Graphs in Machine Learning

Daniele Calandriello

DeepMind Paris, France

Collaborators: Achraf Azize,
Michal Valko

Based on material by: Petar Veličković

https://petar-v.com/communications.html

27 Feb, 2022

MVA 2022/2023

Revisiting SL on graphs

modern problems require modern solutions

Supervised learning on graphs

Perspective so far: one graph, graph is the *bias*

Supervised learning on graphs

Perspective so far: one graph, graph is the *bias*

More perspectives today:

- many graphs, graphs are the *data*
- many sub-graphs, graphs are both *data and bias*

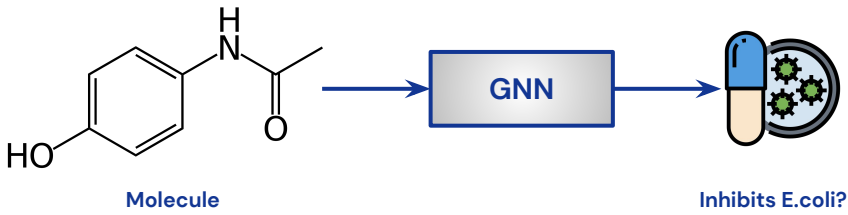
GNNs for molecule classification

Given a molecule, predict whether or not it will have a specific effect

GNNs for molecule classification

Given a molecule, predict whether or not it will have a specific effect

↳ e.g., given a *supervised* dataset of molecules that either cure E. coli or not, predict if unseen (validation/test) molecules cure E. coli



Attempt naive drug discovery

Once trained, the model can be applied to any molecule.

1. Execute on a large dataset of known candidate molecules.

Build large dataset of candidates → classifier with confidence logits

We take the one that is the most confident.

Can do additional filtering on the answer

Attempt naive drug discovery

Once trained, the model can be applied to any molecule.

1. Execute on a large dataset of known candidate molecules.
2. Select the top-100 candidates from your GNN model.

Attempt naive drug discovery

Once trained, the model can be applied to any molecule.

1. Execute on a large dataset of known candidate molecules.
2. Select the top-100 candidates from your GNN model.
3. Have chemists thoroughly investigate those (after some additional filtering).

Attempt naive drug discovery

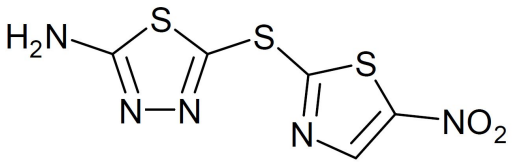
Once trained, the model can be applied to any molecule.

1. Execute on a large dataset of known candidate molecules.
2. Select the top-100 candidates from your GNN model.
3. Have chemists thoroughly investigate those (after some additional filtering).
4. ????

Attempt naive drug discovery

Once trained, the model can be applied to any molecule.

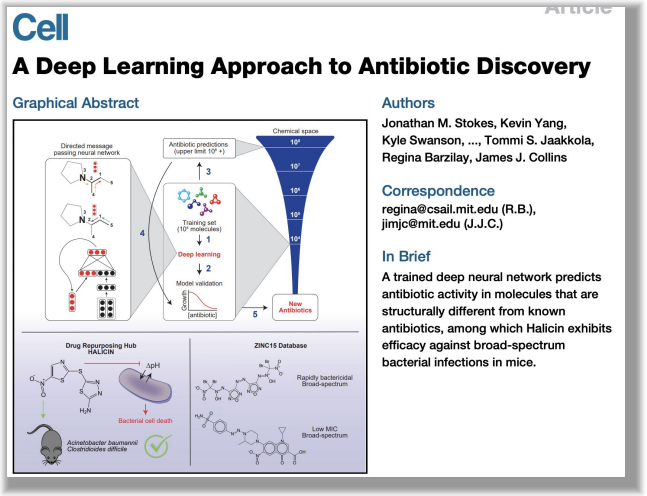
1. Execute on a large dataset of known candidate molecules.
2. Select the top-100 candidates from your GNN model.
3. Have chemists thoroughly investigate those (after some additional filtering).
4. ????
5. ~~Profit!~~ Discovery! A previously overlooked compound is a potent antibiotic!



Halicin

6. become famous

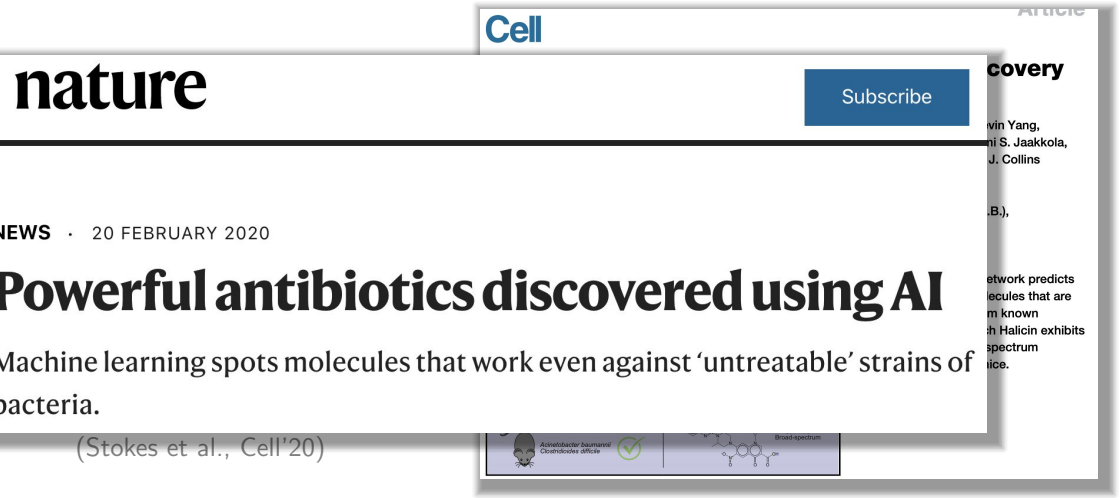
Arguably the most *popularised* success story of graph neural networks to date!



(Stokes et al., Cell'20)

6. become famous

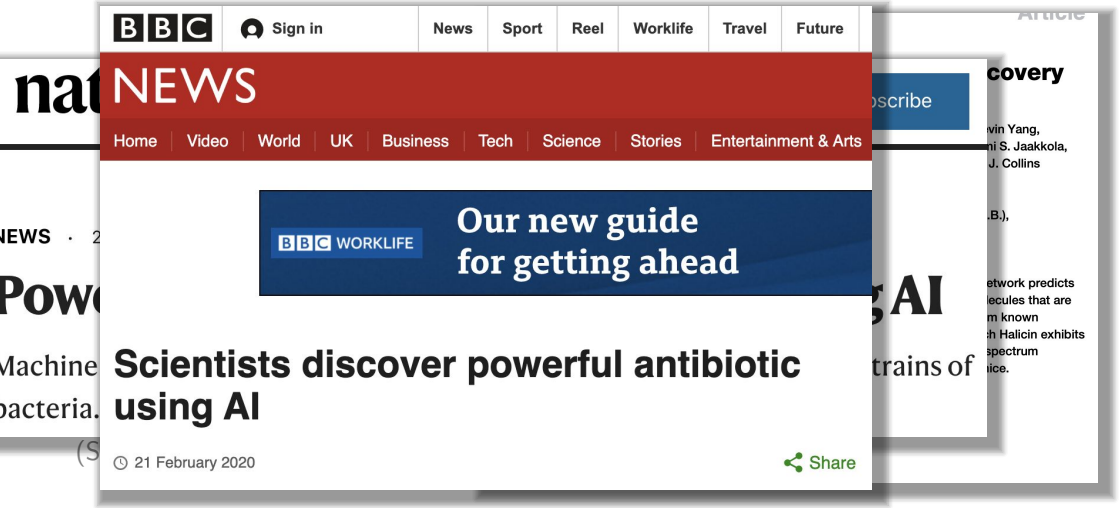
Arguably the most *popularised* success story of graph neural networks to date!



