

DeepMind

Theoretical Foundations of Graph Neural Networks

Petar Veličković

CST Wednesday Seminar
17 February 2021



DeepMind

In this talk:
Neural networks for graph-structured data
(Graph Neural Networks; GNNs)



DeepMind

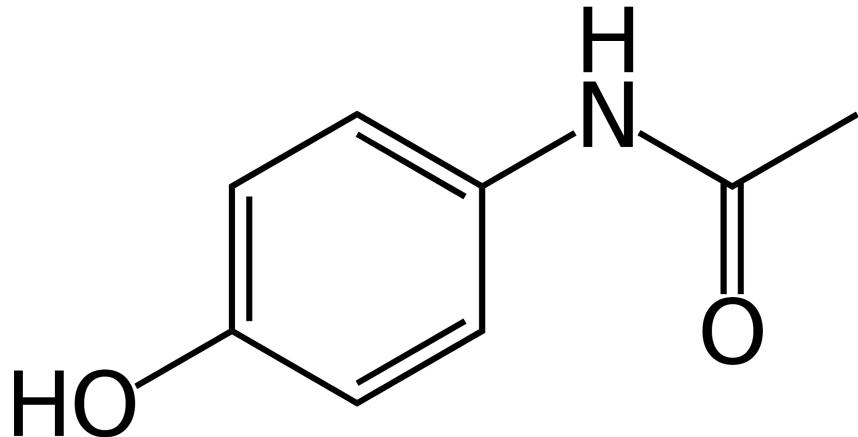
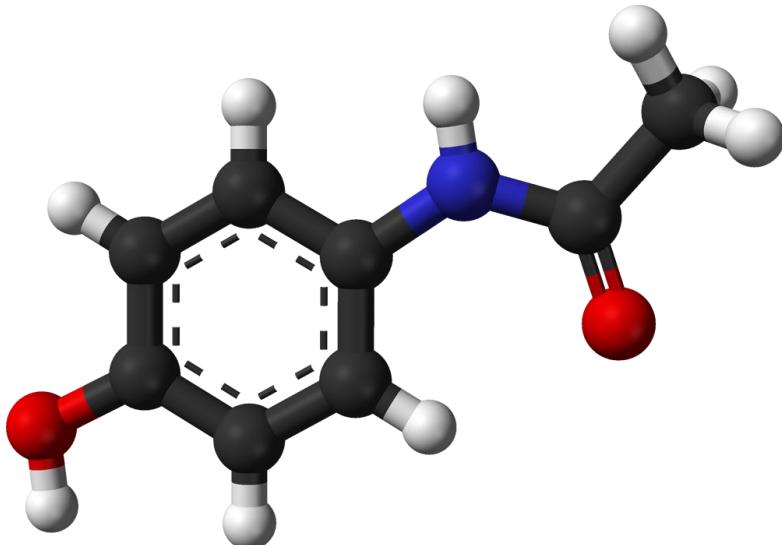
1

Fantastic GNNs in the Wild



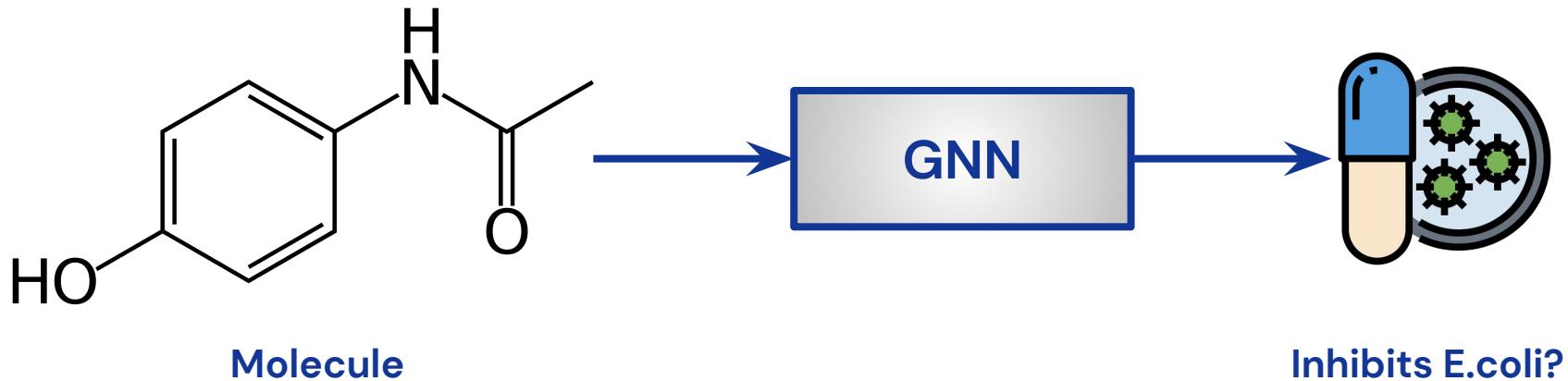
Molecules are graphs!

- 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...**



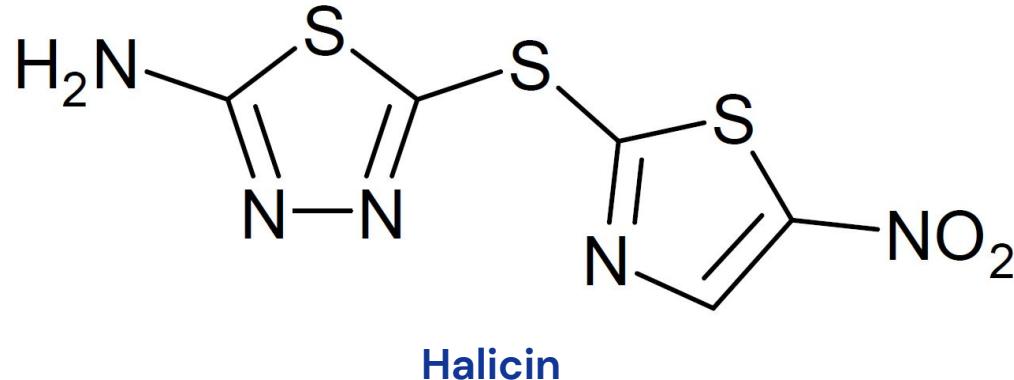
GNNs for molecule classification

- Interesting task to predict is, for example, whether the molecule is a potent **drug**.
 - Can do *binary classification* on whether the drug will inhibit certain bacteria. (*E.coli*)
 - 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.
 - Have chemists thoroughly investigate those (after some additional filtering).
- Discover a previously overlooked compound that is a **highly potent** antibiotic!



...Achieve wide acclaim!

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

Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract

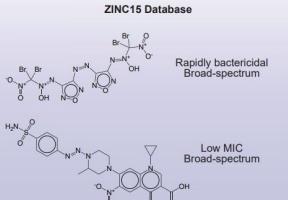
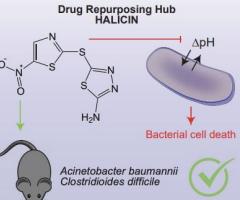
The graphical abstract illustrates a deep learning pipeline for antibiotic discovery. It starts with a 'Directed message passing neural network' (DMPNN) processing two molecules. The output of the DMPNN is used to predict antibiotic activity ('Antibiotic predictions (upper limit $10^{11} +$)'). This prediction is compared against a 'Training set (10^4 molecules)' to validate the model ('Model validation'). The validation process involves 'Growth [antibiotic]' curves, leading to 'New Antibiotics'. A funnel diagram on the right shows the exponential reduction of molecules from 10^8 to 10^4 . The bottom section shows the 'Drug Repurposing Hub HALICIN' targeting *Acinetobacter baumannii* and *Clostridioides difficile*, leading to 'Bacterial cell death'. The 'ZINC15 Database' is also mentioned, highlighting 'Rapidly bactericidal Broad-spectrum' and 'Low MIC Broad-spectrum' antibiotics.

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.

(Stokes et al., Cell'20)





...Achieve wide acclaim!

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

The screenshot shows a news article from the journal **nature**. At the top right is a blue "Subscribe" button. Below the header, the word "ARTICLE" is visible. The main headline reads: **Powerful antibiotics discovered using AI**. The sub-headline states: "Machine learning spots molecules that work even against 'untreatable' strains of bacteria." At the bottom left, it says "(Stokes et al., Cell'20)". A small graphic at the bottom features a mouse icon, the names of two bacteria (*Acinetobacter baumannii* and *Clostridioides difficile*), a green checkmark, and a chemical structure labeled "Broad-spectrum".

ARTICLE

nature

Subscribe

NEWS · 20 FEBRUARY 2020

Powerful antibiotics discovered using AI

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

(Stokes et al., Cell'20)

Acinetobacter baumannii
Clostridioides difficile

Broad-spectrum



...Achieve wide acclaim!

Arguably the most popular

nature

NEWS · 20 FEBRUARY 2020

Powerful an

Machine learning spots
bacteria.

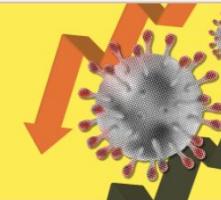
(Stokes et al., Cell'20)

FINANCIAL TIMES

S COMPANIES TECH MARKETS GRAPHICS OPINION WORK & CAREERS LIFE & ARTS HOW TO SPEND IT

CORONAVIRUS BUSINESS UPDATE

Get 30 days' complimentary access to our Coronavirus Business Update newsletter



Intelligence

Robotics



'Death of the office' homeworking claims exaggerated



Anti-social robots harr
increase social distanc

Artificial intelligence

+ Add to myFT

AI discovers antibiotics to treat drug-resistant diseases

Machine learning uncovers potent new drug able to kill 35 powerful bacteria



...Achieve wide acclaim!

The image displays a screenshot of a BBC News website. At the top, there is a navigation bar with the BBC logo, a 'Sign in' button, and links for 'News', 'Sport', 'Reel', 'Worklife', 'Travel', and 'Future'. Below this is a large red banner with the word 'NEWS' in white. Underneath the banner is a secondary navigation bar with links for 'Home', 'Video', 'World', 'UK', 'Business', 'Tech', 'Science', 'Stories', and 'Entertainment & Arts'. A blue box labeled 'BBC WORKLIFE' is positioned on the left side of the main content area. The main headline reads 'Our new guide for getting ahead'. Below the headline, a large article summary is visible, featuring the title 'Scientists discover powerful antibiotic using AI' and a sub-headline 'Machine learning uncovers potent new drug able to kill 35 powerful bacteria'. The date '21 February 2020' is listed next to the sub-headline. To the right of the main content, a vertical sidebar titled 'TIMES' shows a section titled 'HOW TO SPEND IT' with an image of a coronavirus and a downward-pointing arrow. Another article in the sidebar discusses 'Anti-social robots' and their impact on social distancing. The bottom right corner of the image features a small blue circular logo with a white swirl pattern.

