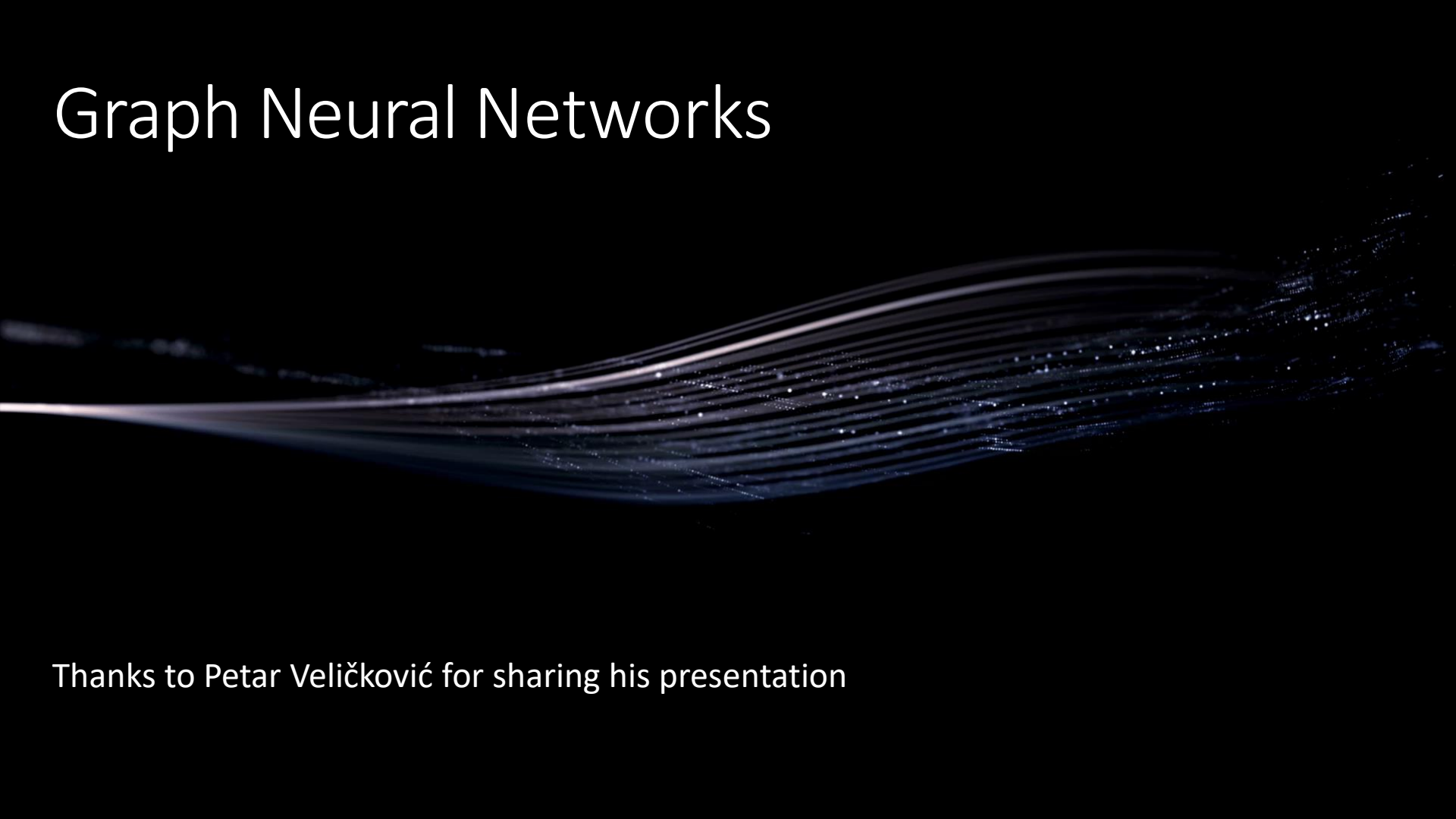
Graph Neural Networks



Thanks to Petar Veličković for sharing his presentation

Talk roadmap

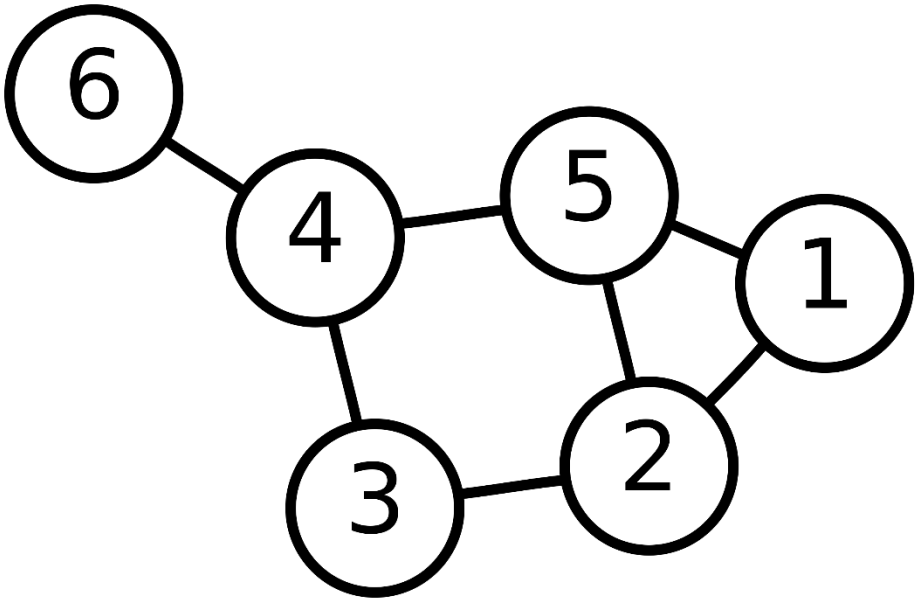
Aim of today: provide good **blueprints** and **contexts** for studying the GNN:

1. What are Graphs? What are they used to?
2. Look to the *past*: how GNN-like models emerged in historical ML research
3. Derive GNNs from first principles
4. Learning on Graph
5. The Message Passing framework

Fantastic GNNs in the Wild

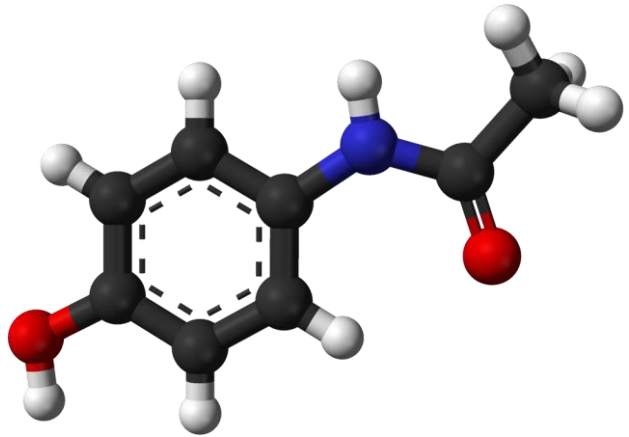
What is a Graph?

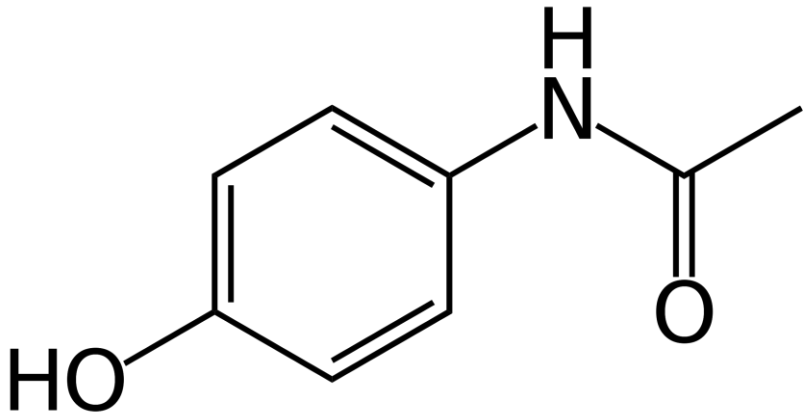
- Mathematical structures used to model pairwise relations between objects
- Graphs $G = (V, E)$ are made up of:
  - *Vertices* V (also called *nodes* or *points*)
  - *Edges* E (also called *links* or *lines*) which connects the vertices



Molecules are graphs!

