GEOMETRIC DEEP LEARNING (**L65**)

Pietro Liò *University of Cambridge*
Petar Veličković *Google DeepMind / University of Cambridge*

Lent Term 2025

CST Part III / MPhil ACS / MPhil MLMI

4. GRAPH NEURAL NETWORKS

Permutation-equivariant learning on graphs

Petar Veličković

Learning on graphs

In the last lecture, we studied how to build neural nets over *sets*
Now we augment the set of nodes with **edges** between them
That is, we consider *graphs* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$

We can represent these edges in an *adjacency matrix*, $\mathbf{A}$, such that:

$$a_{uv} = \begin{cases} 1, & (u, v) \in \mathcal{E} \\ 0, & (u, v) \notin \mathcal{E} \end{cases}$$

Note that the edges are now *part of the domain*!

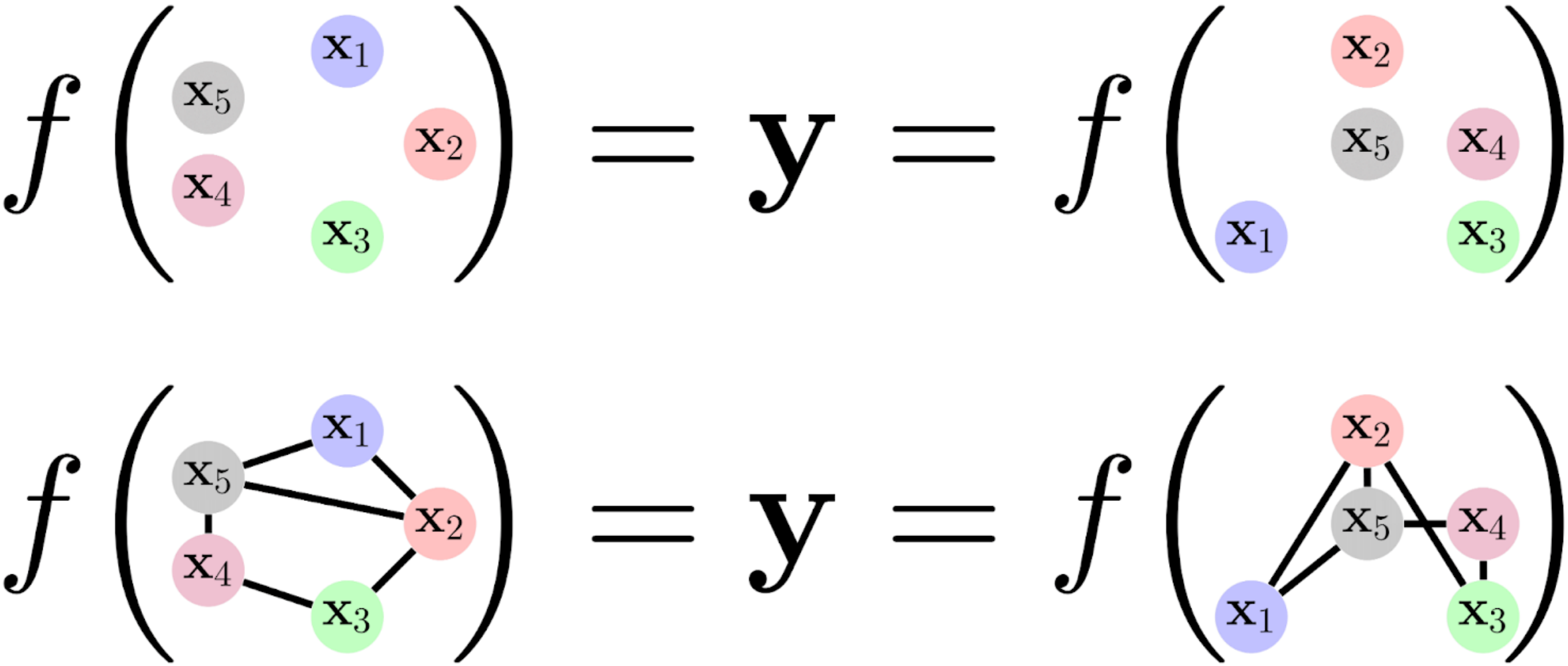
Further additions, e.g. edge features, are possible but **ignored** for now

Our main desiderata (*permutation {in,equi}variance*) still hold!

What's changed?

$$f\left(\mathbf{x}_5 \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_4 \quad \mathbf{x}_3\right) = \mathbf{y} = f\left(\mathbf{x}_2 \quad \mathbf{x}_5 \quad \mathbf{x}_4 \quad \mathbf{x}_1 \quad \mathbf{x}_3\right)$$

What's changed?



Permutation invariance and equivariance on graphs

Main difference: 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 $\mathbf{PAP}^\top$

We arrive at updated definitions of suitable functions over graphs:

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

Equivariance: $$\mathbf{F}(\mathbf{PX}, \mathbf{PAP}^\top) = \mathbf{PF}(\mathbf{X}, \mathbf{A})$$

Locality on graphs: Neighbourhoods

On *sets,* we enforced *locality* by transforming every node **in isolation**

Graphs give us a broader context: a node's *neighbourhood*
For a node u, its (1-hop) neighbourhood, $\mathcal{N}_u$, is commonly defined as:

$$\mathcal{N}_u = \{v : (u, v) \in \mathcal{E} \ \vee (v, u) \in \mathcal{E})\}$$

Accordingly, we can extract neighbourhood features, $\mathbf{X}_{\mathcal{N}_u}$, like so:

$$\mathbf{X}_{\mathcal{N}_u} = \{\{\, \mathbf{x}_v : v \in \mathcal{N}_u \,\}\}$$

and define a *local* function, $f(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$, operating over them.

($\mathbf{X}_{\mathcal{N}_u}$ is a *multiset*; cf. {{ ... }} notation)

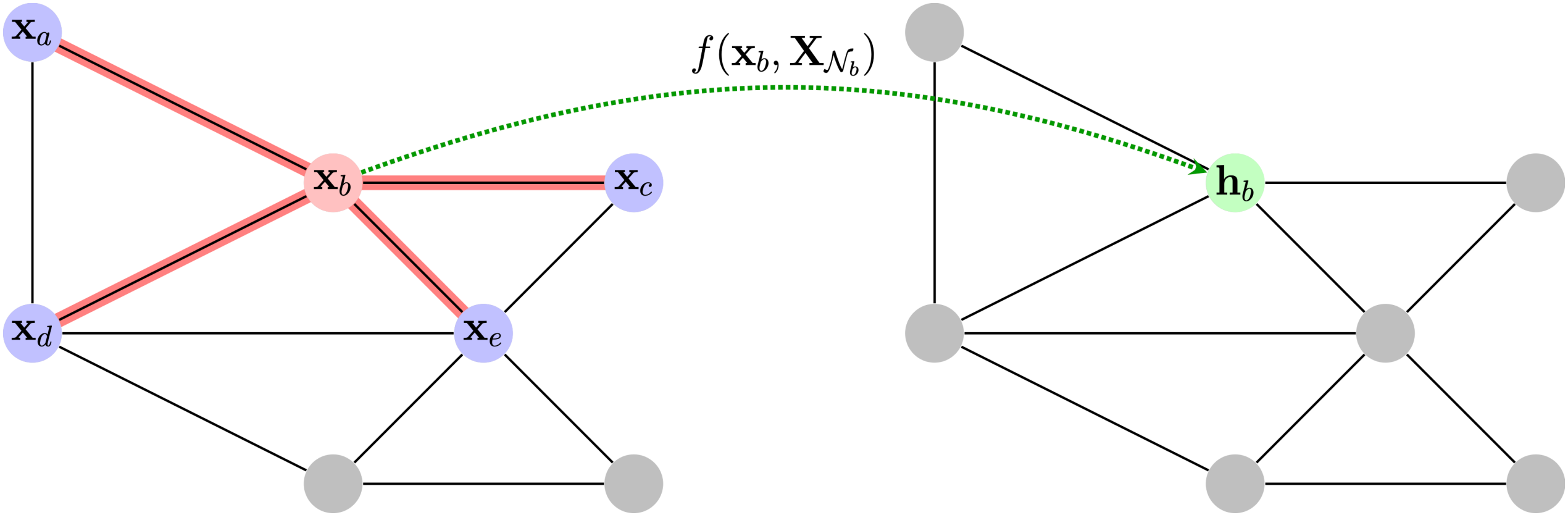
Recipe for graph neural networks

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

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

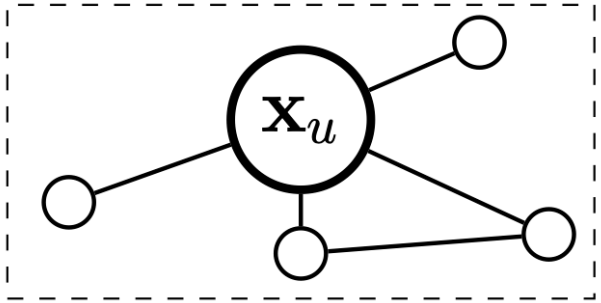
To ensure equivariance, it is sufficient if f does not depend on the **order** of the nodes in $\mathbf{X}_{\mathcal{N}_u}$ (i.e. if it is *permutation invariant* in $\mathbf{X}_{\mathcal{N}_u}$).
Exercise: Prove this!

Recipe for graph neural networks, visualised



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

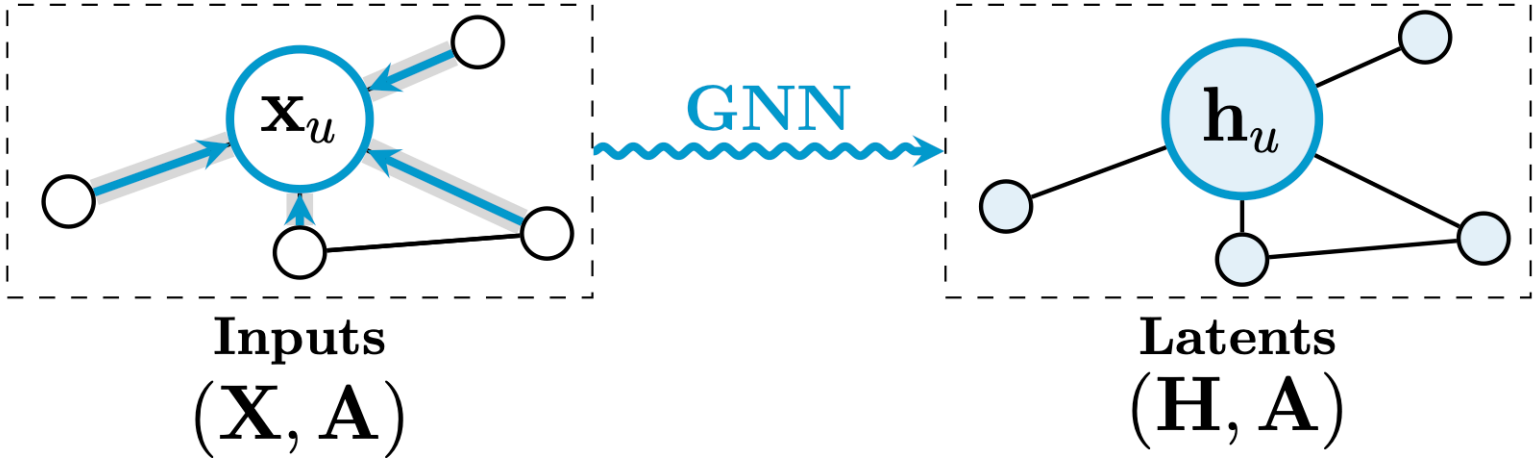
General blueprint for learning on graphs