Argua

NEWS

Home | Video | World | UK | Business | Tech | Science | Stories | Entertainment & Arts

BBC WORKLIFE

Our new guide for getting ahead

Scientists discover powerful antibiotic using AI

Machine learning uncovers potent new drug able to kill 35 powerful bacteria

21 February 2020

Share

(Stokes et al., Cell'20)

Machine learning uncovers potent new drug able to kill 35 powerful bacteria

TIMES

HOW TO SPEND IT

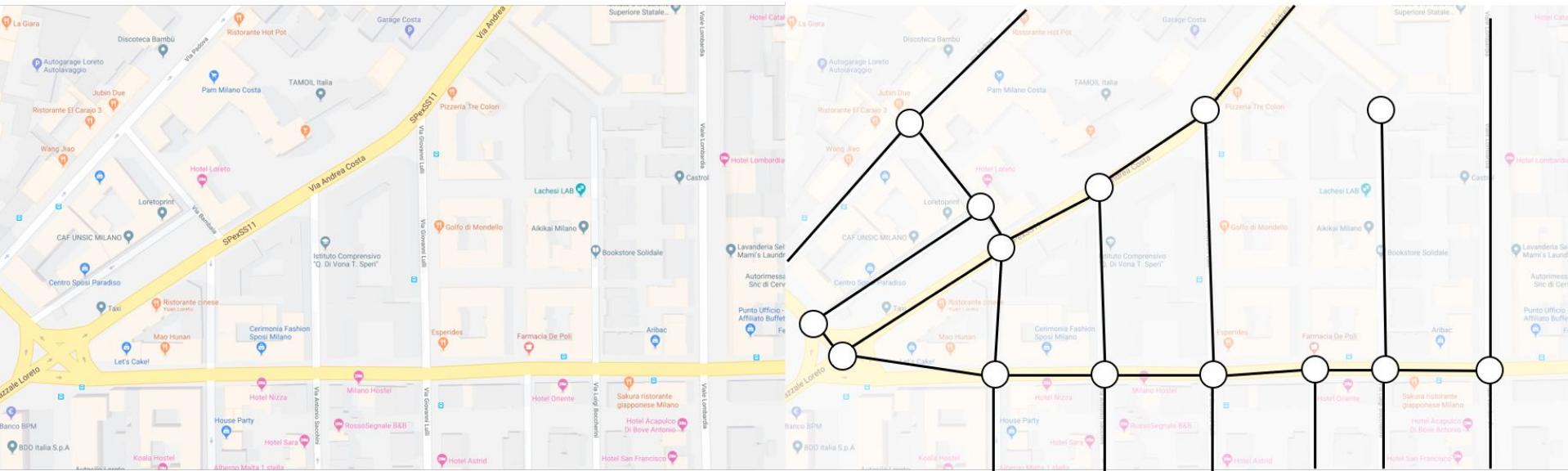
Anti-social robots harr increase social distanc

g-resistant

© 2020

Traffic maps are graphs!

Transportation maps (e.g. the ones found on Google Maps) naturally modelled as **graphs**.

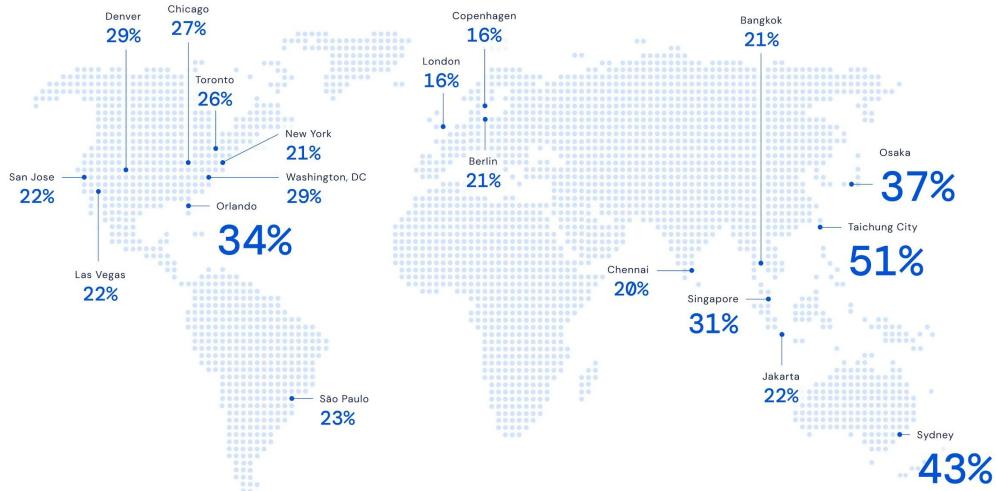


Nodes could be **intersections**, and edges could be **roads**. (Relevant **node features**: road *length*, *current speeds*, *historical speeds*)



DeepMind's ETA Prediction using GNNs in Google Maps

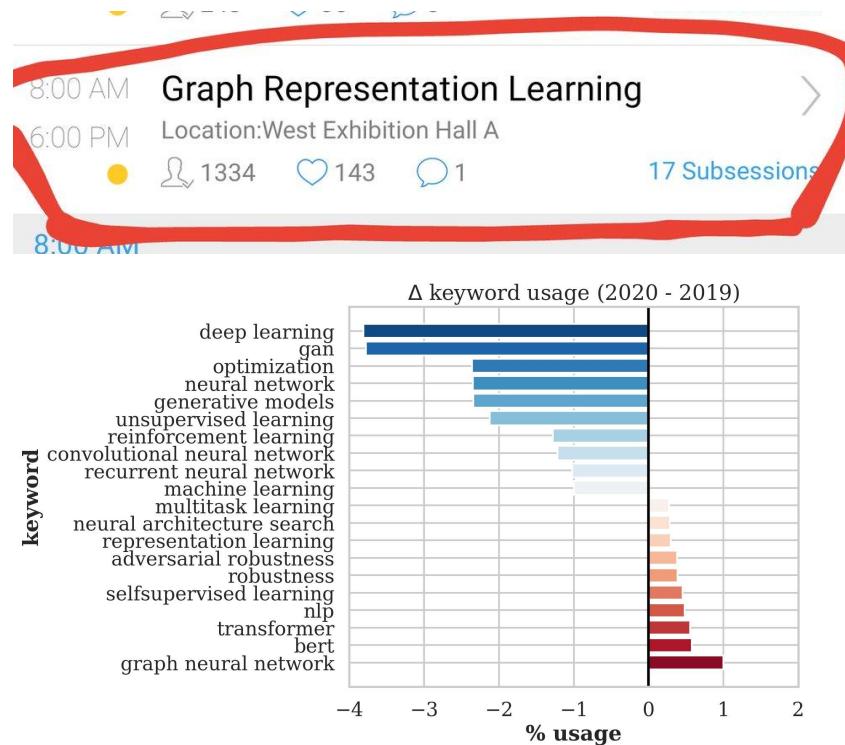
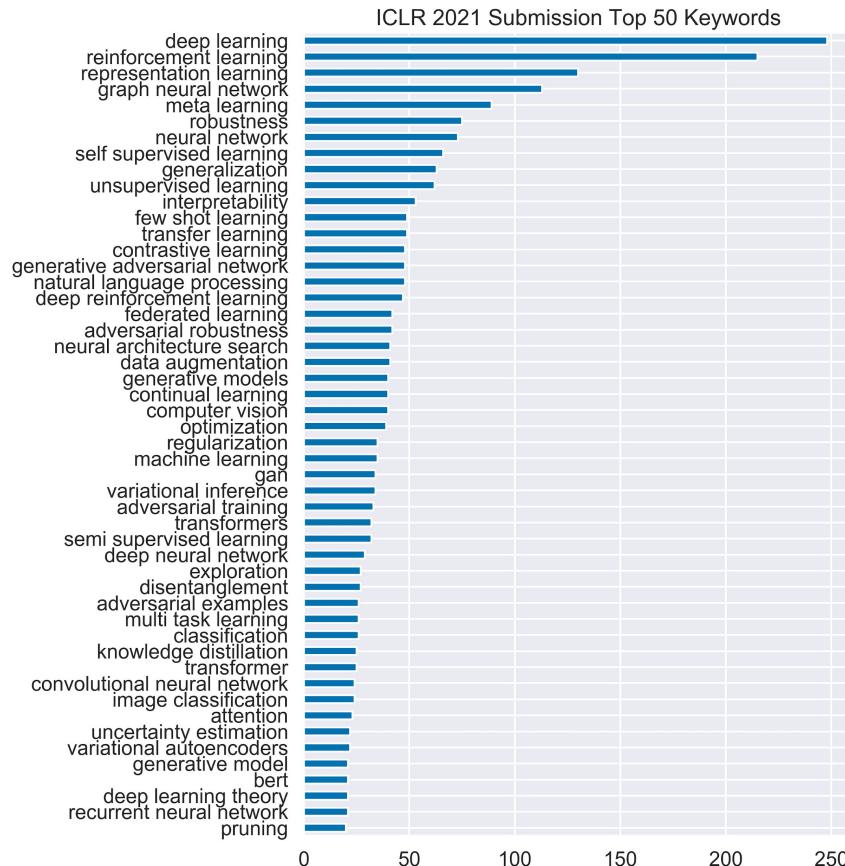
Run GNN on **supersegment** graph to estimate **time of arrival** (ETA) (*graph regression*).



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



GNNs are a very hot research topic



GNNs are currently experiencing their
“ImageNet” moment



Rich ecosystem of libraries



PyTorch
geometric

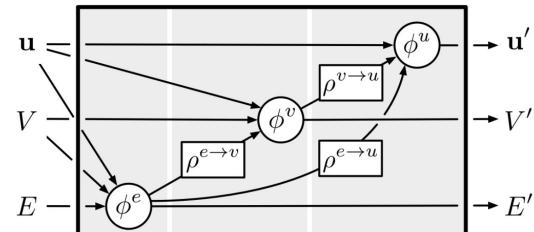
github.com/rusty1s/pytorch_geometric

DGL
dgl.ai



Spektral

graphneural.network



github.com/deepmind/graph_nets



github.com/deepmind/jraph



Rich ecosystem of datasets



ogb.stanford.edu



PyTorch
geometric

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



graphlearning.io

Benchmarking Graph Neural Networks

github.com/graphdeeplearning/benchmarking-gnns



2

Talk roadmap



What will we cover today?

- Hopefully I've given you a convincing argument for **why** GNNs are useful to study
 - For more details and applications, please see e.g. my *EEML 2020* talk
- My aim for today: provide good **blueprints** and **contexts** for studying the field
 - Derive GNNs from first principles
 - Position this in context of several *independently-studied* derivations of GNNs
 - Often using *drastically* different mathematical tools...
 - Look to the *past*: how GNN-like models emerged in historical ML research
 - Look to the *present*: some *immediate* lines of research interest
 - Look to the *future*: how our blueprint generalises beyond graph-structured inputs
- Hopefully my perspective is of use both to newcomers and seasoned GNN practitioners
 - **Any** and **all** feedback very welcome!