- A very natural way to represent molecules is as a **graph**
  - **Atoms** as nodes, **bonds** as edges
  - Features such as **atom type**, **charge**, **bond type**…

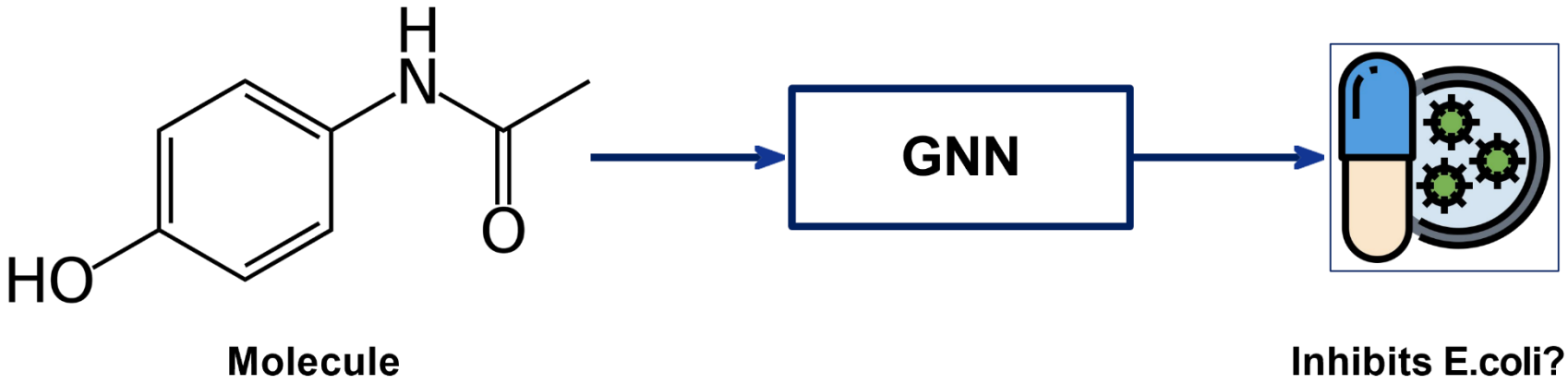




GNNs for molecule classification

Interesting task, is a molecule a potent **drug?**

- *Binary classification* on whether the drug will inhibit a bacteria
- Train on a **curated dataset** for compounds where response is known

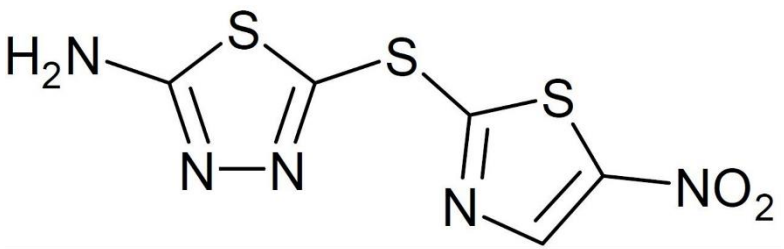


Follow-up study

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

Execute on a large dataset of known candidate molecules	Select the *~top-100* candidates from your GNN model	Chemists thoroughly investigate

Discover a previously overlooked compound that is a **highly potent** antibiotic!



Halicin

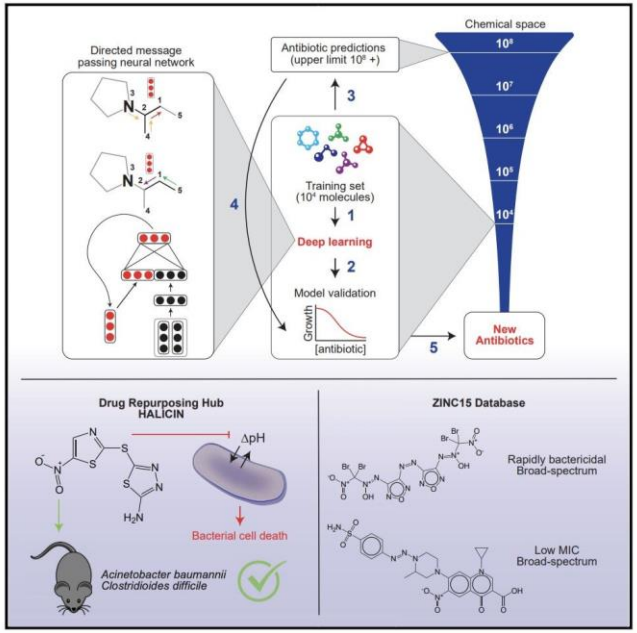
...Achieve wide acclaim!

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

Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract



Authors

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, ..., Tommi S. Jaakkola, Regina Barzilay, James J. Collins

Correspondence

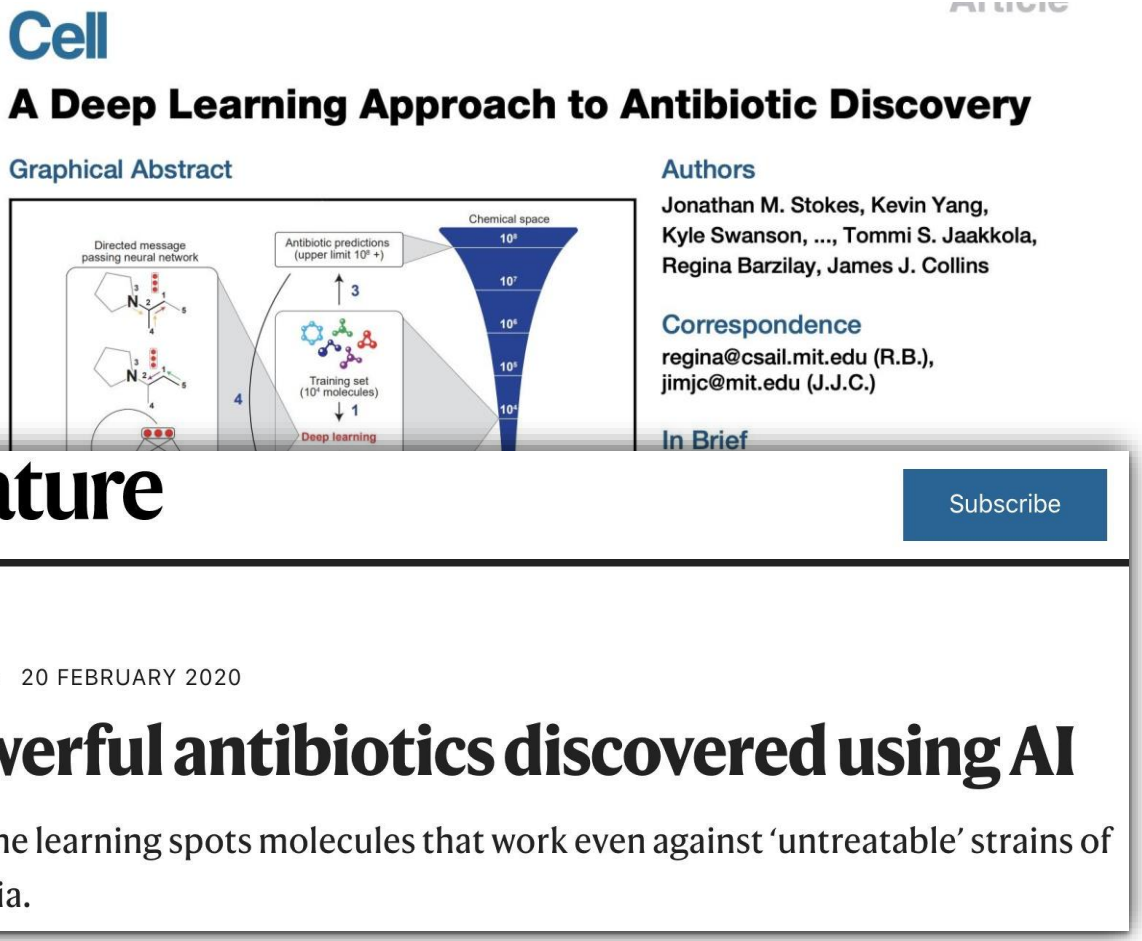
regina@csail.mit.edu (R.B.), jimjc@mit.edu (J.J.C.)

In Brief

A trained deep neural network predicts antibiotic activity in molecules that are structurally different from known antibiotics, among which Halicin exhibits efficacy against broad-spectrum bacterial infections in mice.

...Achieve wide acclaim!

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



Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract

Authors

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, ..., Tommi S. Jaakkola, Regina Barzilay, James J. Collins

Correspondence

regina@csail.mit.edu (R.B.), jimjc@mit.edu (J.J.C.)

