of a molecule's position and rotation (*Special Orthogonal Group*) [3]

Examples

The following are all examples of problems that GDL can (and has) been applied to

- **Classifying an image** - The class of objects in an image are *invariant* under shifts (*Euclidean Group*)[1]
- **Graph Labelling** - Predicting the label of some node in a graph should be invariant under all permutations of the nodes (*Symmetric Group*) [2]
- **Predicting chemical properties from molecules** - Chemical properties are independent of a molecule's position and rotation (*Special Orthogonal Group*) [3]
- **Classification of Radio Galaxies** - Incoming signals to Earth can be represented on the sphere - these signals are independent of their rotation in space (*Special Orthogonal Group*) [4]

Examples

The following are all examples of problems that GDL can (and has) been applied to

- **Classifying an image** - The class of objects in an image are *invariant* under shifts (*Euclidean Group*)[1]
- **Graph Labelling** - Predicting the label of some node in a graph should be invariant under all permutations of the nodes (*Symmetric Group*) [2]
- **Predicting chemical properties from molecules** - Chemical properties are independent of a molecule's position and rotation (*Special Orthogonal Group*) [3]
- **Classification of Radio Galaxies** - Incoming signals to Earth can be represented on the sphere - these signals are independent of their rotation in space (*Special Orthogonal Group*) [4]

Invariant and Equivariant Functions

Once we have a specific domain, Ω and symmetry group, $\mathcal{G}$, we want to build our function by adding *layers* of functions that "respect" the symmetries of Ω. There are two types of functions that can do this:

- **Invariant Functions**: The output of an invariant function is equal for all symmetries of the input.

Invariant and Equivariant Functions

Once we have a specific domain, Ω and symmetry group, $\mathcal{G}$, we want to build our function by adding *layers* of functions that "respect" the symmetries of Ω. There are two types of functions that can do this:

- **Invariant Functions**: The output of an invariant function is equal for all symmetries of the input.
- **Equivariant Functions**: The change in the output of an equivariant function is proportional to the change of the input.

Invariance and Equivariance in CNNs

Invariant and equivariant layers are not new to Deep Learning and are incorporated into Convolutional Neural Networks (CNNs):

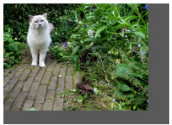
- Convolutional layers in CNNs are **equivariant** to translation of the input images.

Invariance and Equivariance in CNNs

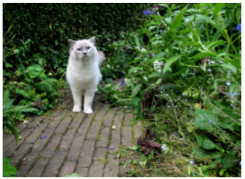
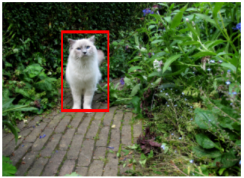
Invariant and equivariant layers are not new to Deep Learning and are incorporated into Convolutional Neural Networks (CNNs):

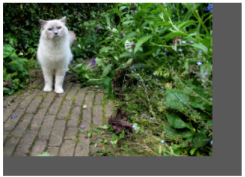
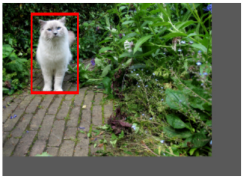
- Convolutional layers in CNNs are **equivariant** to translation of the input images.
- Adding global pooling after the convolutional layers (usually in the form of Global-Average Pooling) makes CNNs **invariant** to translations of the input images.

Invariance Visualized

$$\mathrm{F}(\text{  }) = \mathrm{F}(\text{  })$$

Equivariance Visualized

$$\mathbf{F}\left(\text{  }\right) = \text{  }$$

$$\mathbf{F}\left(\text{  }\right) = \text{  }$$

Deformation Stability I

In reality, asking for global symmetries of the domain is a bit unrealistic

- Nature is rarely going to exactly respect symmetries.

Deformation Stability I

In reality, asking for global symmetries of the domain is a bit unrealistic

- Nature is rarely going to exactly respect symmetries.
- Local deformations of data are not represented. This leaves out cases where input information is almost the same, but no global symmetry can describe the relation between the two signals.