(Stokes et al., Cell'20)

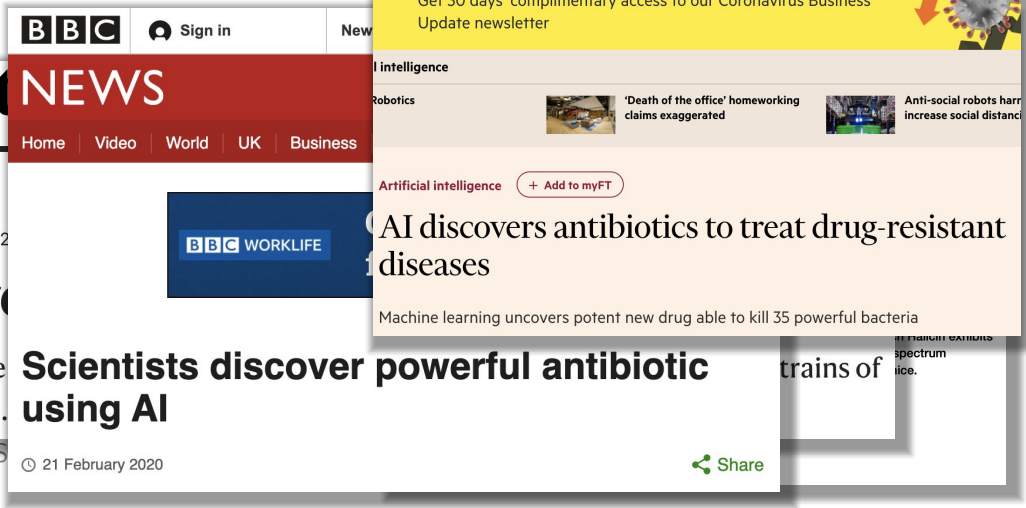
6. become famous

Arguably the most *popularised* success story of graph neural networks to date!



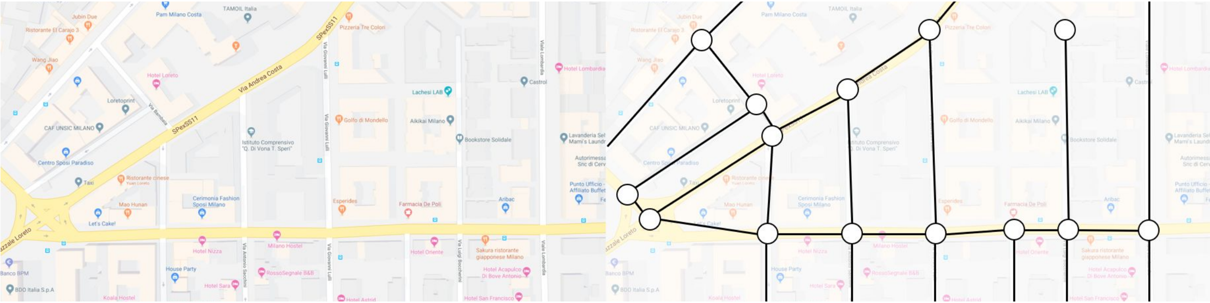
6. become famous

Arguably the most *popularised* s



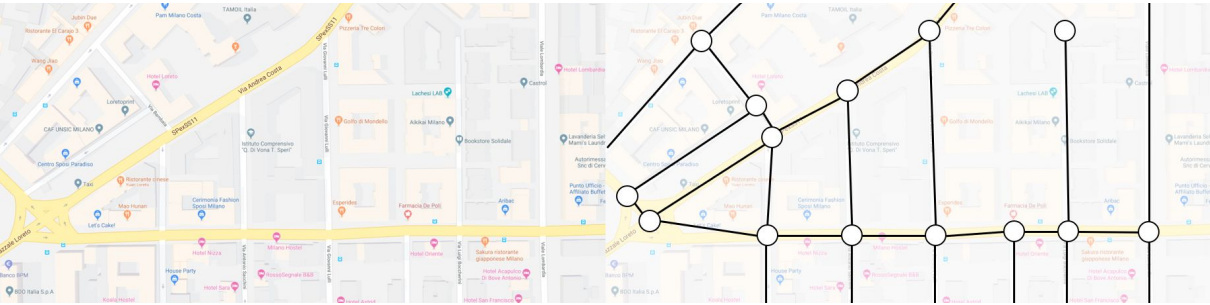
Transportation sub-graph classification

Transportation maps (e.g. Google maps) naturally modelled as graphs



Transportation sub-graph classification

Transportation maps (e.g. Google maps) naturally modelled as graphs



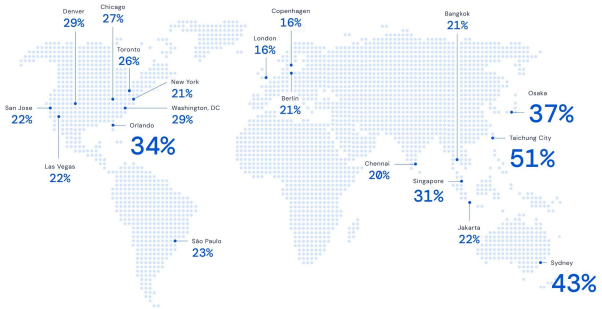
Classify sub-graph properties:

↳ Very livable neighborhood? Dangerous intersection? Traffic trap?

↳ not good → maybe ok...

Transportation sub-graph regression

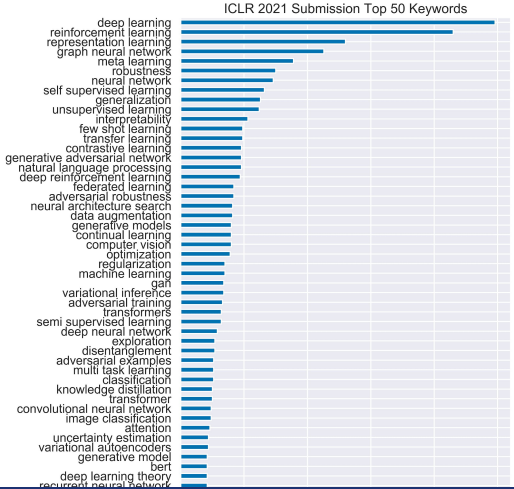
Run GNN on supersegment graph to estimate time of arrival (ETA)
↳ DeepMind's ETA Prediction using GNNs in Google Maps

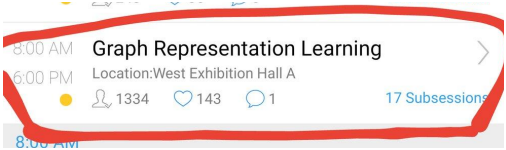


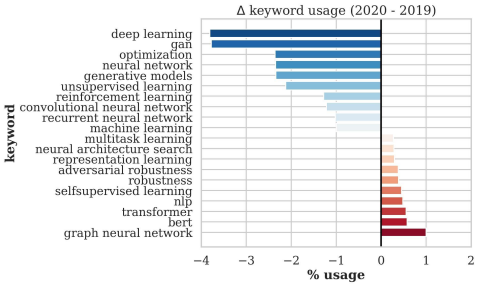
Already deployed in several major cities, significantly reducing negative ETA outcomes!

GNN's imagenet moment

In AGI we have large data to make work CNN
Same for GNN: recently, data set emerge





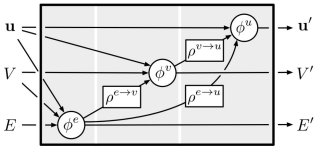


Rich ecosystem of libraries

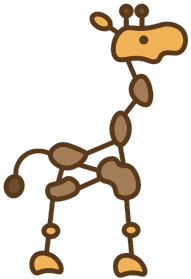
github.com/rusty1s/pytorch_geometric

graphneural.network

dgl.ai



github.com/deepmind/graph_nets



github.com/deepmind/jraph

Rich ecosystem of datasets

ogb.stanford.edu

https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

graphlearning.io

Benchmarking Graph Neural Networks

github.com/graphdeeplearning/benchmarking-gnns

More references

Main Material

Petar Veličković's site (`https://petar-v.com`), this class' material is based on
↳ Theoretical Foundations of Graph Neural Networks.
Everything is Connected: Graph Neural Networks from the Ground Up.