In Brief

nature

Subscribe

NEWS · 20 FEBRUARY 2020

Powerful antibiotics discovered using AI

Machine learning spots molecules that work even against 'untreatable' strains of bacteria.

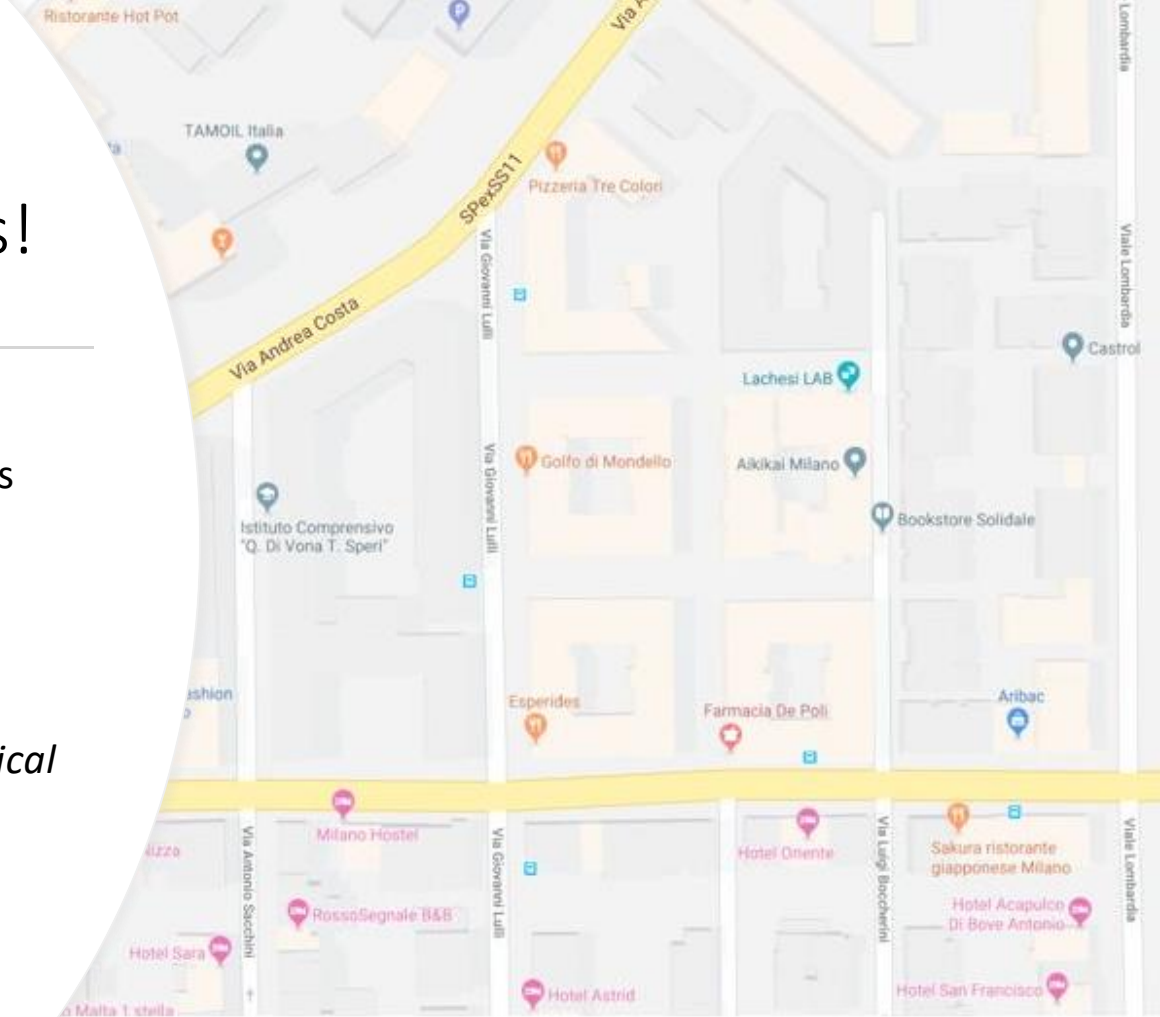
...Achieve wide acclaim!

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

Traffic maps are graphs!

- Transportation maps (e.g. *Google Maps*) can be naturally modelled as **graphs**

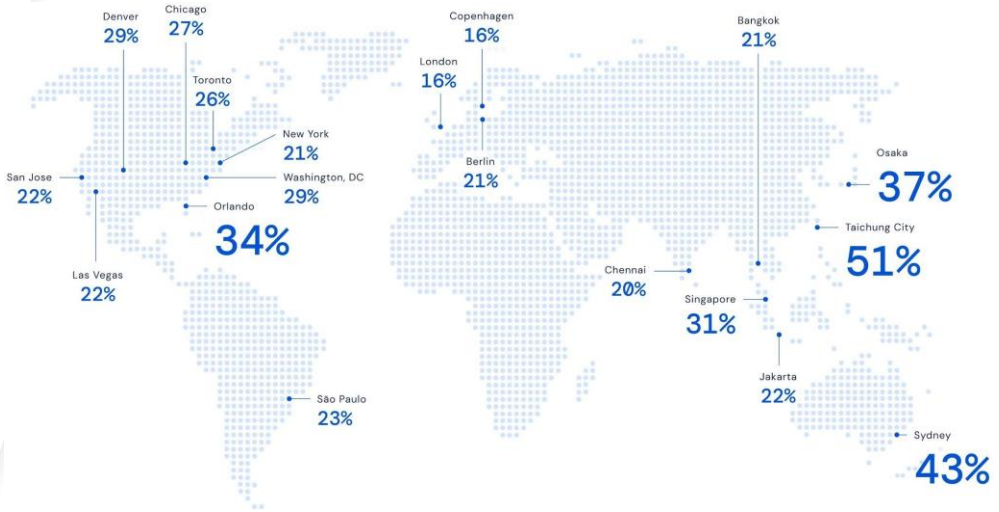
- Nodes could be **intersections**, and edges could be **roads**
  - Relevant **node features**: road *length*, *current speeds*, *historical speeds*)



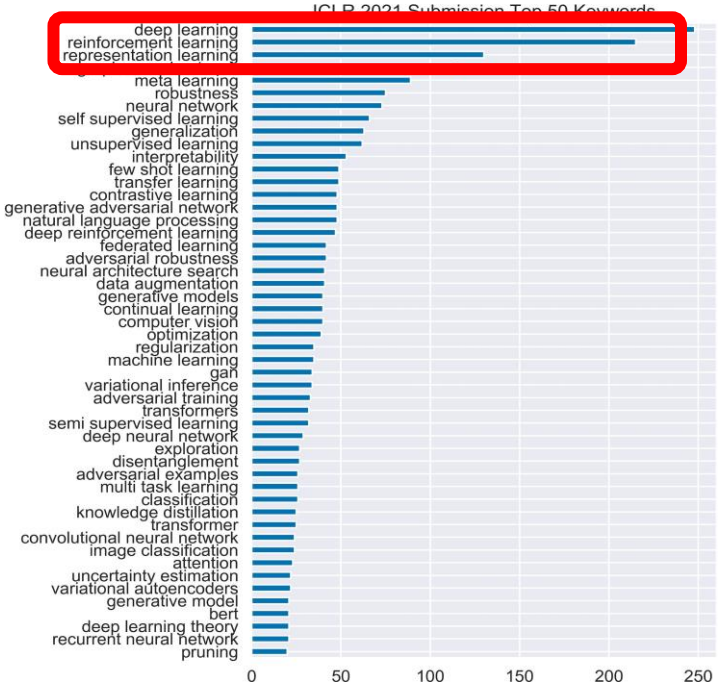
DeepMind's ETA Prediction using GNNs in Google Maps

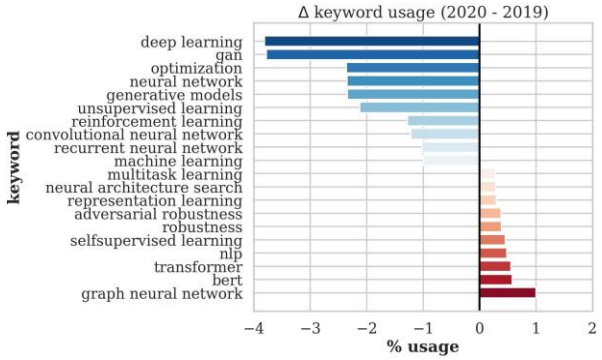
Run GNN on **supersegment** graph to estimate **time of arrival** (ETA) (*graph regression)*.

Already **deployed** in several major cities, significantly improving ETA outcomes!