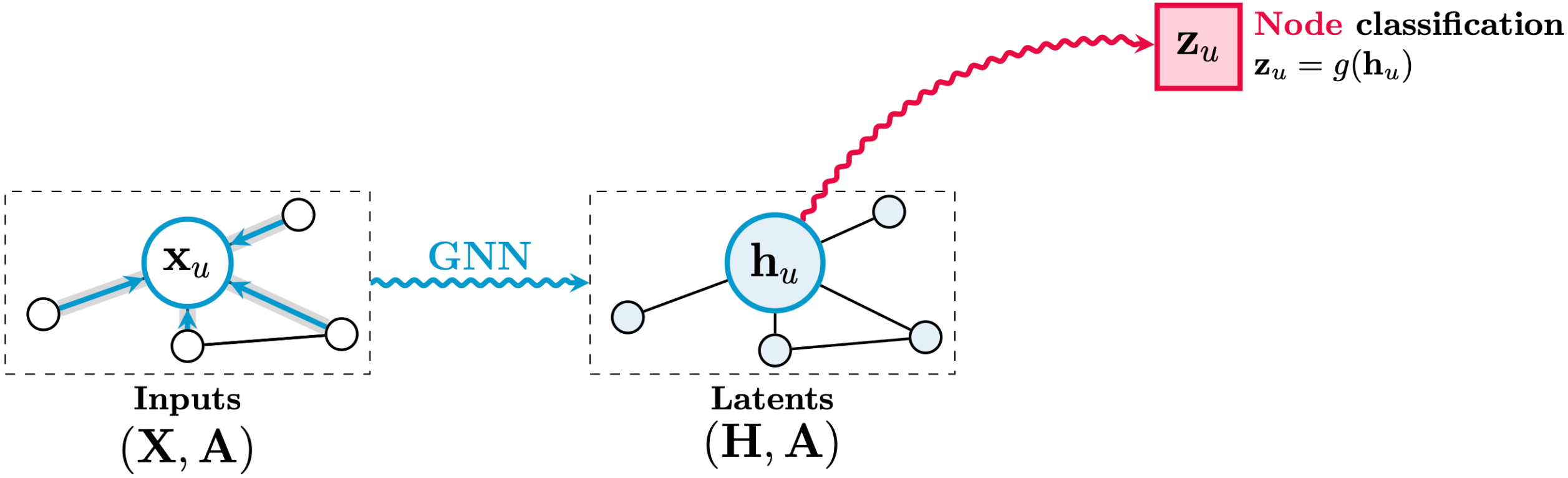
Inputs

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

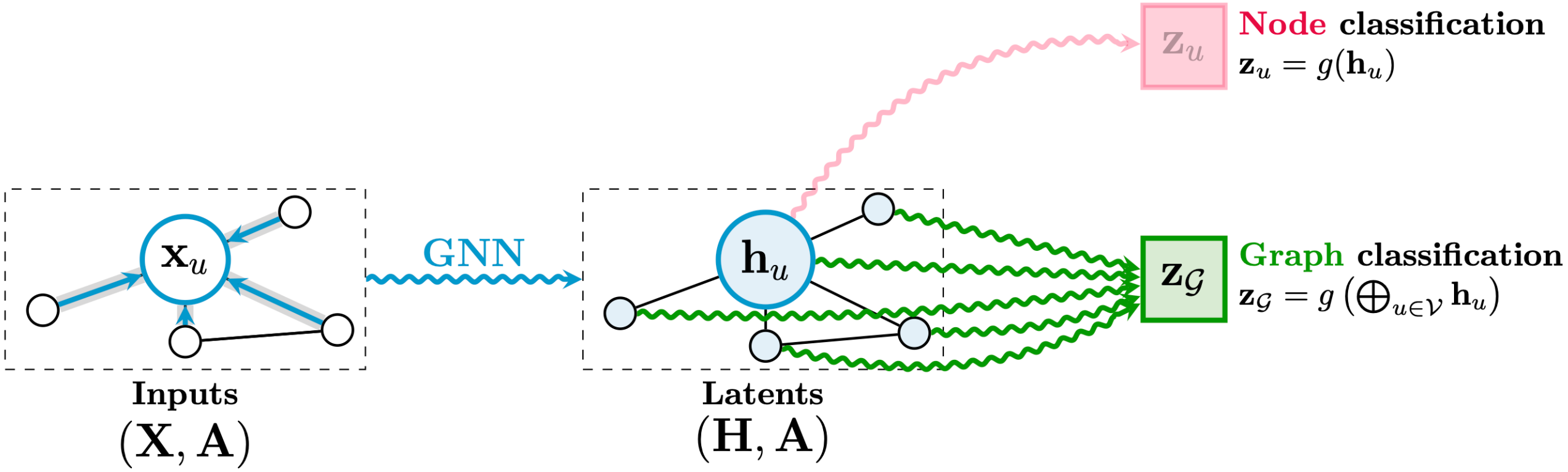
General blueprint for learning on graphs



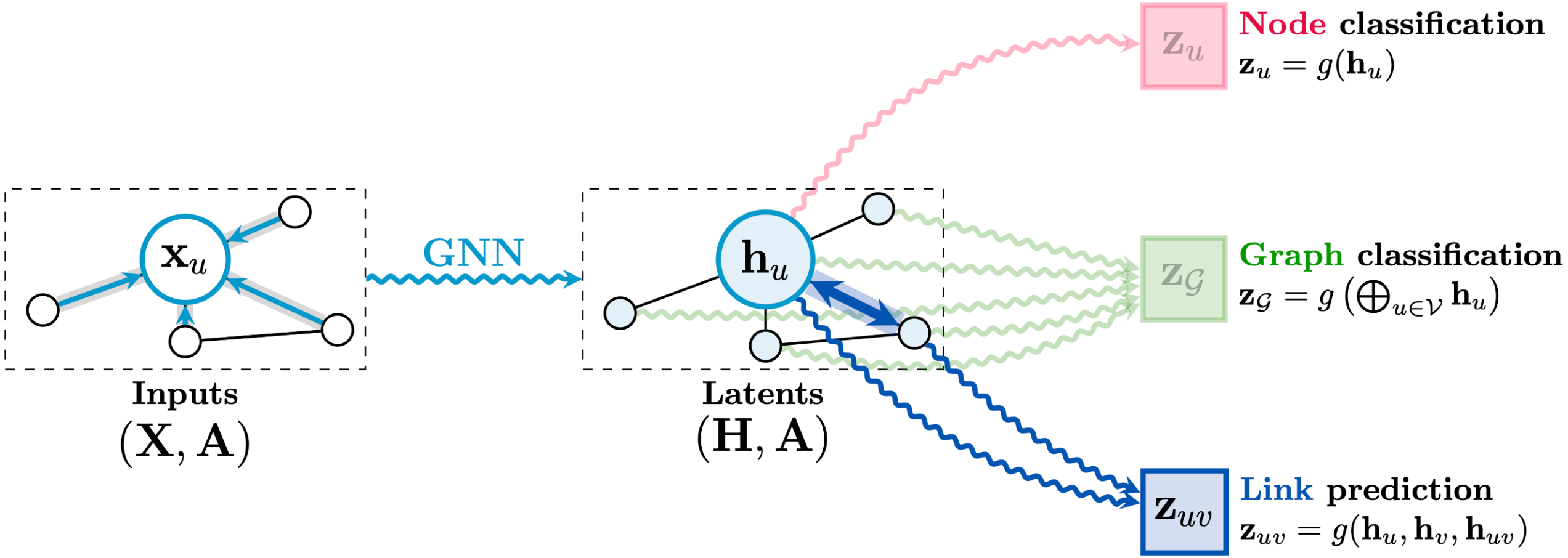
General blueprint for learning on graphs



General blueprint for learning on graphs



General blueprint for learning on graphs



What's in a GNN layer?

We build permutation equivariant functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$ on graphs by shared application of a *local* permutation-invariant $f(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$

Common lingo:
F is a *"GNN layer"*
f is *"diffusion"* | *"propagation"* | *"message passing"*

But **how** do we implement f?
Very intense area of research!

What's in a GNN layer?

We build permutation equivariant functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$ on graphs by shared application of a *local* permutation-invariant $f(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$

Common lingo:
F is a *"GNN layer"*
f is *"diffusion"* / *"propagation"* / *"message passing"*

But **how** do we implement f?
Very intense area of research!
Fortunately, *almost all* of them can be classified across three *"flavours"*

Preliminaries

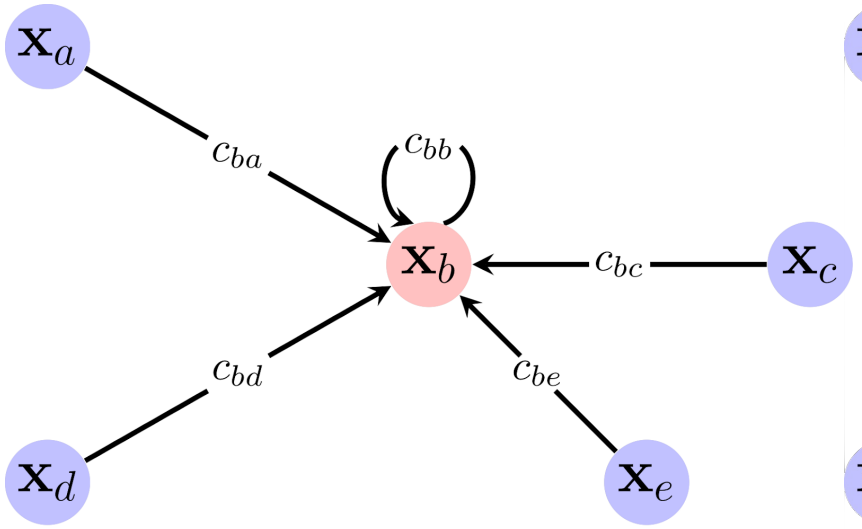
As f is supposed to be a local and permutation-invariant function over the neighbourhood features $\mathbf{X}_{\mathcal{N}_u}$, it effectively needs to be a neural network over **sets**, potentially conditioned by $\mathbf{x}_u$.

Recalling the Deep Sets model and its universality, we can hence assume the following generic equation (with added conditioning):

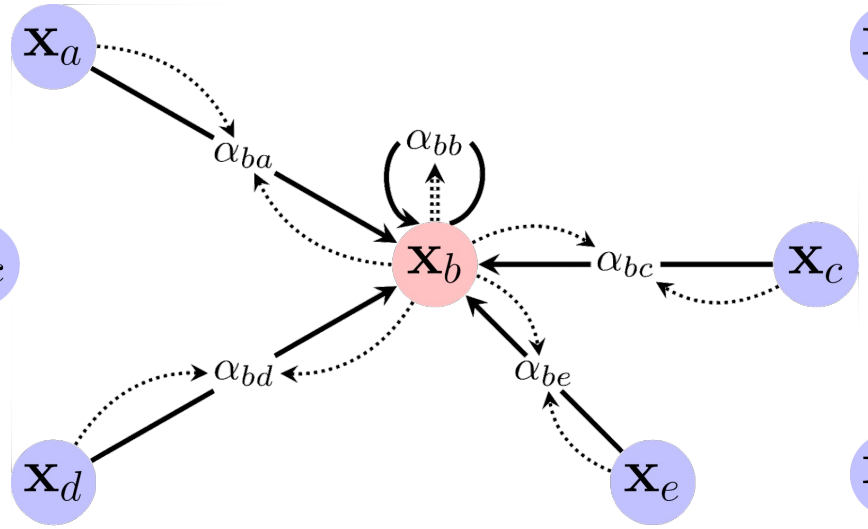
$$f\left(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u}\right) = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right)$$

Note that this induces several free variables $(\mathcal{N}_u, \oplus, \phi, \psi)$
We will primarily focus on the *parametric* ones in today's lecture!
(**NB.** We (for now) assume our GNN does not modify the graph structure!)

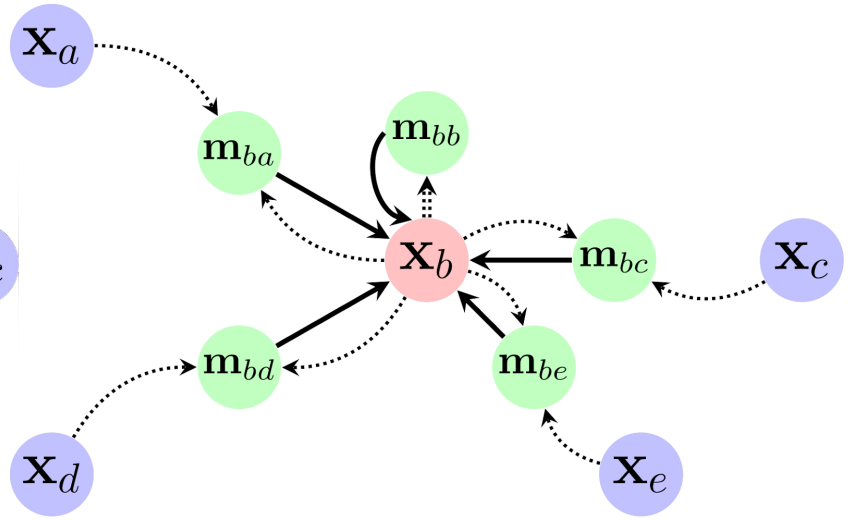
The three "flavours" of GNN layers



Convolutional



Attentional



Message-passing

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\psi(\mathbf{x}_v)\right)$$

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} a(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)\right)$$

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right)$$

Important disclaimer

For much of this lecture, we will be explicitly diving into the *specific instantiations* of the three flavours and *implementing* them in practice

This is designed with *orientation* in mind, while also telling a few *chronological* stories of GNN development + some of the *lessons* learnt

It is **by no means a complete account**!

Rather, it is only meant to give you context to *navigate* and *categorise* the *overwhelming* emerging research developments in GNNs

A note on notation

You will often see the appearance of functions ψ and ϕ in this lecture

They are meant to be neural networks operating over *flat* vector inputs

The simplest example is a fully-connected MLP layer, e.g.:

$$\psi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \qquad \psi(\mathbf{x}, \mathbf{y}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{U}\mathbf{y} + \mathbf{b})$$

$\mathbf{W}$, $\mathbf{U}$, and $\mathbf{b}$ are weights and biases, and σ is an activation function

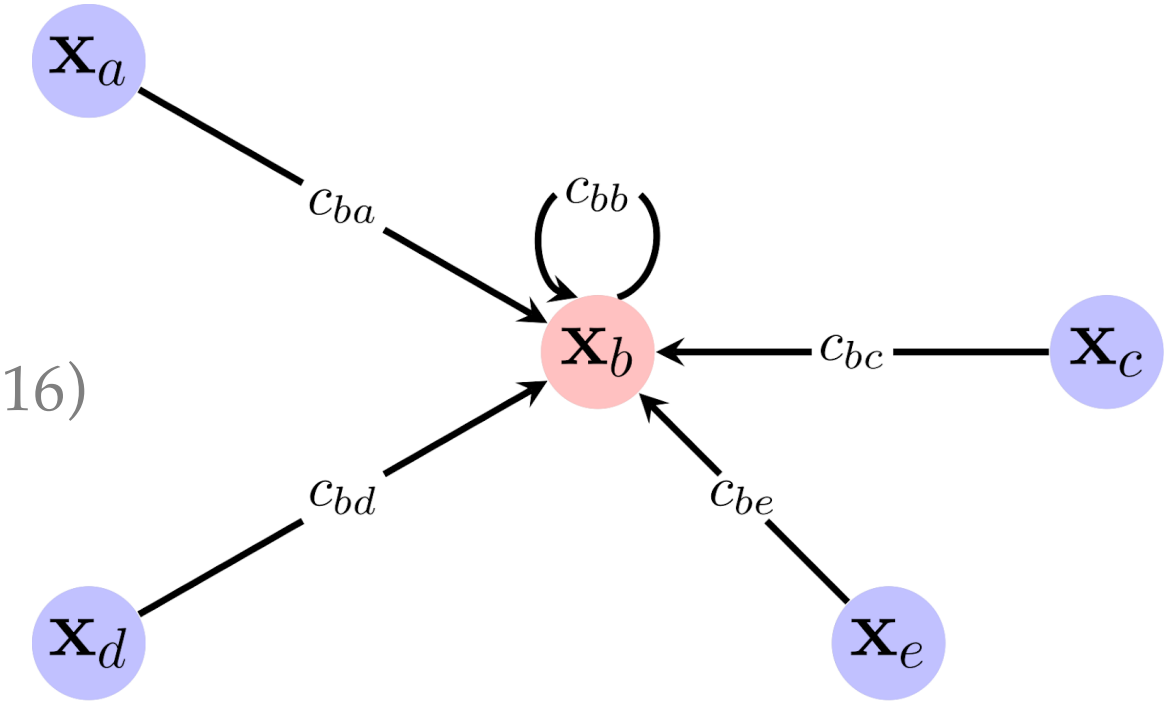
(stacking more layers, e.g. $\psi(\mathbf{x}, \mathbf{y}) = \sigma_2(\mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{U}\mathbf{y} + \mathbf{b}_1) + \mathbf{b}_2)$ is possible, and occasionally *necessary*)

The parameters are usually trainable via *stochastic gradient descent*

Convolutional GNN

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

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\psi(\mathbf{x}_v)\right)$$



Convolutional

Usually, 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
(when edges encode *label similarity*)
Highly **scalable**
Most *industrial* GNN applications currently live here

Setting the convolutional weights

What could be a good coefficient c_{uv}?

