DeepMind

Everything is Connected

Graph Neural Networks from the Ground Up

Petar Veličković

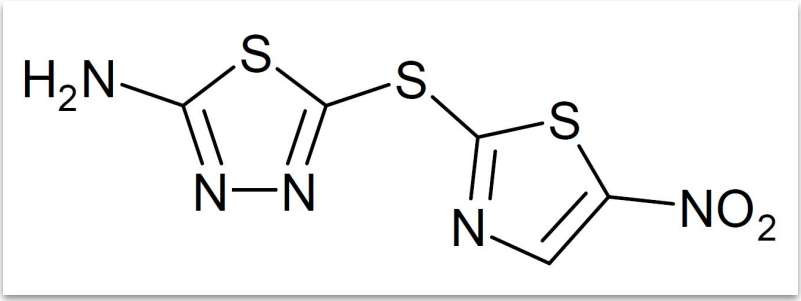
Eastern European Machine Learning Summer School (EEML)
12 July 2021

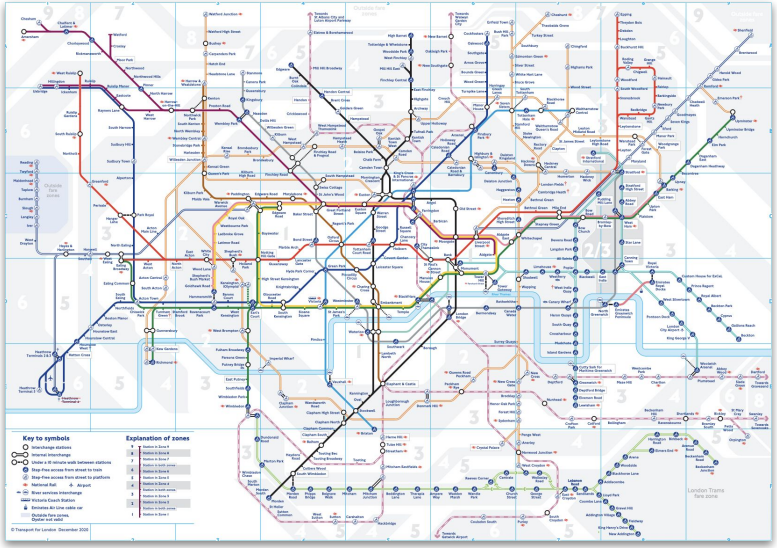
DeepMind

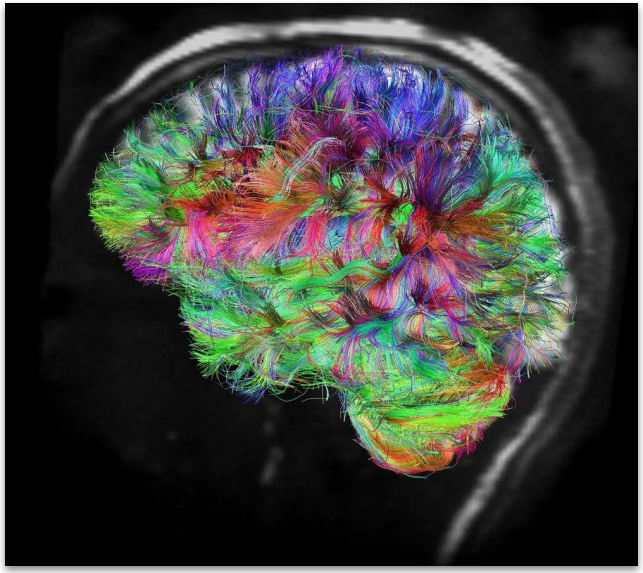
In this talk:
Neural networks for **graph-structured data**