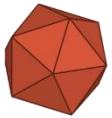
GNNs are a very hot research topic





GNNs are currently experiencing their "ImageNet" moment

Rich ecosystem of libraries



PyTorch geometric

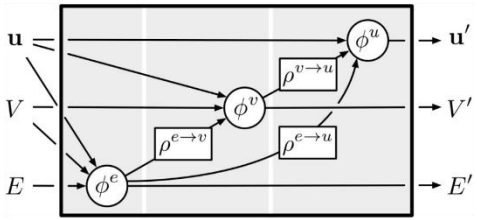
github.com/rusty1s/pytorch_geometric

Spektral

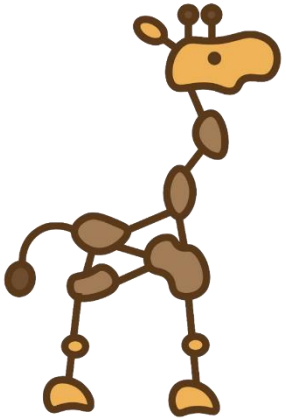
graphneural.network

DGL

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

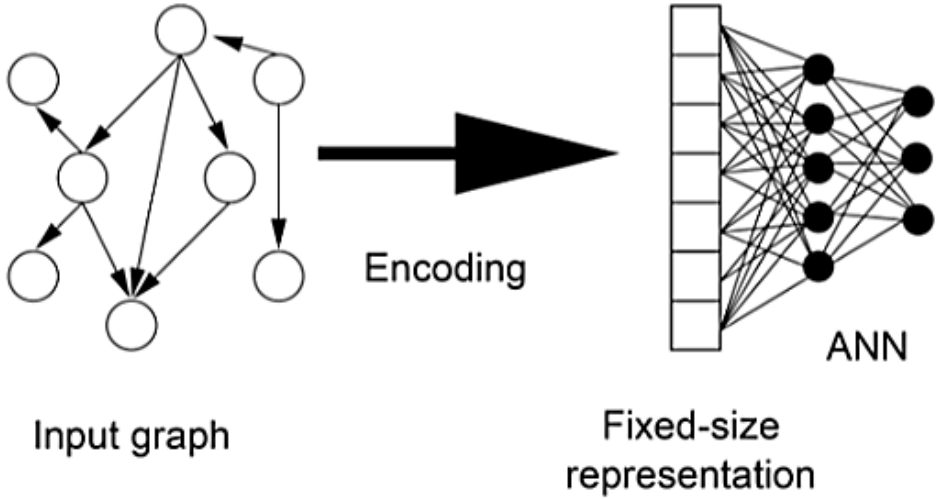
Benchmarking Graph Neural Networks

github.com/graphdeeplearning/benchmarking-gnns

Learning with GNNs – An Historical Perspective

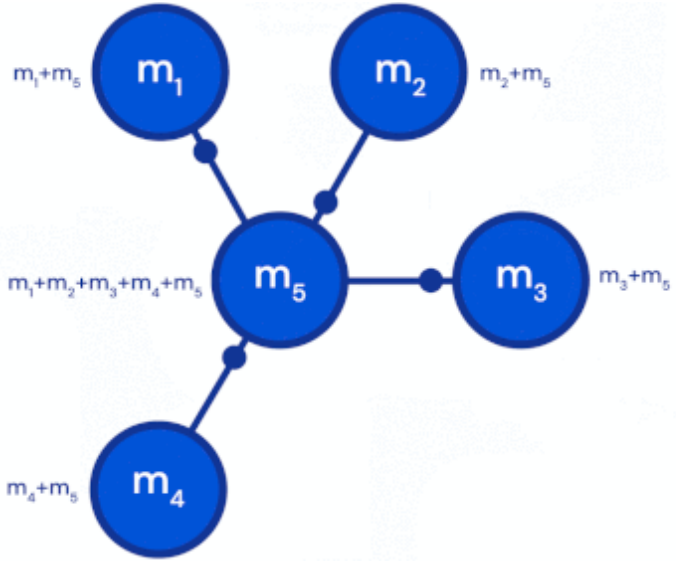
Traditional ML approaches

- ML approaches assume to deal with **flat data**
- *Pre-processing step*, extracting structural information encodings
- Flat representations relying on summary graph statistics, kernel functions, graph traversals, etc.
- **Cons**:
  - loosing useful information,
  - not able to adapt during learning



GNN Advantage

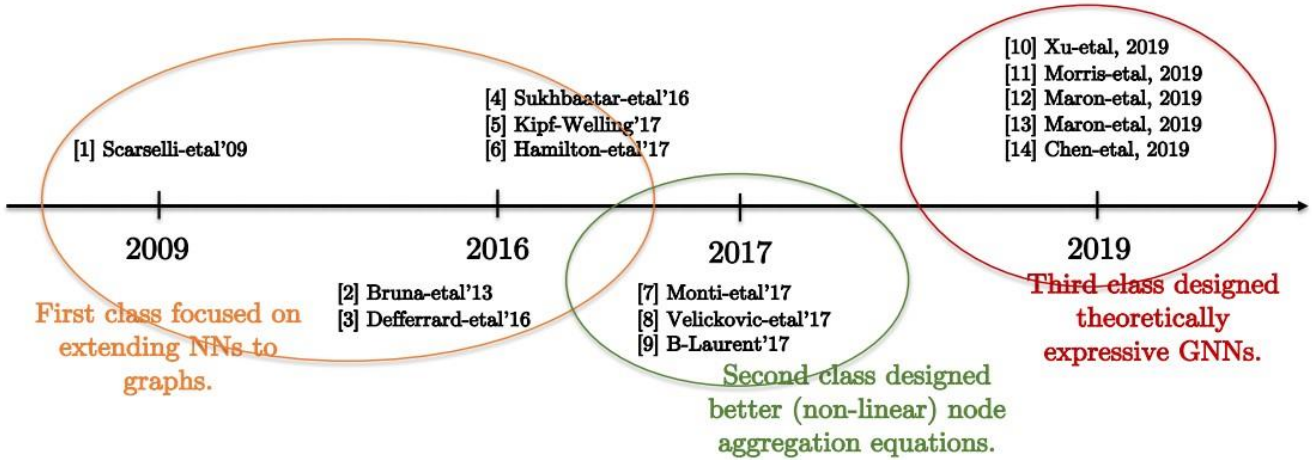
- No need of a preprocessing embedding step
- No limitation on the graph type
- Neural networks exploited to learn how to encode nodes of a graph for a given task
- Consider information local to each node and the whole graph topology



Where did GNN comes from?

- Early forms can be traced to the **early 1990s**, often involving DAG structures.
  - Backpropagation through structure (Goller & Küchler, ICNN'96)
  - Adaptive structure processing (Sperduti & Starita, TNN'97; Frasconi *et al.*, TNN'98)
- First proper treatment of **generic** graph structure processing happens in the 2000s:
  - The **GNN framework** (Gori *et al.*, IJCNN'05; Scarselli *et al.*, TNN'08)
  - The **NN4G framework** (Micheli, TNN'09)
- The GNN model of Gori, Scarselli *et al.* used primarily *recurrent*-style updates
  - Problems in the optimization of the message diffusion process