If we don't know *anything* about the task, perhaps a *constant*?
Let $c_{uv} = 1$, and set $\bigoplus = \sum$.

Conveniently, we can now represent the update rule in *matrix form*!

$$\mathbf{H} = \mathbf{\Phi}\big(\mathbf{X}, \mathbf{A}\mathbf{\Psi}(\mathbf{X})\big)$$

(Here, $\mathbf{\Phi}$ and $\mathbf{\Psi}$ distribute the computation of ϕ and ψ across all nodes)

To make the matrix analogy even clearer: let ψ be a linear layer, and ϕ a (residual) sum followed by an activation function, σ:

$$\mathbf{H} = \sigma(\mathbf{A}\mathbf{X}\mathbf{W}_1 + \mathbf{X}\mathbf{W}_0)$$

Stabilising the operator

We need to resolve a key issue: *explosion* of the features
For most graphs, $\|\mathbf{AX}\| > \|\mathbf{X}\|$

Instead of taking sums, let's take the average! i.e. $c_{uv} = \frac{1}{d_u}$

(Other options, such as *layer normalisation* on the output node features, have seen popularity recently)

The matrix form still works:

$$\mathbf{H} = \sigma(\mathbf{D}^{-1}\mathbf{AXW}_1 + \mathbf{XW}_0)$$

where $\mathbf{D}$ is the *degree matrix*; $d_{uu} = d_u$, and zero otherwise.

Note: this operation is related to the *random-walk Laplacian*!

Symmetric normalisation

Generally, more interesting dynamics emerge with *symmetric normalisation*; that is, $c_{uv} = \sqrt{\frac{1}{d_u d_v}}$. The matrix form now reads:

$$\mathbf{H} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}_1 + \mathbf{X}\mathbf{W}_0\right)$$

Note: this operation is related to the *symmetric normalised Laplacian*!

In fact, we are two steps away from *the most popular GNN layer!*

Motivated by the very easy-to-overfit datasets of its time, let's simplify this layer even further.

Graph convolutional network

First observation: halve the number of parameters if $\mathbf{W}_0 = \mathbf{W}_1 = \mathbf{W}$

$$\mathbf{H} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W} + \mathbf{X}\mathbf{W}\right) = \sigma\left(\left(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{X}\mathbf{W}\right)$$

Second observation: this operator now has largest eigenvalue 2, which can lead to exploding parameters again. Hence, we renormalise it:

$$\mathbf{H} = \sigma\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}\right)$$

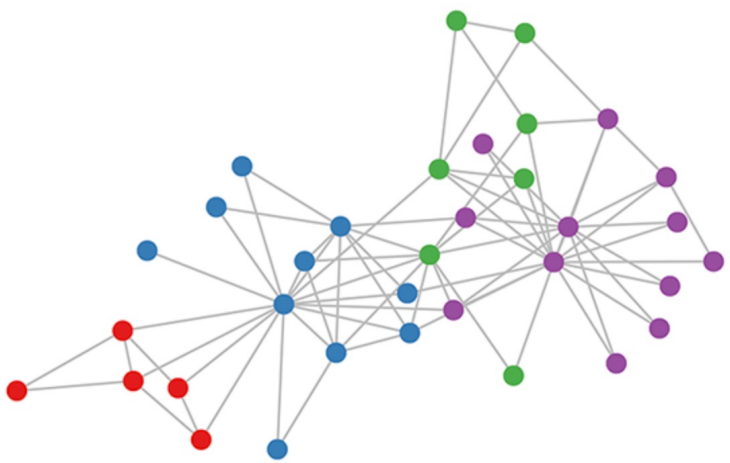
Where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, and $\widetilde{\mathbf{D}}$ is its corresponding degree matrix.
(**N.B.** this step is equivalent to adding u into $\mathcal{N}_u$ and ignoring the residual term in ϕ)

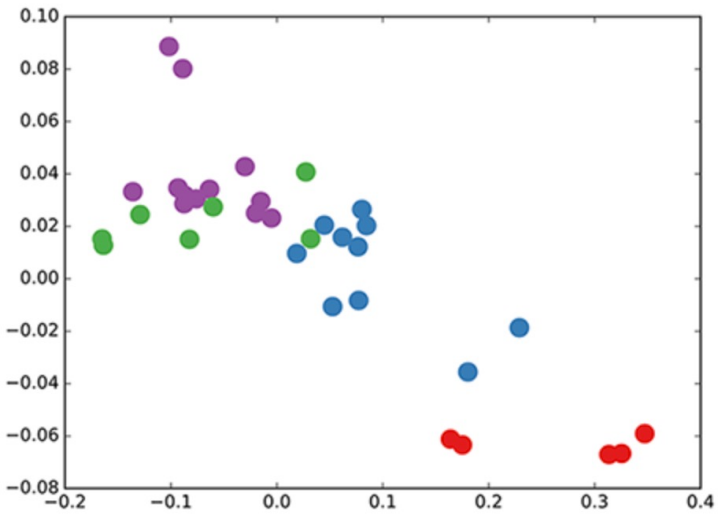
Graph convolutional network (**GCN**; Kipf & Welling, ICLR'17)

Empirical performance of GCNs

While GCNs are *simple,* they already encode a strong *inductive bias*!

Randomly initialised GCN on Zachary's karate club network:





If you are likely to share labels with a neighbour…
Averaging your neighbours can be a powerful predictor!

*Just **how** powerful is aggregation?*

Do we even **need** *deep learning* for many graph datasets?
Let's try to strip the parameters and nonlinearity from a GCN…

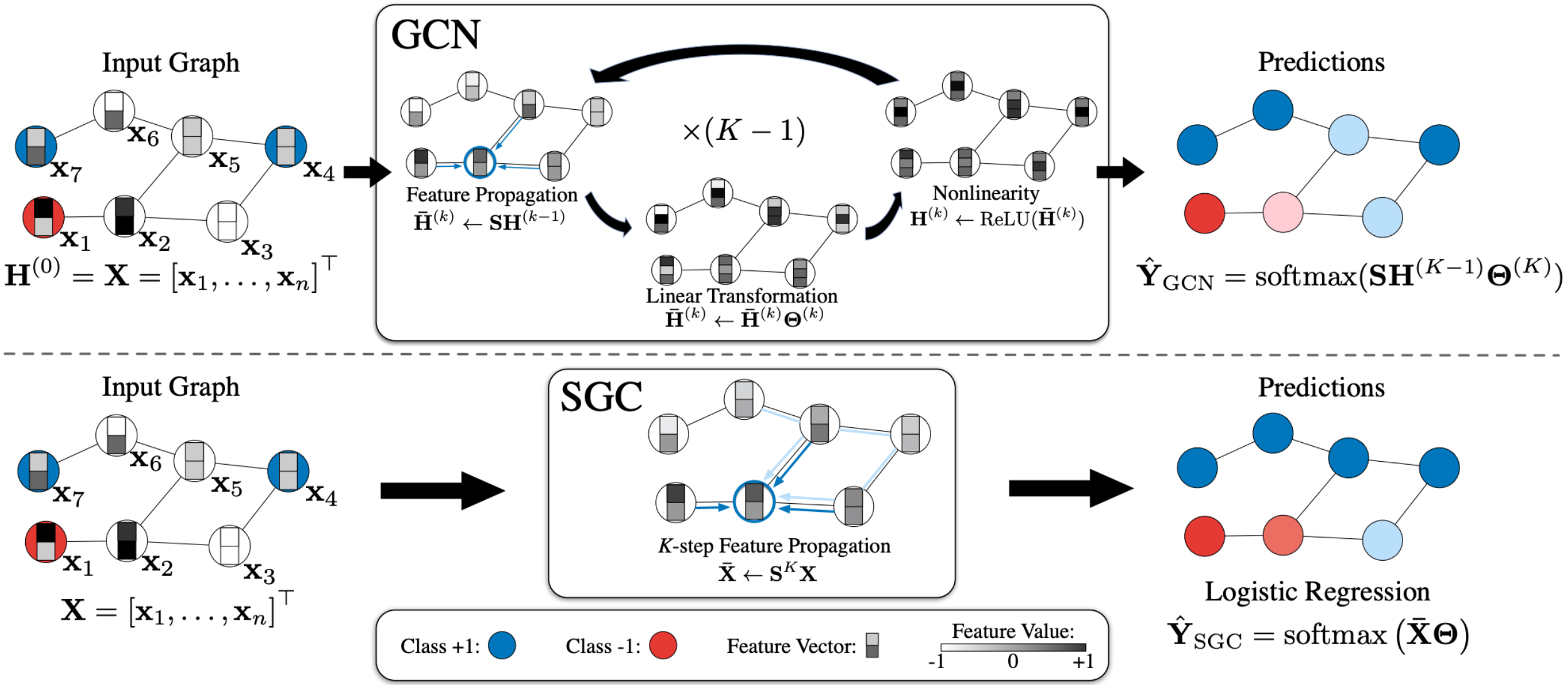
After K steps, the node features become $\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{K}\mathbf{X}$

Now, e.g., to classify nodes, learn a simple *logistic regressor*:

$$\mathrm{softmax}\left(\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{K}\mathbf{X}\mathbf{W}\right)$$

Yielding the *simplified graph convolution* (**SGC**; Wu *et al.*, ICML'19)
Near **state-of-the-art** on many tasks of interest; very efficient to train!

SGC vs. GCN


Input Graph
$\mathbf{x}_7$ $\mathbf{x}_6$ $\mathbf{x}_5$ $\mathbf{x}_4$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$
$\mathbf{H}^{(0)} = \mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$
GCN
Feature Propagation
$\bar{\mathbf{H}}^{(k)} \leftarrow \mathbf{S}\mathbf{H}^{(k-1)}$
$\times(K-1)$
Linear Transformation
$\bar{\mathbf{H}}^{(k)} \leftarrow \bar{\mathbf{H}}^{(k)}\mathbf{\Theta}^{(k)}$
Nonlinearity
$\mathbf{H}^{(k)} \leftarrow \mathrm{ReLU}(\bar{\mathbf{H}}^{(k)})$
Predictions
$\hat{\mathbf{Y}}_{\mathrm{GCN}} = \mathrm{softmax}(\mathbf{S}\mathbf{H}^{(K-1)}\mathbf{\Theta}^{(K)})$
Input Graph
$\mathbf{x}_7$ $\mathbf{x}_6$ $\mathbf{x}_5$ $\mathbf{x}_4$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$
$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$
SGC
K-step Feature Propagation
$\bar{\mathbf{X}} \leftarrow \mathbf{S}^K\mathbf{X}$
Predictions
Logistic Regression
$\hat{\mathbf{Y}}_{\mathrm{SGC}} = \mathrm{softmax}\,(\bar{\mathbf{X}}\mathbf{\Theta})$
Class +1:
Class -1:
Feature Vector:
Feature Value:
-1
0
+1

Multi-hop convolutional GNNs via matrix multiplication

We can represent conv-GNNs using *sparse matrix multiplications*!
(For choices of *nonparametric* $\oplus$ other than $\sum$, this idea still holds; just over a different *semiring*)

This gives them *scalability* benefits compared to other GNNs

It also allows us to easily aggregate over *multiple hops* in one layer!