This twitter thread for more GNN resources
`https://twitter.com/PetarV_93/status/1306689702020382720`

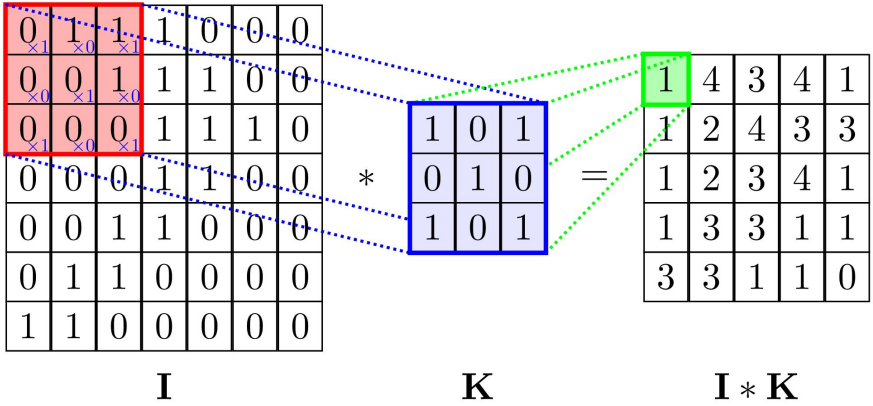
Will Hamilton's grap representation learning textbook (esp. Chapter 7)
`https://www.cs.mcgill.ca/~wlh/grl_book/`

Aleksa Gordić's GitHub repository on GATs
`https://github.com/gordicaleksa/pytorch-GAT`

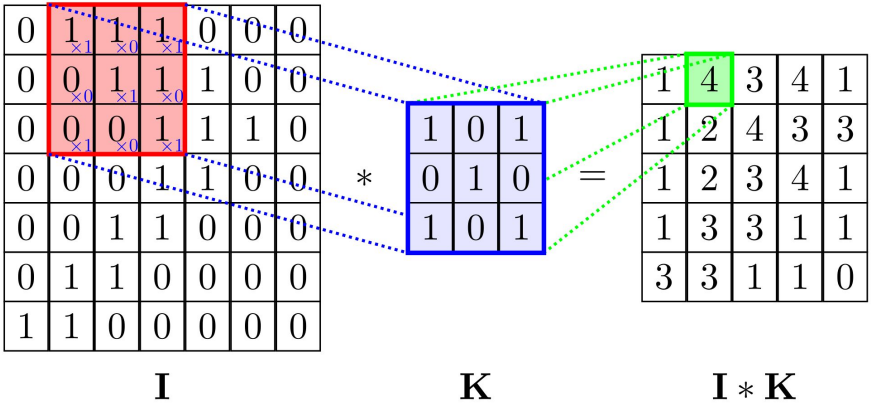
Set neural networks

from first principles

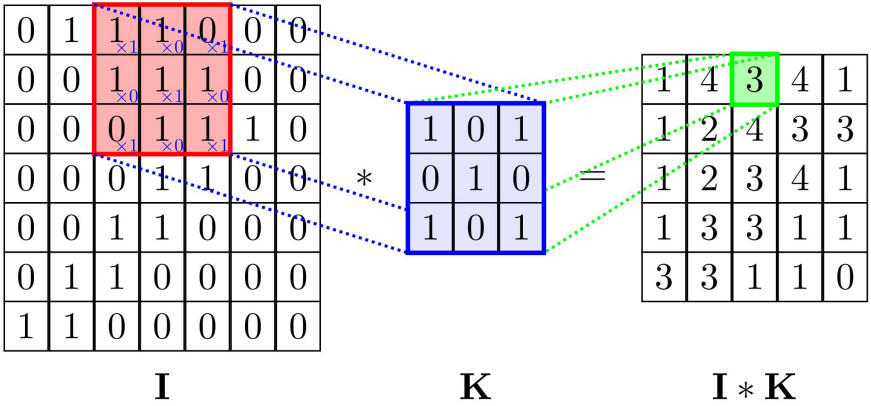
Function invariance in networks



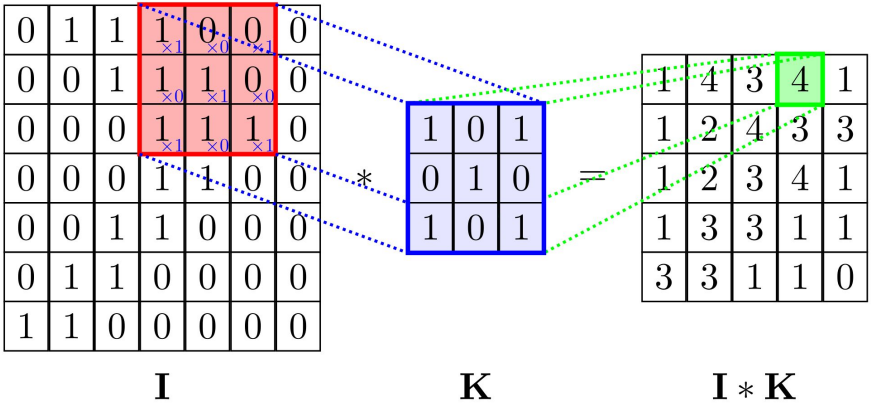
Function invariance in networks



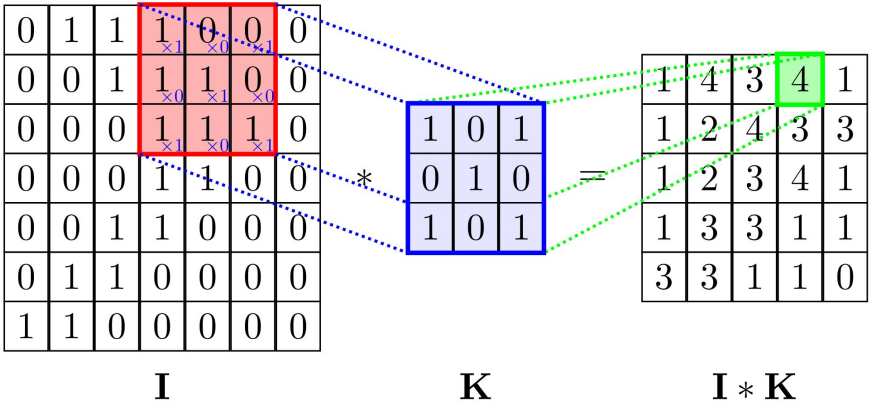
Function invariance in networks



Function invariance in networks



Function invariance in networks



How can we encode a graph vectorially? Which invariances should we preserve?

Set invariance

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

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

where the i-th row of $\mathbf{X}$ corresponds to $\mathbf{x}_i$.

Set invariance

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

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

where the i-th row of $\mathbf{X}$ corresponds to $\mathbf{x}_i$.

Note that, by doing so, we have specified a **node ordering!**

Set invariance

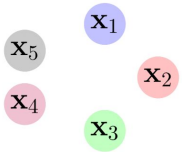
For now, assume the graph has no edges (e.g. set of n nodes). Let $\mathbf{x}_i \in \mathbb{R}^d$ be the features of node i. We can stack them into a node feature matrix of shape $n \times k$:

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

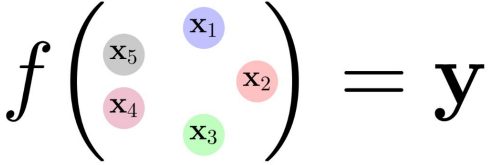
where the i-th row of $\mathbf{X}$ corresponds to $\mathbf{x}_i$.

Note that, by doing so, we have specified a **node ordering!**
↳ We would like the result of any neural networks to not depend on this.

Set invariance

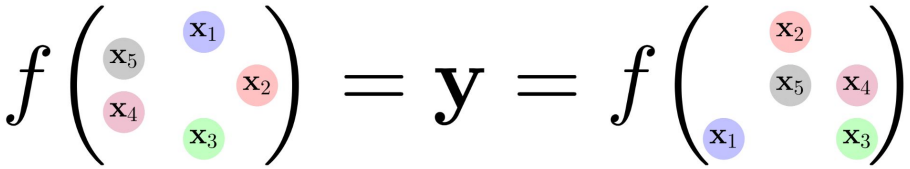


Set invariance



Set invariance

invariance on set permutation.



Permutation invariance

Changing the order of the nodes is called a permutation (there are n! of them).

Permutation invariance

Changing the order of the nodes is called a permutation (there are n! of them).

To explain permutations to the GPU we can use linear algebra

↳ We introduce an $n \times n$ permutation matrix $\mathbf{P}$ with exactly one 1 in every row and column, and zeros everywhere else.

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

Permutation invariance

Changing the order of the nodes is called a permutation (there are n! of them).

To explain permutations to the GPU we can use linear algebra

↳ We introduce an $n \times n$ permutation matrix $\mathbf{P}$ with exactly one 1 in every row and column, and zeros everywhere else.

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

Permutation invariance $\iff$ Applying $\mathbf{P}$ does not modify the result!

Permutation invariance

We say that $f_{\text{inv}}(\mathbf{X})$ is permutation invariant if for all permutation matrices $\mathbf{P}$:

$$f_{\text{inv}}(\mathbf{PX}) = f_{\text{inv}}(\mathbf{X}).$$

Permutation invariance

We say that $f_{\text{inv}}(\mathbf{X})$ is permutation invariant if for all permutation matrices $\mathbf{P}$:

$$f_{\text{inv}}(\mathbf{PX}) = f_{\text{inv}}(\mathbf{X}).$$

One very generic form is the Deep Sets model

$$f_{\text{inv}}(\mathbf{X}) = \phi\left(\sum\nolimits_{i\in\mathcal{V}} \psi(\mathbf{x}_i)\right)$$

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

Permutation invariance

We say that $f_{\text{inv}}(\mathbf{X})$ is permutation invariant if for all permutation matrices $\mathbf{P}$:

$$f_{\text{inv}}(\mathbf{PX}) = f_{\text{inv}}(\mathbf{X}).$$

One very generic form is the Deep Sets model

Deep set function $f_{\text{inv}}(\mathbf{X}) = \phi\left(\sum_{i \in \mathcal{V}} \psi(\mathbf{x}_i)\right)$

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

What if I am trying to classify individual nodes?