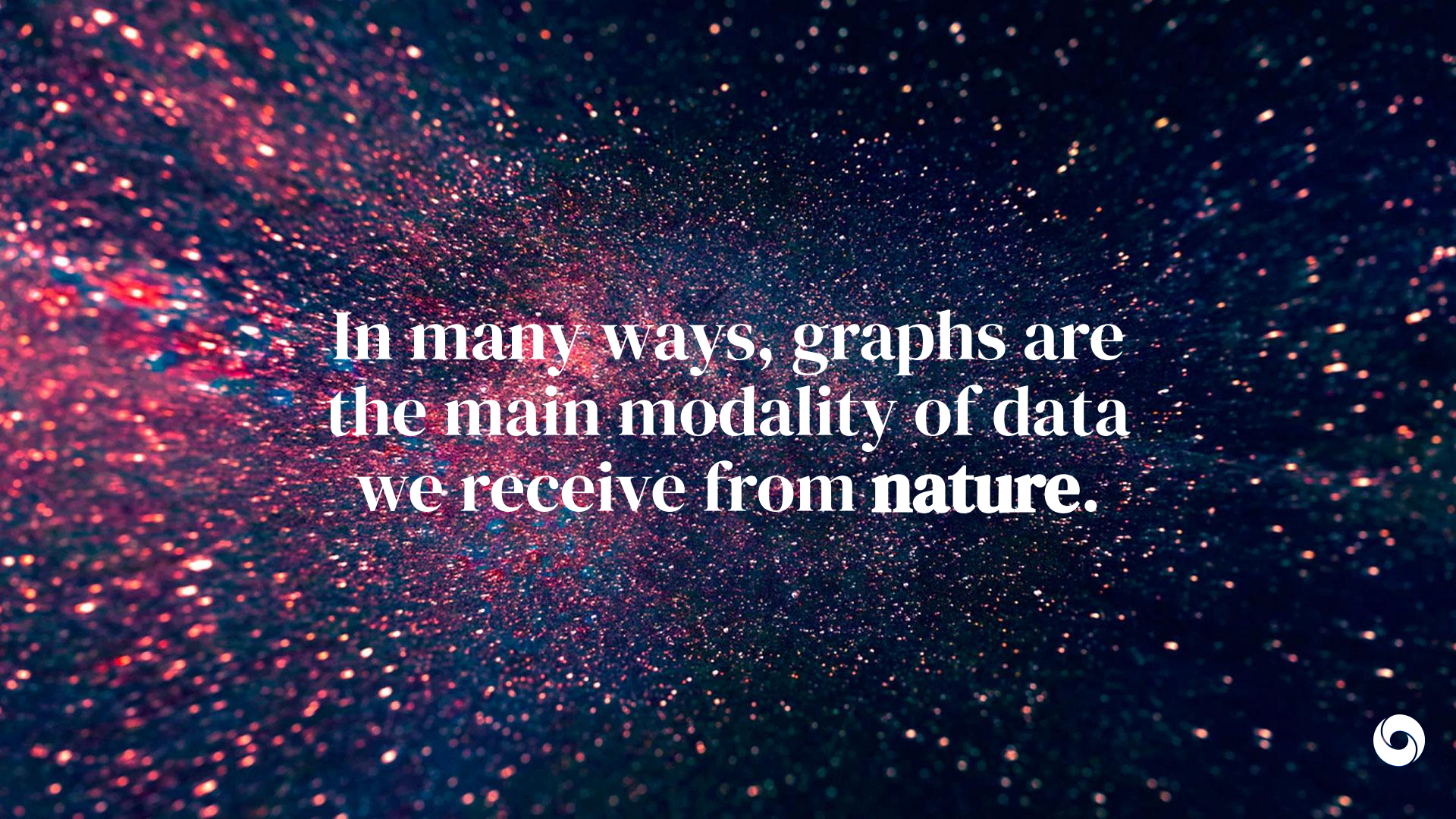
*(Graph Neural Networks; **GNNs**)*

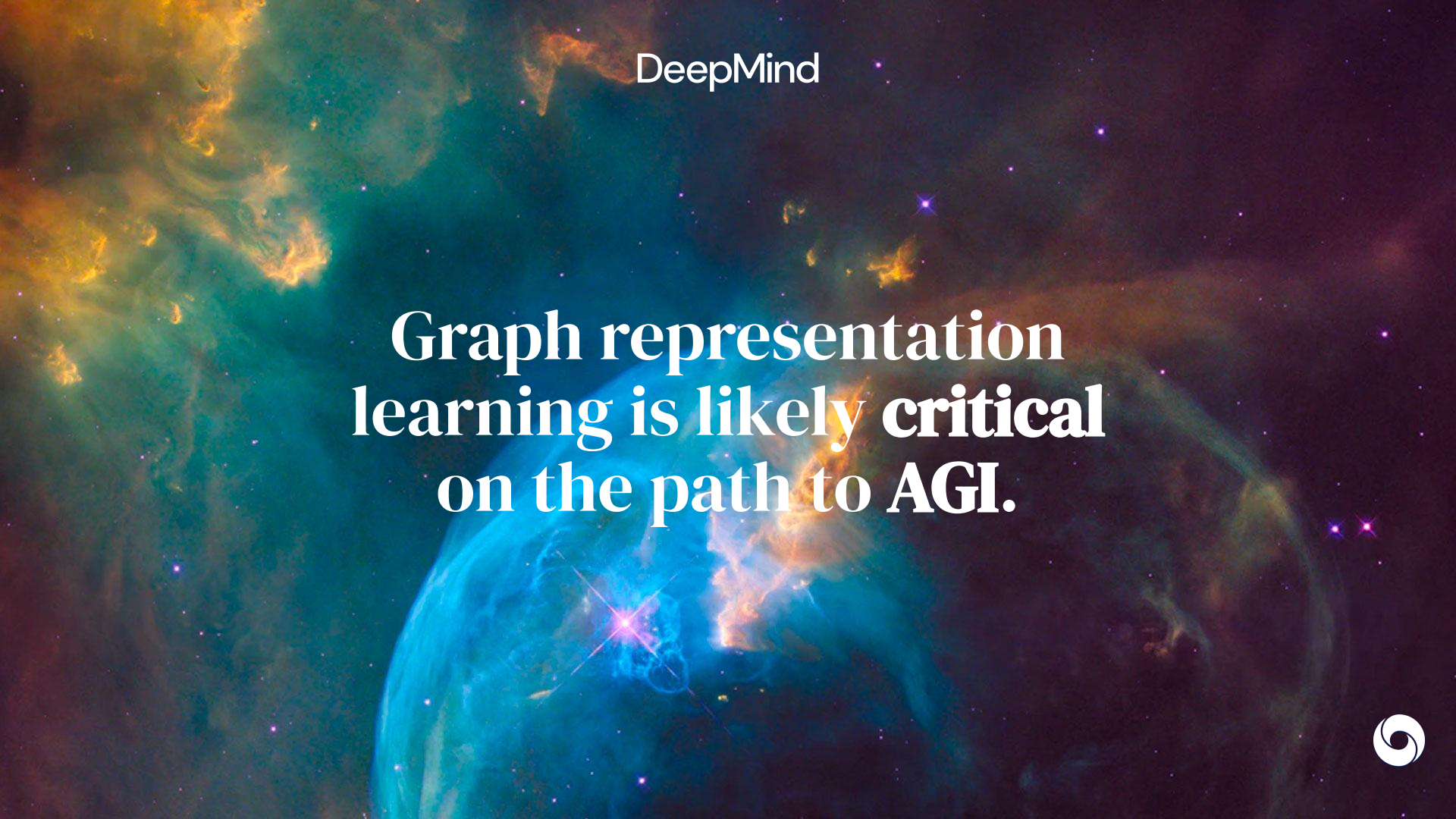
Graphs are **everywhere!**



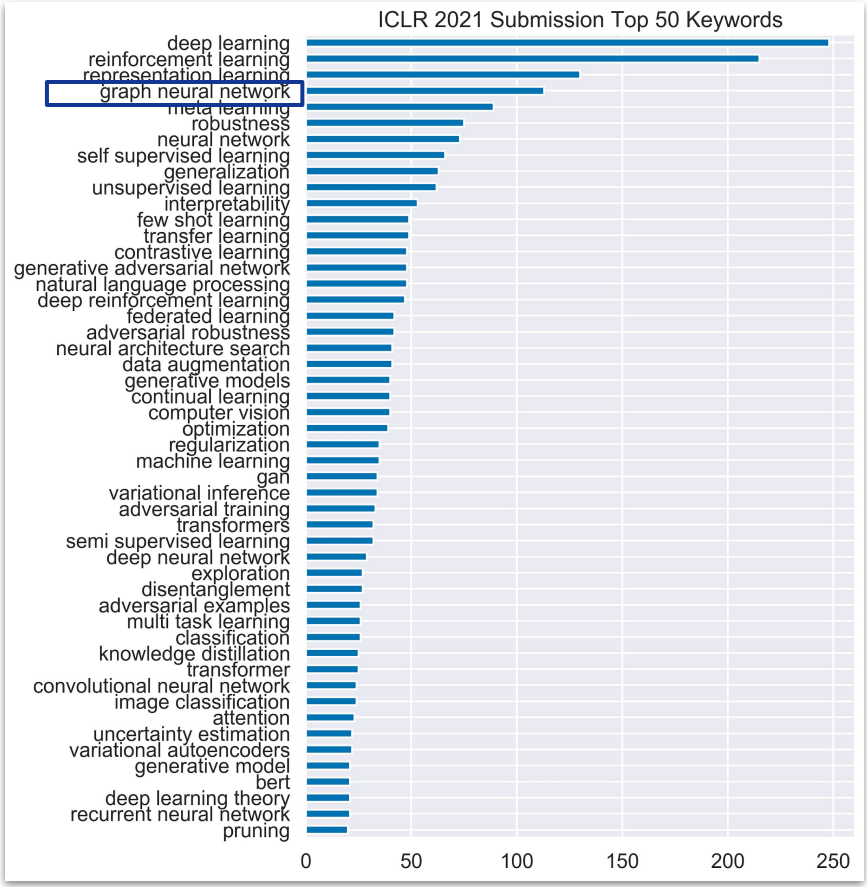


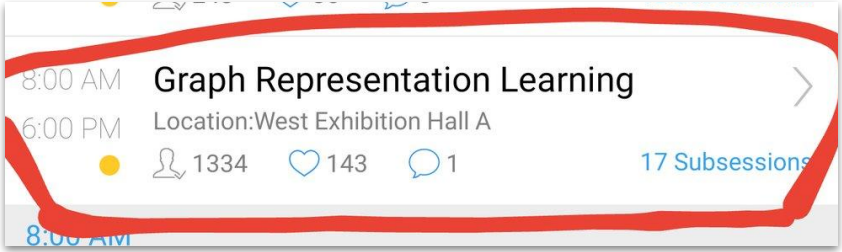


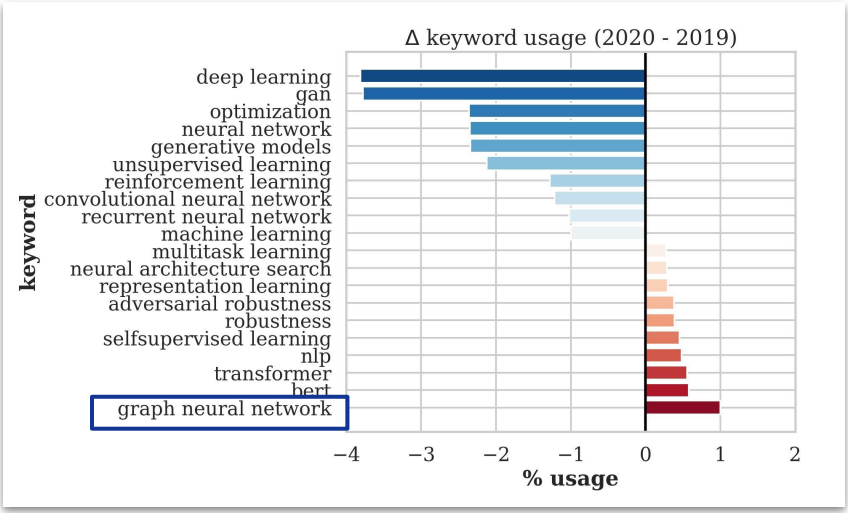

In many ways, graphs are
the main modality of data
we receive from nature.


DeepMind
Graph representation learning is likely critical on the path to AGI.