What is the content based on?

- GNN derivation + further horizons inspired by my work on **geometric deep learning**
 - Ongoing collaboration with Joan Bruna, Michael Bronstein and Taco Cohen
- Various contexts of GNN study inspired by Will Hamilton's **GRL Textbook** (esp. Chapter 7)
 - https://www.cs.mcgill.ca/~wlh/grl_book/
 - Highly recommended!
- Historical contexts developed with input of several researchers
 - Thanks to Yoshua Bengio, Marco Gori, Jürgen Schmidhuber, Christian Merkwirth and Marwin Segler
- But of course, any errors and omissions are mine alone.



Disclaimer before advancing

- My talk **content** is geared to a *general* Computer Science audience
 - We will construct “useful” functions operating over graphs
 - We will use concepts commonly encountered in a CS curriculum
- **Implementation** requires background in machine learning with deep neural networks
 - Useful resource to get started: “**Deep Learning**” by Goodfellow, Bengio and Courville
 - <https://www.deeplearningbook.org/>
- I recently compiled a list of many useful GNN resources in a **Twitter thread**
 - https://twitter.com/PetarV_93/status/1306689702020382720
- When you feel ready, I **highly** recommend Aleksa Gordić’s GitHub repository on GATs:
 - <https://github.com/gordicaleksa/pytorch-GAT>
 - Arguably the most *gentle* introduction to GNN implementations



3

Towards GNNs from first principles



Towards a neural network for graphs

- We will now work towards defining a GNN from first principles
- What properties are useful for operating meaningfully on graphs?
- Specifically: what **symmetries** and **invariances** must a GNN preserve?
 - Let's revisit a known example...



Convolution on images

0 $\times 1$	1 $\times 0$	1 $\times 1$	1	0	0	0
0 $\times 0$	0 $\times 1$	1 $\times 0$	1	1	0	0
0 $\times 1$	0 $\times 0$	0 $\times 1$	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

I

*

1	0	1
0	1	0
1	0	1

K

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

I * K



Convolution on images

0	1 $\times 1$	1 $\times 0$	1 $\times 1$	0	0	0
0	0 $\times 0$	1 $\times 1$	1 $\times 0$	1	0	0
0	0 $\times 1$	0 $\times 0$	1 $\times 1$	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

I

1	0	1
0	1	0
1	0	1

K

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

I * K



Convolution on images

0	1	1 $\times 1$	1 $\times 0$	0 $\times 1$	0	0
0	0	1 $\times 0$	1 $\times 1$	1 $\times 0$	0	0
0	0	0 $\times 1$	1 $\times 0$	1 $\times 1$	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

I

1	0	1
0	1	0
1	0	1

K

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

I * K



Convolution on images

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

I

1	0	1
0	1	0
1	0	1

K

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

I * K



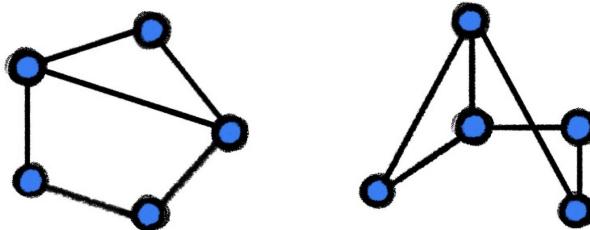
Convolutional neural network invariances

- 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
- That is, we would like to get the same results for two isomorphic graphs



- To see how to enforce this, we will define new terms...



4

Permutation invariance and equivariance



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 $n \times k$:

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

- That is, 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.



Permutations and permutation matrices

- It will be useful to think about the operations that **change** the node order
 - Such operations are known as **permutations** (there are $n!$ of them)
 - e.g. a permutation $(2, 4, 1, 3)$ means $\mathbf{y}_1 \leftarrow \mathbf{x}_{2'}, \mathbf{y}_2 \leftarrow \mathbf{x}_{4'}, \mathbf{y}_3 \leftarrow \mathbf{x}_{1'}, \mathbf{y}_4 \leftarrow \mathbf{x}_{3'}$.
- To stay within linear algebra, each permutation defines an $n \times n$ **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 \mathbf{X} , like so:

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



Permutation *invariance*

- We want to design functions $f(\mathbf{X})$ over sets that will not depend on the order
- Equivalently, applying a permutation matrix shouldn't modify the result!
- We arrive at a useful notion of permutation invariance. We say that $f(\mathbf{X})$ is *permutation invariant* if, for *all* permutation matrices P :

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

- One very generic form is the *Deep Sets* model (Zaheer et al., NeurIPS'17): $f(\mathbf{X}) = \phi \left(\sum_{i \in \mathcal{V}} \psi(\mathbf{x}_i) \right)$ where ψ and ϕ are (learnable) functions, e.g. MLPs.
 - The **sum** aggregation is *critical*! (other choices possible, e.g. **max** or **avg**)



Permutation *equivariance*

- Permutation-*invariant* models are a good way to obtain set-level outputs
- What if we would like answers at the **node** level?
 - We want to still be able to **identify** node outputs, which a permutation-invariant aggregator would destroy!
- We may instead seek functions that don't **change** the node order
 - i.e. if we permute the nodes, it doesn't matter if we do it **before** or **after** the function!
- Accordingly, we say that $f(\mathbf{X})$ is permutation equivariant if, for all permutation matrices \mathbf{P} :

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



General blueprint for learning on sets

- Equivariance mandates that each node's row is unchanged by f . That is, we can think of equivariant set functions as transforming each node input \mathbf{x}_i into a *latent* vector \mathbf{h}_i :

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

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

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

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

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



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

5

Learning on graphs



Learning on graphs

- Now we augment the set of nodes with **edges** between them.
 - That is, we consider general $E \subseteq V \times V$.
- We can represent these edges with an **adjacency matrix**, A , such that:

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

- Further additions (e.g. *edge features*) are possible but **ignored** for simplicity.
- Our main desiderata (*permutation {in,equi}variance*) still hold!



Permutation invariance and equivariance on graphs

- The main difference: node permutations now also accordingly act on the **edges**
- We need to appropriately permute both **rows** and **columns** of \mathbf{A}
 - When applying a permutation matrix \mathbf{P} , this amounts to $\mathbf{P}\mathbf{A}\mathbf{P}^T$
- We arrive at updated definitions of suitable functions $f(\mathbf{X}, \mathbf{A})$ over graphs:

Invariance: $f(\mathbf{P}\mathbf{X}, \mathbf{P}\mathbf{A}\mathbf{P}^T) = f(\mathbf{X}, \mathbf{A})$

Equivariance: $f(\mathbf{P}\mathbf{X}, \mathbf{P}\mathbf{A}\mathbf{P}^T) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$



Locality on graphs: neighbourhoods

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

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

N.B. we do not explicitly consider *directed* edges, and often we assume $i \in \mathcal{N}_i$

- Accordingly, we can extract the *multiset* of **features** in the neighbourhood

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

and define a *local* function, g , as operating over this multiset: $g(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$.



A recipe for graph neural networks

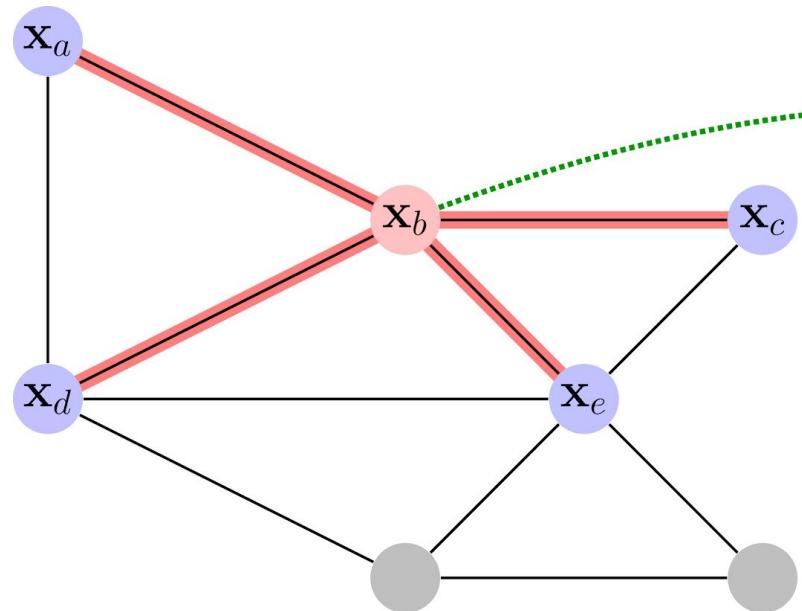
- Now we can construct permutation equivariant functions, $f(\mathbf{X}, \mathbf{A})$, by appropriately applying the local function, g , over *all* neighbourhoods:

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

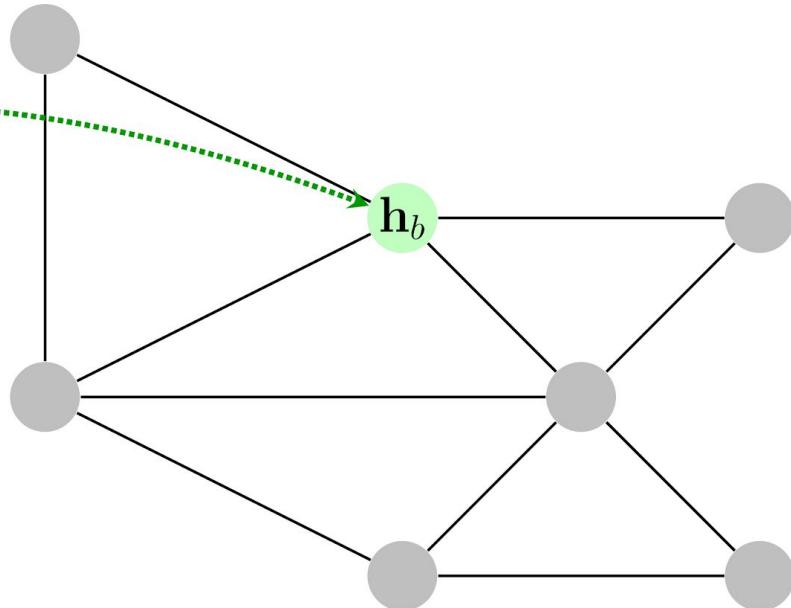
- To ensure equivariance, we need g to not depend on the **order** of the vertices in $\mathbf{X}_{\mathcal{N}_i}$
 - Hence, g should be permutation **invariant**!