GNN - Timeline

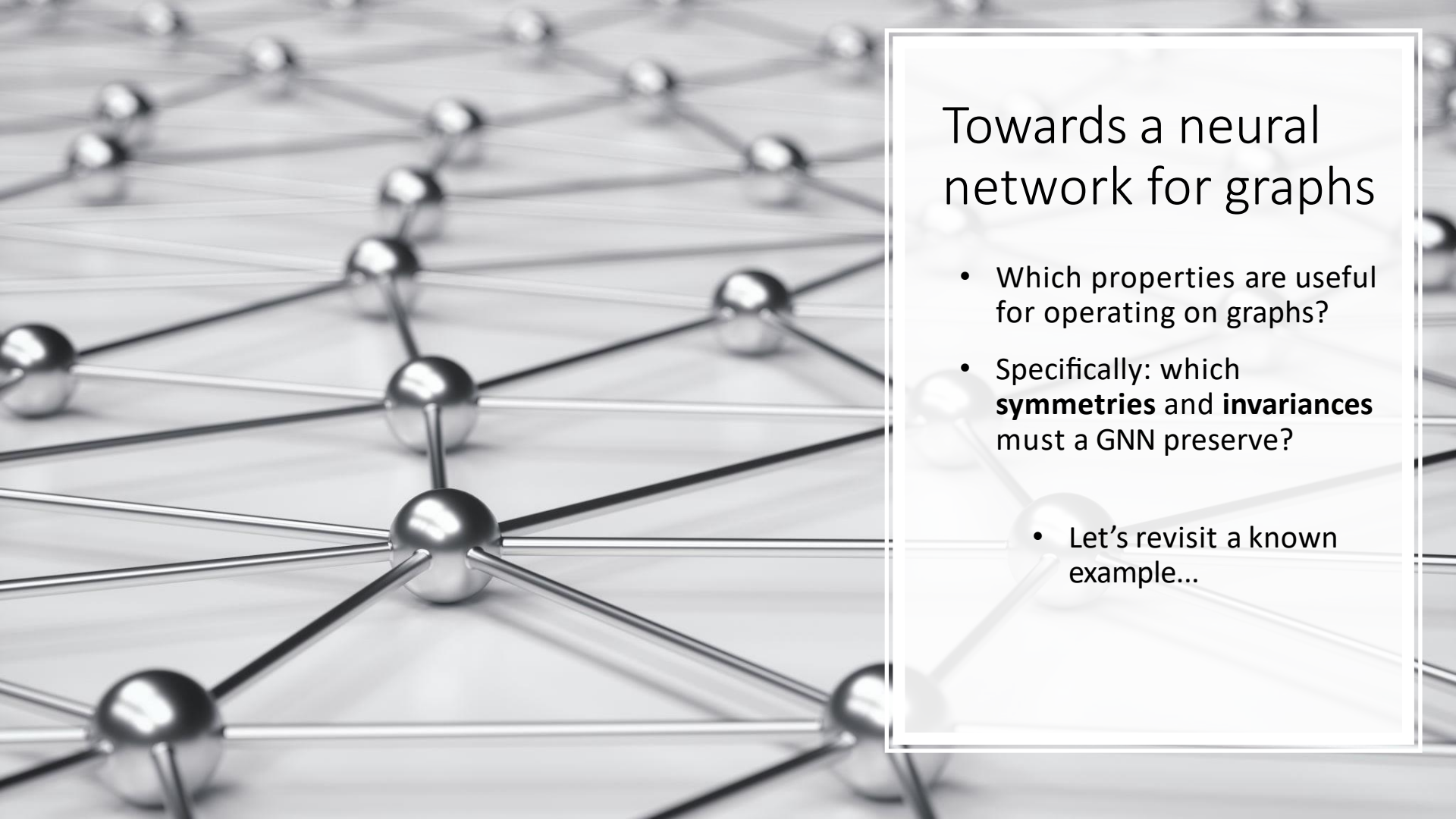


Developing powerful GNNs for real-world adoption of graph deep learning.

[1] Scarselli, Gori, Tsoi, Hagenbuchner, Monfardini, The Graph Neural Network Model, 2009
[2] Bruna, Zaremba, Szlam, LeCun, Spectral networks and locally connected networks on graphs, 2013
[3] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016
[4] Sukhbaatar, Szlam, Fergus, Learning multiagent communication with backpropagation, 2016
[5] Kipf, Welling, Semi-supervised classification with graph convolutional networks, 2017
[6] Hamilton, Ying, Leskovec, Inductive representation learning on large graphs, 2017
[7] Monti, Boscaini, Masci, Rodola, Svoboda, Bronstein, Geometric deep learning on graphs using mixture model cnns, 2017
[8] Velickovic, Cucurull, Casanova, Romero, Lio, Bengio, Graph attention networks, 2017
[9] Bresson, Laurent, Residual gated graph convnets, 2017
[10] Xu, Hu, Leskovec, Jegelka, How powerful are graph neural networks?, 2019
[11] Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, Grohe, Weisfeiler and leman go neural: Higher-order graph networks, 2019
[12] Maron, Ben-Hamu, Shamir, Lipman, Invariant and equivariant graph networks, 2019
[13] Maron, Ben-Hamu, Serviansky, Lipman, Provably powerful graph networks, 2019
[14] Chen, Villar, Chen, Bruna, On the equivalence graph isomorphism testing and function approximation with gnns, 2019

Apologize for not citing more works (4000+ GNN papers).

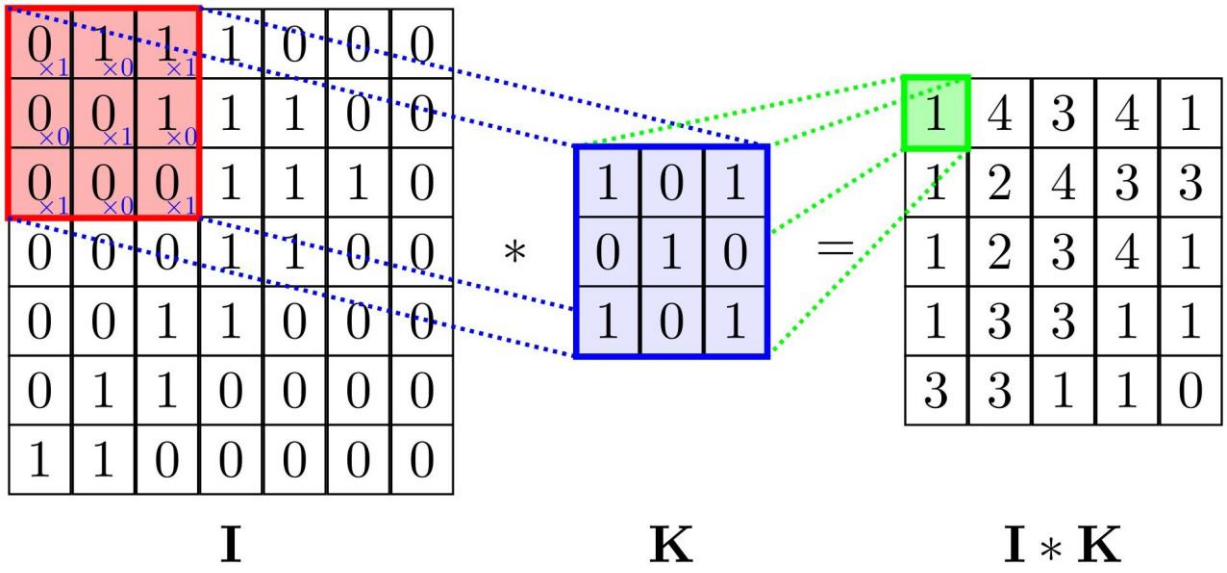
Towards GNNs from first principles



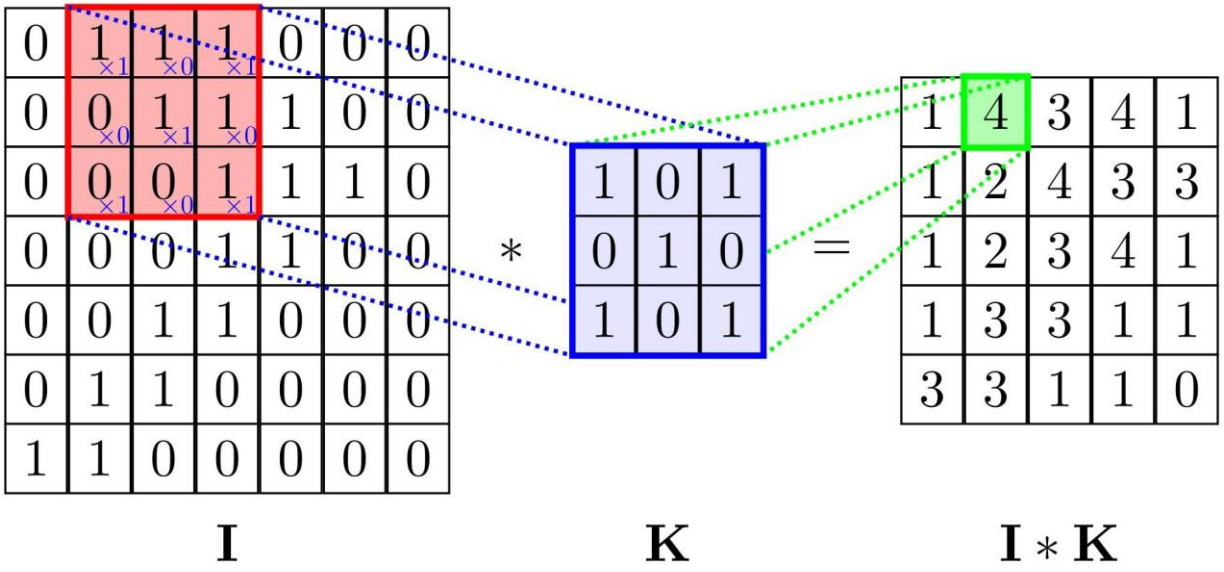
Towards a neural network for graphs

- Which properties are useful for operating on graphs?
- Specifically: which **symmetries** and **invariances** must a GNN preserve?
    - Let's revisit a known example...