A **very hot** research topic

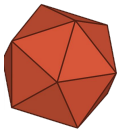






GRL is currently experiencing its "ImageNet" moment

Rich ecosystem of libraries



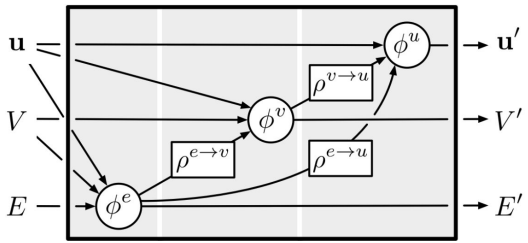
PyTorch geometric

github.com/rusty1s/pytorch_geometric

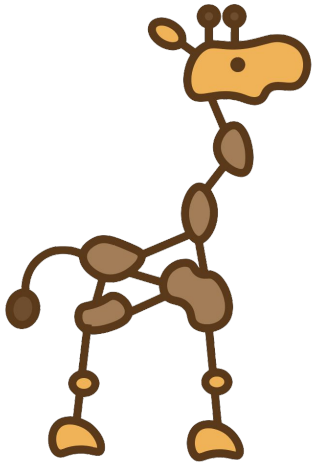
Spektral

graphneural.network

dgl.ai



github.com/deepmind/graph_nets



github.com/deepmind/jraph

Rich ecosystem of datasets

ogb.stanford.edu

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

graphlearning.io

Benchmarking Graph Neural Networks

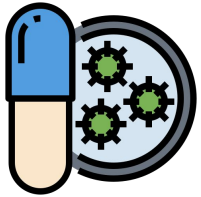
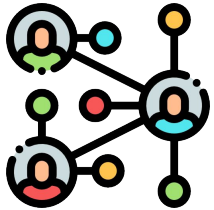
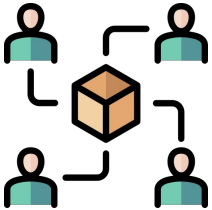
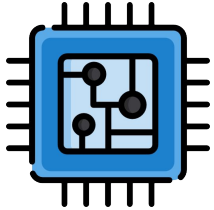
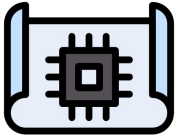
github.com/graphdeeplearning/benchmarking-gnns

DeepMind

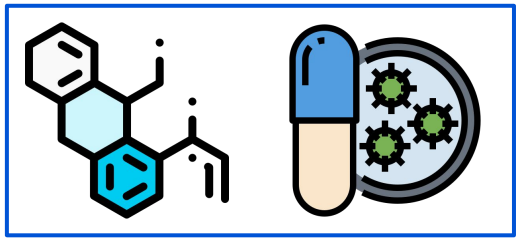
1 Fantastic GNNs in the Wild

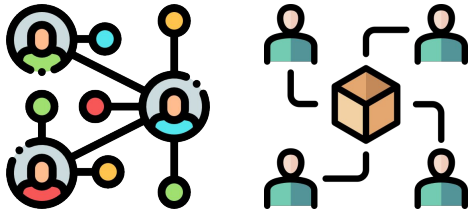
(Compressed & updated talk from EEML 2020)

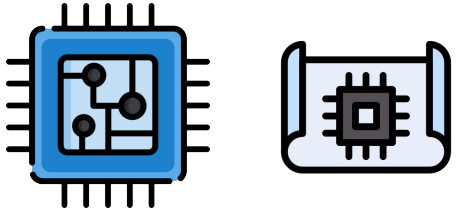
Impactful applications in **science** and **industry**

Impactful applications in **science** and **industry**



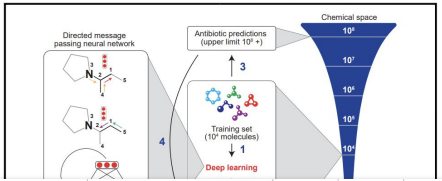




Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract



Authors

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

Corresponde

regina@csail.m
jimjc@mit.edu

In Brief

BBC Sign in News Sport Reel Worklife

NEWS

Home Video World UK Business Tech Science Stories

BBC WORKLIFE Our new guide for getting ahea

Scientists discover powerful antibiotic using AI

21 February 2020 Share

FINANCIAL TIMES

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

l intelligence

Robotics

'Death of the office' homeworking claims exaggerated

Anti-social robots harr increase social distanci

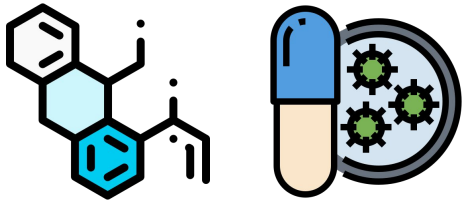
Artificial intelligence + Add to myFT

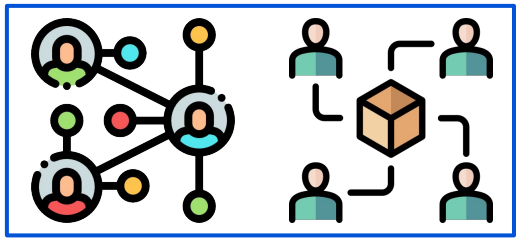
AI discovers antibiotics to treat drug-resistant diseases

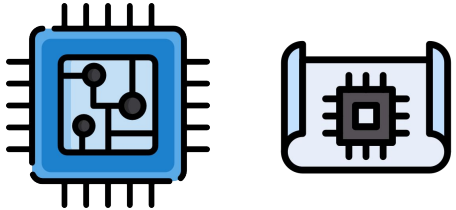
Machine learning uncovers potent new drug able to kill 35 powerful bacteria

Virtual drug screening

Impactful applications in **science** and **industry**







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

Pinterest Engineering Follow

Aug 15, 2018 · 8 min read

amazon | science

PUBLICATION

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

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

2020

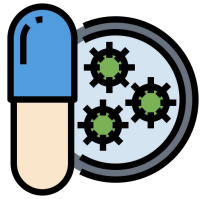
Food Discovery with Uber Eats: Using Graph Learning to Power Recommendations

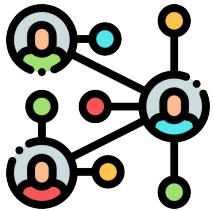
Ankit Jain, Isaac Liu, Ankur Sarda, and Piero Molino December 4, 2019

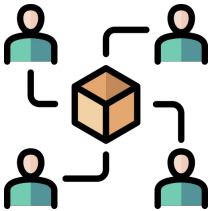
0

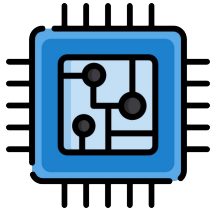
Recommender systems

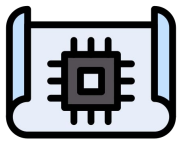
Impactful applications in **science** and **industry**











nature

Explore content | Journal information | Publish with us | Subscribe

nature > articles > article

Article | Published: 09 June 2021

A graph placement methodology for fast chip design

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

GOOGLE | TECH | ARTIFICIAL INTELLIGENCE

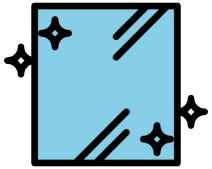
Google is using AI to design its next generation of AI chips more quickly than humans can

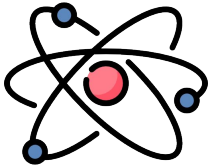
Designs that take humans months can be matched or beaten by AI in six hours

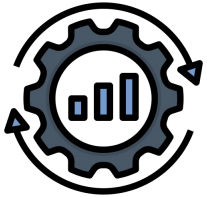
By James Vincent | Jun 10, 2021, 9:13am EDT

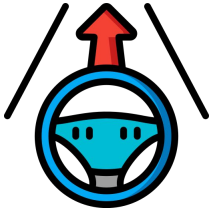
Chip design (TPUv5)