A recipe for graph neural networks, visualised



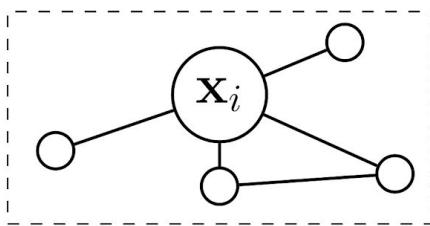
$$g(\mathbf{x}_b, \mathbf{X}_{\mathcal{N}_b})$$



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



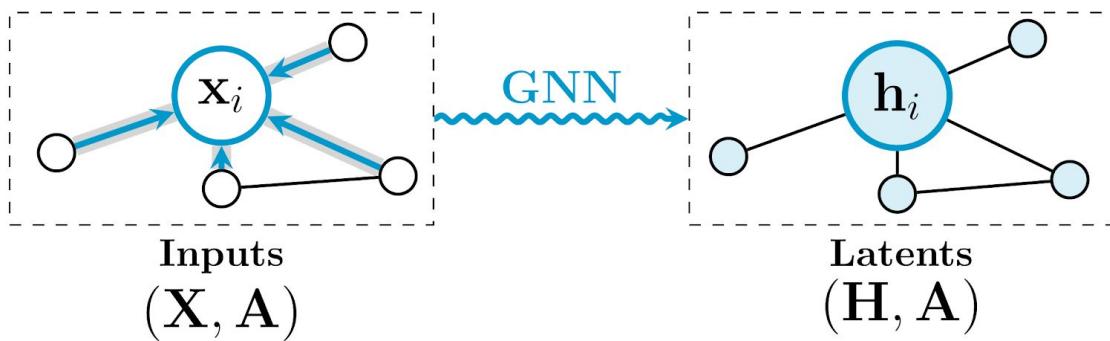
How to use GNNs?



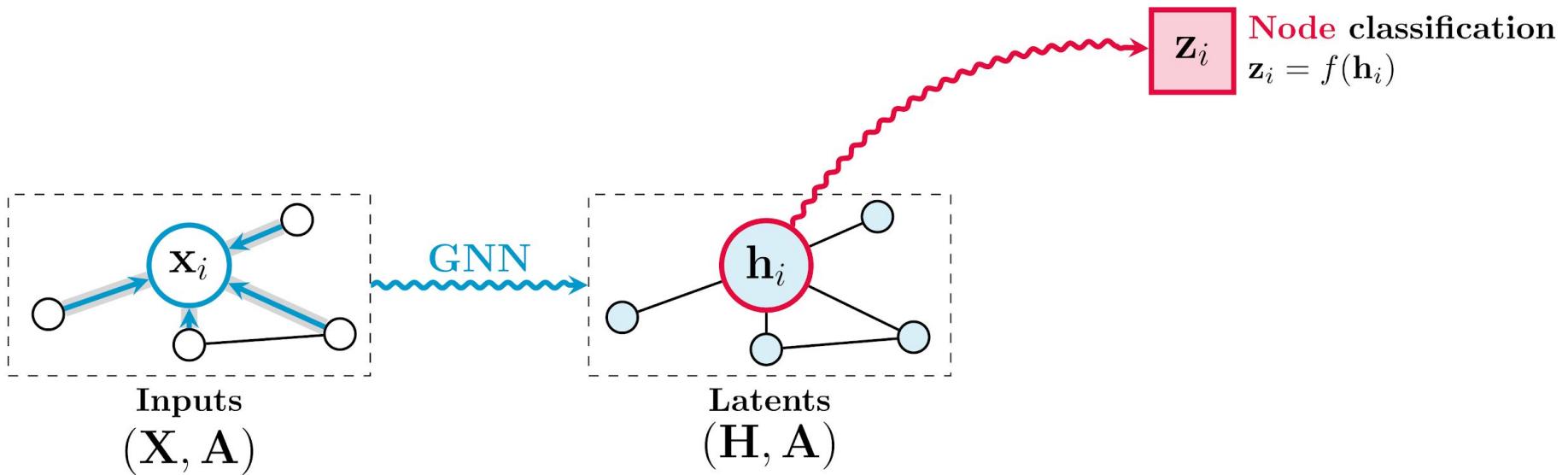
Inputs
(\mathbf{X} , \mathbf{A})



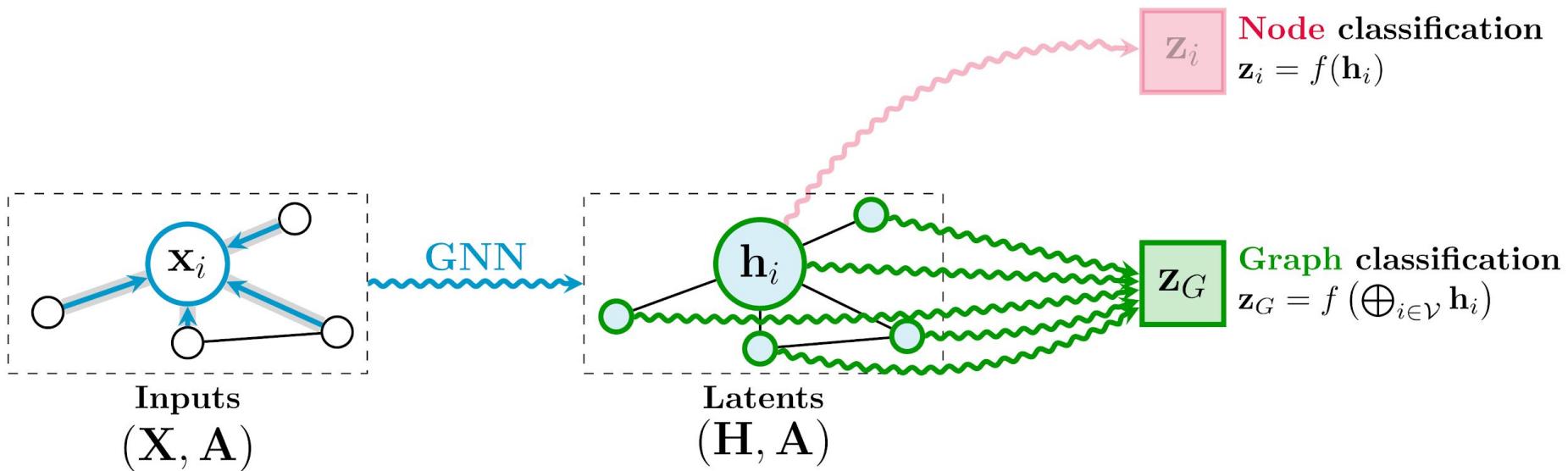
How to use GNNs?



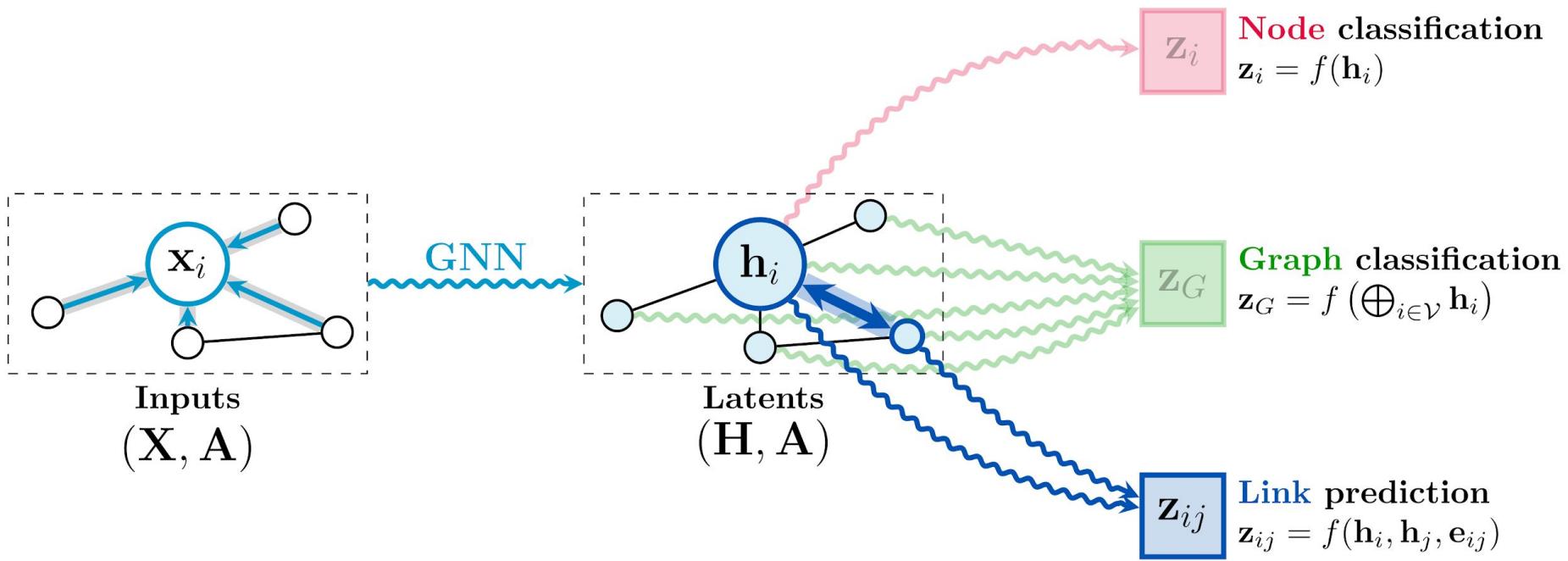
How to use GNNs?



How to use GNNs?



How to use GNNs?



6

Message passing on graphs

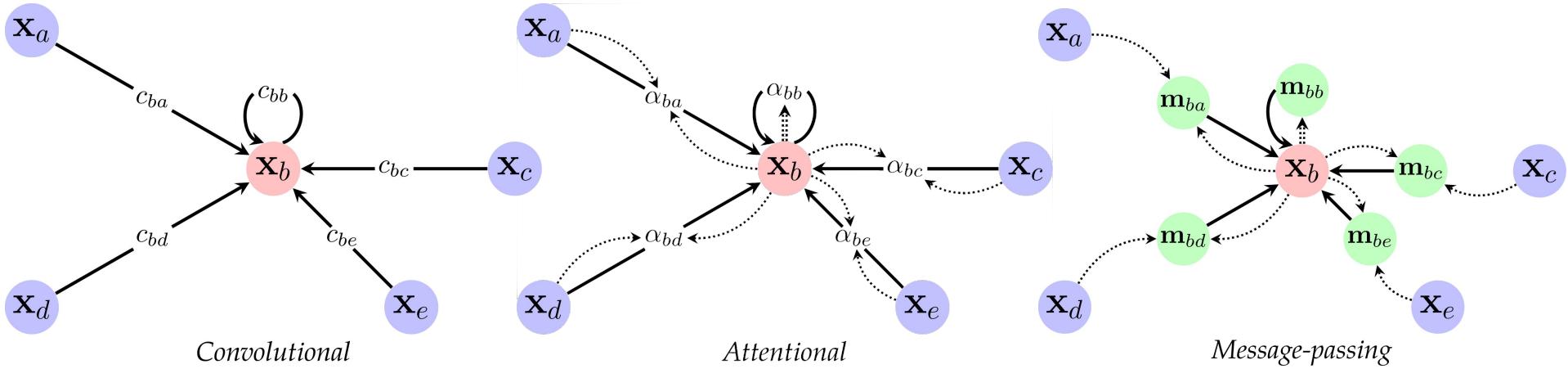


What's in a GNN layer?