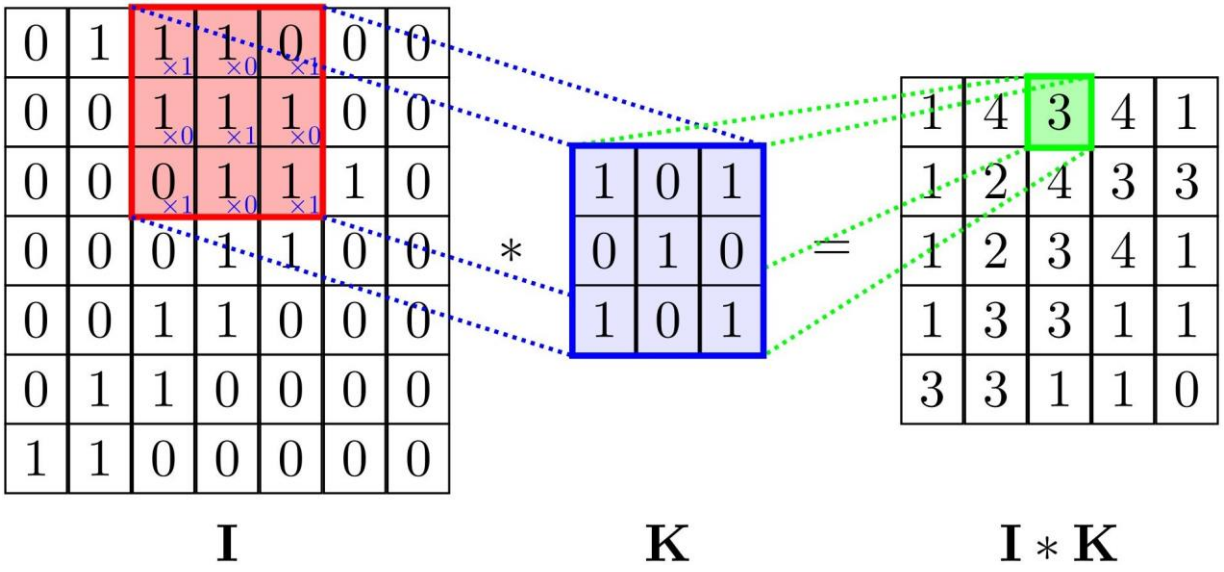
Convolution on images



Convolution on images



Convolution on images

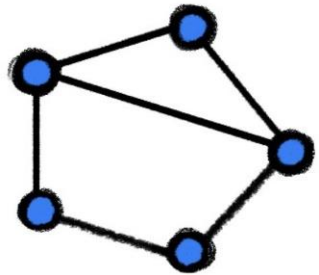


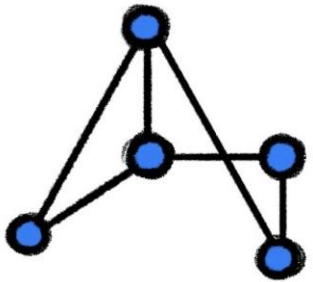
Convolutional Neural Network Invariances

- Convolutional neural nets respect **translational invariance**
- Patterns are interesting irrespective of *where* they are in the image
- **Locality**: neighbouring pixels relate much more strongly than distant ones
- What about **arbitrary** graphs?

Isomorphism-preserving transformation

- The nodes of a graph are not assumed to be in any order
- We would like to get the same results for two isomorphic graphs
- To see how to enforce this, we will define new terms...





Learning on sets: Setup

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

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

- That is, the *i*th row of **X** corresponds to $\mathbf{x}_i$

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

- We would like the result of any neural networks to not depend on this.

Permutations matrices

Permutations are the operations that change the node order

- e.g., a permutation (2, 4, 1, 3) means $\mathbf{y}_1 \leftarrow \mathbf{x}_2$, $\mathbf{y}_2 \leftarrow \mathbf{x}_4$, $\mathbf{y}_3 \leftarrow \mathbf{x}_1$, $\mathbf{y}_4 \leftarrow \mathbf{x}_3$
- there are **n!** of them

In linear algebra, each permutation defines an *n* x *n* matrix

- Such matrices are called **permutation matrices**
- They have exactly one 1 in every row and column, and zeros everywhere else
- Their effect when left-multiplied is to permute the rows of **X**, like so:

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

Permutation *invariance & equivariance*

Permutation *invariance*:

- We want to design functions f(X) over sets that will not depend on the order of the input

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

- Example of permutation invariant operation: Sum, Avg, Max
- Good models to obtain *set-level* outputs

Permutation *equivariant*:

- We want to still be able to **identify** node outputs, which a permutation-invariant aggregator would destroy!

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

- if we permute the nodes, it doesn't matter if we do it **before** or **after** the function

General *blueprint* for learning on sets

- The Equivariant set functions transform each node input $\mathbf{x}_i$ into a *latent* vector $\mathbf{h}_i$ (embedding):

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

- ψ is a function **applied in isolation to every node**.
- Stacking $\mathbf{h}_i$ yields $\mathbf{H} = f(\mathbf{X})$.

- General *blueprint*:
  stacking **equivariant** function(s), potentially aggregated with an **invariant** operator -- yields many useful functions on sets!

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

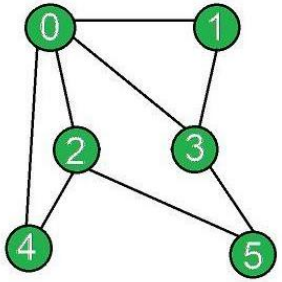
- Here, $\bigoplus$ is a permutation-invariant **aggregator** (such as sum, avg or max).

Learning on graphs

- Now we augment the set of nodes with **edges** between them: $E \subseteq V \times V$
- We represent these edges with an **adjacency matrix**, **A**:

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

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



	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	0	1	1
3	1	1	0	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

Permutation invariance and equivariance on graphs

- Node permutations now also valid on the **edges**
- We need to appropriately permute both **rows** and **columns** of **A**
  - When permutation **P**, on a matrix it amounts to $\mathbf{P}\mathbf{A}\mathbf{P}^T$
- We update definitions of suitable functions f(**X, A**) over graphs:

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

Locality on graphs: Neighborhoods!

- On **sets**, we enforced equivariance applying functions to node **in isolation**
- **Graphs** give us a broader context: a node's **neighbourhood**
  - For a node *i*, its (1-hop) neighbourhood is defined as follows:

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

N.B. we consider *undirected* edges, and often we assume $i \in N_i$

- We extract and aggregate the *multiset* of **features** in the neighbourhood

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

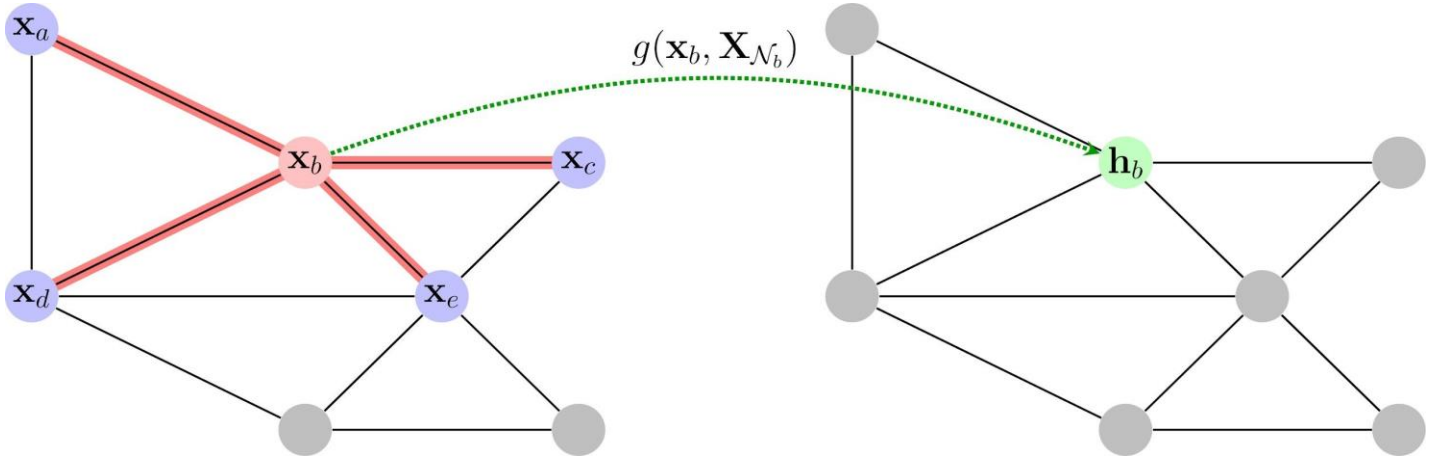
A recipe for graph neural networks

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

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

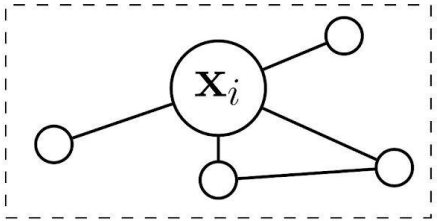
- To ensure equivariance, we need *g* to not depend on the **order** of vertices in $\mathbf{X}_{\text{Ni}}$
  - Hence, g should be permutation **invariant**!