Impactful applications **from DeepMind**

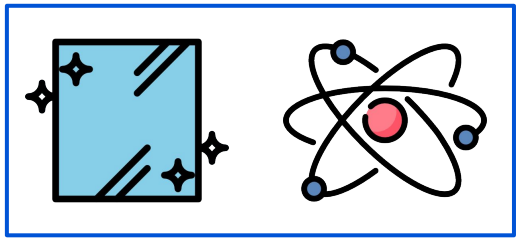


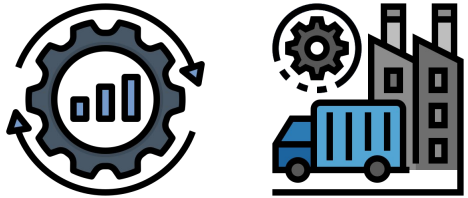


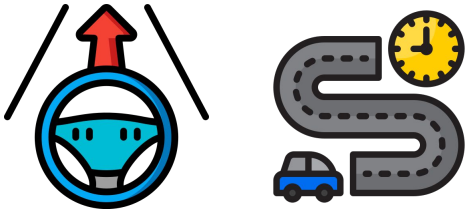


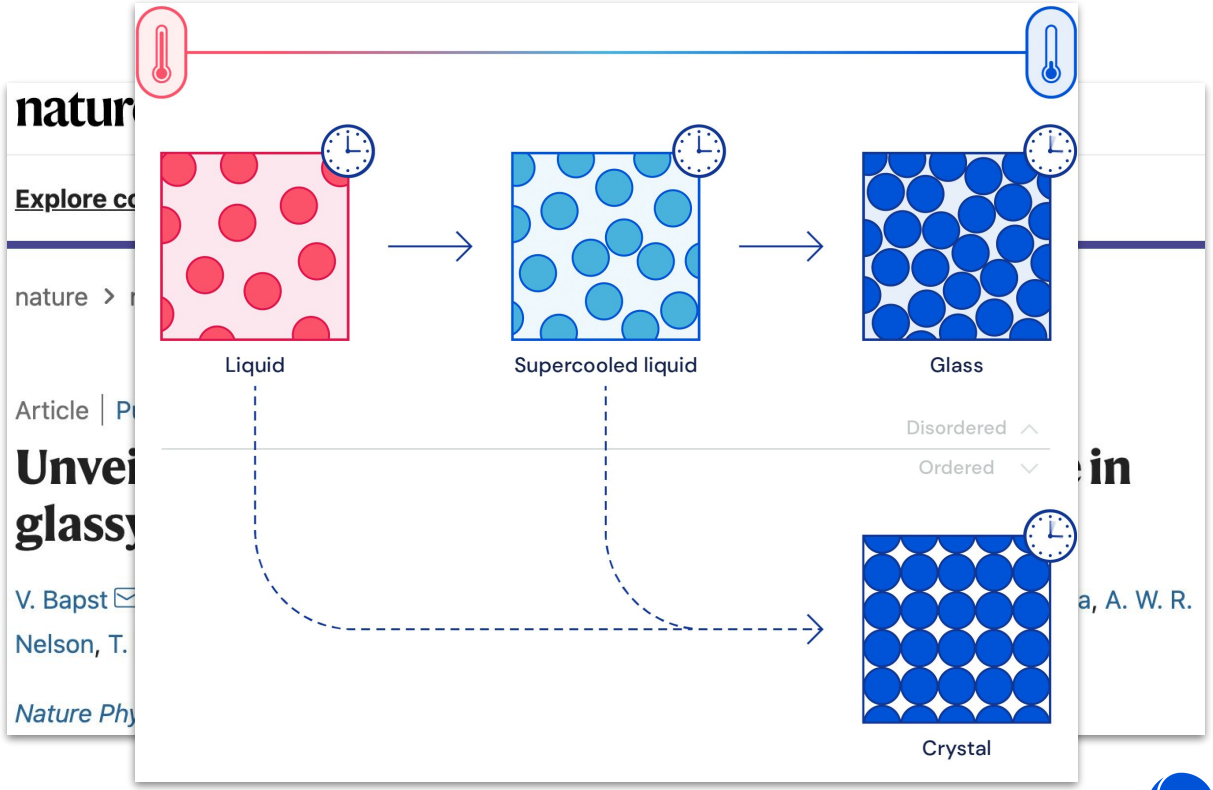


Impactful applications **from DeepMind**



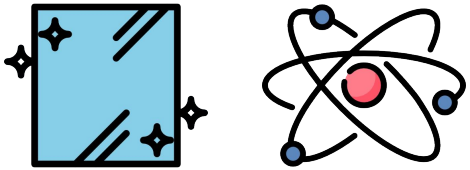


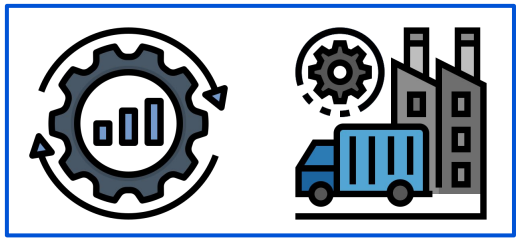


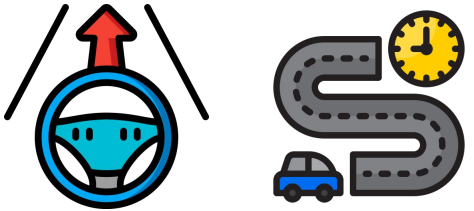


Glassy dynamics

Impactful applications **from DeepMind**

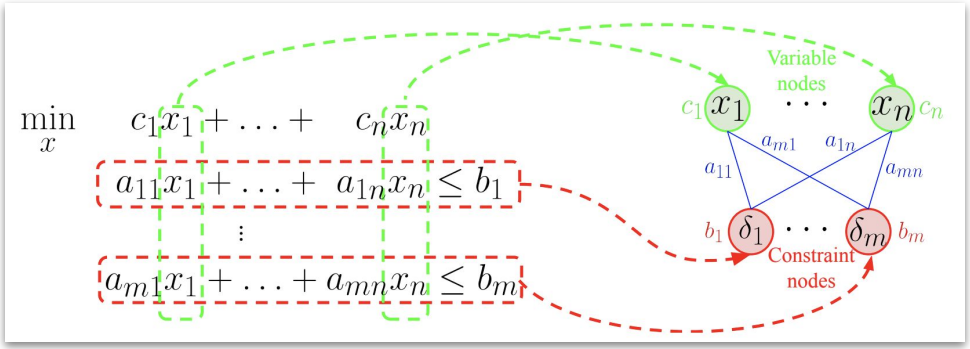






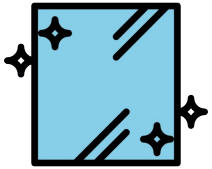
Solving Mixed Integer Programs Using Neural Networks

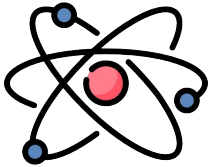
Vinod Nair*†[1], Sergey Bartunov*[1], Felix Gimeno*[1], Ingrid von Glehn*[1], Pawel Lichocki*[2], Ivan Lobov*[1], Brendan O'Donoghue*[1], Nicolas Sonnerat*[1], Christian Tjandraatmadja*[2], Pengming Wang*[1], Ravichandra Addanki[1], Tharindi Hapuarachchi[1], Thomas Keck[1], James Keeling[1], Pushmeet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

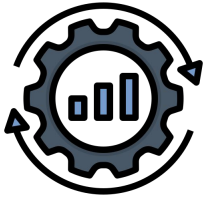


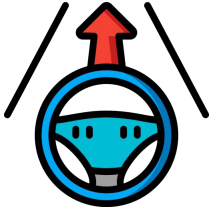
Combinatorial optimisation

Impactful applications **from DeepMind**



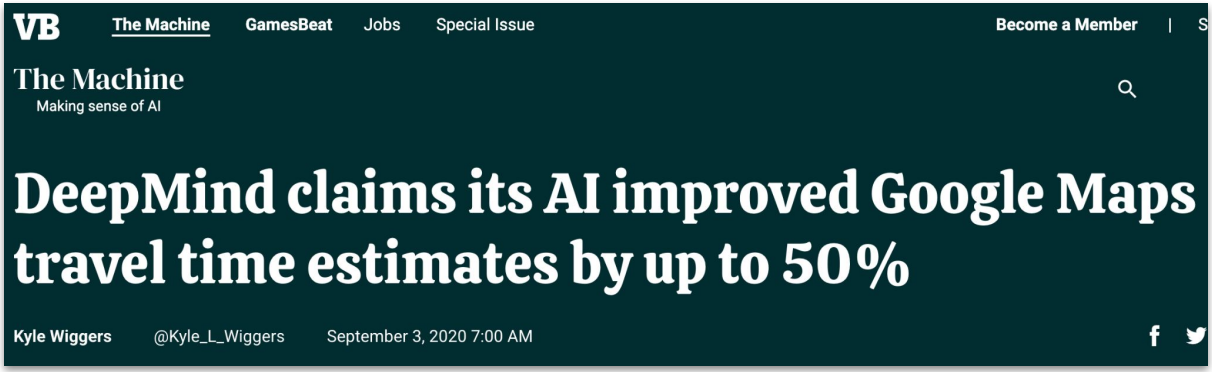






ETA Prediction with Graph Neural Networks in Google Maps

Austin Derrow-Pinion[1], Jennifer She[1], David Wong[2*], Oliver Lange[3], Todd Hester[4*], Luis Perez[5*], Marc Nunkesser[3], Seongjae Lee[3], Xueying Guo[3], Brett Wiltshire[1], Peter W. Battaglia[1], Vishal Gupta[1], Ang Li[1], Zhongwen Xu[6*], Alvaro Sanchez-Gonzalez[1], Yujia Li[1] and Petar Veličković[1]
[1]DeepMind [2]Waymo [3]Google [4]Amazon [5]Facebook AI [6]Sea AI Lab *work done while at DeepMind
{derrowap,jenshe,wongda,petarv}@google.com



Travel-time Prediction in Google Maps

DeepMind

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 aims for today: **empower** you to make immediate **contributions** to GRL
  - Followed by Nikola's Colab tutorial to facilitate implementation!