Deformation Stability I

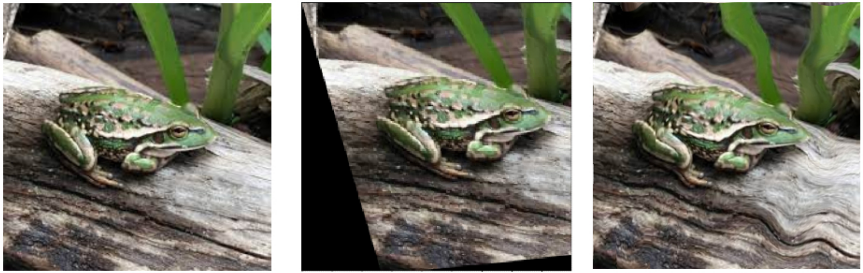
In reality, asking for global symmetries of the domain is a bit unrealistic

- Nature is rarely going to exactly respect symmetries.
- Local deformations of data are not represented. This leaves out cases where input information is almost the same, but no global symmetry can describe the relation between the two signals.

(4 frame difference)

Deformation Stability II

Instead of just considering global, exact symmetries, we want to build layers that are invariant (equivariant) to these local symmetries too. The authors call these **local deformations**:



Scale Separation I

Even restricting to functions respecting our symmetries is still not enough:

- Still too many invariant (equivariant) functions!

Scale Separation I

Even restricting to functions respecting our symmetries is still not enough:

- Still too many invariant (equivariant) functions!
- Can be computationally difficult (Ex: CNN without any pooling or downsampling)

Scale Separation I

Even restricting to functions respecting our symmetries is still not enough:

- Still too many invariant (equivariant) functions!
- Can be computationally difficult (Ex: CNN without any pooling or downsampling)

We can try to improve on this by exploiting the **multi-scale structure** of our learning problems, a popular approach in many domains of science.

Scale Separation II

GDL incorporates scale separation priors by introducing **Locally Equivariant Functions** and **Coarsening Functions**:

Scale Separation II

GDL incorporates scale separation priors by introducing **Locally Equivariant Functions** and **Coarsening Functions**:

- Locally Equivariant Function: equivariant functions with local support (Ex: convolutional kernels with small support, such as 3x3 filters)
- Coarsening Function: function assimilating "close" points (Ex: Local max-pooling layers in CNNs to downsample image/filter dimensions)

Scale Separation II

GDL incorporates scale separation priors by introducing **Locally Equivariant Functions** and **Coarsening Functions**:

- Locally Equivariant Function: equivariant functions with local support (Ex: convolutional kernels with small support, such as 3x3 filters)
- Coarsening Function: function assimilating "close" points (Ex: Local max-pooling layers in CNNs to downsample image/filter dimensions)
- A single layer of local filters cannot learn long range interactions but stacking many of such layers allows the model to propagate from local to global interactions

Scale Separation II

GDL incorporates scale separation priors by introducing **Locally Equivariant Functions** and **Coarsening Functions**:

- Locally Equivariant Function: equivariant functions with local support (Ex: convolutional kernels with small support, such as 3x3 filters)
- Coarsening Function: function assimilating "close" points (Ex: Local max-pooling layers in CNNs to downsample image/filter dimensions)
- A single layer of local filters cannot learn long range interactions but stacking many of such layers allows the model to propagate from local to global interactions
- Composing these operators allows us to build up equivariant functions that are computationally efficient while still respecting the symmetries and deformations of the signals.

Blueprint of Geometric Deep Learning

Let Ω be the domain and $\mathcal{G}$ be a symmetry group for Ω. We consider Ω' to be a coarsened version of Ω. We define the following building blocks:

- Linear $\mathcal{G}$-equivariant layer, B

Blueprint of Geometric Deep Learning

Let Ω be the domain and $\mathcal{G}$ be a symmetry group for Ω. We consider Ω' to be a coarsened version of Ω. We define the following building blocks:

- Linear $\mathcal{G}$-equivariant layer, B
- Non-linearity σ applied elementwise, needed for learning non-linear functions

Blueprint of Geometric Deep Learning

Let Ω be the domain and $\mathcal{G}$ be a symmetry group for Ω. We consider Ω' to be a coarsened version of Ω. We define the following building blocks:

- Linear $\mathcal{G}$-equivariant layer, B
- Non-linearity σ applied elementwise, needed for learning non-linear functions
- Coarsening Layer (Local Pooling), P

Blueprint of Geometric Deep Learning

Let Ω be the domain and $\mathcal{G}$ be a symmetry group for Ω. We consider Ω' to be a coarsened version of Ω. We define the following building blocks:

- Linear $\mathcal{G}$-equivariant layer, B
- Non-linearity σ applied elementwise, needed for learning non-linear functions
- Coarsening Layer (Local Pooling), P
- $\mathcal{G}$-invariant layer (Global Pooling), A

Blueprint of Geometric Deep Learning

Let Ω be the domain and $\mathcal{G}$ be a symmetry group for Ω. We consider Ω' to be a coarsened version of Ω. We define the following building blocks:

- Linear $\mathcal{G}$-equivariant layer, B
- Non-linearity σ applied elementwise, needed for learning non-linear functions
- Coarsening Layer (Local Pooling), P
- $\mathcal{G}$-invariant layer (Global Pooling), A

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

$$f = A \circ \sigma_J \circ B_J \circ P_{J-1} \circ \cdots \circ P_1 \circ \sigma_1 \circ B_1$$

Deriving Neural Networks on Graphs I

- Graphs are very general structures that can appear in many domains including Chemistry, Physics, Social Networks

Deriving Neural Networks on Graphs I

- Graphs are very general structures that can appear in many domains including Chemistry, Physics, Social Networks
- Many different tasks on graph-structured data:

Deriving Neural Networks on Graphs I

- Graphs are very general structures that can appear in many domains including Chemistry, Physics, Social Networks
- Many different tasks on graph-structured data:
  - **Graph-Level**: Predicting the properties/labels for the entire graph (Ex: Predicting chemical properties of a molecule)

Deriving Neural Networks on Graphs I

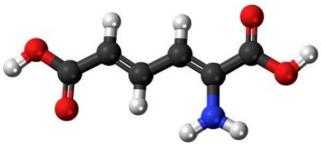
- Graphs are very general structures that can appear in many domains including Chemistry, Physics, Social Networks
- Many different tasks on graph-structured data:
  - **Graph-Level**: Predicting the properties/labels for the entire graph (Ex: Predicting chemical properties of a molecule)
  - **Node-Level**: Predicting the properties/labels for a specific node (Ex: Classification of individuals in a social network)

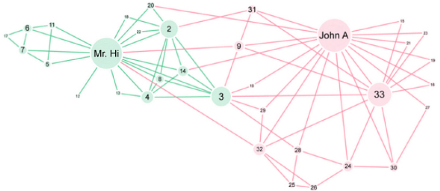
Deriving Neural Networks on Graphs I

- Graphs are very general structures that can appear in many domains including Chemistry, Physics, Social Networks
- Many different tasks on graph-structured data:
  - **Graph-Level**: Predicting the properties/labels for the entire graph (Ex: Predicting chemical properties of a molecule)
  - **Node-Level**: Predicting the properties/labels for a specific node (Ex: Classification of individuals in a social network)
  - **Edge-Level**: Predicting whether two nodes are linked by an edge (Ex: Predicting connections or relations between individuals in social networks)

Deriving Neural Networks on Graphs I

- Graphs are very general structures that can appear in many domains including Chemistry, Physics, Social Networks
- Many different tasks on graph-structured data:
  - **Graph-Level**: Predicting the properties/labels for the entire graph (Ex: Predicting chemical properties of a molecule)
  - **Node-Level**: Predicting the properties/labels for a specific node (Ex: Classification of individuals in a social network)
  - **Edge-Level**: Predicting whether two nodes are linked by an edge (Ex: Predicting connections or relations between individuals in social networks)





Deriving Neural Networks on Graphs II

- Graph, $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a collection of nodes, $\mathcal{N}$ and edges, $\mathcal{E}$

Deriving Neural Networks on Graphs II

- Graph, $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a collection of nodes, $\mathcal{N}$ and edges, $\mathcal{E}$
- Node features are stacked into a design matrix, $X \in \mathbb{R}^{n \times d}$

Deriving Neural Networks on Graphs II

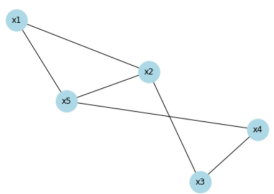
- Graph, $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a collection of nodes, $\mathcal{N}$ and edges, $\mathcal{E}$
- Node features are stacked into a design matrix, $X \in \mathbb{R}^{n \times d}$
- Edges are encoded as an Adjacency Matrix, A, where $a_{uv} = 1$ when $(u, v) \in \mathcal{E}$.

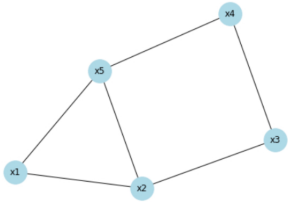
Deriving Neural Networks on Graphs II

- Graph, $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a collection of nodes, $\mathcal{N}$ and edges, $\mathcal{E}$
- Node features are stacked into a design matrix, $X \in \mathbb{R}^{n \times d}$
- Edges are encoded as an Adjacency Matrix, A, where $a_{uv} = 1$ when $(u, v) \in \mathcal{E}$.
- **Key Property**: The nodes in a graph do not have a specific ordering.