For example:

$$\mathbf{H} = \sigma\left(\sum_{k=0}^{K} \mathbf{A}^k \mathbf{X} \mathbf{W}_k\right)$$

will combine information from the K-hop neighbourhood!
(**N.B.** this layer still fits the conv-GNN framework; we need only re-define $\mathcal{N}_u$ and set c_{uv} accordingly)

Chebyshev Networks

A popular multi-hop conv-GNN from Defferrard *et al.* (NeurIPS'16):

$$\mathbf{H} = \sigma\left(\sum_{k=0}^{K} \alpha_k \left(\frac{2}{\lambda_{\max}}\mathbf{L}_{\text{sym}} - \mathbf{I}\right)^k \mathbf{X}\mathbf{W}_k\right)$$

where $\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the symmetric graph Laplacian and α_k is the order-k coefficient of its *Chebyshev polynomial*

($\lambda_{\max}$ is the largest eigenvalue of $\mathbf{L}$; the $\frac{2}{\lambda_{\max}}$ factor is designed to protect against exploding outputs)

N.B. GCN can be interpreted as a ChebyNet with $K = 1, \lambda_{\max} \approx 2$

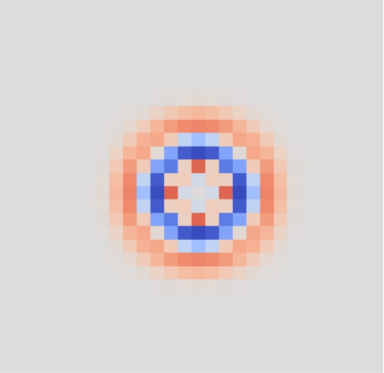
Chebyshev polynomials are convenient as they offer a sparse and scalable multi-hop method, which is quite performant in practice

To what extent are ChebyNets convolution-like?

One motivation for considering stronger GNNs comes from *images*
What happens when a ChebyNet is applied to an image graph?

Assume every pixel connected to its four immediate neighbours
Then the weights of a 3 x 3 conv kernel around a pixel would look like:

$$\begin{pmatrix} w_2 & w_1 & w_2 \\ w_1 & w_0 & w_1 \\ w_2 & w_1 & w_2 \end{pmatrix}$$



(to see why, note that images are *regular graphs*: c_{uv} is a constant)

Such filters are *radial*, and are fundamentally limited in expressivity
ChebyNets *(hence GCNs also!)* cannot represent all image CNNs.
(See Huszár, *How powerful are Graph Convolutions?*)

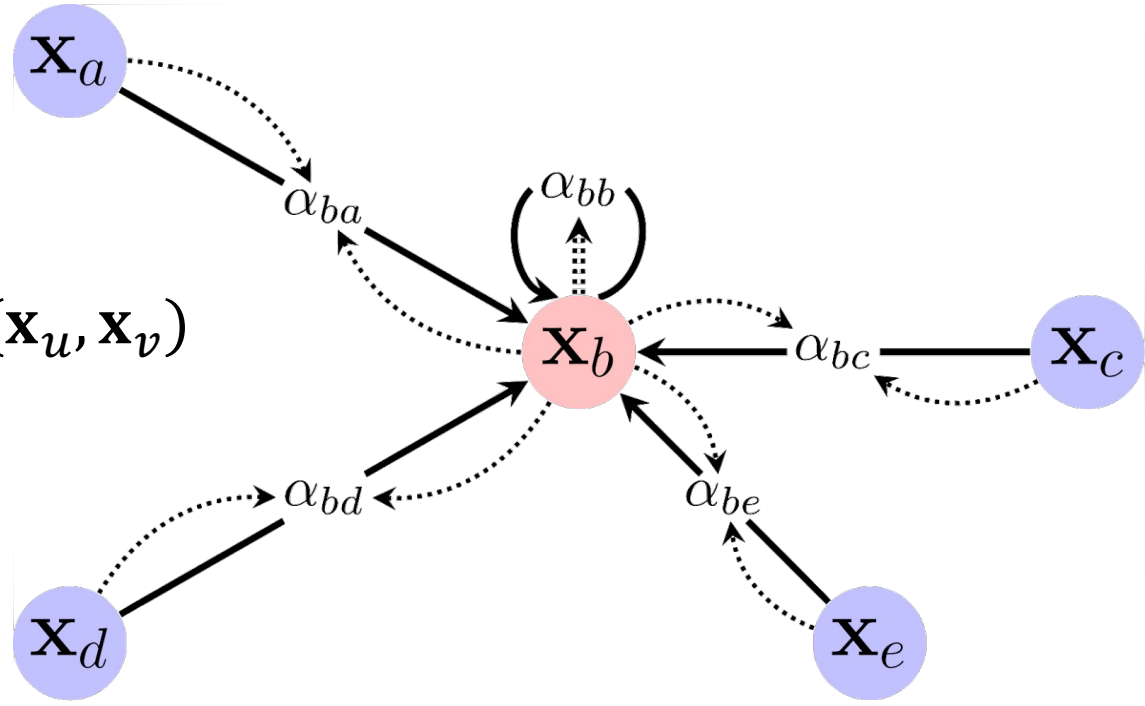
Attentional GNN

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

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} a(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)\right)$$

Attention weights computed as $\alpha_{uv} = a(\mathbf{x}_u, \mathbf{x}_v)$

- MoNet (Monti *et al.*, CVPR'17)
- GAT (Veličković *et al.*, ICLR'18)
- GATv2 (Brody *et al.*, ICLR'22)



Attentional

Useful as "middle ground"
w.r.t. *capacity, scale, interpretability*
Edges need not encode homophily
But still computing only a **scalar** per edge

Mixture model CNNs

The need for general-purpose *anisotropic aggregation* was detected by the *mixture model CNN* (**MoNet**; Monti *et al.*, CVPR'17)

MoNet approaches this topic from the point of view of *meshes*:

$$\mathbf{h}_u = \sigma\left(\sum_{v \in \mathcal{N}_u} w\big(\mathbf{e}(u,v)\big)\mathbf{W}\mathbf{x}_v\right)$$

$\mathbf{e} : \mathcal{V}^2 \to \mathbb{R}^k$ is a *pseudo-coordinate function*

$w : \mathbb{R}^k \to \mathbb{R}$ is a *weighting function*

MoNet is very general

$$\mathbf{h}_u = \sigma\left(\sum_{v \in \mathcal{N}_u} w\big(\mathbf{e}(u,v)\big)\mathbf{W}\mathbf{x}_v\right)$$

$\mathbf{e} : \mathcal{V}^2 \to \mathbb{R}^k$ is a *pseudo-coordinate function*
$w : \mathbb{R}^k \to \mathbb{R}$ is a *weighting function*

It is clear to see how *all* isotropic GNNs fit within this framework.
For example, we can recover GCNs by setting:

$$\mathbf{e}(u,v) = [d_u, d_v]^\top \qquad w(\mathbf{e}) = \left(1 - \left|1 - \frac{1}{\sqrt{e_1}}\right|\right)\left(1 - \left|1 - \frac{1}{\sqrt{e_2}}\right|\right)$$

And many other standard *anisotropic* methods, such as image CNNs
(which GCNs / ChebyNets could not!)

$\mathbf{e}$ extracts a vector-based representation of the (u,v) edge
w converts this vector into an aggregation coefficient

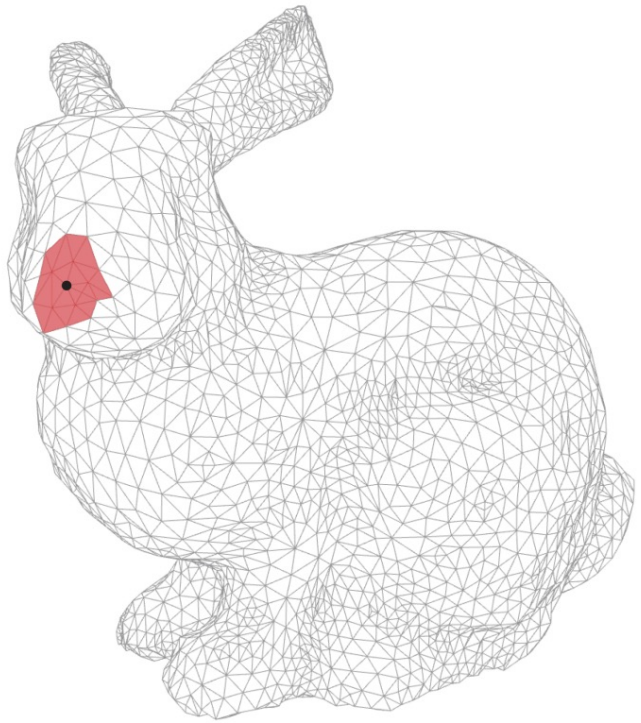
What is missing?

$$\mathbf{h}_u = \sigma\left(\sum_{v \in \mathcal{N}_u} w\big(\mathbf{e}(u,v)\big)\mathbf{W}\mathbf{x}_v\right)$$

$\mathbf{e} : \mathcal{V}^2 \to \mathbb{R}^k$ is a *pseudo-coordinate function*
$w : \mathbb{R}^k \to \mathbb{R}$ is a *weighting function*

While powerful, MoNet's motivation still came from the *mesh* domain, where nodes are expected to have *coordinates*

What does the MoNet paper do for *graph* inputs?



What is missing?

$$\mathbf{h}_u = \sigma\left(\sum_{v \in \mathcal{N}_u} w\big(\mathbf{e}(u,v)\big)\mathbf{W}\mathbf{x}_v\right)$$

$\mathbf{e} : \mathcal{V}^2 \to \mathbb{R}^k$ is a *pseudo-coordinate function*
$w : \mathbb{R}^k \to \mathbb{R}$ is a *weighting function*

For graphs, the MoNet paper instead uses only *simple structure* in $\mathbf{e}$:

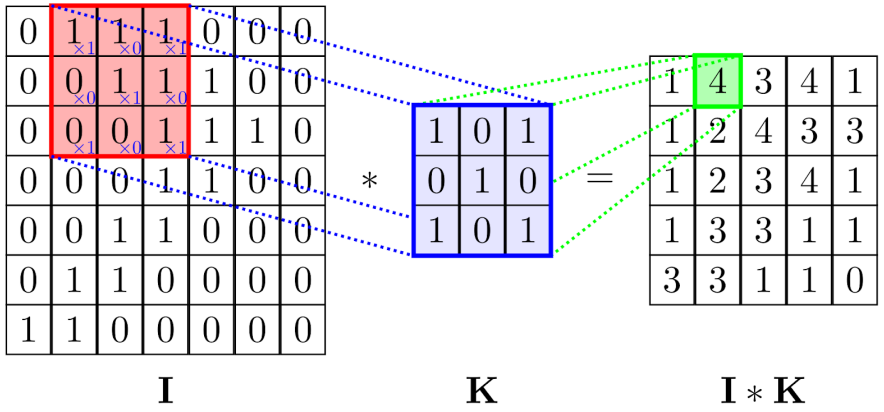
$$\mathbf{e}(u,v) = \tanh\left(\mathbf{A}\left[\frac{1}{\sqrt{d_u}}, \frac{1}{\sqrt{d_v}}\right]^\top + \mathbf{b}\right)$$

$\mathbf{A}, \mathbf{b}$ are *learnable*

And therefore, **still** behaved *isotropically* for regular graphs / images!

FYI: the weighting function MoNet used was a Gaussian kernel: $w(\mathbf{e}) = \exp\left(-\frac{1}{2}(\mathbf{e}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{e}-\boldsymbol{\mu})\right)$

*Towards a **truly** convolutional GNN*



Our aim is to *generalise* **CNNs** to graphs

Allow *different* neighbours to be weighted *differently*
even if they are *structurally identical*!

MoNet's generalised *weighting functions* are *sufficient* to support this

BUT the *input* to the weighting function was thus far always *structural*
Therefore, hopeless when nodes are structurally identical

To achieve this, a *paradigm shift* was needed
Where else can we find information to **disambiguate** the neighbours?

Features-as-coordinates: graph attention networks

The nodes' **feature vectors**, $\mathbf{x}_u$, may also hold identifying information

Representing *features as coordinates* allows us to move away from the mesh-based angle and into the realm of *attention;* deciding how much to *attend* to each neighbour based on its content

The mesh angle *will* make a very important comeback towards the end of the course ☺

Embodied by *graph attention networks* (**GAT**; Veličković *et al.* (ICLR'18))

$$\mathbf{h}_u = \sigma\left(\sum_{v \in \mathcal{N}_u} \alpha(\mathbf{x}_u, \mathbf{x}_v)\mathbf{W}\mathbf{x}_v\right)$$

Where $\alpha : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ is the *attention mechanism*

in the MoNet framework, set $\mathbf{e}(u, v) = \mathbf{x}_u \| \mathbf{x}_v$.

Graph attention networks

With this change in thinking, we now have satisfied *all* of the requirements for a GNN that generalises image CNNs
(local, computationally efficient, anisotropic, …)

The GAT design comes with a few additional perks:

- Supports arbitrary weighted aggregation
- Still computing only *one* scalar per edge
- Potential for *interpretability* and *structure discovery*: attention function computes a direct measure of affinity between neighbours (careful!)

Accordingly, GATs have seen popularity in *scientific applications*
Generally seen as a *"sweet spot"* between scalability and expressivity
(This also relates to the success of the Transformer architecture; a link we will explore in future lectures)

Selection of GAT applications

Interaction data are identifiable even across long periods of time

Ana-Maria Creţu, Federico Monti, Stefano Marrone, Xiaowen Dong, Michael Bronstein & Yves-Alexandre de Montjoye

Nature Communications **13**, Article number: 313 (2022) | Cite this article

P-Companion: A Principled Framework for Diversified Complementary Product Recommendation

Junheng Hao[1], Tong Zhao[2], Jin Li[2], Xin Luna Dong[2]
Christos Faloutsos[2,3], Yizhou Sun[1], Wei Wang[1]
University of California, Los Angeles[1], Amazon.com[2], Carnegie Mellon University[3]

Network medicine framework for identifying drug-repurposing opportunities for COVID-19

Deisy Morselli Gysi, Ítalo do Valle, Marinka Zitnik, Asher Ameli, Xiao Gan, Onur Varol, …
+ See all authors and affiliations

PNAS May 11, 2021 118 (19) e2025581118; https://doi.org/10.1073/pnas.2025581118

TacticAI: an AI assistant for football tactics

Zhe Wang, Petar Veličković, Daniel Hennes, Nenad Tomašev, Laurel Prince, Michael Kaisers, Yoram Bachrach, Romuald Elie, Li Kevin Wenliang, Federico Piccinini, William Spearman, Ian Graham, Jerome Connor, Yi Yang, Adrià Recasens, Mina Khan, Nathalie Beauguerlange, Pablo Sprechmann, Pol Moreno, Nicolas Heess, Michael Bowling, Demis Hassabis & Karl Tuyls

Nature Communications **15**, Article number: 1906 (2024) | Cite this article

Fake News Detection on Social Media using Geometric Deep Learning

Federico Monti[1,2] **Fabrizio Frasca**[1,2] **Davide Eynard**[1,2] **Damon Mannion**[1,2]
Michael M. Bronstein[1,2,3]
[1]Fabula AI United Kingdom [2]USI Lugano Switzerland [3]Imperial College United Kingdom

VectorNet: Encoding HD Maps and Agent Dynamics from Vectorized Representation

Jiyang Gao[1*] Chen Sun[2*] Hang Zhao[1] Yi Shen[1]
Dragomir Anguelov [1] Congcong Li[1] Cordelia Schmid [2]
[1]Waymo LLC [2] Google Research

Which attention mechanism to use?