A recipe for graph neural networks, visualised



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

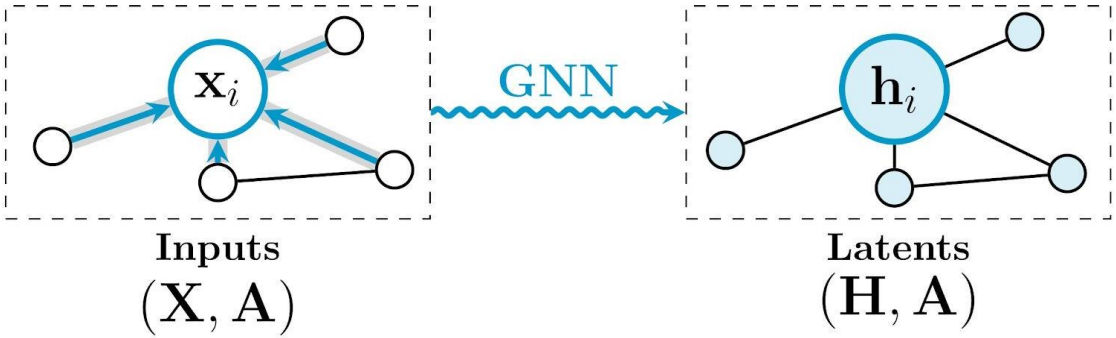
How to use GNNs?



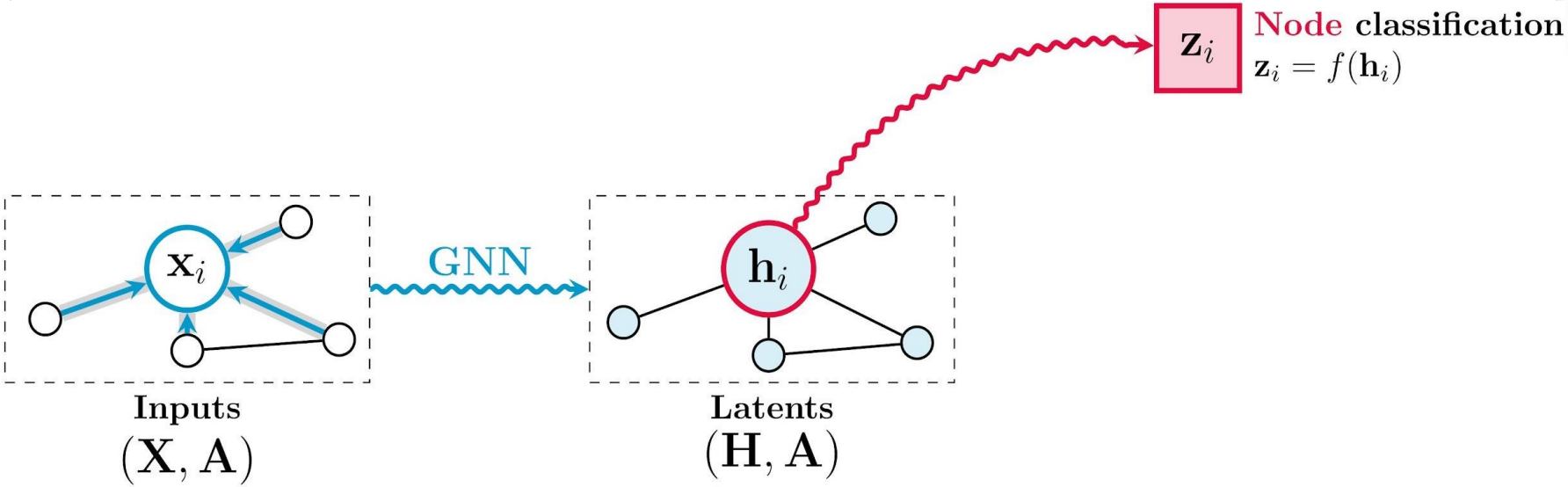
Inputs

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

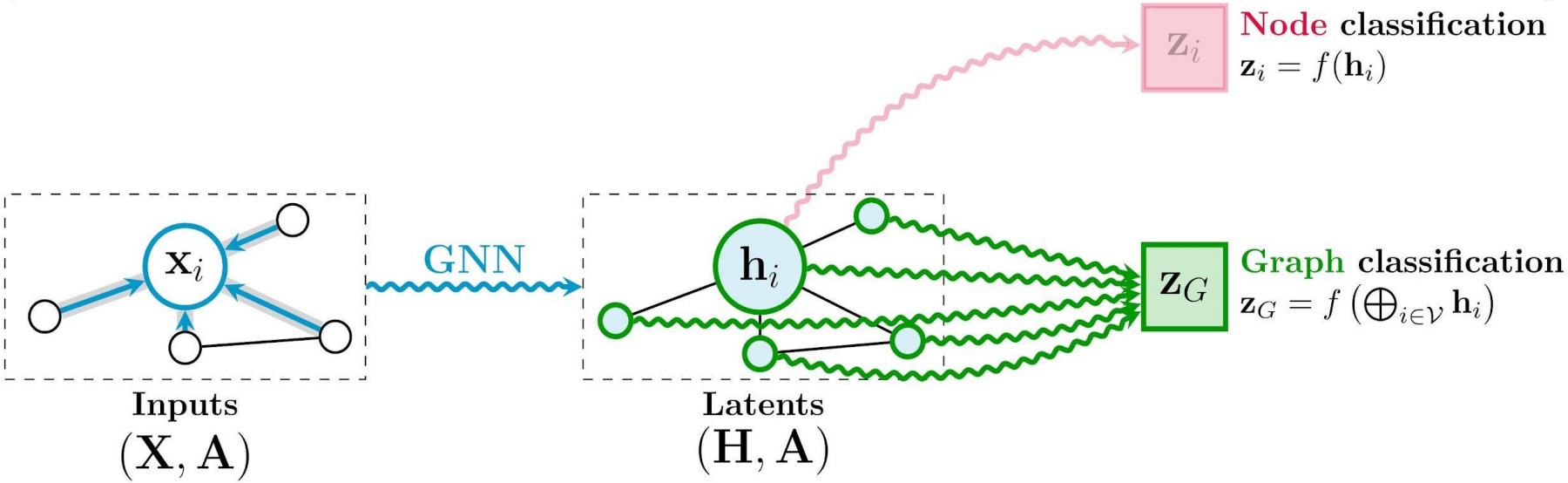
How to use GNNs?