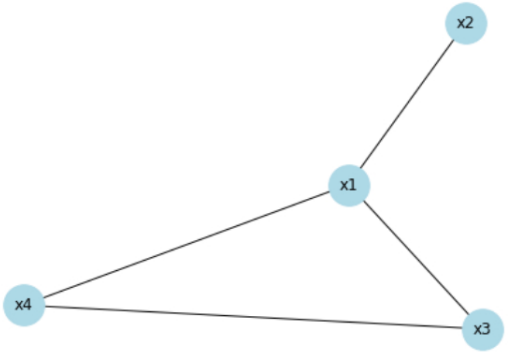
Deriving Neural Networks on Graphs II

- Graph, $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is a collection of nodes, $\mathcal{N}$ and edges, $\mathcal{E}$
- Node features are stacked into a design matrix, $X \in \mathbb{R}^{n \times d}$
- Edges are encoded as an Adjacency Matrix, A, where $a_{uv} = 1$ when $(u, v) \in \mathcal{E}$.
- **Key Property**: The nodes in a graph do not have a specific ordering.





Graph and Adjacency Matrix



$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

The Symmetric Group

- The permutations of a set of n elements is called the Symmetric Group, S_n. This can act on our graph by permuting the order of the nodes.

The Symmetric Group

- The permutations of a set of n elements is called the Symmetric Group, S_n. This can act on our graph by permuting the order of the nodes.
- An element from S_n must act on both the node features, X, *and* the adjacency matrix, A.

The Symmetric Group

- The permutations of a set of n elements is called the Symmetric Group, S_n. This can act on our graph by permuting the order of the nodes.
- An element from S_n must act on both the node features, X, *and* the adjacency matrix, A.
- Example: $P = P_{(2413)} \in S_4$ maps 1 to 2, 2 to 4, etc.

The Symmetric Group

- The permutations of a set of n elements is called the Symmetric Group, S_n. This can act on our graph by permuting the order of the nodes.
- An element from S_n must act on both the node features, X, *and* the adjacency matrix, A.
- Example: $P = P_{(2413)} \in S_4$ maps 1 to 2, 2 to 4, etc.
- We can represent this transformation as a matrix multiplication:

$$PX = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_4 \\ x_1 \\ x_3 \end{pmatrix}$$

- PAP^T is the adjacency matrix of the transformed graph, PX.

Recipe for Graph Neural Networks I

- Locality in graphs is defined by **Node Neighbourhoods**:

$$X_{\mathcal{N}_u} = \{x_v : (u, v) \in \mathcal{E}\} \text{ (1-hop neighbourhood)}$$

Recipe for Graph Neural Networks I

- Locality in graphs is defined by **Node Neighbourhoods**:

$$X_{\mathcal{N}_u} = \{x_v : (u, v) \in \mathcal{E}\} \text{ (1-hop neighbourhood)}$$

- We define a local function, ϕ operating on a node, x_u and its 1-hop neighbourhood $X_{\mathcal{N}_u}$.

Recipe for Graph Neural Networks I

- Locality in graphs is defined by **Node Neighbourhoods**:

$$X_{\mathcal{N}_u} = \{x_v : (u, v) \in \mathcal{E}\} \text{ (1-hop neighbourhood)}$$

- We define a local function, ϕ operating on a node, x_u and its 1-hop neighbourhood $X_{\mathcal{N}_u}$.
- Applying ϕ to every node in our graph allows us to build a permutation equivariant layer:

$$F(X, A) = \begin{bmatrix} -\phi(x_1, X_{\mathcal{N}_1})- \\ \vdots \\ -\phi(x_n, X_{\mathcal{N}_n})- \end{bmatrix}$$

Recipe for Graph Neural Networks I

- Locality in graphs is defined by **Node Neighbourhoods**:

$$X_{\mathcal{N}_u} = \{x_v : (u, v) \in \mathcal{E}\} \text{ (1-hop neighbourhood)}$$

- We define a local function, ϕ operating on a node, x_u and its 1-hop neighbourhood $X_{\mathcal{N}_u}$.
- Applying ϕ to every node in our graph allows us to build a permutation equivariant layer:

$$F(X, A) = \begin{bmatrix} -\phi(x_1, X_{\mathcal{N}_1})- \\ \vdots \\ -\phi(x_n, X_{\mathcal{N}_n})- \end{bmatrix}$$

- To ensure permutation equivariance of F, ϕ must be permutation **invariant**.