Permutation invariance

We say that $f_{\text{inv}}(\mathbf{X})$ is permutation invariant if for all permutation matrices $\mathbf{P}$:

$$f_{\text{inv}}(\mathbf{PX}) = f_{\text{inv}}(\mathbf{X}).$$

One very generic form is the Deep Sets model

$$f_{\text{inv}}(\mathbf{X}) = \phi\left(\sum\nolimits_{i \in \mathcal{V}} \psi(\mathbf{x}_i)\right)$$

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

What if I am trying to classify individual nodes?
↳ We also need functions that don't change the node order!

Permutation equivariance

We say that $f_{\text{equ}}(\mathbf{X})$ is permutation equivariant if, for all permutation matrices $\mathbf{P}$:

$$f_{\text{equ}}(\mathbf{PX}) = \mathbf{P}f_{\text{equ}}(\mathbf{X}).$$

Permutation equivariance

We say that $f_{\text{equ}}(\mathbf{X})$ is permutation equivariant if, for all permutation matrices $\mathbf{P}$:

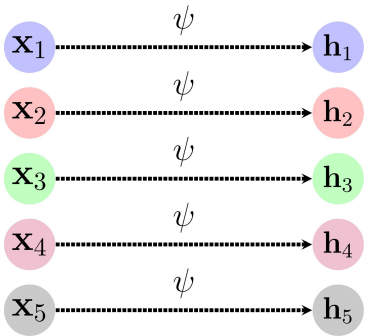
$$f_{\text{equ}}(\mathbf{PX}) = \mathbf{P}f_{\text{equ}}(\mathbf{X}).$$

We can think of equivariant f_{equ} as transforming each node input $\mathbf{x}_i$ into a latent vector $\mathbf{h}_i$:

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

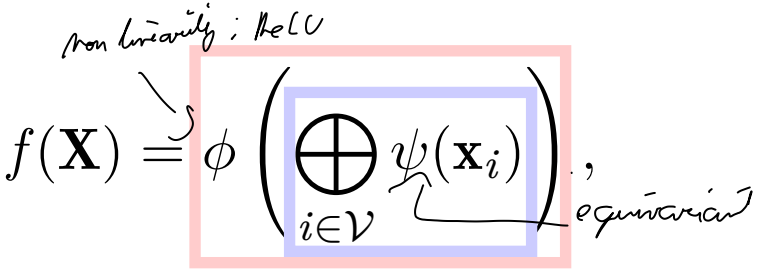
where ψ is any function.

Stacking $\mathbf{h}_i$ yields $\mathbf{H} = f_{\text{equ}}(\mathbf{X})$.



One-layer set neural network

We arrive at a general blueprint: (stacking) equivariant function(s), potentially with an invariant final aggregator yields (m)any useful functions on sets!



were, $\oplus$ is a permutation-invariant aggregator (such as sum, avg or max).

Graph neural networks

from first principles

From set to graphs

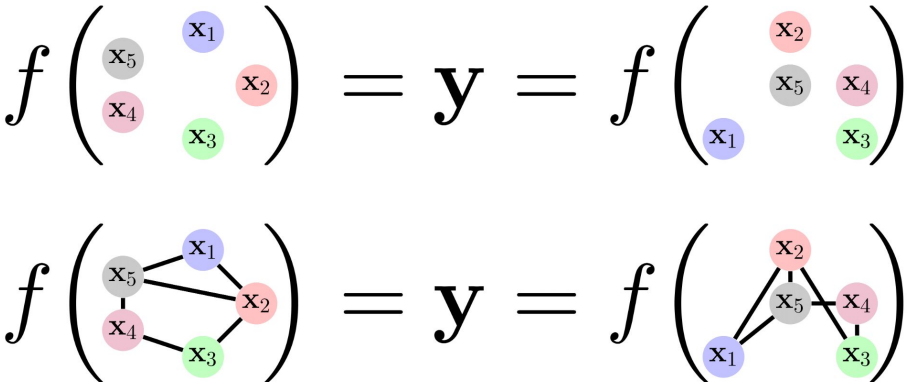
Now we augment the set of nodes with edges $\mathcal{E}$ between them, that we represent with the usual adjacency matrix $\mathbf{A}$.

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

and need to find appropriate $f(\mathbf{X}, \mathbf{A})$.

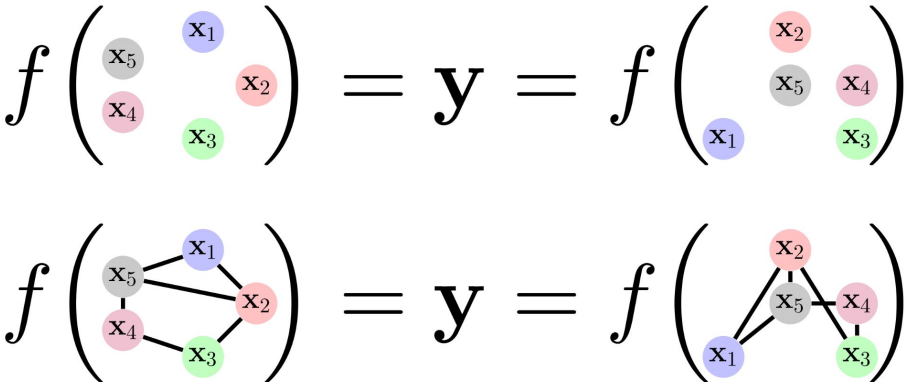
From set to graphs

In theory: invariance to graph isomorphism!



From set to graphs

In theory: invariance to graph isomorphism!



In practice: permutation {in,equi}variance again!

Permutation {in, equi}variance on graphs

We need to appropriately permute both rows and columns of $\mathbf{A}$:
↳ this amounts to applying the permutation matrix $\mathbf{P}$ on both sides $\mathbf{PAP}^\top$

Permutation {in, equi}variance on graphs

We need to appropriately permute both rows and columns of $\mathbf{A}$:
↳ this amounts to applying the permutation matrix $\mathbf{P}$ on both sides $\mathbf{PAP}^\mathsf{T}$

Invariance:

$$f(\mathbf{PX}, \mathbf{PAP}^\mathsf{T}) = f(\mathbf{X}, \mathbf{A})$$

Permutation {in, equi}variance on graphs

We need to appropriately permute both rows and columns of $\mathbf{A}$:
↳ this amounts to applying the permutation matrix $\mathbf{P}$ on both sides $\mathbf{PAP}^\top$

Invariance:

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

Equivariance:

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

More desirable properties: locality

Sets do not have a concept of locality, but graphs have neighborhoods

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

where for simplicity we assume unweighted, undirected edges and $i \in \mathcal{N}_i$.

More desirable properties: locality

Sets do not have a concept of locality, but graphs have neighborhoods

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

where for simplicity we assume unweighted, undirected edges and $i \in \mathcal{N}_i$.

We will also define the feature multiset associated with $\mathcal{N}_i$

$$\mathbf{X}_{\mathcal{N}_i} = \left\{\mathbf{x}_i : j \in \mathcal{N}_i\right\},$$

and will indicate **local** functions g as $g(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$.

A recipe for graph neural networks

We can construct permutation equivariant functions $f(\mathbf{X}, \mathbf{A})$ by appropriately applying g over all local neighbourhoods:

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

To ensure equivariance g must not depend on the order of the vertices in $\mathbf{X}_{\mathcal{N}_i}$

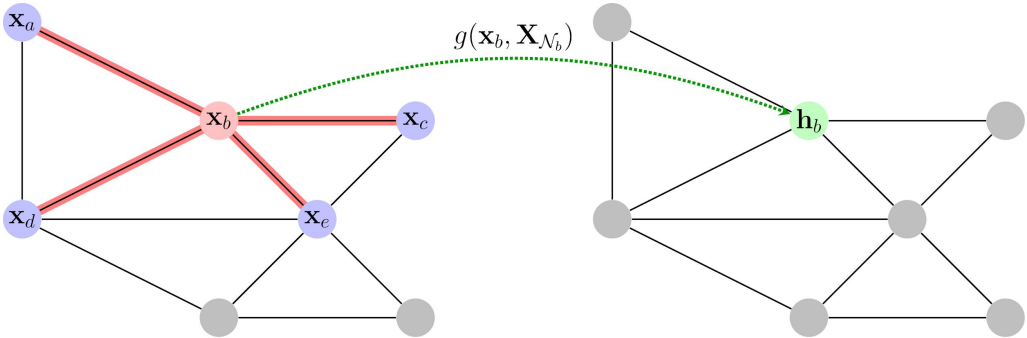
A recipe for graph neural networks

We can construct permutation equivariant functions $f(\mathbf{X}, \mathbf{A})$ by appropriately applying g over all local neighbourhoods:

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

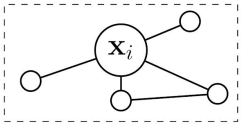
To ensure equivariance g must not depend on the order of the vertices in $\mathbf{X}_{\mathcal{N}_i}$
↳ g should be permutation invariant!