The GAT paper is, in principle, not enforcing a particular function α
Most generally, it should be a deep MLP

But in practice, to prevent overfitting on the (now deprecated) datasets of the time, the function α had to be substantially *weakened*

Therefore, the GAT paper implements *linear attention:*

$$e(\mathbf{x}_u, \mathbf{x}_v) = \mathrm{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{x}_u \| \mathbf{x}_v\right]\right); \quad \alpha(\mathbf{x}_u, \mathbf{x}_v) = \frac{\exp e(\mathbf{x}_u, \mathbf{x}_v)}{\sum_{w \in \mathcal{N}_u} \exp e(\mathbf{x}_u, \mathbf{x}_w)}$$

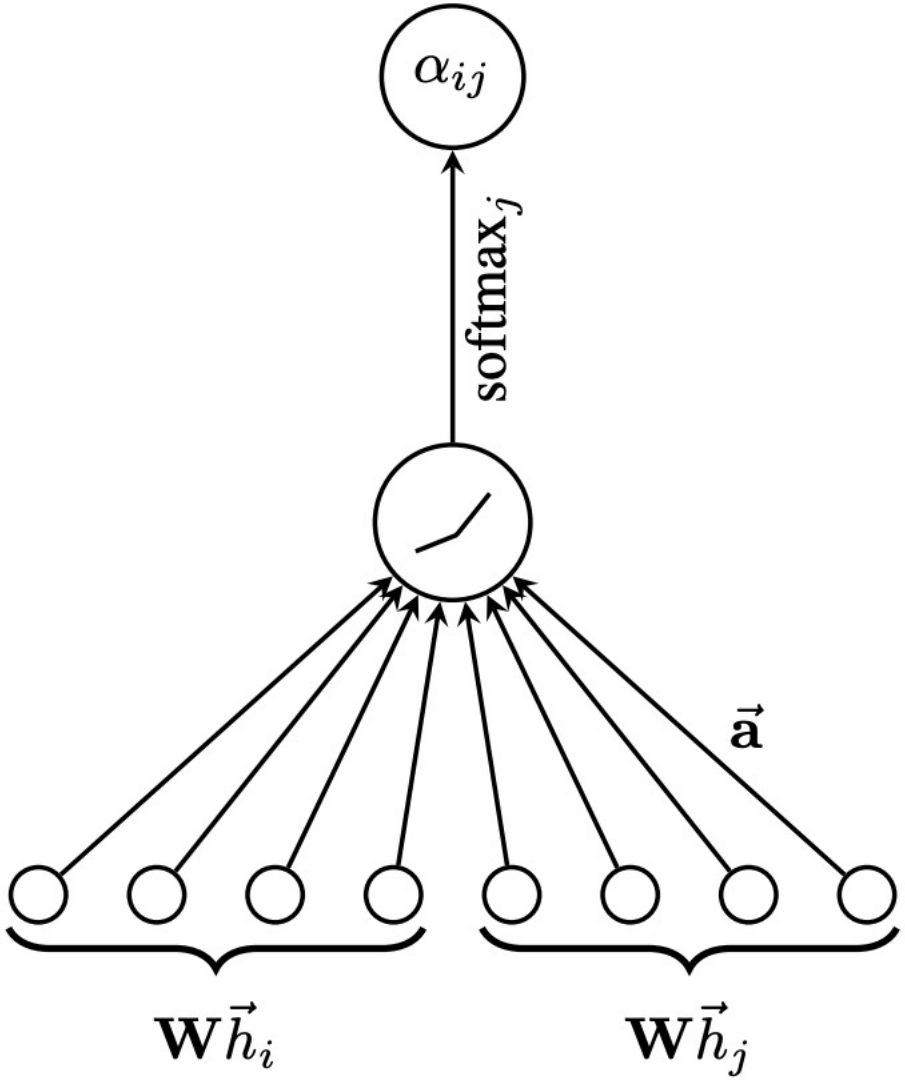
Occasionally, the use of linear attention is *synonymous* with "GAT"

GAT linear attention

$$e(\mathbf{x}_u, \mathbf{x}_v) = \mathrm{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{x}_u \| \mathbf{x}_v\right]\right)$$

$$\alpha(\mathbf{x}_u, \mathbf{x}_v) = \frac{\exp e(\mathbf{x}_u, \mathbf{x}_v)}{\sum_{u \in \mathcal{N}(w)} \exp e(\mathbf{x}_u, \mathbf{x}_w)}$$

Note: the original GAT paper (right) performed an additional *multiplication* of node features with **W**. This addition does **not** increase expressive power!



Static attention

However, in recent years, the field has moved on from simple benchmarks where nonlinear attention would overfit

Specifically, a key issue of linear attention is that it is *static*
There always exists *one* node, $f \in \mathcal{V}$, whose features $\mathbf{x}_f$ *maximise* $e(\mathbf{x}_u, \mathbf{x}_f)$, *regardless* of the receiver features $\mathbf{x}_u$!

This node is exactly the node that optimizes $\mathbf{a}^\top \mathbf{x}_f$
The features of the receiver only provide an *additive* factor of $\mathbf{a}^\top \mathbf{x}_u$ so they cannot affect the *order* of the coefficients

Hidden **assumption**: there exists a *global* ranking of node "influences"
Does not always hold!

Dynamic attention: GATv2

The static attention issue went unnoticed until quite recently!

It was both identified and patched by Brody *et al.* (ICLR'22)
They propose the **GATv2** attention mechanism, as follows:

$$e(\mathbf{x}_u, \mathbf{x}_v) = \mathbf{a}^\top \mathrm{LeakyReLU}\big(\mathbf{W}\big[\mathbf{x}_u \| \mathbf{x}_v\big]\big)$$

(the softmax is still applied to the coefficients within $\alpha(\mathbf{x}_u, \mathbf{x}_v)$)

Since this is effectively a two-layer MLP, it is a *universal approximator*
Hence it can learn **any** attention function, including *dynamic* ones

Conveniently, Brody *et al.* also prove that *dot-product attention* (as seen in Transformers) is **not** always able to compute dynamic attention

The need for dynamic attention: Dictionary Lookup

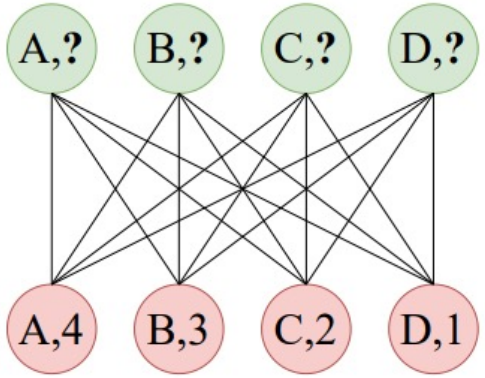


Figure 2: The DICTIONARY-LOOKUP problem of size k=4: every node in the bottom row has an alphabetic *attribute* ($\{A, B, C, ...\}$) and a numeric *value* ($\{1, 2, 3, ...\}$); every node in the upper row has only an attribute; the goal is to predict the value for each node in the upper row, using its attribute.

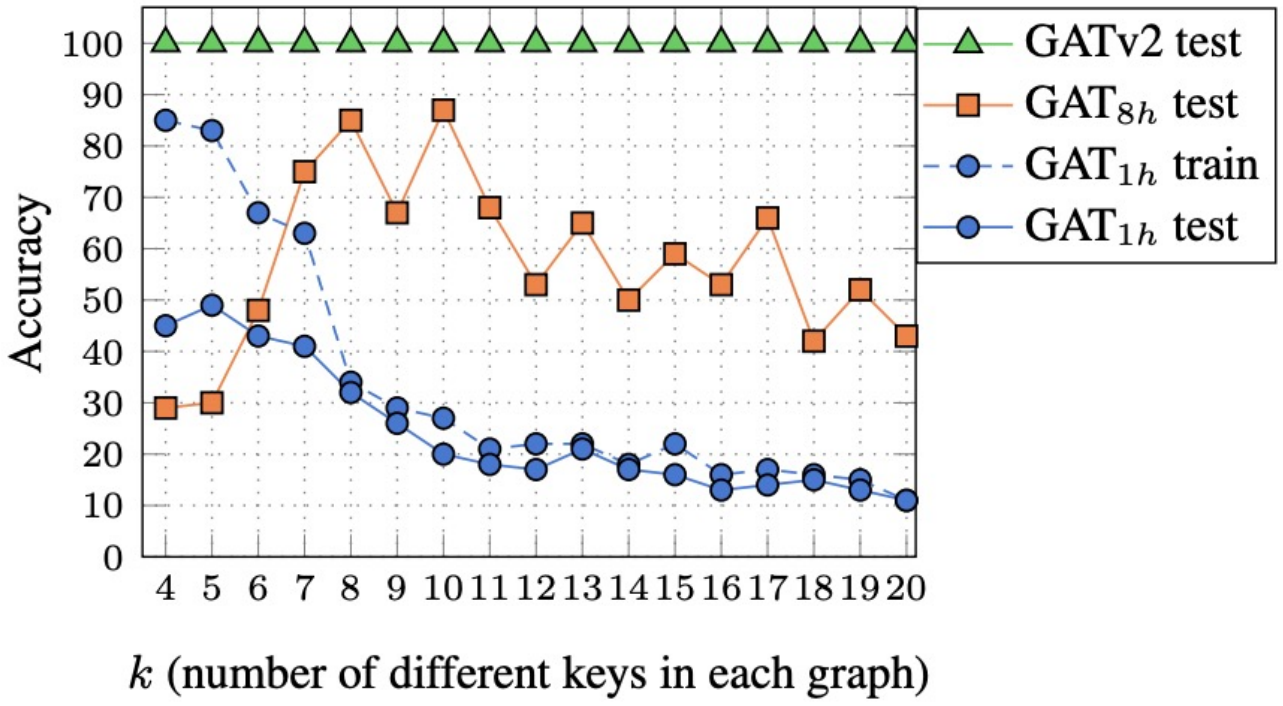
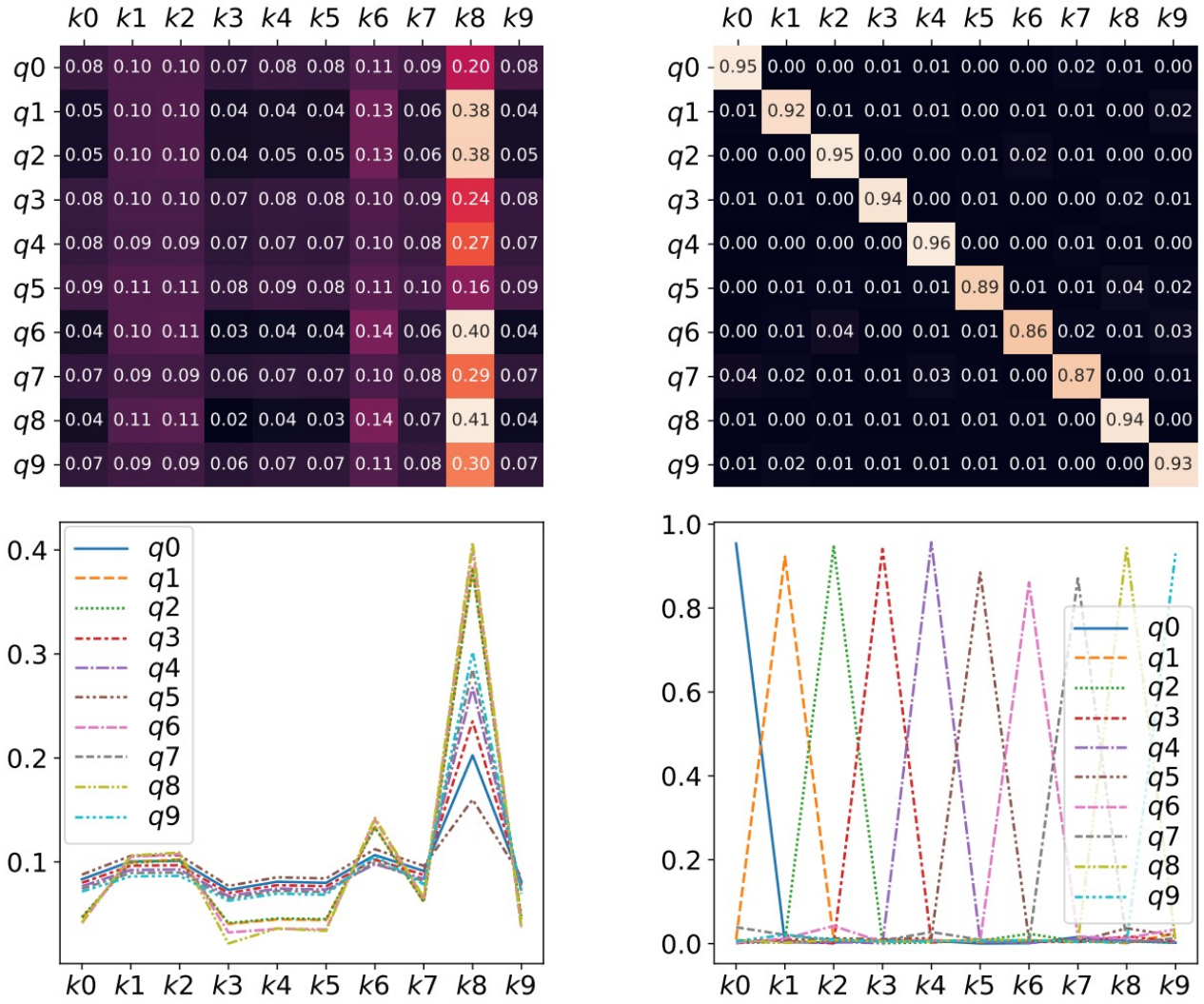


Figure 3: The DICTIONARYLOOKUP problem: GATv2 easily achieves 100% train and test accuracies even for k=100 and using only a single head.