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



The three “flavours” of GNN layers



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

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

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

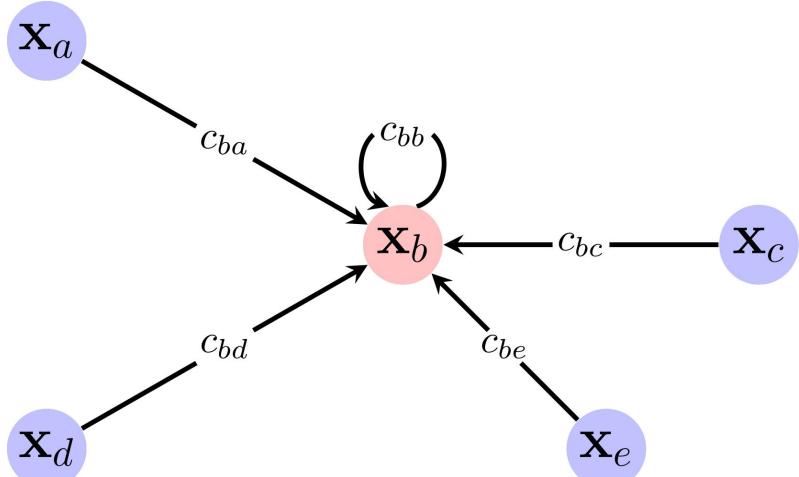


Convolutional GNN

- 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)
 - SGC (Wu et al., ICML'19)
- Useful for **homophilous** graphs and **scaling up**
 - When edges encode *label similarity*

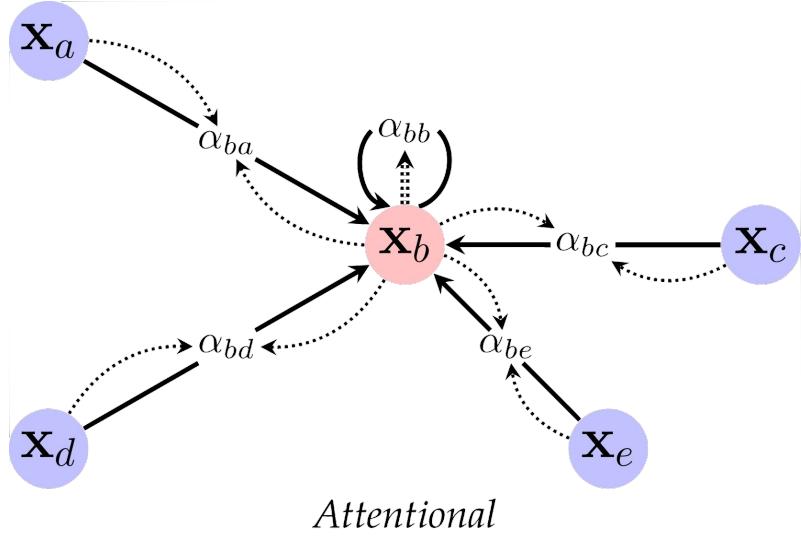


Attentional GNN

- 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)
 - GaAN (Zhang et al., UAI'18)
- Useful as “middle ground” w.r.t. **capacity** and **scale**
 - Edges need not encode homophily
 - But still compute scalar value in each edge

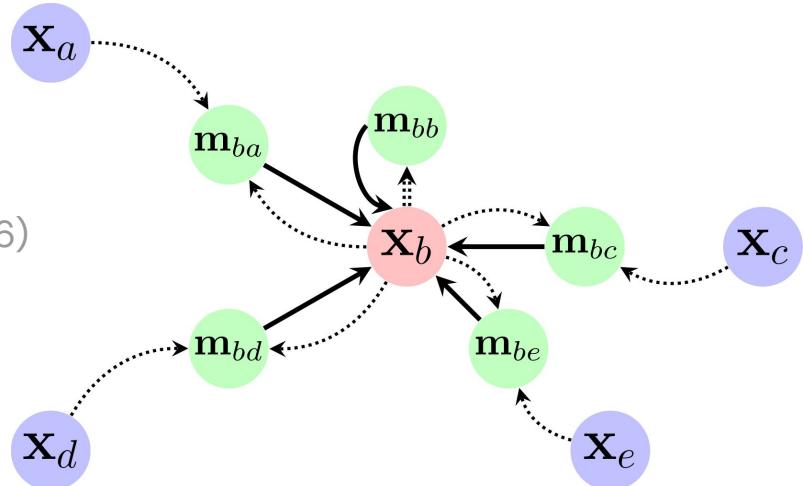


Message-passing GNN

- 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)
 - GraphNets (Battaglia et al., 2018)
- Most **generic** GNN layer
 - May have *scalability* or *learnability* issues
 - Ideal for *computational chemistry, reasoning and simulation*



Message-passing



DeepMind

7

Perspectives on GNNs



This framework looks quite clean, *but...*

- We didn't **start** researching GNNs from a blueprint like this.
- Graphs naturally arise **across** the sciences
 - Different disciplines found different tools to process them
- To give you a feel of the scale of diversity, I will now **survey** several prior and concurrent approaches to graph representation learning + to what extent they map to this blueprint.
- If you've read up on graph machine learning before, there's a good chance you will have seen at least some of these.



DeepMind

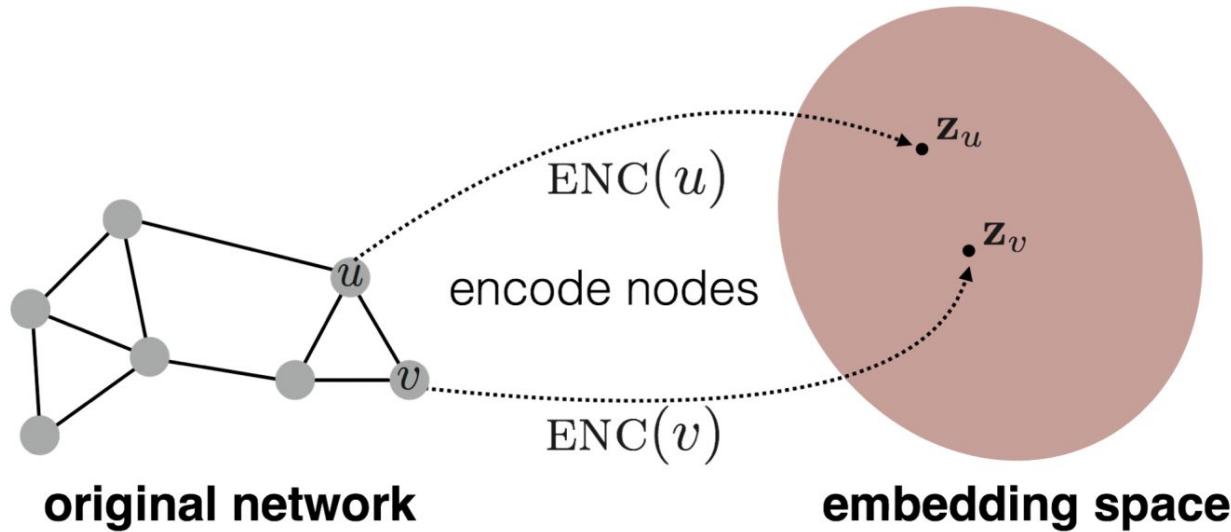
I

Node
embedding
techniques



Node embedding techniques

- Some of the earliest “successes” of deep learning on graphs relied on finding good ways to **embed** nodes into vectors \mathbf{h}_u (below: \mathbf{z}_u) using an *encoder function*
 - At the time, implemented as a **look-up table!**



What's in a *good* representation?

- What makes an embedding “good”?
 - Graphs carry interesting **structure**!
 - Good node representations should **preserve** it.
- Simplest notion of graph structure is an **edge**.
 - Features of nodes i and j should be predictive of existence of edge (i, j) !
- Yields a straightforward unsupervised objective
 - Optimise \mathbf{h}_i and \mathbf{h}_j to be **nearby** iff $(i, j) \in E$.
 - Can use standard *binary cross-entropy* loss:

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



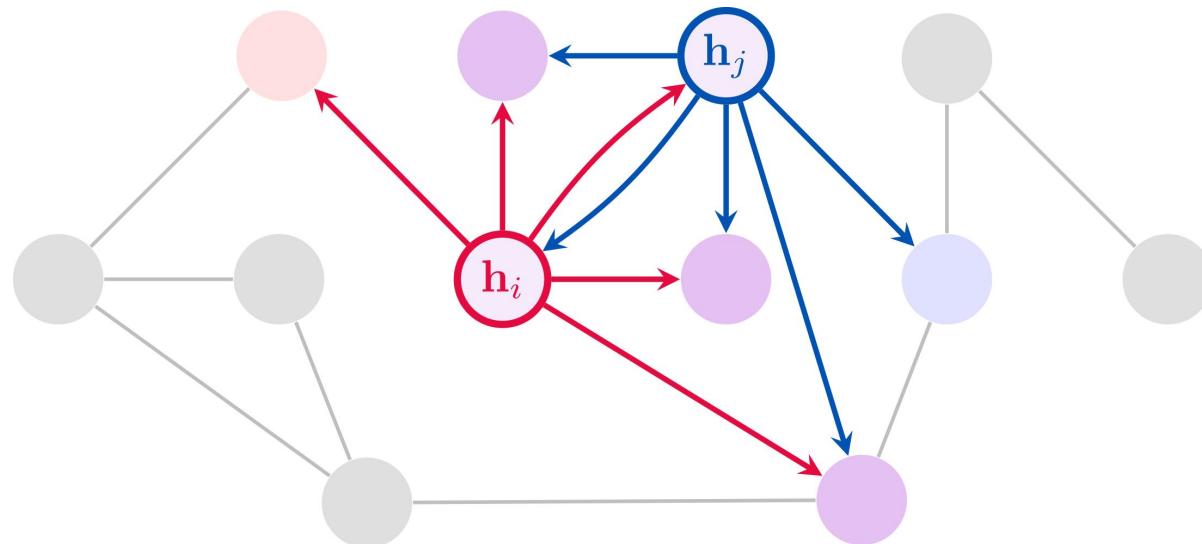
Random-walk objectives

- This link-prediction objective is a special case of **random-walk** objectives.
- **Redefine** the condition from $(i, j) \in E$ to i and j co-occur in a (short) random walk.
- Dominated unsupervised graph representation learning prior to GNNs!
 - DeepWalk (Perozzi *et al.*, KDD'14)
 - node2vec (Grover & Leskovec, KDD'16)
 - LINE (Tang *et al.*, WWW'15)



Local objectives emulate Conv-GNNs

- Random walk objectives inherently capture **local** similarities.
- But a (convolutional) GNN summarises *local patches* of the graph!
 - Neighbouring nodes tend to highly overlap in n -step neighbourhoods;
 - Therefore, a conv-GNN enforces similar features for neighbouring nodes **by design**.