A recipe for graph neural networks, visualised

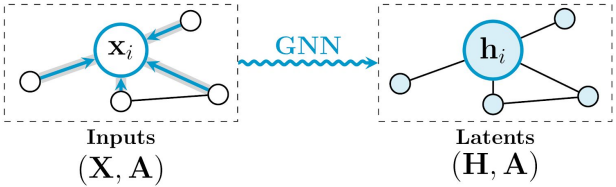


$$\mathbf{X}_{\mathcal{N}_b} = \{\!\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}\!\}$$

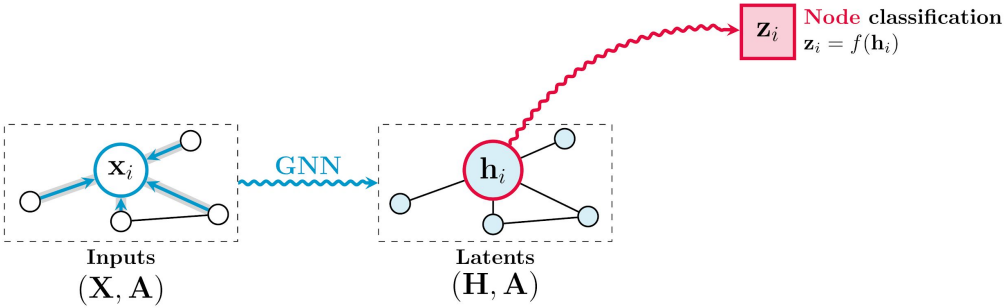
General blueprint for learning on graphs



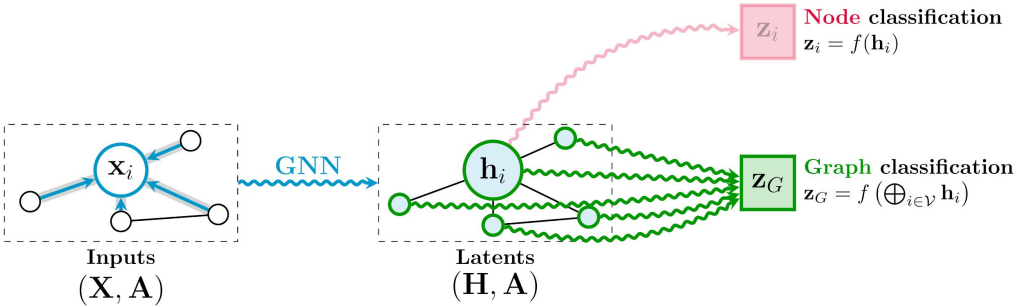
General blueprint for learning on graphs



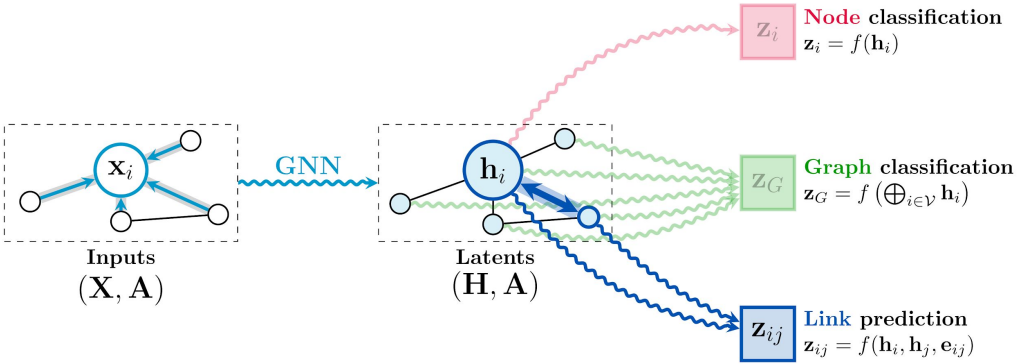
General blueprint for learning on graphs



General blueprint for learning on graphs



General blueprint for learning on graphs



Message passing on graphs

three flavours for all tastes

What's in a GNN layer?

We will call a "GNN layer" f any generic equivariant function $f(\mathbf{X}, \mathbf{A})$ of the form

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

with $g(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$ invariant.

What's in a GNN layer?

We will call a "GNN layer" f any generic equivariant function $f(\mathbf{X}, \mathbf{A})$ of the form

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

with $g(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$ invariant.

Defining a GNN layer $\iff$ Defining g function

What's in a GNN layer?

We will call a "GNN layer" f any generic equivariant function $f(\mathbf{X}, \mathbf{A})$ of the form

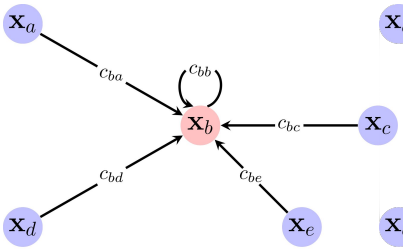
$$f(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} — & g(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & — \\ — & g(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & — \\ & \vdots & \\ — & g(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & — \end{bmatrix}$$

with $g(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$ invariant.

$$\text{Defining a GNN layer} \iff \text{Defining } g \text{ function}$$

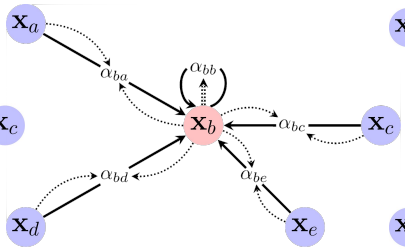
g is referred to as the "diffusion", "propagation", or "message passing" function, depending on context.

The three "flavours" of GNN layers



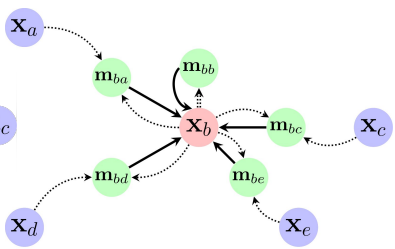
Convolutional

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



Attentional

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



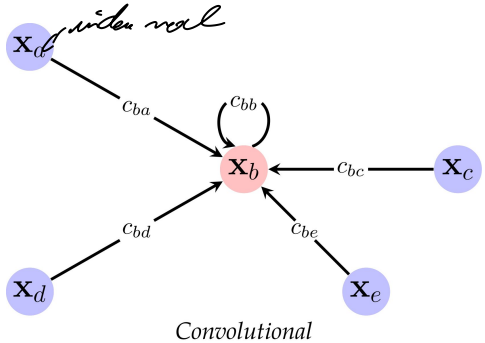
Message-passing

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

Convolutional GNNs

Features of neighbours aggregated with fixed weights, c_{ij}

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



Convolutional

Usually, the weights depend directly on $\mathbf{A}$.

↳ ChebyNet (Defferrard et al., NeurIPS'16)
GCN (Kipf & Welling, ICLR'17)
SGC (Wu et al., ICML'19)

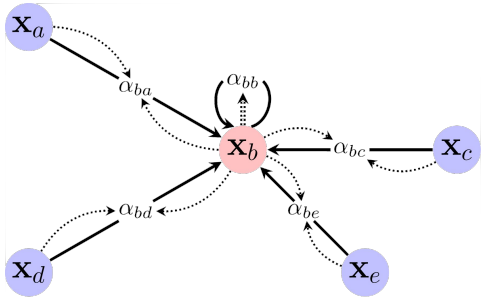
Useful for homophilous graphs and **scaling up**

Attentional GNNs

Features of neighbours aggregated with implicit weights (via attention)

weight between nodes are parametrized by a (can be an MLP)

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



Attentional

Attention weight computed as $a_{ij} = a(\mathbf{x}_i, \mathbf{x}_j)$

↳ MoNet (Monti et al., CVPR'17)
GAT (Veličković et al., ICLR'18)
GaAN (Zhang et al., UAI'18)