Static and dynamic attention



(a) Attention in standard GAT (Veličković et al. (2018)) (b) Attention in GATv2, our fixed version of GAT

Note on GAT scalability

Comparing GAT and GATv2 highlights also a *storage complexity* aspect

Recall the attention mechanisms:

GAT: $e(\mathbf{x}_u, \mathbf{x}_v) = \text{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{x}_u \| \mathbf{x}_v\right]\right)$

GATv2: $e(\mathbf{x}_u, \mathbf{x}_v) = \mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{W}\left[\mathbf{x}_u \| \mathbf{x}_v\right]\right)$

While they seem to use the same operations in a different order, GATv2 requires *explicitly materialising* the concatenation $\left[\mathbf{x}_u \| \mathbf{x}_v\right]$

Therefore, storage complexity proportional to the number of *edges* is incurred; often, this is significantly larger than the number of *nodes*

Exercise: Implement GAT attention with $O(\mathcal{V})$ storage complexity

Multi-head attention

All formulations of attentional GNNs so far only used *one* learnable attention mechanism; learns one mode of interaction!

It is beneficial to consider multiple modes of interaction at once: deploy *multi-head attention* (Vaswani *et al.*, NeurIPS'17)

$$\mathbf{h}_u = \mathop{\Big\|}_{k=1}^{K} \sigma\left(\sum_{v \in \mathcal{N}_u} \alpha_k(\mathbf{x}_u, \mathbf{x}_v) \mathbf{W}_k \mathbf{x}_v \right)$$

This allows GATs to learn multiple modes of interaction
(and ameliorates the static attention issue to an extent)

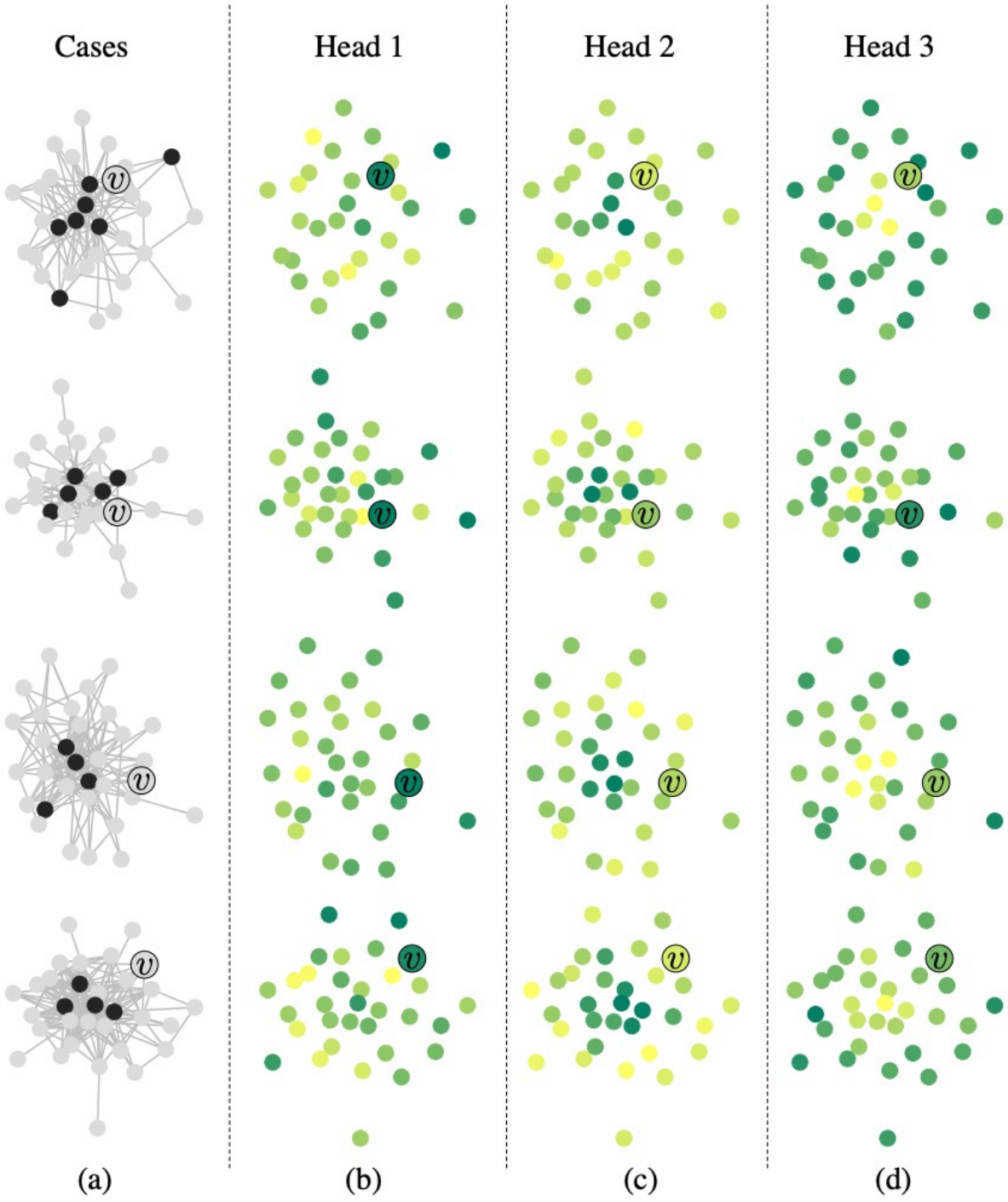
Analysing the attention heads

DeepInf (Qiu *et al.*, KDD'18)

The *first* qualitative study that interprets a GAT model

Task is to classify whether node v will perform some action in a social network (e.g. liking/retweeting)

Different *heads* learn to focus on different *aspects* of v's neighbourhood



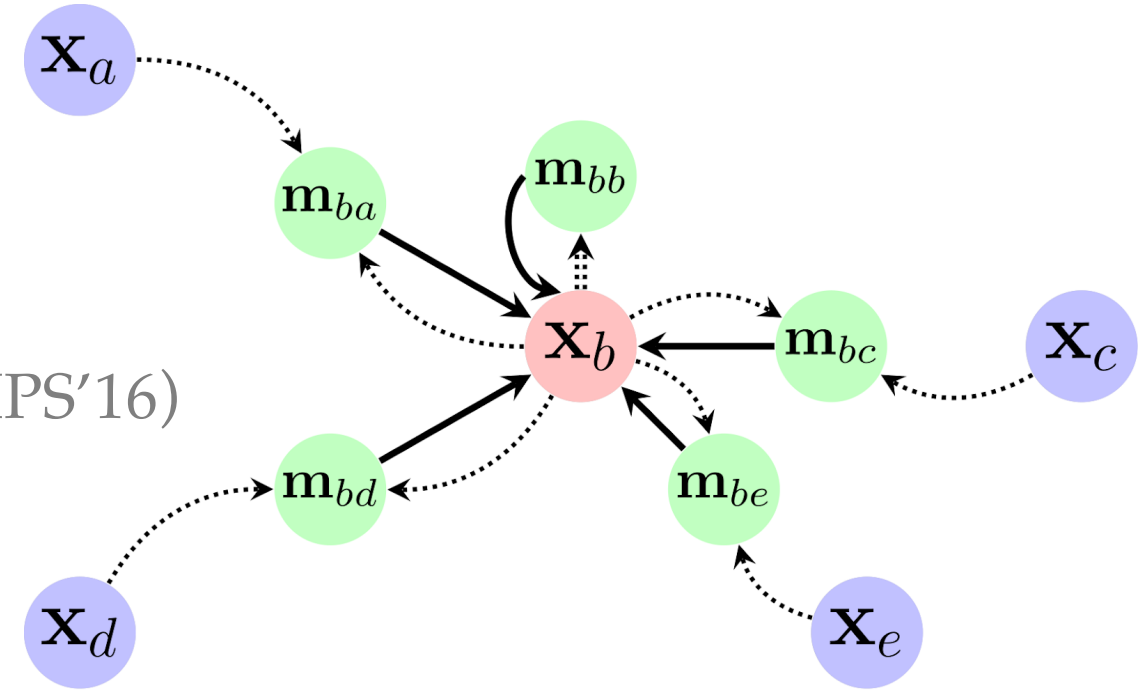
Message-passing GNN

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

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right)$$

Messages computed as $\mathbf{m}_{uv} = \psi(\mathbf{x}_u, \mathbf{x}_v)$

- Interaction Nets (Battaglia *et al.*, NeurIPS'16)
- MPNN (Gilmer *et al.*, ICML'17)
- GraphNets (Battaglia *et al.*, 2018)



Message-passing

Most **generic** GNN layer
Edges give "recipe" for passing data
May have *scalability* or *learnability* issues
Ideal for *computational chemistry*, *reasoning* and *simulation* tasks

Towards the most expressive GNN

On our journey through convolutional and attentional GNNs, we have gradually increased *expressive power* (at the expense of scalability)

The latest GATv2 model, in fact, materialises *vectors* per *edge*

At this point, a complex process is encoded over the edges, **but** it is still used only to determine a *weighted combination* of the neighbours

The ”next step” in generality:
the *quantities being combined* themselves depend on *both* nodes
The receiver can explicitly *condition* what it receives from the sender!

This leads us to *message-passing GNNs*

Interaction networks

The need for arbitrary vector-based messages appeared early in *physics*

It allows us to easily encode various kinds of interactions (e.g. *forces*) between the nodes in the graph

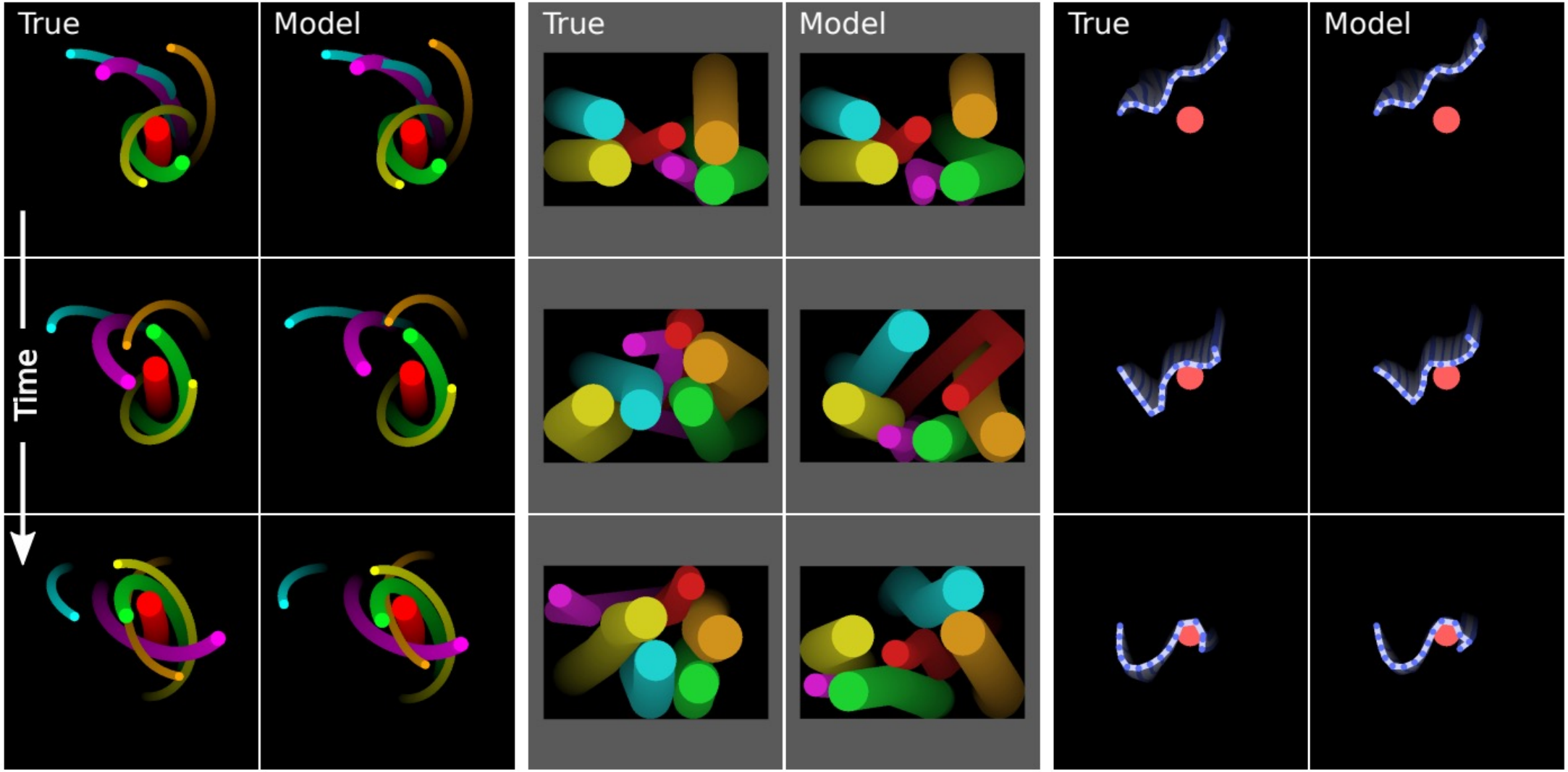
Eponymous *interaction network* of Battaglia *et al*. (NeurIPS'16)

Apply GNNs to predict future trajectories of n-body systems, bouncing balls, and strings

Physics, especially *simulations*, remains one of the key domains that stimulate the development of expressive GNNs

(We will revisit physics simulations from various aspects throughout the course)

Interaction networks in action



Computational chemistry

Another area that drives modern GNN design is *chemistry*
Molecules lend themselves to a graph representation quite naturally
Applications in *quantum chemistry, drug design, material science*…

In fact, it can be argued that computational chemists invented the first general-purpose GNNs!