Recipe for Graph Neural Networks II

- **Convolutional**[2]: Neighbourhood features aggregation is done with **fixed** weights:

$$\text{Ex: } \phi(x_j, X_{\mathcal{N}_j}) = \sum_{i \in \mathcal{N}(j)} c_{ij} \psi(x_i)$$

Recipe for Graph Neural Networks II

- **Convolutional**[2]: Neighbourhood features aggregation is done with **fixed** weights:

$$\text{Ex: } \phi(x_j, X_{\mathcal{N}_j}) = \sum_{i \in \mathcal{N}(j)} c_{ij} \psi(x_i)$$

- **Attentional**[5]: Neighbourhood features aggregated with **learned** weights:

$$\text{Ex: } \phi(x_j, X_{\mathcal{N}_j}) = \sum_{i \in \mathcal{N}(j)} \alpha(x_i, x_j) \psi(x_i)$$

Recipe for Graph Neural Networks II

- **Convolutional**[2]: Neighbourhood features aggregation is done with **fixed** weights:

$$\text{Ex: } \phi(x_j, X_{\mathcal{N}_j}) = \sum_{i \in \mathcal{N}(j)} c_{ij} \psi(x_i)$$

- **Attentional**[5]: Neighbourhood features aggregated with **learned** weights:

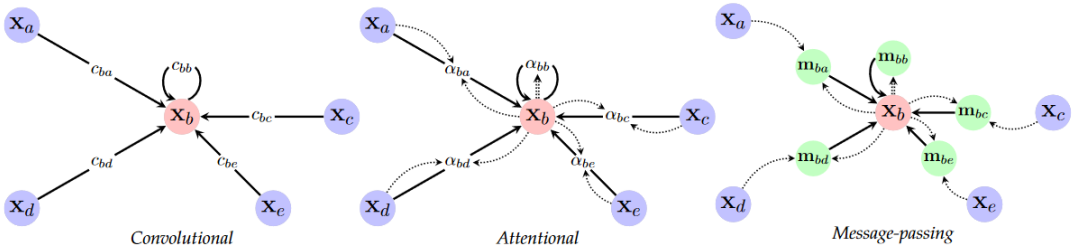
$$\text{Ex: } \phi(x_j, X_{\mathcal{N}_j}) = \sum_{i \in \mathcal{N}(j)} \alpha(x_i, x_j) \psi(x_i)$$

- **Message-Passing**[3]: Neighbourhood features aggregated through vector-valued function, φ:

$$\text{Ex: } \phi(x_j, X_{\mathcal{N}_j}) = \sum_{i \in \mathcal{N}(j)} \varphi(x_i, x_j)$$

Summary of Graph Neural Networks

- GNN layers are built by constructing locally equivariant functions operating on the 1-hop neighbourhoods of a node.
- Stacking GNN layers allows propagating of interactions across larger distances.
- **Connection**: Transformers are attentional GNNs on a complete graph.
- Increasing generality: convolutional $\subseteq$ attentional $\subseteq$ message-passing



GNNs in Practice

- Pinterest (Convolutional)
- Uber Eats (Convolutional)
- Google Maps (Attentional)

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

initial layer 0 embeddings are equal to node features

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k > 0$$

k^{th} layer embedding of v

non-linearity

average of neighbor's previous layer embeddings

previous layer embedding of v

$$z_v = \mathbf{h}_v^{last}$$

Review

The book was a bit of a mixed bag:

- **The Good**: Interesting concepts, clear presentation of CNNs, GNNs, good motivation.

Review

The book was a bit of a mixed bag:

- **The Good**: Interesting concepts, clear presentation of CNNs, GNNs, good motivation.
- **The Less-Good**: Authors make some large claims and don't expand/cite them. Claim book is useful for ML practitioners but do not expand on this.

Thanks for Listening!

More GDL Links

- Interactive article on GNNs
- GDL100 Lectures (By authors of the text)
- Practical GNN/GDL Tutorials: One, Two
- PyTorch Geometric Tutorials

[1] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[2] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[3] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].

[4] Anna M M Scaife and Fiona Porter. "Fanaroff–Riley classification of radio galaxies using group-equivariant convolutional neural networks". In: *Monthly Notices of the Royal Astronomical Society* 503.2 (Feb. 2021), pp. 2369–2379. ISSN: 0035-8711. DOI: 10.1093/mnras/stab530. eprint: https://academic.oup.com/mnras/article-pdf/503/2/2369/36733567/stab530.pdf. URL: https://doi.org/10.1093/mnras/stab530.

[5] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].