Useful as "middle ground" w.r.t. capacity and scale

↳ Edges still scalars but need not encode homophily

Message-Passing GNNs

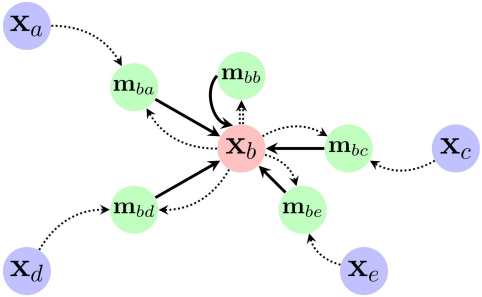
Compute arbitrary vectors ("messages") to be sent across edges

arbitrarily encodes

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

Messages computed as $\psi(\mathbf{x}_i, \mathbf{x}_j)$

↳ Interaction Networks (Battaglia et al., NeurIPS'16)
MPNN (Gilmer et al., ICML'17)
GraphNets (Battaglia et al., 2018)



Message-passing

Most generic GNN layer

↳ May have scalability or learnability issues
Ideal for computational chemistry, reasoning and simulation

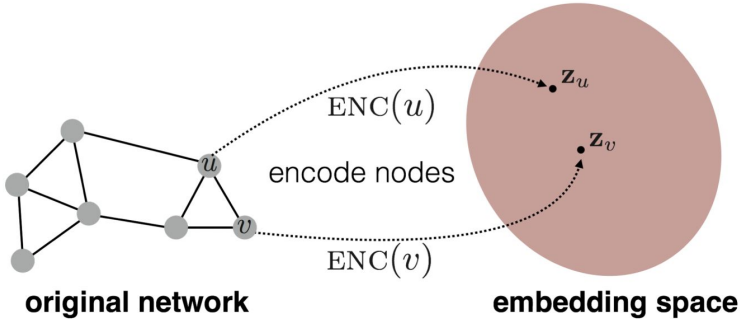
From ad-hoc node embeddings to GNNs

Current GNN frameworks got to permutation $\{$in, equi$\}$variance mostly in **hindsight**.

From ad-hoc node embeddings to GNNs

Current GNN frameworks got to permutation {in, equi}variance mostly in **hindsight**.

First "popular" DNN applications on graphs were embeddings pre-training:
↳ part of the everything2vec hype cycle



Node embedding as edge presence classification

Graphs are about edges, thus features of nodes should be predictive of edge presence

Node embedding as edge presence classification

Graphs are about edges, thus features of nodes should be predictive of edge presence
↳ Can use standard binary cross-entropy loss!

$$\min_{\{\mathbf{H}\}} \sum_{(i,j)\in\mathcal{E}} \log(\sigma(\mathbf{h}_i^\top \mathbf{h}_j)) + \sum_{(i,j)\notin\mathcal{E}} \log(1-\sigma(\mathbf{h}_i^\top \mathbf{h}_j))$$

Node embedding as edge presence classification

Graphs are about edges, thus features of nodes should be predictive of edge presence
↳ Can use standard binary cross-entropy loss!

$$\min_{\{\mathbf{H}\}} \sum_{(i,j)\in\mathcal{E}} \log(\sigma(\mathbf{h}_i^\top \mathbf{h}_j)) + \sum_{(i,j)\notin\mathcal{E}} \log(1-\sigma(\mathbf{h}_i^\top \mathbf{h}_j))$$

1-NN is limiting, how do we include global structure?

Node embedding as edge presence classification

Graphs are about edges, thus features of nodes should be predictive of edge presence
↳ Can use standard binary cross-entropy loss!

$$\min_{\{\mathbf{H}\}} \sum_{(i,j)\in\mathcal{E}} \log(\sigma(\mathbf{h}_i^\top \mathbf{h}_j)) + \sum_{(i,j)\notin\mathcal{E}} \log(1-\sigma(\mathbf{h}_i^\top \mathbf{h}_j))$$

1-NN is limiting, how do we include global structure? Co-occurrence in random walk!

Node embedding as edge presence classification

Graphs are about edges, thus features of nodes should be predictive of edge presence
↳ Can use standard binary cross-entropy loss!

$$\min_{\{\mathbf{H}\}} \sum_{(i,j)\in\mathcal{E}} \log(\sigma(\mathbf{h}_i^\top \mathbf{h}_j)) + \sum_{(i,j)\notin\mathcal{E}} \log(1-\sigma(\mathbf{h}_i^\top \mathbf{h}_j))$$

1-NN is limiting, how do we include global structure? Co-occurrence in random walk!
↳ DeepWalk (Perozzi et al., KDD'14)
node2vec (Grover & Leskovec, KDD'16)
LINE (Tang et al., WWW'15)

Node embedding as edge presence classification

Graphs are about edges, thus features of nodes should be predictive of edge presence
↳ Can use standard binary cross-entropy loss!

$$\min_{\{\mathbf{H}\}} \sum_{(i,j)\in\mathcal{E}} \log(\sigma(\mathbf{h}_i^\top \mathbf{h}_j)) + \sum_{(i,j)\notin\mathcal{E}} \log(1-\sigma(\mathbf{h}_i^\top \mathbf{h}_j))$$

1-NN is limiting, how do we include global structure? Co-occurrence in random walk!
↳ DeepWalk (Perozzi et al., KDD'14)
node2vec (Grover & Leskovec, KDD'16)
LINE (Tang et al., WWW'15)

Not the only time GNNs and NLP's path overlapped

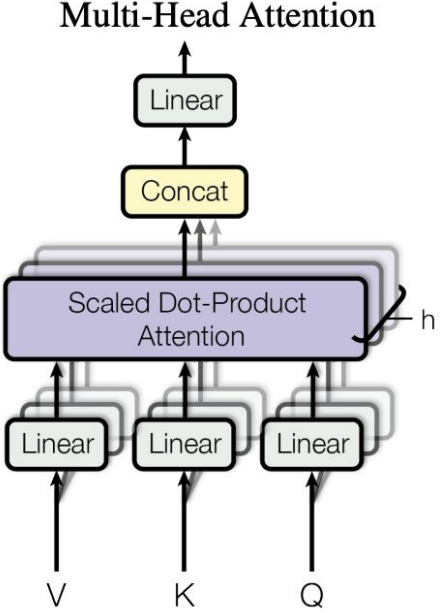
A note on transformers

Transformers are Graph Neural Networks!
↳ Fully-connected graph
Attentional flavour

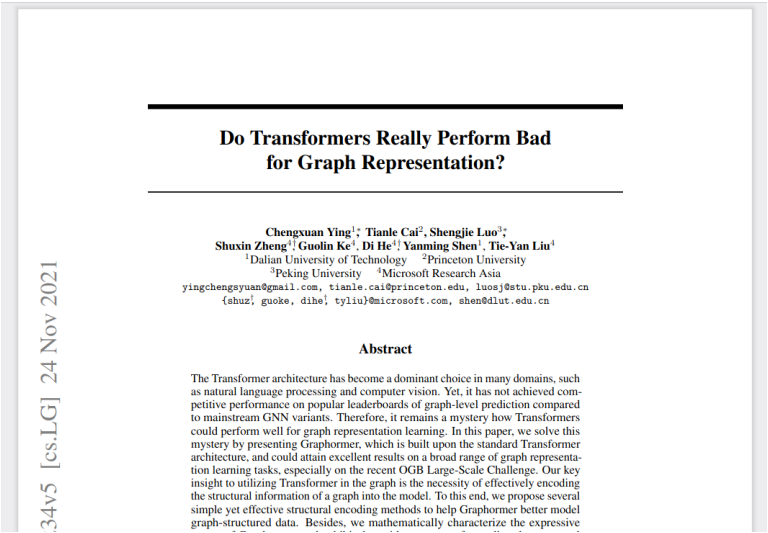
The sequential structural information is injected through the positional embeddings.
Dropping them yields a fully-connected GAT model.

Attention can be seen as inferring soft adjacency.

See Joshi (The Gradient; 2020).



The wild west of heuristics



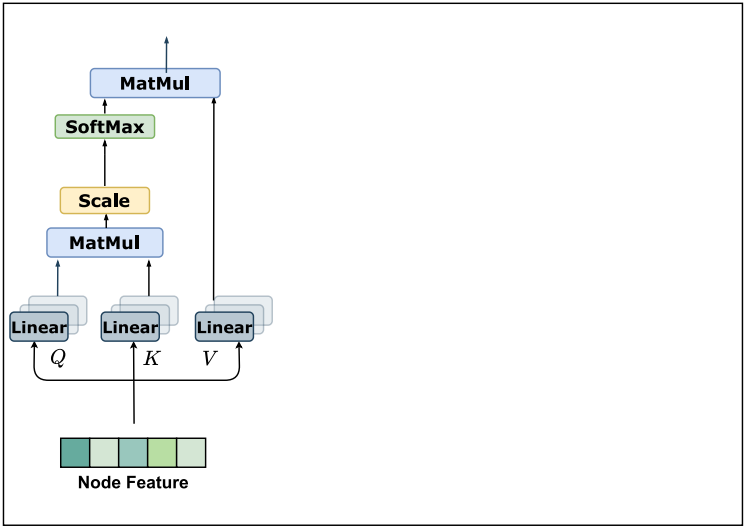
Do Transformers Really Perform Bad for Graph Representation?