Local objectives emulate Conv-GNNs

- Random walk objectives inherently capture **local** similarities.
- But a (convolutional) GNN summarises *local patches* of the graph!
 - Neighbouring nodes tend to highly overlap in n -step neighbourhoods;
 - Therefore, a conv-GNN enforces similar features for neighbouring nodes **by design**.
- From a representation perspective, DeepWalk-style models **emulate** a convolutional GNN!
- **Corollary 1:** Random-walk objectives can **fail** to provide useful signal to GNNs!
- **Corollary 2:** At times, DeepWalk can be matched by an **untrained conv-GNN**!
 - First spotted within DGI (Veličković *et al.*, ICLR'19)
 - Independently verified by SGC (Wu *et al.*, ICML'19)



Parallels to NLP

- Note clear **correspondence** between *node embedding* techniques and **word embedding** techniques in NLP
 - nodes ~ words
 - random walks ~ sentences
 - “node2vec” ~ “word2vec”
 - The optimisation objectives are *near-equal!*
- This correspondence continues even nowadays, with recent unsupervised graph representation learning techniques borrowing concepts from BERT.
 (“Strategies for pre-training graph neural networks”; Hu, Liu *et al.*, ICLR’20)
- Speaking of NLP...



DeepMind

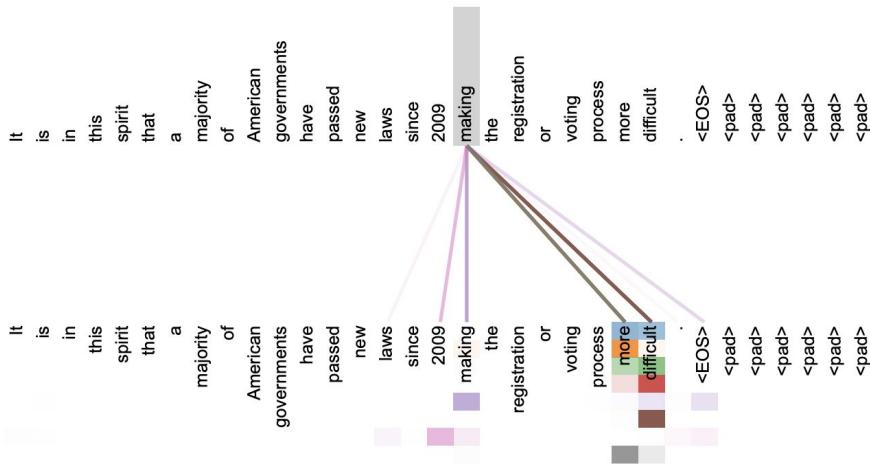
III

Natural Language Processing



Parallels from NLP

- It's not only that NLP feeds into GNN design...
- Words in a sentence **interact**
 - Nontrivially and **non-sequentially**
 - We may want to use a **graph** over them
 - But *what is this graph?*
- A common assumption is to assume a **complete graph**
 - Then let the network *infer* relations
- If you're at all involved with NLP, this should sound **familiar...**



A note on Transformers

Transformers **are** Graph Neural Networks!

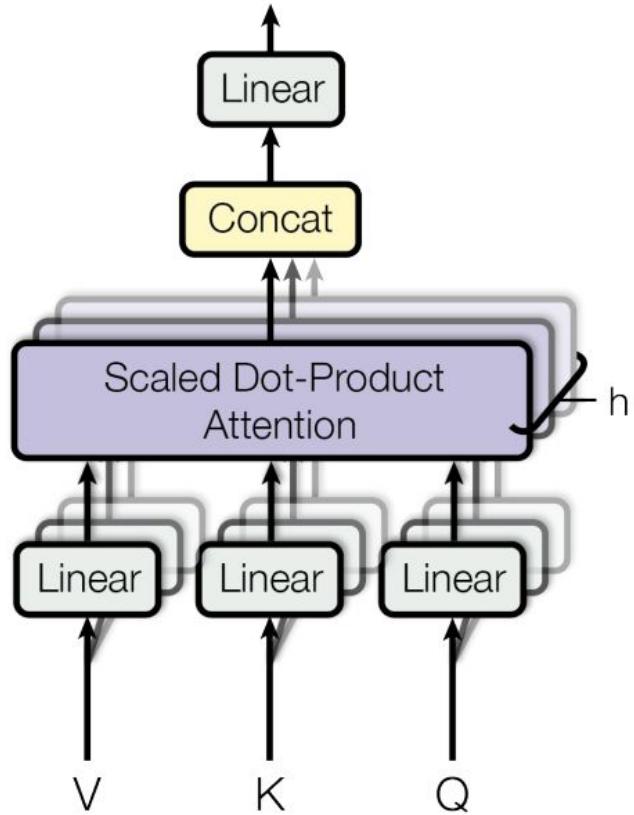
- Fully-connected graph
- Attentional flavour

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).

Multi-Head Attention



DeepMind

III

Spectral GNNs



Look to the Fourier transform

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

$$(x \star y)(\xi) = \hat{x}(\xi) \cdot \hat{y}(\xi) \quad \hat{x}(\xi) = \int_{-\infty}^{+\infty} x(u) e^{-i\xi u} du$$

"convolution in the time domain is multiplication in the frequency domain"

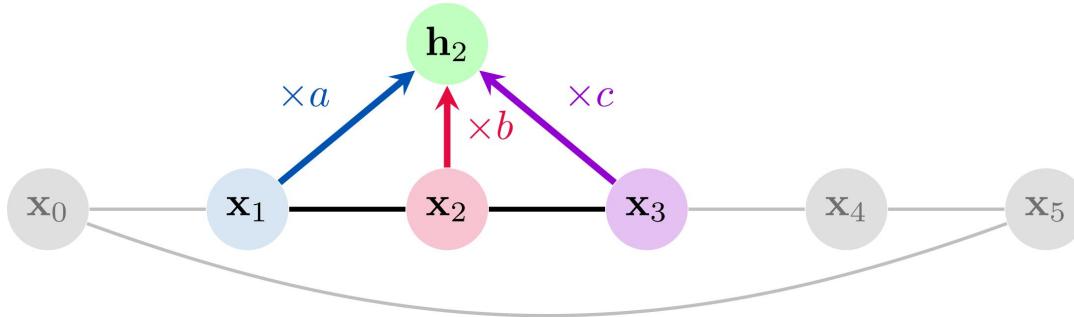
- This could give us a 'detour' to defining convolutions on graphs
 - Pointwise multiplication is **easy!**
 - But what are the 'domains' in this case?
- We will first see how graphs arise in **discrete** sequences.



Rethinking the convolution on *sequences*

*for easier handling of boundary conditions

- We can imagine a sequence as a *cyclical grid graph*, and a **convolution** over it:



- NB this defines a **circulant** matrix $\mathbf{C}([b, c, 0, 0, \dots, 0, a])$ s.t. $\mathbf{H} = f(\mathbf{X}) = \mathbf{C}\mathbf{X}$

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



Properties of circulants, and their eigenvectors

- Circulant matrices **commute!**
 - That is, $\mathbf{C}(\mathbf{v})\mathbf{C}(\mathbf{w}) = \mathbf{C}(\mathbf{w})\mathbf{C}(\mathbf{v})$, for any parameter vectors \mathbf{v}, \mathbf{w} .
- Matrices that commute are **jointly diagonalisable**.
 - That is, the eigenvectors of one are eigenvectors of *all* of them!
- Conveniently, the eigenvectors of circulants are the *discrete Fourier basis*

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

- This can be easily computed by studying $\mathbf{C}([0, 1, 0, 0, 0, \dots])$, which is the **shift** matrix.



The DFT and the convolution theorem

- If we stack these Fourier basis vectors into a matrix: $\Phi = (\phi_0, \dots, \phi_{n-1})$
 - We recover the *discrete Fourier transform* (DFT), as multiplication by Φ^* . (conjugate transpose)
- We can now eigendecompose *any* circulant as $\mathbf{C}(\theta) = \Phi \Lambda \Phi^*$
 - Where Λ is a diagonal matrix of its eigenvalues, $\hat{\theta}$
- The convolution theorem naturally follows:

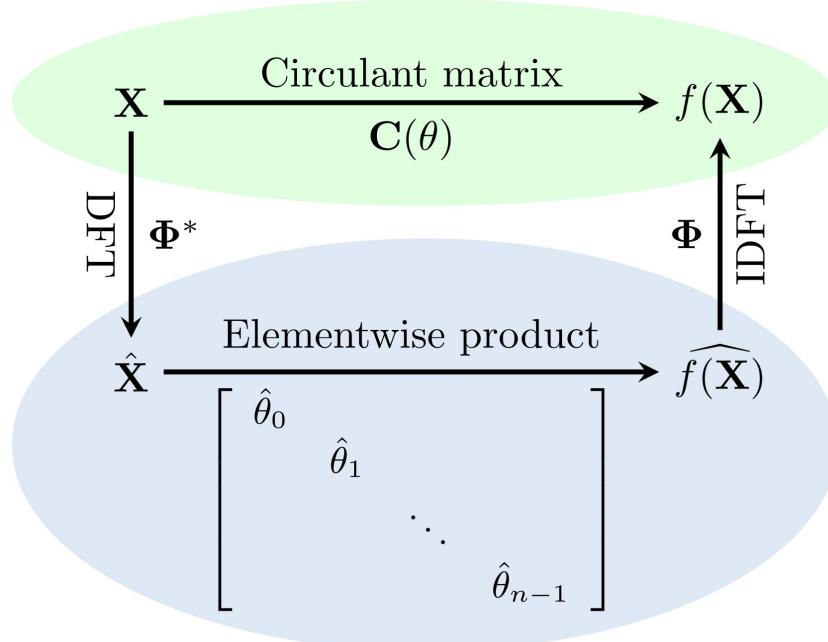
$$f(\mathbf{X}) = \mathbf{C}(\theta)\mathbf{X} = \Phi \Lambda \Phi^* \mathbf{X} = \Phi \begin{bmatrix} \hat{\theta}_0 & & \\ & \ddots & \\ & & \hat{\theta}_{n-1} \end{bmatrix} \Phi^* \mathbf{X} = \Phi(\hat{\theta} \circ \hat{\mathbf{X}})$$

- Now, as long as we know Φ , we can express our convolution using $\hat{\theta}$ rather than θ



What we have covered so far

Spatial



Spectral



Key idea: we don't **need** to know the circulant if we know its eigenvalues!