- **Derive** GNNs from **first principles** (permutation invariance and equivariance)
  - Similar in spirit to my previous talk at Cambridge

- **Categorise** existing GNNs into three spatial *"flavours"*

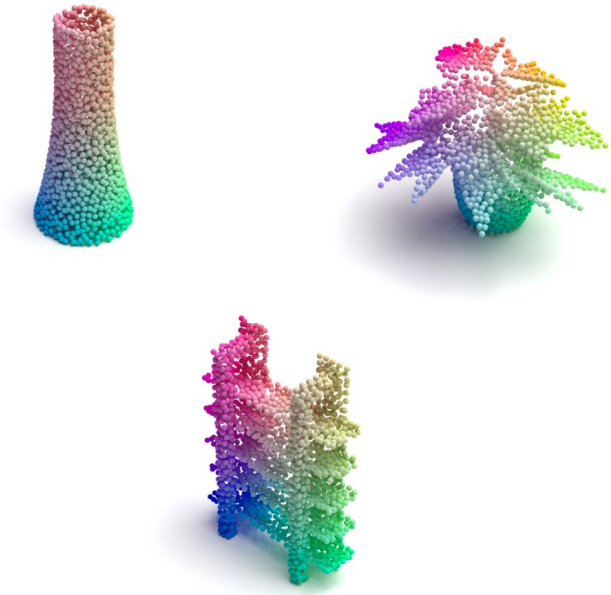
- **Present** interesting open problems at the *"bleeding edge"*

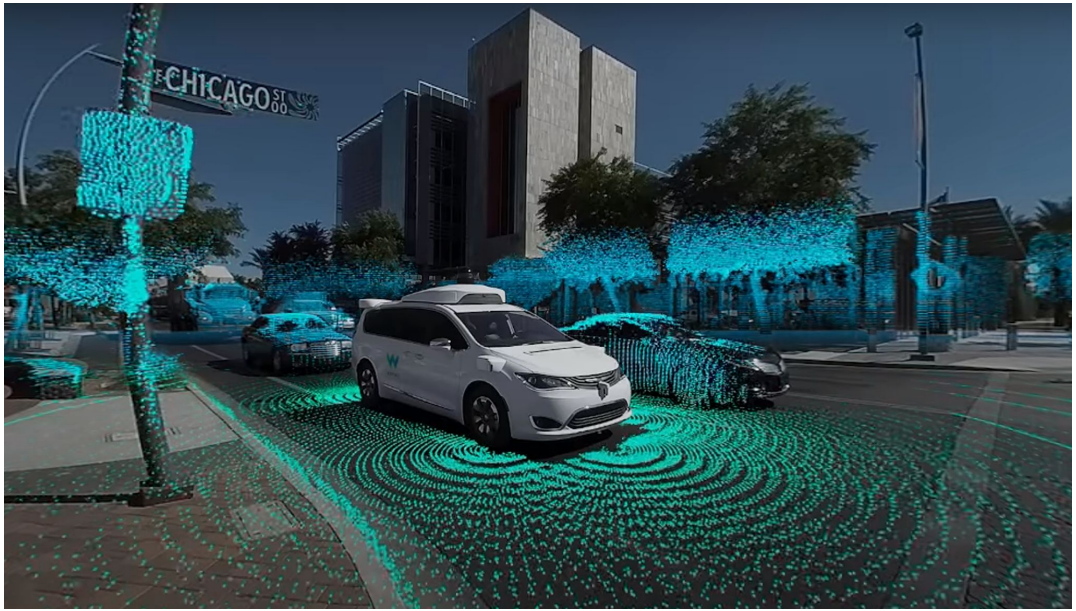
Agenda

- Talk by me on GNN foundations (this talk!)
- Lab session by Nikola Jovanović (right after this talk!)
- Mentoring session on GNNs with me (right after the lab! :))
- Ask questions in **#graph-ml-geometric-dl** at **any** point!
  - We'll do our best to get around to them, either online or offline :)
- Have fun with graphs!

Where do we begin?

- We will first look at **graphs without connections** (**sets**)
  - Much *simpler* to analyse
  - Most conclusions will naturally *carry over* to graphs
  - Still very *relevant*! (point clouds / LiDAR)





DeepMind

3 Permutation invariance and equivariance

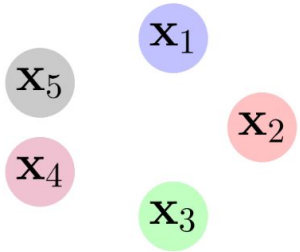
Learning on sets: Setup

- 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* x *k*:

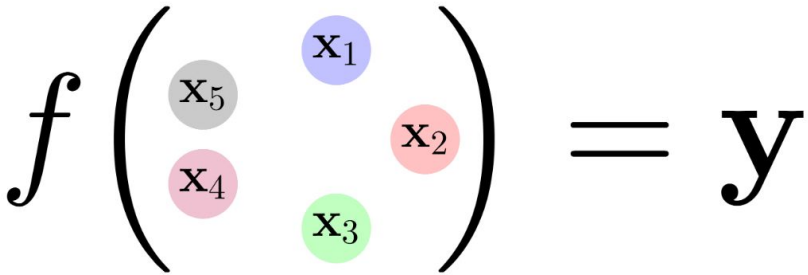
$$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$$

- That is, the *i*th row of **X** corresponds to $\mathbf{x}_i$
- Note that, by doing so, we have specified a **node ordering**!
  - We would like the result of any neural networks to not depend on this.

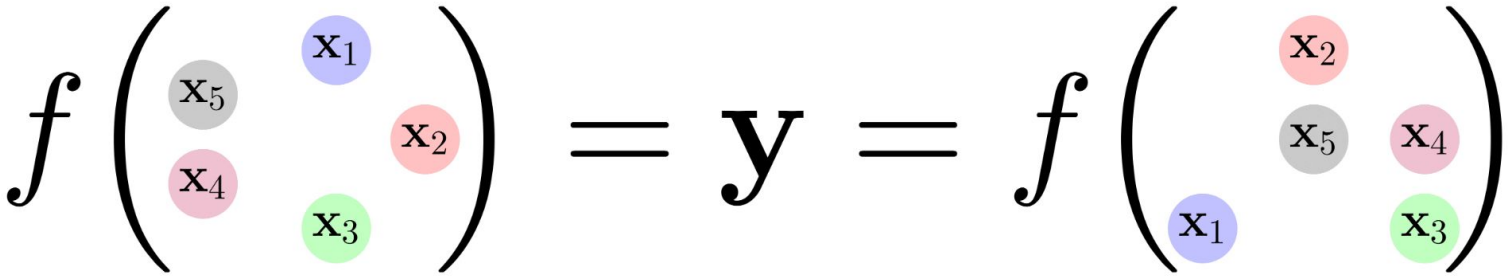
What do we want?



What do we want?



What do we want?



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

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

Permutation *invariance*

- We want to design functions f(**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(**X**) is permutation *invariant* if, for *all* permutation matrices **P**:

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

- One very generic form is the *Deep Sets* model (Zaheer et al., NeurIPS'17): $f(\mathbf{X}) = \phi\left(\sum_{i \in \mathcal{V}} \psi\left(\mathbf{x}_i\right)\right)$ 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!