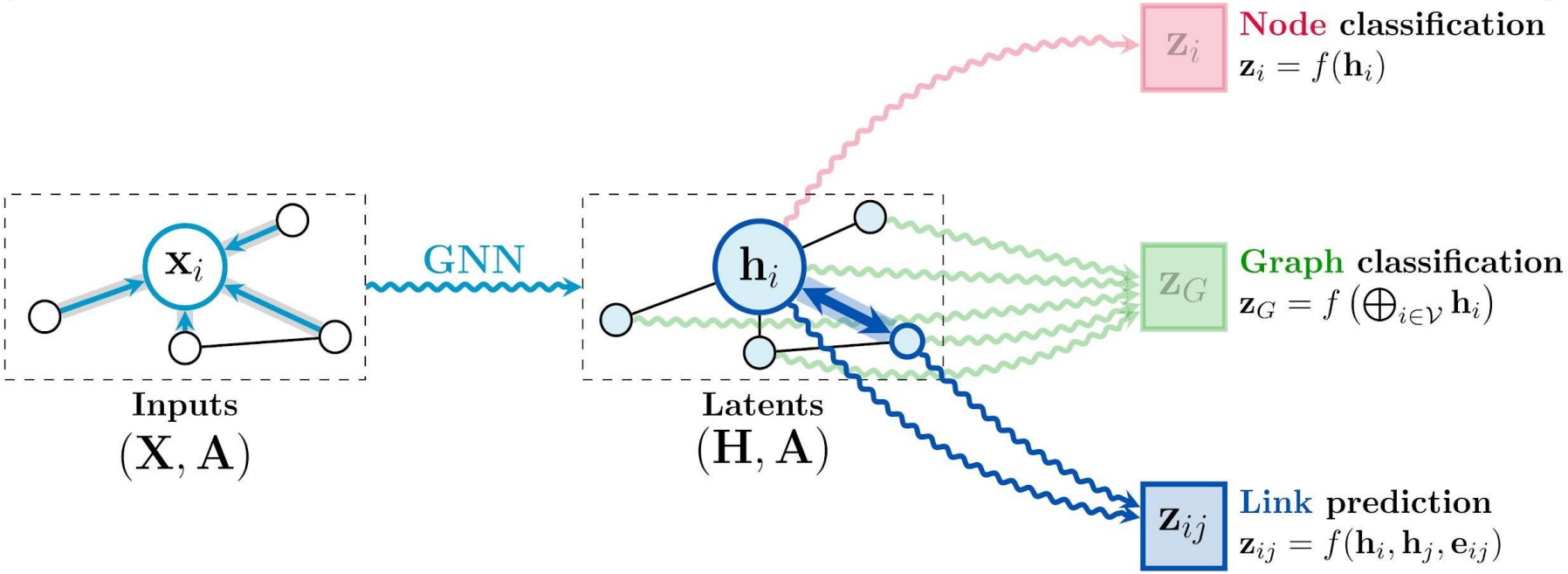
How to use GNNs?



How to use GNNs?



How to use GNNs?

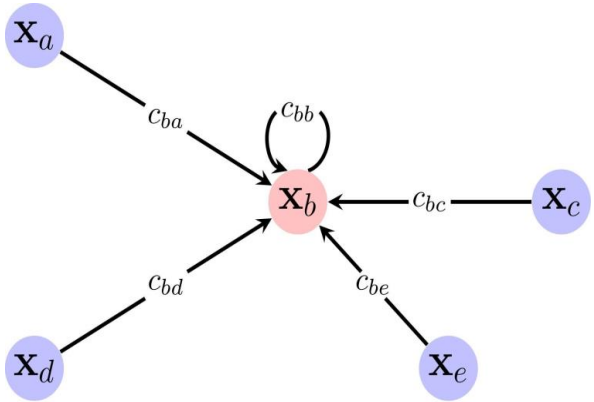


Message Passing on Graphs

What's in a GNN layer?

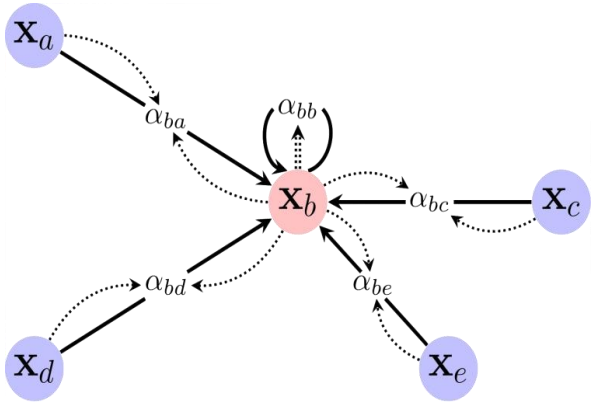
- We construct permutation-equivariant functions f(**X**, **A**) over graphs by shared application of a local permutation-invariant g($\mathbf{x}_i$, $\mathbf{X}_{Ni}$).
  - We often refer to f as "GNN layer", g as "diffusion", "propagation", "message passing"
- Now we look at ways in which we can actually and concretely **define** g.
  - **Very intense** area of research!
- Fortunately, *almost all* proposed layers can be classified as one of three ***spatial*** "*flavours*".

The 3 “flavours” of GNN layers



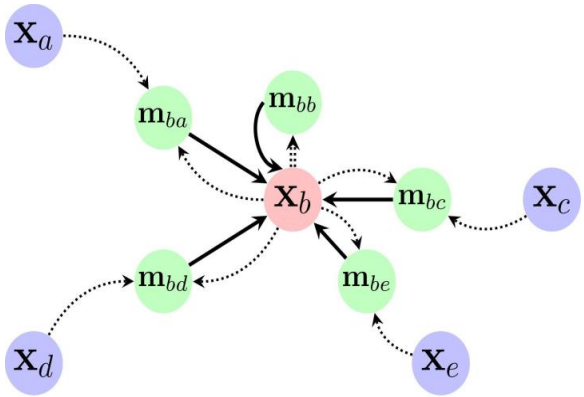
Convolutional

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



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

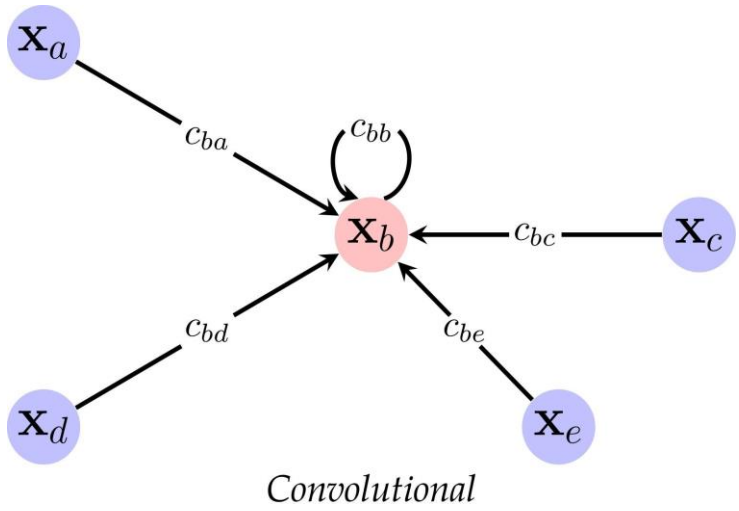
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

Convolutional GNN

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

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

- Usually, the weights depend directly on **A**.
  - ChebyNet (Defferrard *et al.*, NeurIPS'16)
  - GCN (Kipf & Welling, ICLR'17)
- Useful for **homophilous** graphs and **scaling up**
  - When edges encode *label similarity*



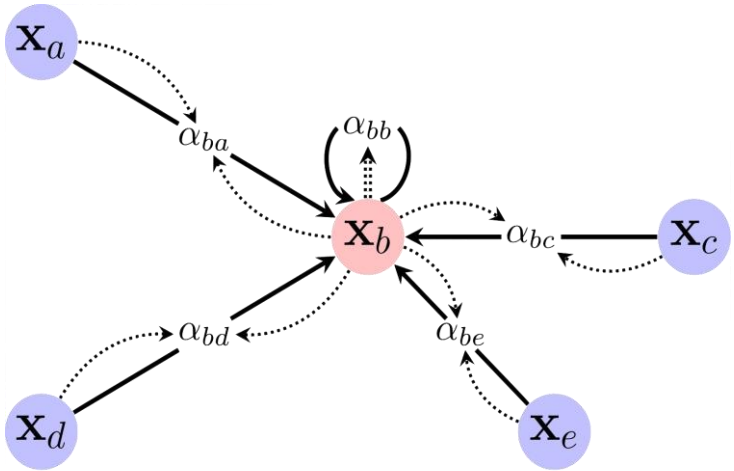
Convolutional

Attentional GNN

- Features of neighbours aggregated with **implicit** weights (via *attention*)

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

- Attention weight computed as $\alpha_{ij} = a(\mathbf{x}_i, \mathbf{x}_j)$
  - MoNet (Monti *et al.*, CVPR'17)
  - GAT (Veličković *et al.*, ICLR'18)
- Useful as "middle ground" w.r.t. **capacity** and **scale**
  - Edges need not encode homophily
  - But still compute *scalar* value in each edge



Message-passing GNN

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

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

- Messages computed as $\mathbf{m}_{ij} = \psi(\mathbf{x}_i, \mathbf{x}_j)$
  - Interaction Networks (Battaglia *et al.*, NeurIPS'16)
  - MPNN (Gilmer *et al.*, ICML'17)
- Most **generic** GNN layer
  - May have *scalability* or *learnability* issues
  - Ideal for *computational chemistry*, *reasoning* and *simulation*