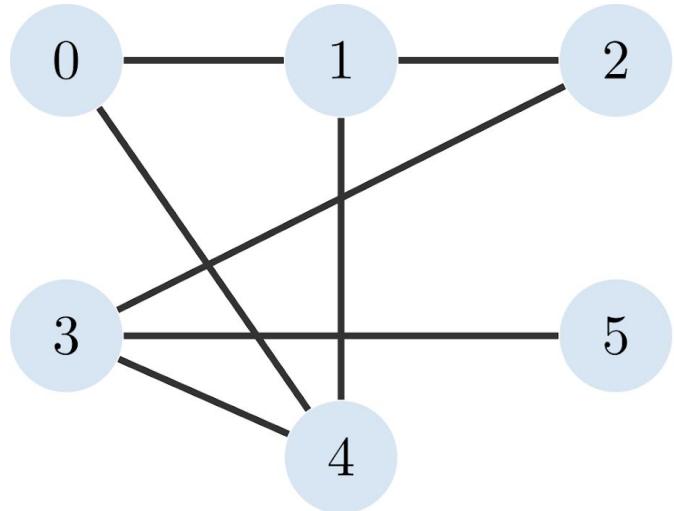
Credits to Michael Bronstein

What about graphs?

- On graphs, convolutions of interest need to be more generic than circulants.
 - But we can still use the concept of **joint eigenbases**!
 - If we know a “graph Fourier basis”, Φ , we can only focus on learning the eigenvalues.
- For grids, we wanted our convolutions to commute with *shifts*.
 - We can think of the shift matrix as an **adjacency matrix** of the grid
 - This generalises to non-grids!
 - For the grid convolution on n nodes, Φ was always the same (n -way DFT).
 - Now **every graph** will have its own Φ !
- Want our convolution to commute with A , but we cannot always eigendecompose A !
 - Instead, use the **graph Laplacian matrix**, $L = D - A$, where D is the degree matrix.
 - Captures all adjacency properties in mathematically convenient way!



Example Laplacian



$$\mathbf{L} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$



Graph Fourier Transform

- Assuming undirected graphs, L is:
 - **Symmetric** ($L^T = L$)
 - **Positive semi-definite** ($x^T L x \geq 0$ for all $x \in \mathbb{R}^{|V|}$)
 - This means we will be able to *eigendecompose* it!
- This allows us to re-express $L = \Phi \Lambda \Phi^*$, as before.
 - Changing the eigenvalues in Λ expresses **any** operation that commutes with L .
 - Commonly referred to as the **graph Fourier transform** (Bruna et al., ICLR'14)
- Now, to convolve with some feature matrix X , do as follows (the diagonal can be **learnable**):

$$f(X) = \Phi \begin{bmatrix} \hat{\theta}_0 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \hat{\theta}_{n-1} \end{bmatrix} \Phi^* X$$



Spectral GNNs in practice

- However, directly learning the eigenvalues is typically inappropriate:
 - Not **localised**, doesn't **generalise** to other graphs, computationally **expensive**, etc.
- Instead, a common solution is to make the eigenvalues related to Λ , the eigenvalues of L
 - Commonly by a degree- k polynomial function, p_k
 - Yielding $f(x) = \Phi p_k(\Lambda) \Phi^* x = p_k(L)x$
 - Popular choices include:
 - *Cubic splines* (Bruna et al., ICLR'14)
 - *Chebyshev polynomials* (Defferrard et al., NeurIPS'16)
 - *Cayley polynomials* (Levie et al., Trans. Sig. Proc.'18)
- **NB** by using a polynomial in L , we have defined a **conv-GNN!**
 - With coefficients defined by $c_{ij} = (p_k(L))_{ij}$
 - Most efficient spectral approaches “*spatialise*” themselves in similar ways
 - The “spatial-spectral” divide is often **not really a divide!**



The Transformer positional encodings and beyond

- Lastly, another look at Transformers.
- Transformers signal that the input is a **sequence** of words by using *positional embeddings*
 - Sines/cosines sampled depending on position
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$
- **Very** similar to the DFT eigenvectors!
- Positional embeddings could hence be interpreted as eigenvectors of the grid graph
 - Which is the only assumed ‘underlying’ connectivity between the words
- We can use this idea to run Transformers over *general* graph structures!
 - Just feed some eigenvectors of the graph Laplacian (columns of Φ)
 - See the **Graph Transformer** from Dwivedi & Bresson (2021)



DeepMind

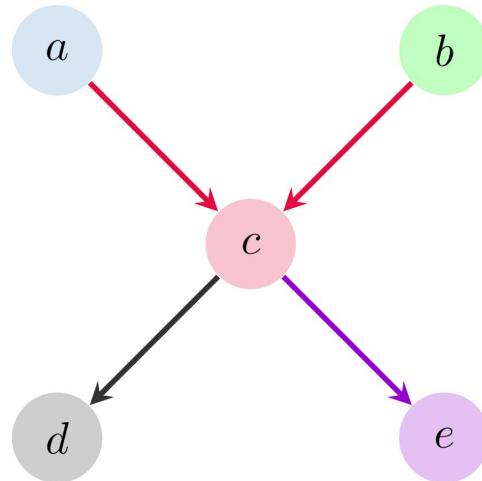
IV

Probabilistic Graphical Models



Probabilistic modelling

- We've so far used edges in a graph to mean any kind of relation between nodes
- Taking a more **probabilistic** view, we can treat nodes as *random variables*, and interpret edges as *dependencies* between their distributions.
 - This gives rise to **probabilistic graphical models** (PGMs)
 - They help us *ignore* relations between variables when computing **joint** probabilities



$$p(a, b, c, d, e) = p(a)p(b)p(c|a, b)p(d|c)p(e|c)$$



Markov random fields

- One particular PGM of interest here is the *Markov random field (MRF)*.
 - It allows us to decompose the joint into a product of *edge potentials*
- Specifically, we assume nodes are represented by inputs \mathbf{X} and latents \mathbf{H}
 - Inputs and latents are related for every node in isolation
 - Latents are related according to the edges of the graph
- This yields the following decomposition of the joint

$$p(\mathbf{X}, \mathbf{H}) \propto \prod_{u \in \mathcal{V}} \Phi(\mathbf{x}_u, \mathbf{h}_u) \prod_{(u,v) \in \mathcal{E}} \Psi(\mathbf{h}_u, \mathbf{h}_v)$$

where Φ and Ψ are real-valued *potential functions*.



Mean-field inference

- To embed nodes, we need to sample from the posterior, $p(\mathbf{H} | \mathbf{X})$.
 - Generally *intractable*, even if we know the exact potential functions.
- One popular method of resolving this is **mean-field variational inference**
 - Assume that posterior can be approximated by a product of node-level densities

$$p(\mathbf{H} | \mathbf{X}) \approx \prod_{u \in \mathcal{V}} q(\mathbf{h}_u)$$

where q is a well-defined density, that is easy to compute and sample (e.g. Gaussian).

- We then obtain the parameters of q by minimising the distance (e.g. KL-divergence) to the true posterior, $\text{KL}(\prod_u q(\mathbf{h}_u) || p(\mathbf{H} | \mathbf{X}))$
- Minimising the KL is intractable, but it admits a favourable approximate algorithm



GNNs strike again!

- Using variational inference techniques (out of scope), we can *iteratively* update q , starting from some initial guess $q^{(0)}(\mathbf{h})$, as follows:

$$\log q^{(t+1)}(\mathbf{h}_i) = c_i + \log \Phi(\mathbf{x}_i, \mathbf{h}_i) + \sum_{j \in \mathcal{N}_i} \int_{\mathbb{R}^k} q^{(t)}(\mathbf{h}_j) \log \Psi(\mathbf{h}_i, \mathbf{h}_j) \, d\mathbf{h}_i$$

- See anything familiar? :)



GNNs strike again!

- Using variational inference techniques (out of scope), we can *iteratively* update q , starting from some initial guess $q^{(0)}(\mathbf{h})$, as follows:

$$\log q^{(t+1)}(\mathbf{h}_i) = c_i + \log \Phi(\mathbf{x}_i, \mathbf{h}_i) + \sum_{j \in \mathcal{N}_i} \int_{\mathbb{R}^k} q^{(t)}(\mathbf{h}_j) \log \Psi(\mathbf{h}_i, \mathbf{h}_j) d\mathbf{h}_i$$

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

- This aligns **very nicely** with computations of a (message-passing) GNN!



GNNs and PGMs, more broadly

- Based on this idea, **structure2vec** (Dai *et al.*, ICML'16) embed mean-field inference within computations of a GNN.
 - Key difference: in PGMs, we expect potential functions specified and known upfront
 - Here, they are defined implicitly, within the latents of a GNN.
- The structure2vec GNN itself is not unlike a typical MPNN.
- Recently, there are other approaches that unify GNNs with PGM-like computations:
 - CRF-GNNs (Gao *et al.*, KDD'19)
 - GMNNs (Qu *et al.*, ICML'19)
 - ExpressGNN (Zhang *et al.*, ICLR'20)
 - Tail-GNNs (Spalević *et al.*, ICML'20 GRL+)



DeepMind

V

Graph Isomorphism Testing



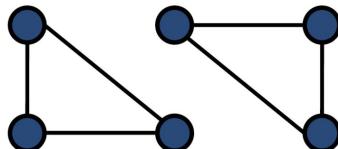
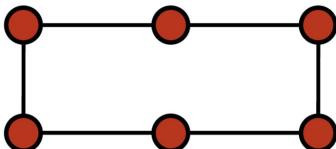
How *powerful* are Graph Neural Networks?

- GNNs are a powerful tool for processing real-world graph data
 - But they won't solve **any** task specified on a graph accurately!
- Canonical example: deciding *graph isomorphism*
 - Am I able to use my GNN to **distinguish** two *non-isomorphic* graphs? ($\mathbf{h}_{G1} \neq \mathbf{h}_{G2}$)
 - If I can't, any kind of task discriminating them is *hopeless*
- Permutation invariance mandates that two *isomorphic* graphs will always be indistinguishable, so this case is OK.