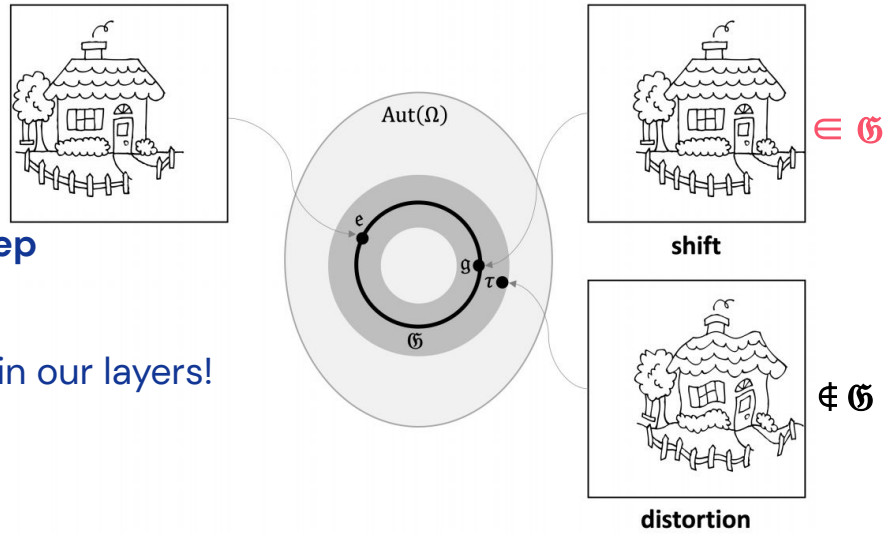
- Accordingly, we say that f(**X**) is **permutation equivariant** if, for all permutation matrices **P**:

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

Important constraint: Locality

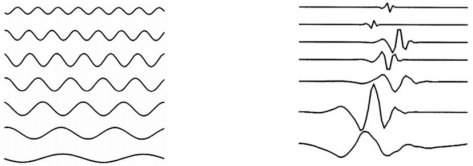
- Want signal to be **stable** under slight *deformations* of the domain

- Highly beneficial to compose **local** operations to model larger-scale ones
  - local ops won't globally propagate errors
  - e.g. CNNs with 3 x 3 kernels, but **very deep**



- Accordingly, we would like to support ***locality*** in our layers!

- cf. Fourier Transform vs. Wavelets



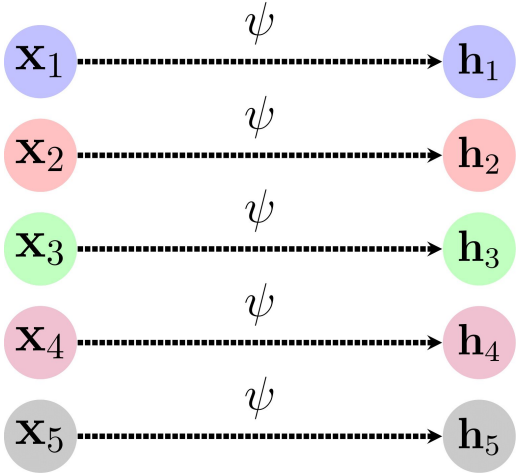
What does this mean for sets?

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 $\mathbf{h}_i$:

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

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



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

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 $\mathbf{h}_i$:

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

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!

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

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)

DeepMind

4 Learning on graphs

Learning on graphs

- Now we augment the set of nodes with **edges** between them.
  - That is, we consider general E ⊆ V x V.

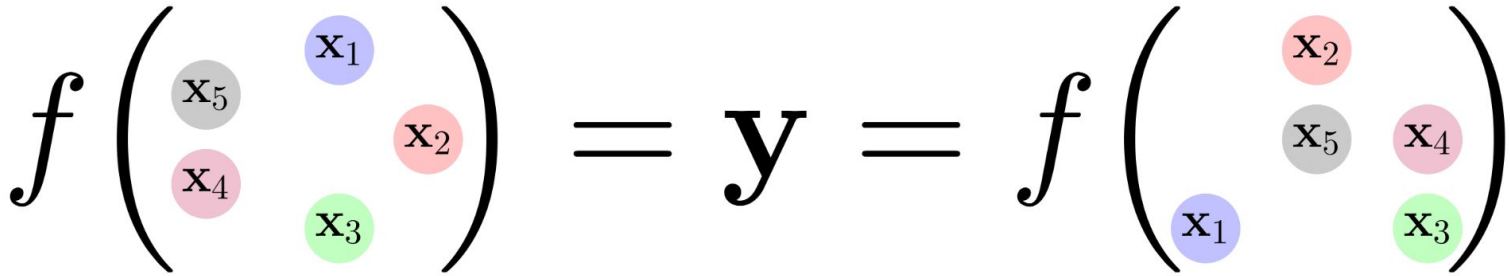
- We can represent these edges with an **adjacency matrix**, **A**, such that:

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

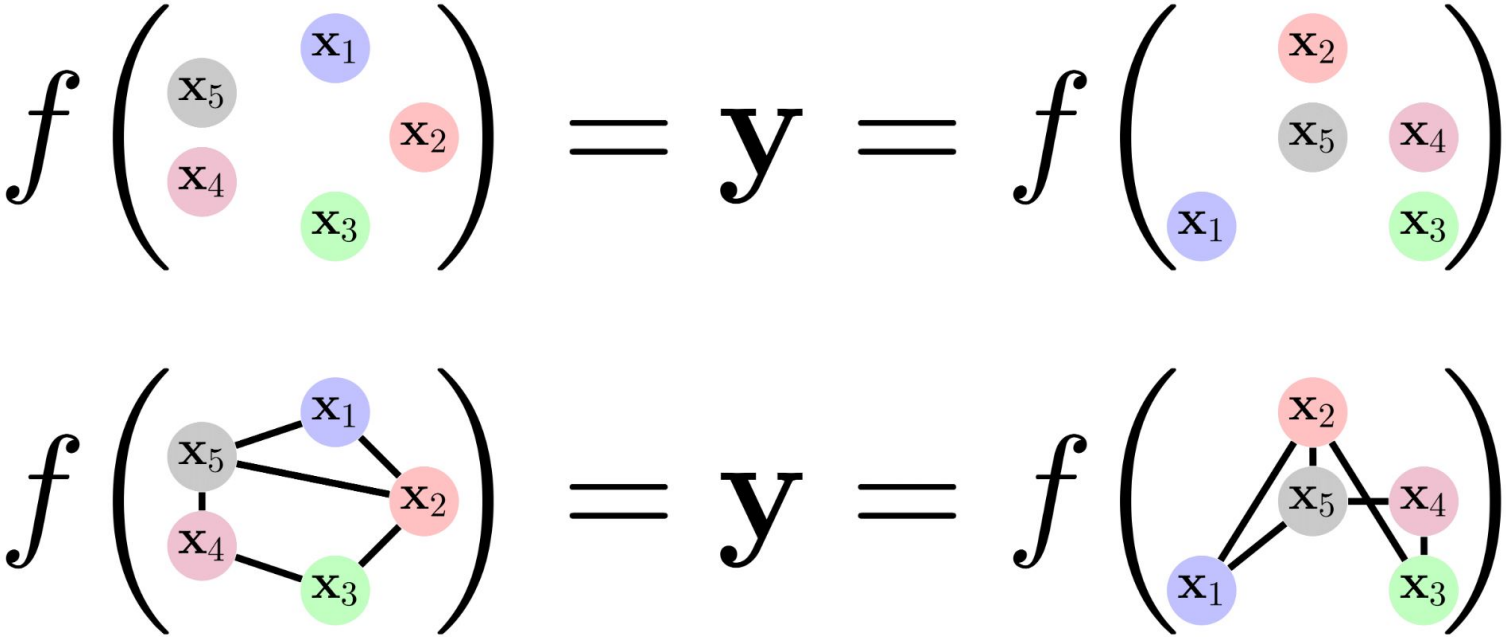
- Further additions (e.g. *edge features*) are possible but **ignored** for simplicity.

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

What's changed?



What's changed?



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 **A**
  - When applying a permutation matrix **P**, this amounts to $\mathbf{PAP}^\intercal$
- We arrive at updated definitions of suitable functions f(**X, A**) over graphs:

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

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

Locality on graphs: **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 $i \in N_i$

- 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: $g(\mathbf{x}_i, \mathbf{X}_{Ni})$.