Chengxuan Ying[1]*, Tianle Cai[2], Shengjie Luo[3]*,
Shuxin Zheng[4]†, Guolin Ke[4], Di He[4]†, Yanming Shen[1], Tie-Yan Liu[4]
[1]Dalian University of Technology [2]Princeton University
[3]Peking University [4]Microsoft Research Asia
yingchengsyuan@gmail.com, tianle.cai@princeton.edu, luosj@stu.pku.edu.cn
{shuz†, guoke, dihe†, tyliu}@microsoft.com, shen@dlut.edu.cn

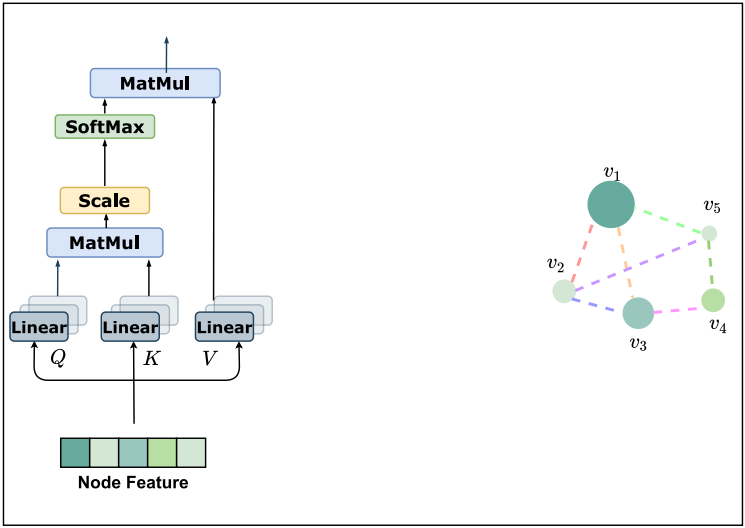
Abstract

The Transformer architecture has become a dominant choice in many domains, such as natural language processing and computer vision. Yet, it has not achieved competitive performance on popular leaderboards of graph-level prediction compared to mainstream GNN variants. Therefore, it remains a mystery how Transformers could perform well for graph representation learning. In this paper, we solve this mystery by presenting Graphormer, which is built upon the standard Transformer architecture, and could attain excellent results on a broad range of graph representation learning tasks, especially on the recent OGB Large-Scale Challenge. Our key insight to utilizing Transformer in the graph is the necessity of effectively encoding the structural information of a graph into the model. To this end, we propose several simple yet effective structural encoding methods to help Graphormer better model graph-structured data. Besides, we mathematically characterize the expressive

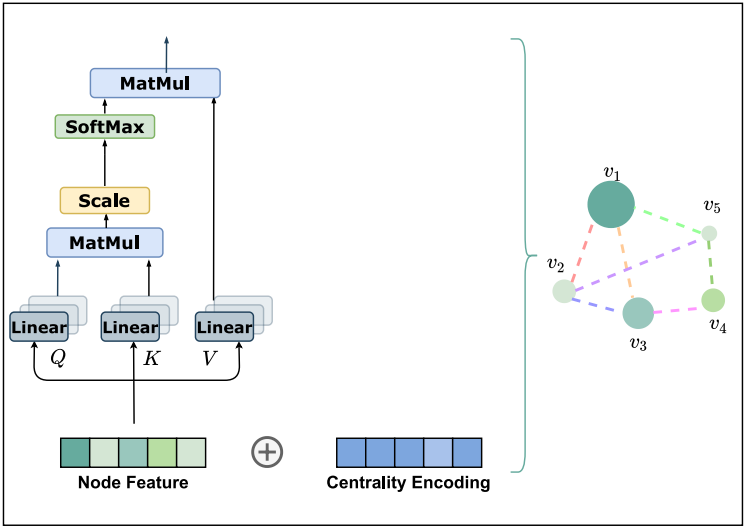
From Transformer to (a) Graphformer



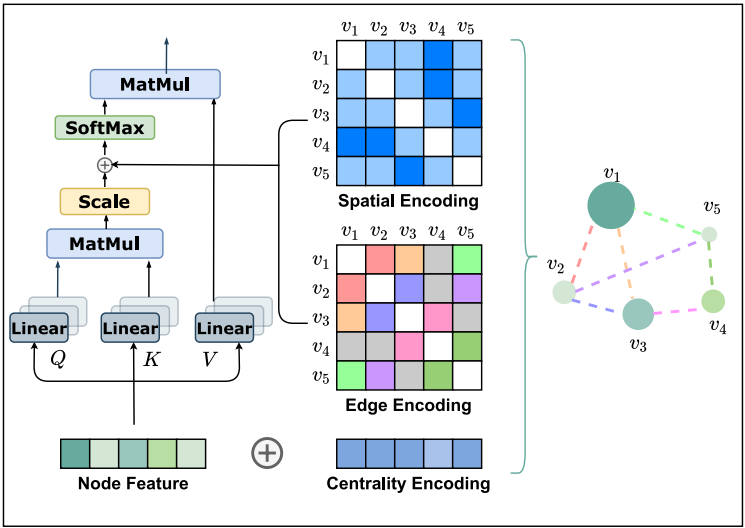
From Transformer to (a) Graphformer



From Transformer to (a) Graphformer



From Transformer to (a) Graphformer



From Transformer to (a) Graphformer: modifying the input

Centrality encodings: $\mathbf{h}_i^{(0)} = \mathbf{x}_i + \mathbf{z}^-_{\mathsf{deg}^-(v_i)} + \mathbf{z}^+_{\mathsf{deg}^+(v_i)}$.

From Transformer to (a) Graphformer: modifying the attention

$$\mathbf{A}_{ij} = (\mathbf{h}_i \mathbf{W}_Q)(\mathbf{h}_j \mathbf{W}_K)^T / \sqrt{d} + \overbrace{b_{s(v_i,v_j)}}^{\text{Spatial encodings}} + \underbrace{c_{ij}}_{\text{Edge encodings}},$$

From Transformer to (a) Graphformer: modifying the attention

$$\mathbf{A}_{ij} = (\mathbf{h}_i \mathbf{W}_Q)(\mathbf{h}_j \mathbf{W}_K)^T / \sqrt{d} + \overbrace{b_{s(v_i,v_j)}}^{\text{Spatial encodings}} + \underbrace{c_{ij}}_{\text{Edge encodings}},$$

Spatial encodings $b_{s(v_i,v_j)}$ are a learnable scalar indexed by $s(v_i, v_j)$.

From Transformer to (a) Graphformer: modifying the attention

$$\mathbf{A}_{ij} = (\mathbf{h}_i \mathbf{W}_Q)(\mathbf{h}_j \mathbf{W}_K)^T / \sqrt{d} + \overbrace{b_{s(v_i,v_j)}}^{\text{Spatial encodings}} + \underbrace{c_{ij}}_{\text{Edge encodings}},$$

Spatial encodings $b_{s(v_i,v_j)}$ are a learnable scalar indexed by $s(v_i, v_j)$.

↳ they use $s(v_i, v_j) =$ length of shortest path.

From Transformer to (a) Graphformer: modifying the attention

$$\mathbf{A}_{ij} = (\mathbf{h}_i \mathbf{W}_Q)(\mathbf{h}_j \mathbf{W}_K)^T / \sqrt{d} + \overbrace{b_{s(v_i,v_j)}}^{\text{Spatial encodings}} + \underbrace{c_{ij}}_{\text{Edge encodings}},$$

Spatial encodings $b_{s(v_i,v_j)}$ are a learnable scalar indexed by $s(v_i, v_j)$.
↳ they use $s(v_i, v_j) =$ length of shortest path.

Edge encodings are scalars $c_{ij} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_{e_n} (\mathbf{w}_n^E)^T$

From Transformer to (a) Graphformer: modifying the attention

$$\mathbf{A}_{ij} = (\mathbf{h}_i \mathbf{W}_Q)(\mathbf{h}_j \mathbf{W}_K)^T / \sqrt{d} + \overbrace{b_{s(v_i,v_j)}}^{\text{Spatial encodings}} + \underbrace{c_{ij}}_{\text{Edge encodings}},$$

Spatial encodings $b_{s(v_i,v_j)}$ are a learnable scalar indexed by $s(v_i, v_j)$.
↳ they use $s(v_i, v_j)$ = length of shortest path.