Weisfeiler-Leman Test

- Simple but powerful way of distinguishing: pass **random hashes** of **sums** along the edges
 - Connection to conv-GNNs spotted very early; e.g. by GCN (Kipf & Welling, ICLR'17)
- It explains why untrained GNNs work well!
 - Untrained \sim random hash
- The test does **fail** at times, however:



Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, \dots, h_N^{(0)})$

Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, \dots, h_N^{(T)})$

$t \leftarrow 0;$

repeat

for $v_i \in \mathcal{V}$ **do**

$h_i^{(t+1)} \leftarrow \text{hash} \left(\sum_{j \in \mathcal{N}_i} h_j^{(t)} \right);$

$t \leftarrow t + 1;$

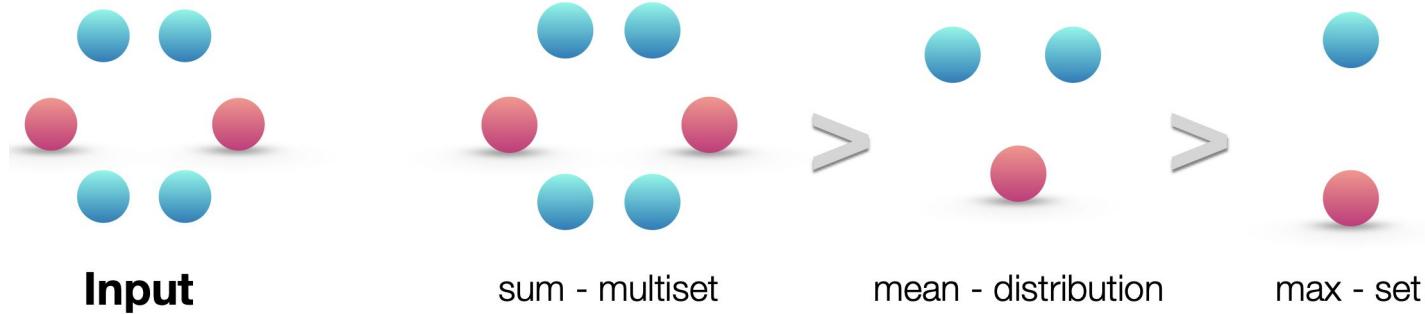
until stable node coloring is reached;

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



GNNs are no more powerful than 1-WL

- Over *discrete* features, GNNs can only be **as powerful** as the 1-WL test described before!
- One important condition for maximal power is an *injective* aggregator (e.g. **sum**)

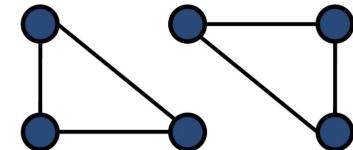
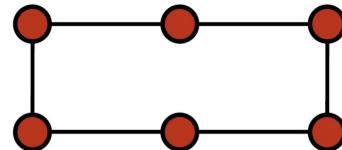


- Graph isomorphism network (**GIN**; Xu et al., ICLR'19) proposes a simple, maximally-expressive GNN, following this principle

$$h_v^{(k)} = \text{MLP}^{(k)} \left(\left(1 + \epsilon^{(k)} \right) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \right)$$



Higher-order GNNs

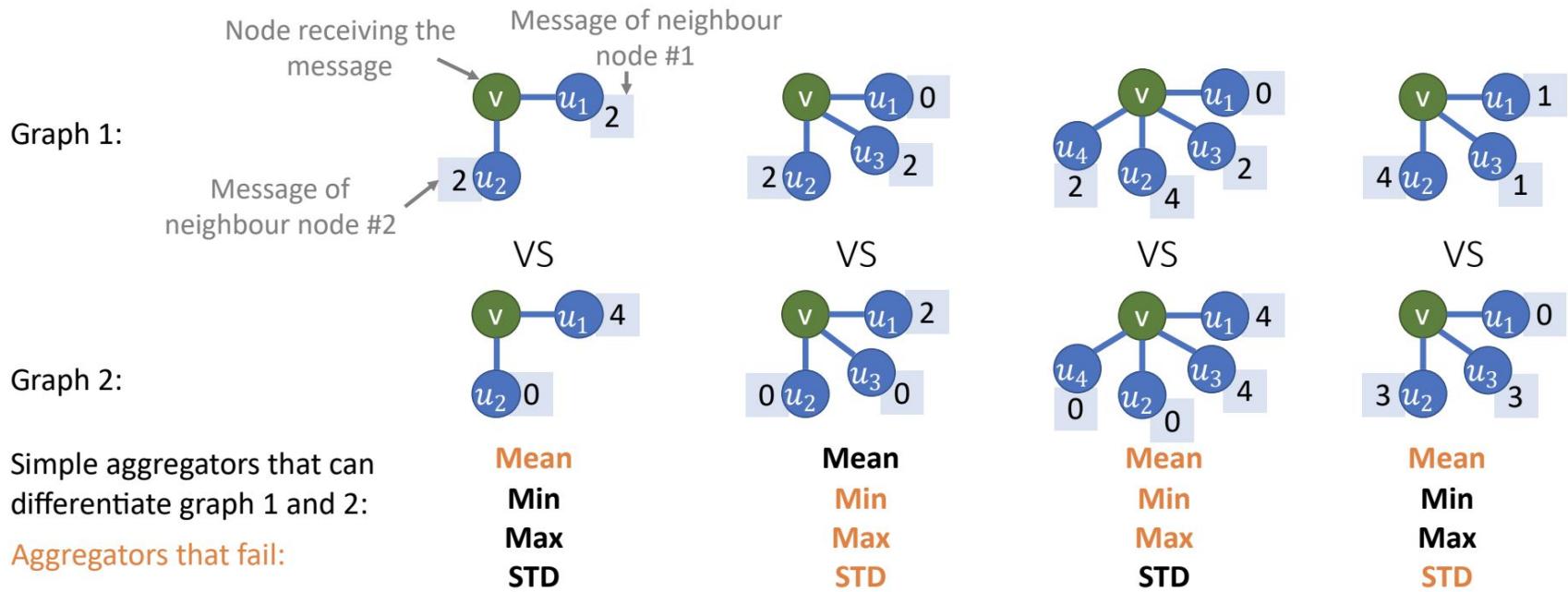


- We can make GNNs stronger by analysing **failure cases** of 1-WL!
- For example, just like 1-WL, GNNs cannot detect **closed triangles**
 - Augment nodes with randomised/positional features
 - Explored by RP-GNN (Murphy et al., ICML'19) and P-GNN (You et al., ICML'19)
 - See also: Sato et al. (SDM'21)
 - Can also literally **count** interesting subgraphs (Bouritsas et al., 2020)
- k -WL labels *subgraphs* of k nodes together.
 - Exploited by 1-2-3-GNNs (Morris et al., AAAI'19)
- Further avenues of interest:
 - Invariant and equivariant GNNs (Maron et al. (ICLR'19))
 - Directional graph networks (DGNs) (Beaini, Passaro et al. (2020))



Going beyond discrete features

- What happens when features are **continuous**? (real-world apps / latent GNN states)
 - ... the proof for injectivity of sum (hence GINs' expressivity) **falls apart**



Which is best? Neither.

- There doesn't seem to be a clear single "winner" aggregator here...
- In fact, we prove in the PNA paper (Corso, Cavalleri et al., NeurIPS'20) that **there isn't one!**

Theorem 1 (Number of aggregators needed). *In order to discriminate between multisets of size n whose underlying set is \mathbb{R} , at least n aggregators are needed.*

- The proof is (in my opinion) **really cool!** (relies on **Borsuk–Ulam theorem**)
- PNA proposes empirically powerful **combination** of aggregators for general-purpose GNNs:

$$\bigoplus = \underbrace{\begin{bmatrix} I \\ S(D, \alpha = 1) \\ S(D, \alpha = -1) \end{bmatrix}}_{\text{scalers}} \otimes \underbrace{\begin{bmatrix} \mu \\ \sigma \\ \max \\ \min \end{bmatrix}}_{\text{aggregators}}$$



DeepMind

VI

Geometric Deep Learning

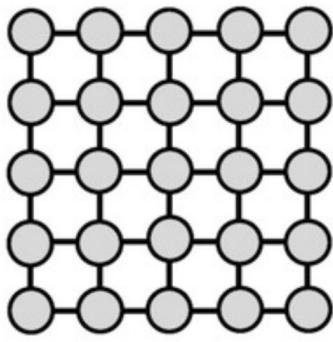


Remark on geometric deep learning

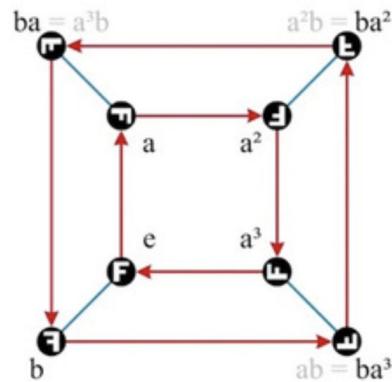
- We used the blueprint of *invariances* and *equivariances* to describe GNNs
 - In fact, it is remarkably powerful! By combining an appropriate
 - **Local** and **equivariant** layer specified over *neighbourhoods*
 - Activation functions
 - (Potentially: **pooling** layers that coarsen the structure)
 - **Global** and **invariant** layer over the entire domain
- we recover many standard architectures (including CNNs and Transformers!)
- But also a more general class of **geometric** deep learning architectures



The “Four Gs” of geometric deep learning



Grids



Groups



Graphs



“Gauges”



Some architectures of interest

Geometric Deep Learning Blueprint instances of interest			
Domain, Ω	Metric, g	Symmetry group, \mathcal{G}	Architecture
Grids	L_∞	Translations	CNNs
Spheres	Great circle	3D rotations, $SO(3)$	Spherical CNNs
SO(3)	Geodesic		
Gauges	Geodesic	{Gauge transform.}	Gauge Equivariant Mesh CNNs
Graphs	Shortest path	Permutations, Σ_n	GNNs
Sets	$+\infty$	Permutations, Σ_n	Deep Sets
	0	Permutations, Σ_n	Transformers



DeepMind

VII

Historical
concepts



Where did GNNs come from?

- Early forms can be traced to the **early 1990s**, often involving DAG structures.
 - Labeling RAAM (Sperduti, NeurIPS'94)
 - 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
 - Updated for modern best practices by **gated GNNs** (Li *et al.*, ICLR'16)



DeepMind

VIII

Computational
Chemistry



“Chemistry disrupts ML, not the other way around”

- Important and **concurrent** GNN development line came from **computational chemistry**
 - Very relevant to the area, as molecules are naturally modelled as graphs
- GNN-like models of *molecular property prediction* arise, also, in the 1990s
 - Examples include **ChemNet** (Kireev, CICS'95) and (Baskin *et al.*, CICS'97)
- “**Molecular Graph Networks**” (Merkwirth and Lengauer, CIM'05) already propose many elements commonly found in modern MPNNs
- This drive continued well into the 2010s:
 - GNNs for molecular fingerprinting (Duvenaud *et al.*, NeurIPS'15)
 - GNNs for quantum chemistry (Gilmer *et al.*, ICML'17)
- Lastly, recall (**Stokes *et al.*, Cell'20**): chemistry is to-this-day a **leading** outlet for GNNs!



DeepMind

Thank you!

Questions?

petarv@google.com | <https://petar-v.com>

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