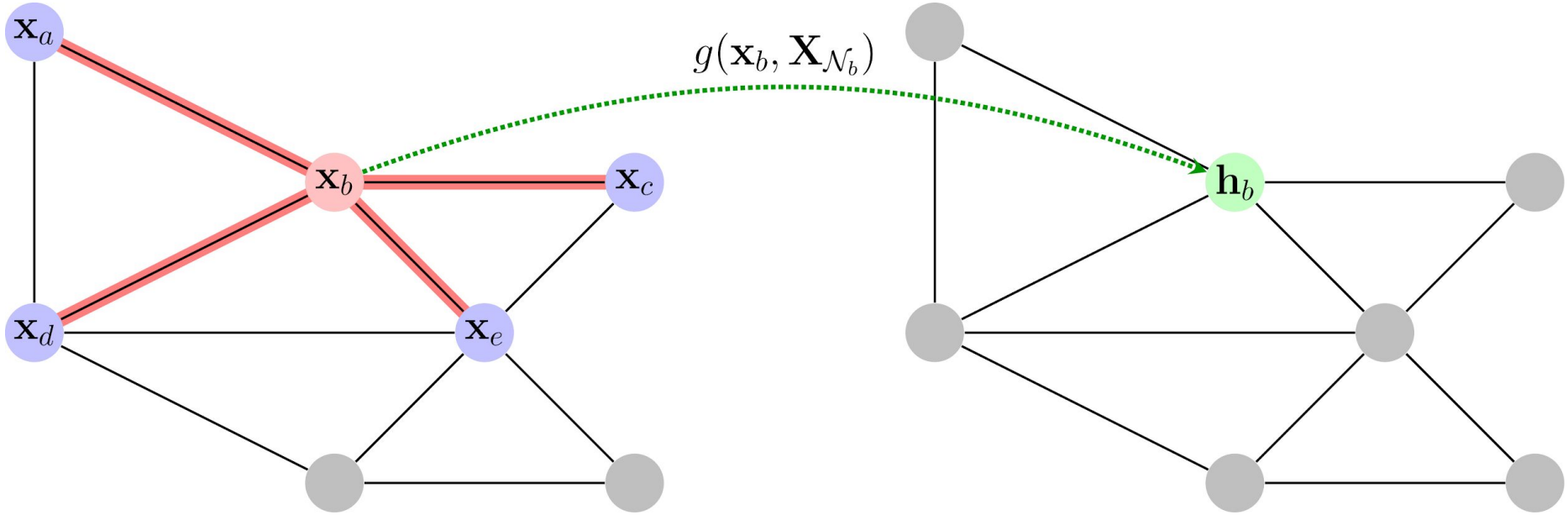
A recipe for **graph** neural networks

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

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

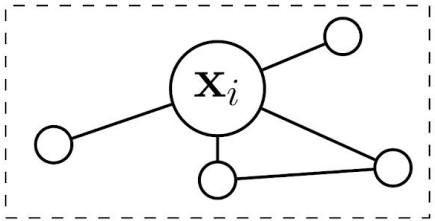
- To ensure equivariance, we need g to not depend on the **order** of the vertices in $\mathbf{X}_{\text{Ni}}$
  - Hence, g should be permutation **invariant**!

A recipe for **graph** neural networks, visualised



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

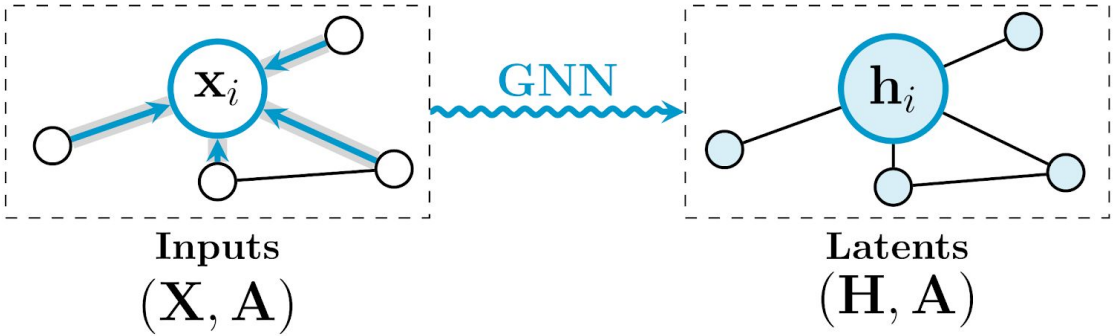
General blueprint for learning on **graphs**



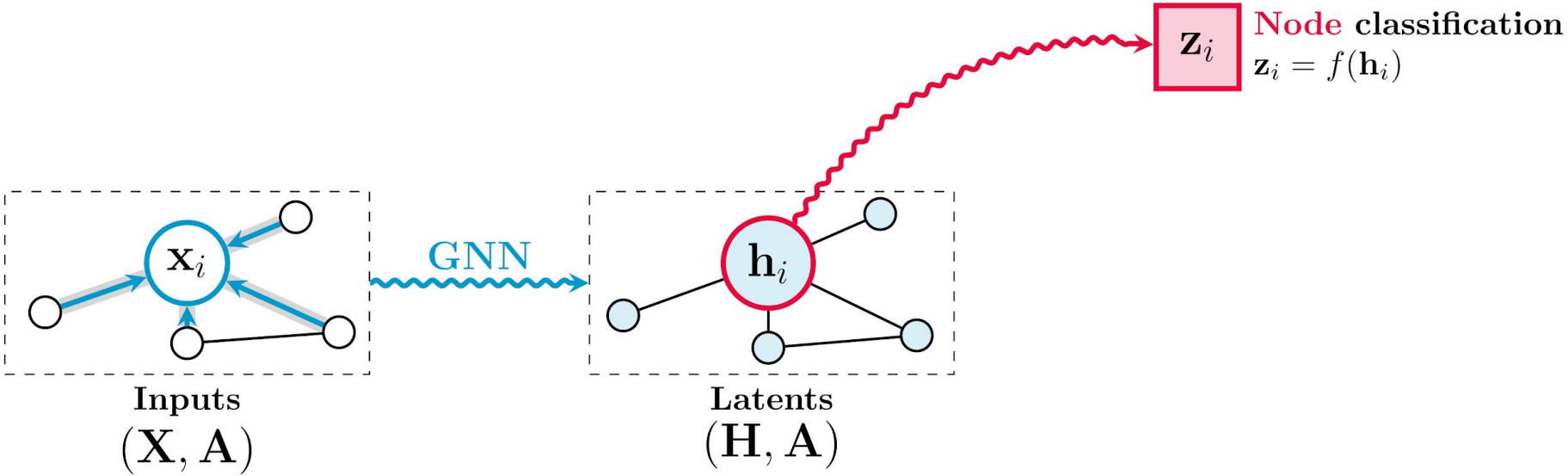
Inputs

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

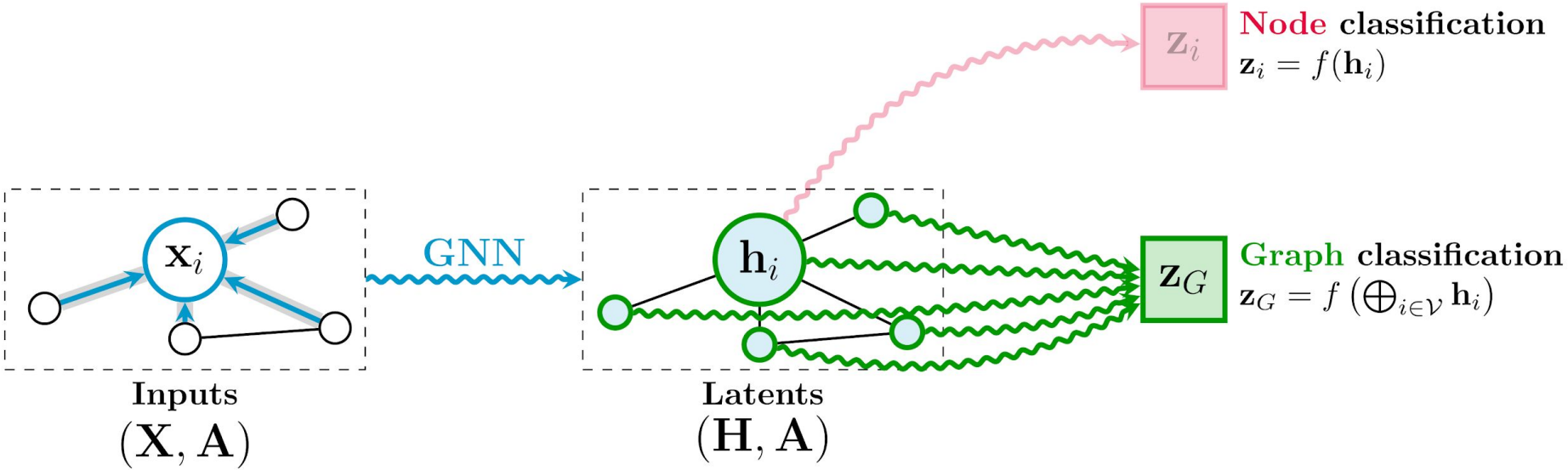
General blueprint for learning on **graphs**