- ChemNet (Kireev *et al.*, CICS'95)
- Baskin *et al.* (CICS'97)
- Molecular Graph Networks (Merkwirth and Lengauer, CIM'05)

This drive continued well into the 2010s:

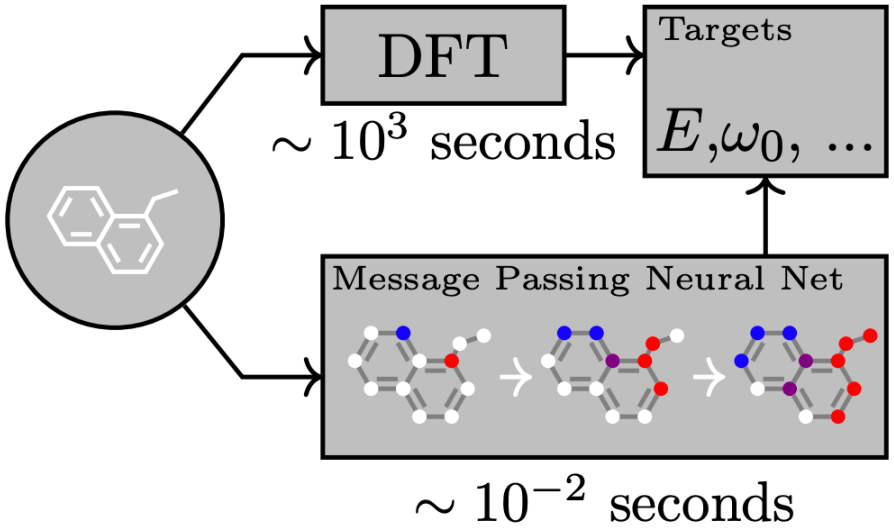
- Molecular fingerprinting GNNs (Duvenaud *et al.*, NeurIPS'15)
- GNNs for quantum chemistry (Gilmer *et al.*, ICML'17)

Neural message passing for quantum chemistry

In this work, Gilmer *et al*. tackle head-on the task of quantum property predictions from small-molecule datasets (such as QM9)

Their **target**: replace expensive DFT simulations with learnt GNN models

Contribution is also *theoretical*: Categorise **all** existing GNNs at the time into the *MPNN* framework



This framework was generic enough to reach *chemical accuracy* on 11 out of 13 of the tasks within QM9, after a thorough architecture scan.

General attributed graphs

Especially here, graphs can convey *rich* information at all granularities

We previously *ignored* non-node data for simplicity

Now we will assume the following possible set of features:

- *Node* features, $\mathbf{x}_u \in \mathbb{R}^k$ (e.g. atom type, charge, nb. of hydrogens)

General attributed graphs

Especially here, graphs can convey *rich* information at all granularities

We previously *ignored* non-node data for simplicity

Now we will assume the following possible set of features:

- *Node* features, $\mathbf{x}_u \in \mathbb{R}^k$
- *Edge* features, $\mathbf{x}_{uv} \in \mathbb{R}^l$ (e.g. bond type, is in a ring?)

General attributed graphs

Especially here, graphs can convey *rich* information at all granularities

We previously *ignored* non-node data for simplicity

Now we will assume the following possible set of features:

- *Node* features, $\mathbf{x}_u \in \mathbb{R}^k$
- *Edge* features, $\mathbf{x}_{uv} \in \mathbb{R}^l$
- *Graph* features, $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^m$ (e.g. molecular weight, fingerprints)

General attributed graphs

Especially here, graphs can convey *rich* information at all granularities

We previously *ignored* non-node data for simplicity

Now we will assume the following possible set of features:

- *Node* features, $\mathbf{x}_u \in \mathbb{R}^k$
- *Edge* features, $\mathbf{x}_{uv} \in \mathbb{R}^l$
- *Graph* features, $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^m$

Analogously defining latents $\mathbf{h}_u, \mathbf{h}_{uv}, \mathbf{h}_{\mathcal{G}}$

It is possible to extend this further (e.g. *hypergraphs*)
but such inputs can usually be represented as an instance of the above

Graph Networks

We will now define a general blueprint for a spatial GNN, which generalises all the flavours referenced before

We use the *Graph Network* (Battaglia *et al.*, 2018) as our basis
This is because it operates over *generic attributed* graphs

INs and MPNNs can be derived by *restricting* of its equations slightly

Dataflow:
- Update edge features (using graph + relevant nodes)
- Update node features (using updated relevant edges + graph)
- Update graph features (using updated nodes + edges)

+ extensive usage of *skip connections*!

Graph Networks

Update **edge** features (using graph + relevant nodes)

$$\mathbf{h}_{uv} = \psi\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Graph Networks

Update **edge** features (using graph + relevant nodes)

$$\mathbf{h}_{uv} = \psi\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Update **node** features (using updated relevant edges + graph)

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \mathbf{h}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Graph Networks

Update **edge** features (using graph + relevant nodes)

$$\mathbf{h}_{uv} = \psi\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Update **node** features (using updated relevant edges + graph)

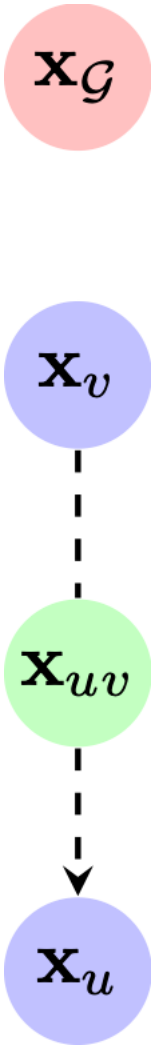
$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \mathbf{h}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Update **graph** features (using updated nodes + edges)

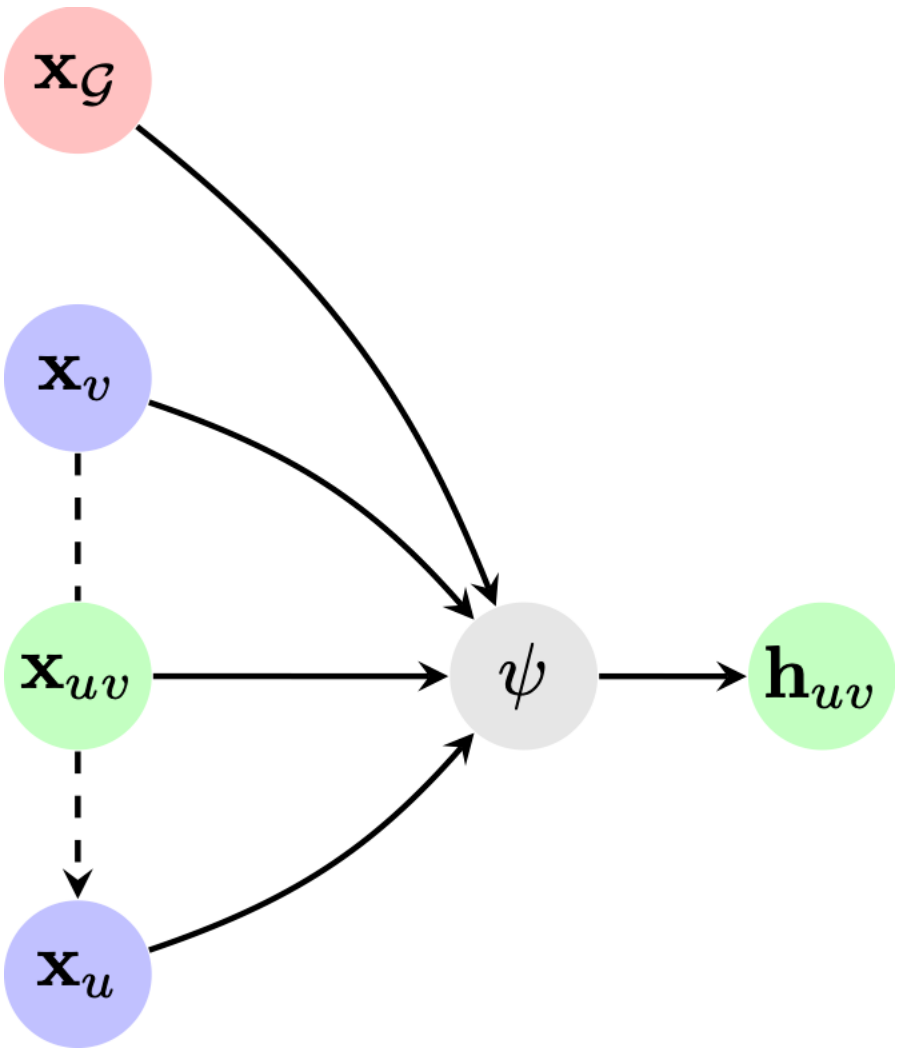
$$\mathbf{h}_{\mathcal{G}} = \rho\left(\bigoplus_{u \in \mathcal{V}} \mathbf{h}_u, \bigoplus_{(u,v) \in \mathcal{E}} \mathbf{h}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

(A very similar formulation treats graph features as a "master node")