Edge encodings are scalars $c_{ij} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_{e_n} (\mathbf{w}_n^E)^T$, based on the features
↳ SP $= (e_1, e_2, ..., e_N)$ is (one of) the shortest path from v_i to v_j,

From Transformer to (a) Graphformer: modifying the attention

$$\mathbf{A}_{ij} = (\mathbf{h}_i \mathbf{W}_Q)(\mathbf{h}_j \mathbf{W}_K)^T / \sqrt{d} + \overbrace{b_{s(v_i,v_j)}}^{\text{Spatial encodings}} + \underbrace{c_{ij}}_{\text{Edge encodings}},$$

Spatial encodings $b_{s(v_i,v_j)}$ are a learnable scalar indexed by $s(v_i, v_j)$.
↳ they use $s(v_i, v_j) =$ length of shortest path.

Edge encodings are scalars $c_{ij} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_{e_n} (\mathbf{w}_n^E)^T$
↳ SP $= (e_1, e_2, ..., e_N)$ is (one of) the shortest path from v_i to v_j,
$\mathbf{x}_{e_n}$ the feature of the n-th edge e_n in SP$_{ij}$,
$\mathbf{w}_n^E \in \mathbb{R}^{d_E}$ the learned n-th weight embedding,

From Transformer to (a) Graphformer: why?

In theory: connections to virtual global sink node, analysis of expressive power

From Transformer to (a) Graphformer: why?

In theory: connections to virtual global sink node, analysis of expressive power

In practice: wins on OGB!

Table 1: Results on PCQM4M-LSC. * indicates the results are cited from the official leaderboard [21].

method	#param.	train MAE	validate MAE
GCN [26]	2.0M	0.1318	0.1691 (0.1684*)
GIN [54]	3.8M	0.1203	0.1537 (0.1536*)
GCN-VN [26, 15]	4.9M	0.1225	0.1485 (0.1510*)
GIN-VN [54, 15]	6.7M	0.1150	0.1395 (0.1396*)
GINE-VN [5, 15]	13.2M	0.1248	0.1430
DeeperGCN-VN [30, 15]	25.5M	0.1059	0.1398
GT [13]	0.6M	0.0944	0.1400
GT-Wide [13]	83.2M	0.0955	0.1408
$\text{Graphormer}_{\text{SMALL}}$	12.5M	0.0778	0.1264
Graphormer	47.1M	**0.0582**	**0.1234**

Wrapping up

more applications and open questions

Bleeding edge applications

PinSage: A new graph convolutional neural network for web-scale recommender systems

Pinterest Engineering Follow

Aug 15, 2018 · 8 min read

Food Discovery with Uber Eats: Using Graph Learning to Power Recommendations

Ankit Jain, Isaac Liu, Ankur Sarda, and Piero Molino December 4, 2019

0

amazon | science

PUBLICATION

P-Companion: A principled framework for diversified complementary product recommendation

By Junheng Hao, Tong Zhao, Jin Li, Xin Luna Dong, Christos Faloutsos, Yizhou Sun, Wei Wang

2020

Bleeding edge applications

nature

Explore content ⌄ Journal information ⌄ Publish with us ⌄ Subscribe

nature › articles › article

Article | Published: 09 June 2021

A graph placement methodology for fast chip design

Azalia Mirhoseini ✉, Anna Goldie ✉, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean

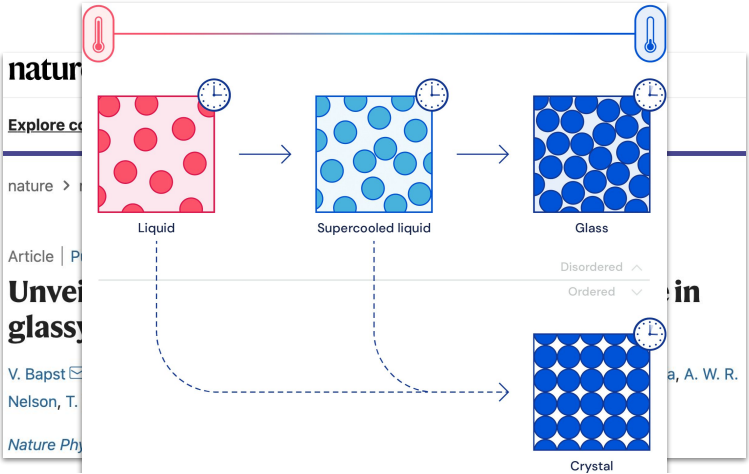
GOOGLE \ TECH \ ARTIFICIAL INTELLIGENCE \

Google is using AI to design its next generation of AI chips more quickly than humans can

Designs that take humans months can be matched or beaten by AI in six hours

By James Vincent | Jun 10, 2021, 9:13am EDT

Bleeding edge applications

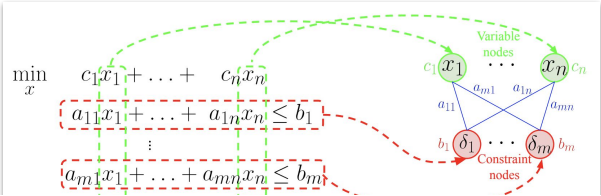


Bleeding edge applications

Solving Mixed Integer Programs Using Neural Networks

Vinod Nair*†[1], Sergey Bartunov*[1], Felix Gimeno*[1], Ingrid von Glehn*[1], Pawel Lichocki*[2], Ivan Lobov*[1], Brendan O'Donoghue*[1], Nicolas Sonnerat*[1], Christian Tjandraatmadja*[2], Pengming Wang*[1], Ravichandra Addanki[1], Tharindi Hapuarachchi[1], Thomas Keck[1], James Keeling[1], Pushmeet Kohli[1], Ira Ktena[1], Yujia Li[1], Oriol Vinyals[1], Yori Zwols[1]

[1]DeepMind, [2]Google Research



GNNs and beyond

GNNs are flexible blueprints based on few properties:
↳ locality, permutation invariance, permutation equivariance

GNNs and beyond

GNNs are flexible blueprints based on few properties:
↳ locality, permutation invariance, permutation equivariance

Still remarkably powerful!
↳ generalization of many standard architectures (including CNNs and Transformers!)

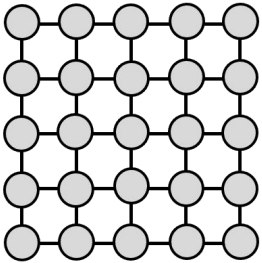
GNNs and beyond

GNNs are flexible blueprints based on few properties:
↳ locality, permutation invariance, permutation equivariance

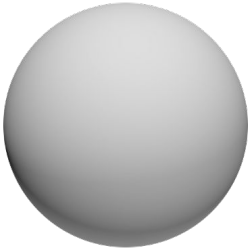
Still remarkably powerful!
↳ generalization of many standard architectures (including CNNs and Transformers!)

Can we generalize it even further?
↳ full graph isomorphism
general class of geometric deep learning architectures

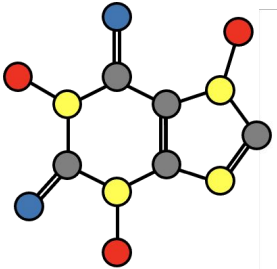
Geometric deep learning



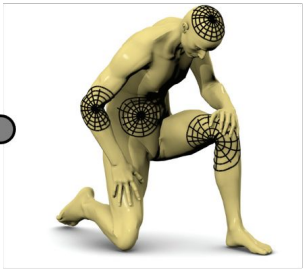
Images & Sequences



Homogeneous spaces

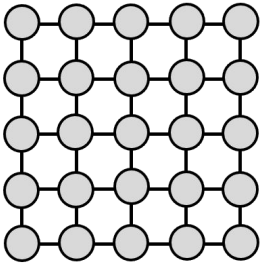


Graphs & Sets

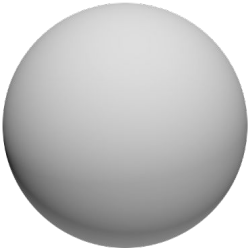


Manifolds, Meshes & Geometric graphs

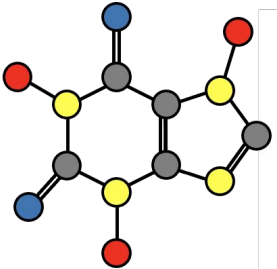
The five "G"s of geometric deep learning



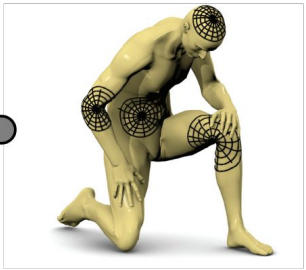
Grids



Groups



Graphs



Geodesics & Gauges

The five "G"s of geometric deep learning

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

Credit to Michael Bronstein

Daniele Calandriello
dcalandriello@google.com
ENS Paris-Saclay, MVA 2022/2023
https://sites.google.com/view/daniele-calandriello/