Graph Networks, visualised

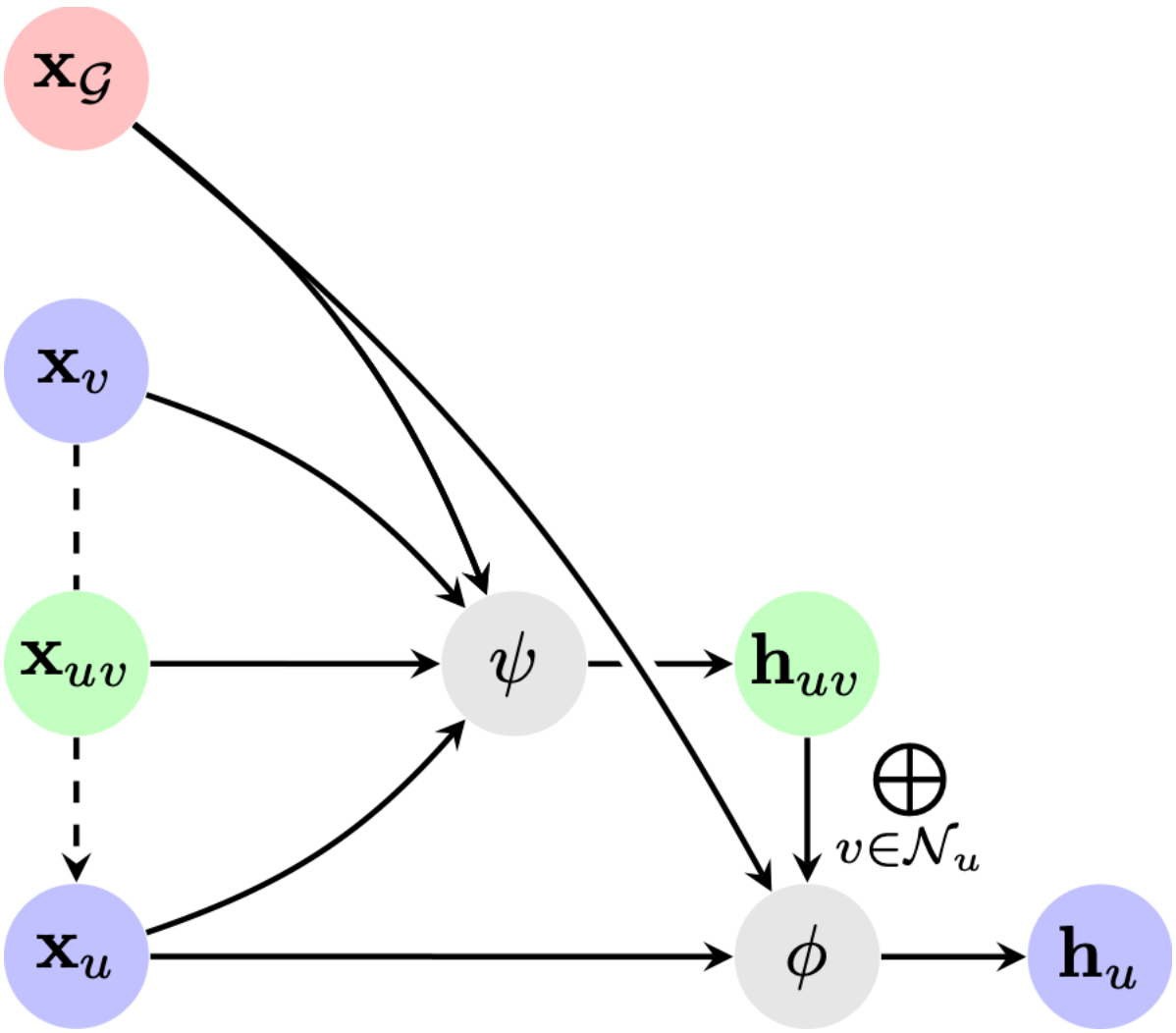


Graph Networks, visualised



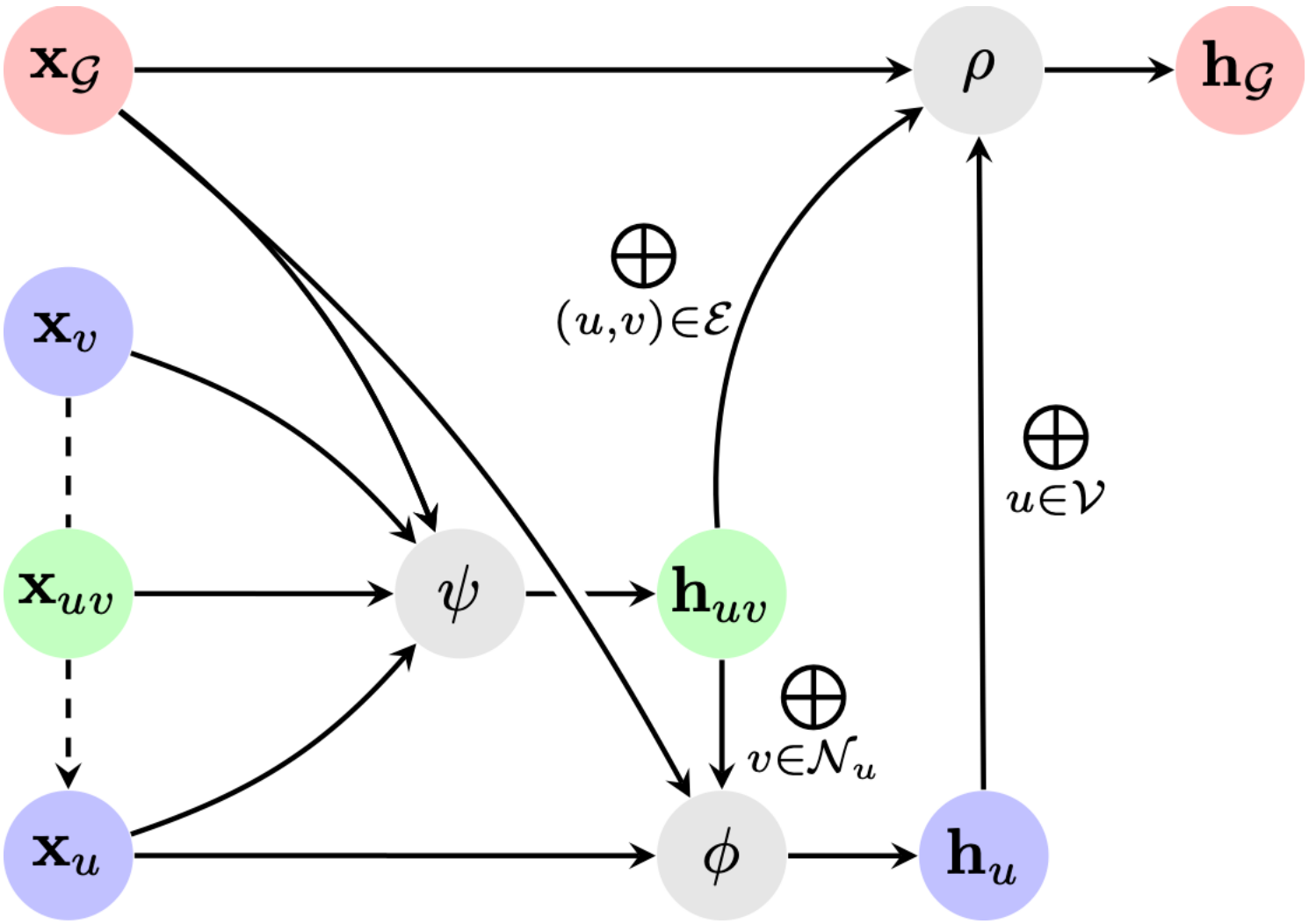
$$\mathbf{h}_{uv} = \psi\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Graph Networks, visualised



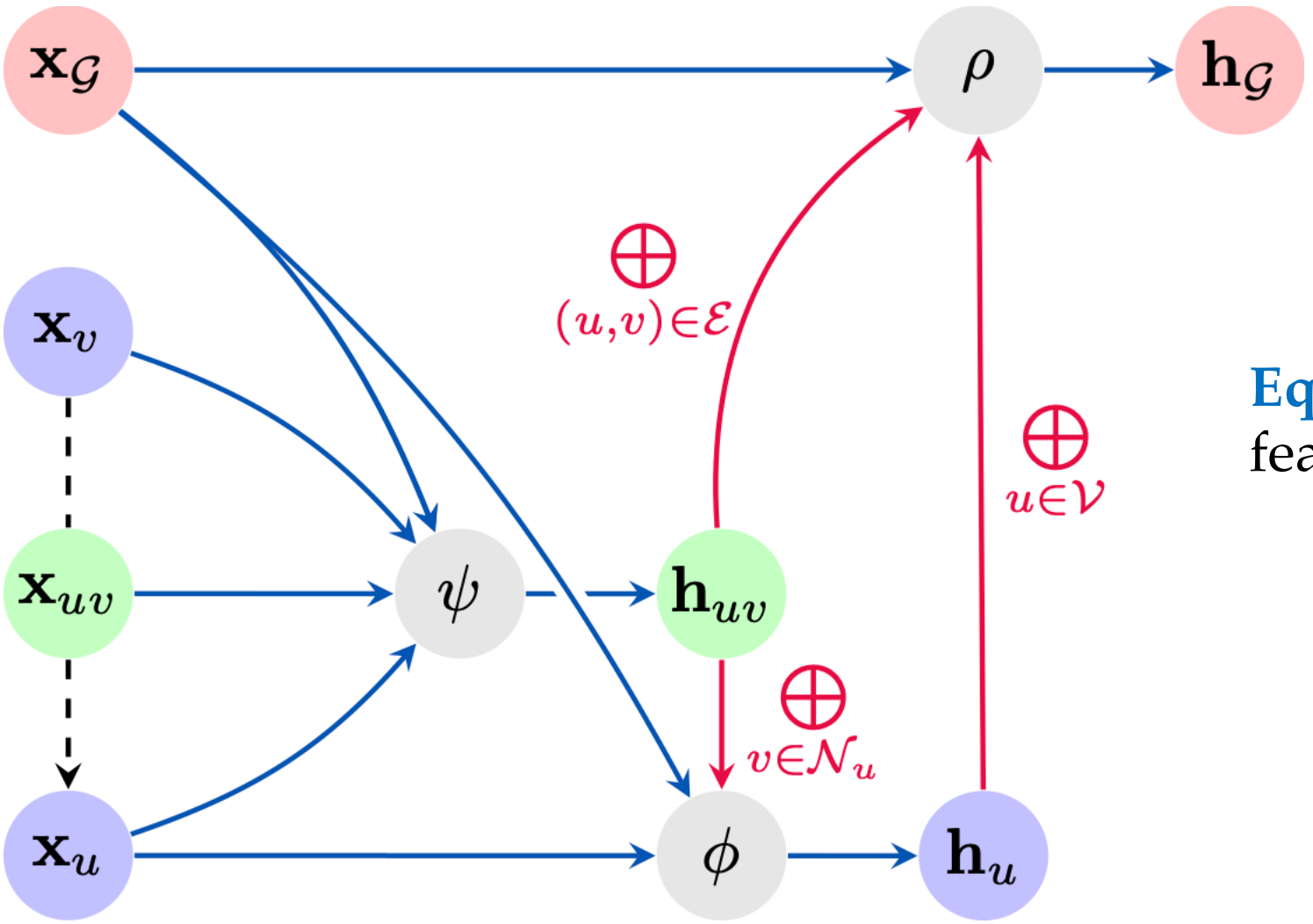
$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \mathbf{h}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Graph Networks, visualised



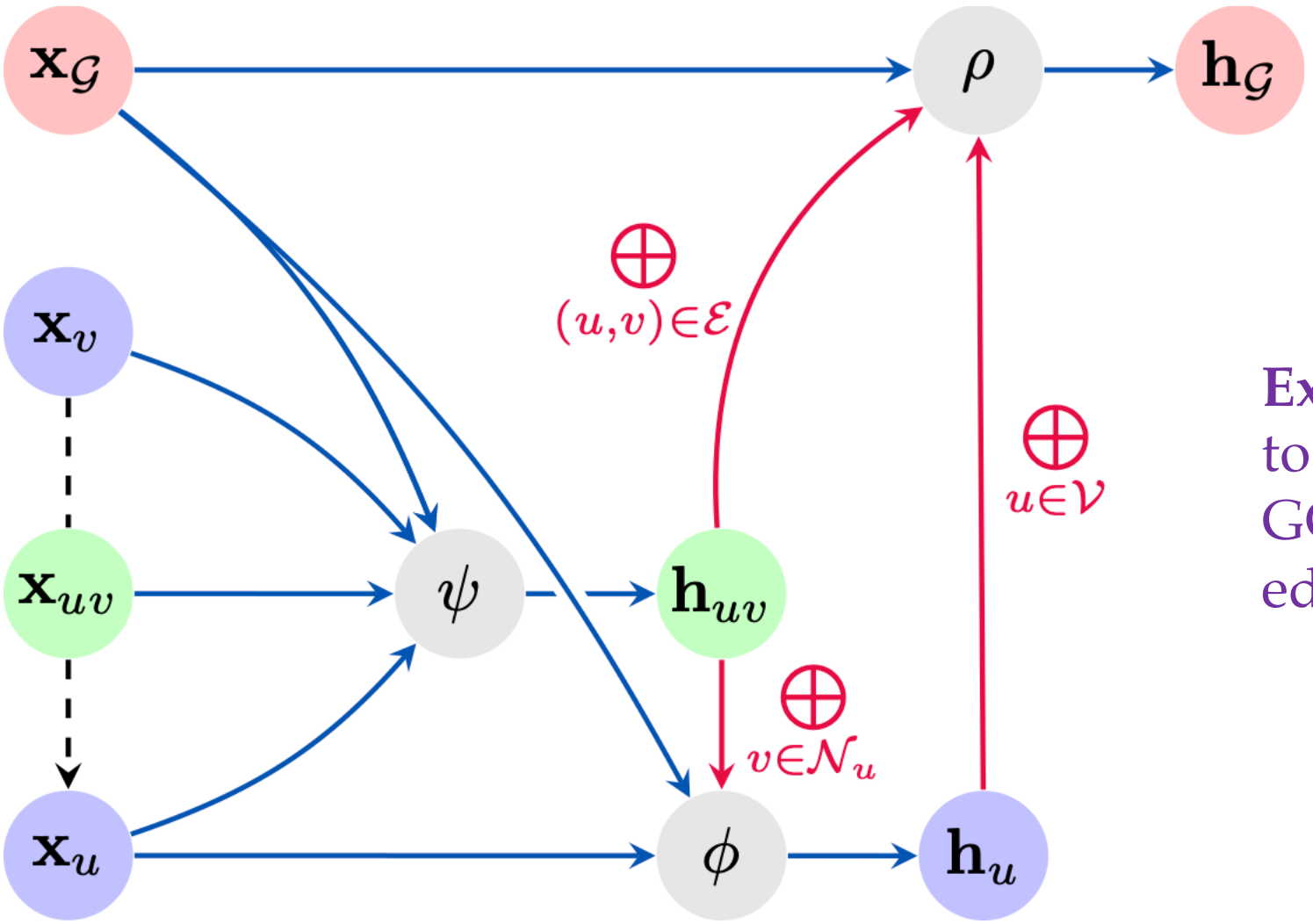
$$\mathbf{h}_{\mathcal{G}} = \rho\left(\bigoplus_{u\in\mathcal{V}} \mathbf{h}_u, \bigoplus_{(u,v)\in\mathcal{E}} \mathbf{h}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Graph Networks, visualised



Equivariant and **invariant** layers feature extensively in GNs

Graph Networks, visualised



Exercise: Specify functions ψ, ϕ, ρ to obtain standard GNNs (e.g. GCN, GAT, MPNN), assuming edge and graph features are given.

One final food for thought

After introducing Deep Sets, we've been able to rigorously *prove* that *any* set function satisfying permutation invariance must be expressible as a Deep Set model

Is our GNN equally general? Can we represent *any* permutation equivariant function over graphs in one of the three flavours?

We will ponder this question in future lectures, from several angles
It will be a useful topic for the practical as well!

What have we covered?

A deep dive into *graph neural networks* (GNNs), primarily by analysing various strategies for building *message functions*:

- *Convolutional* GNNs: GCN, SGC, Chebyshev Networks
- *Attentional* GNNs: MoNet, GAT, GATv2
- *Message-passing* GNNs: IN, MPNN, Graph Networks

The *Graph Network* architecture over generic attributed graphs

Applications: social networks, physics, computational chemistry

What's next?

Practical *(to be released on Moodle ~Wednesday!)*
graph manipulation, geometric GNNs, open-ended paper review
Prepared by Miruna Crețu, Iulia Duță,
Rishabh Jain, Chaitanya Joshi and Dr Paul Scherer

Deep Dives 1 next Monday
featuring Charlie Harris, Chaitanya Joshi and Dr Dobrik Georgiev

Afterwards, we will explore the other key "moving parts" in GNNs
This will allow us to *connect* GNNs to many other important areas of computer science, e.g.: *NLP*, *signal processing* and *classical algorithms*

(If you haven't reached out to your project advisor(s), **please do so**!)