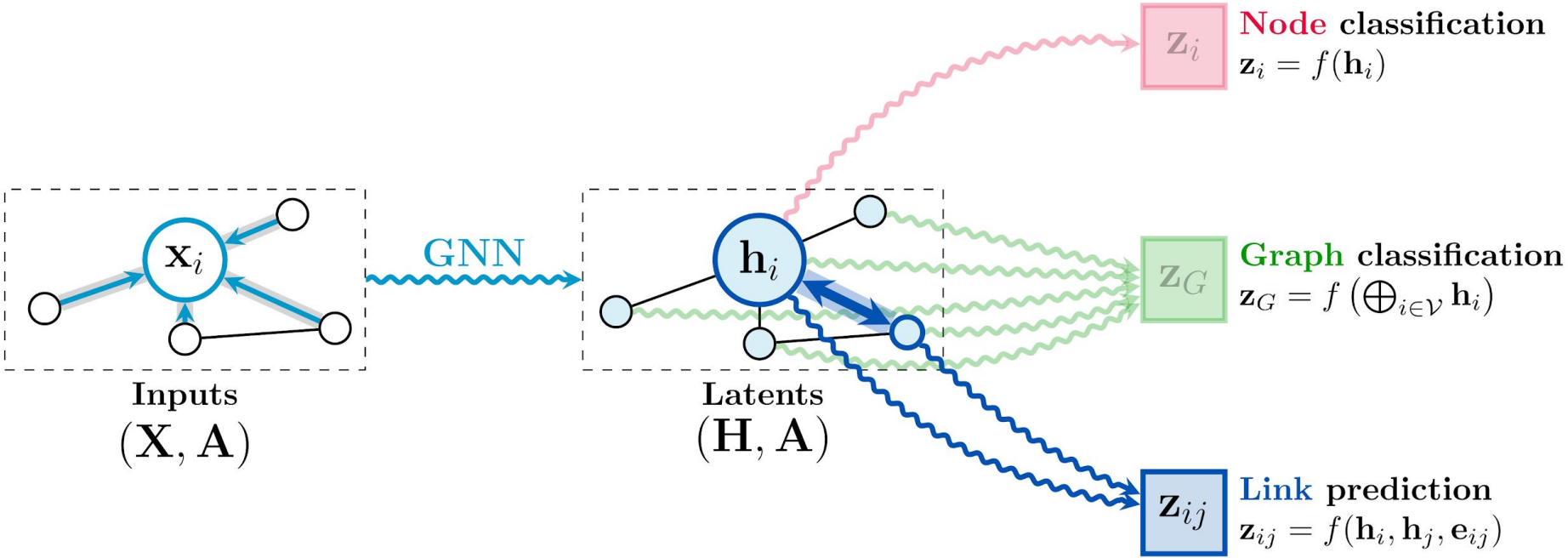
General blueprint for learning on **graphs**



General blueprint for learning on **graphs**



General blueprint for learning on **graphs**



DeepMind

5 Message passing on graphs

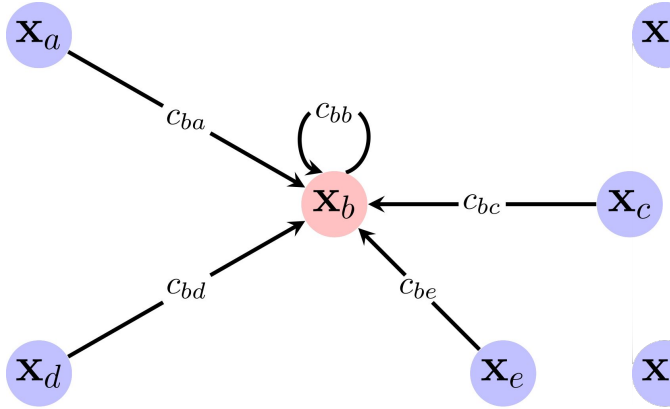
What's in a GNN layer?

- As mentioned, we construct permutation-equivariant functions f(**X**, **A**) over graphs by shared application of a local permutation-invariant g($\mathbf{x}_i$, $\mathbf{X}_{Ni}$).
  - We often refer to f as "GNN layer", g as "diffusion", "propagation", "message passing"

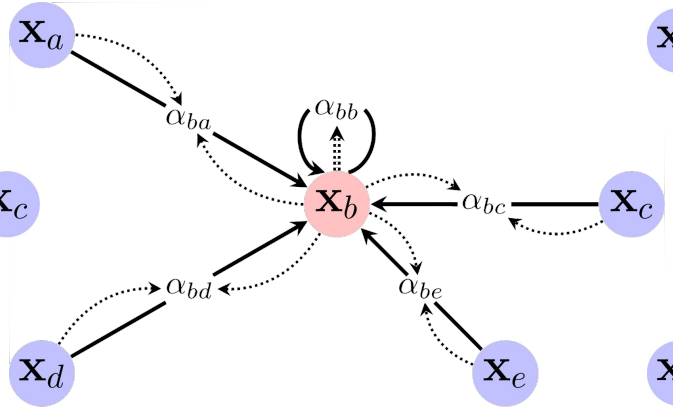
- Now we look at ways in which we can actually concretely **define** g.
  - **Very intense** area of research!

- Fortunately, *almost all* proposed layers can be classified as one of three ***spatial*** "*flavours*".

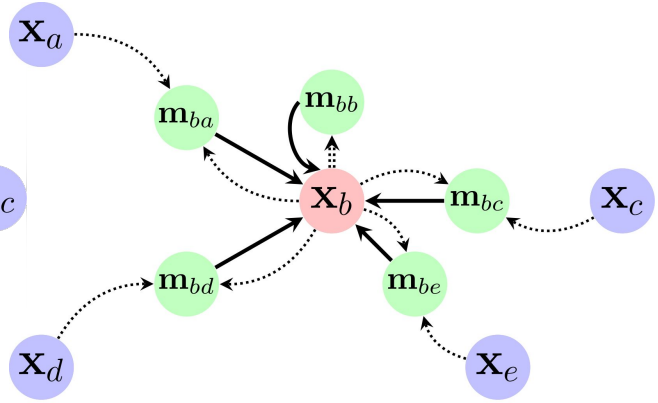
The three “flavours” of GNN layers



Convolutional



Attentional



Message-passing

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

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

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

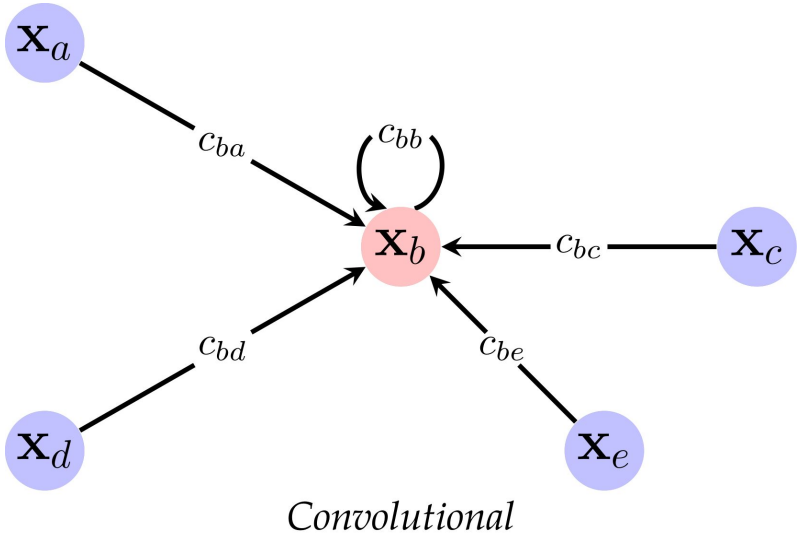
Convolutional GNN

- 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*



Convolutional

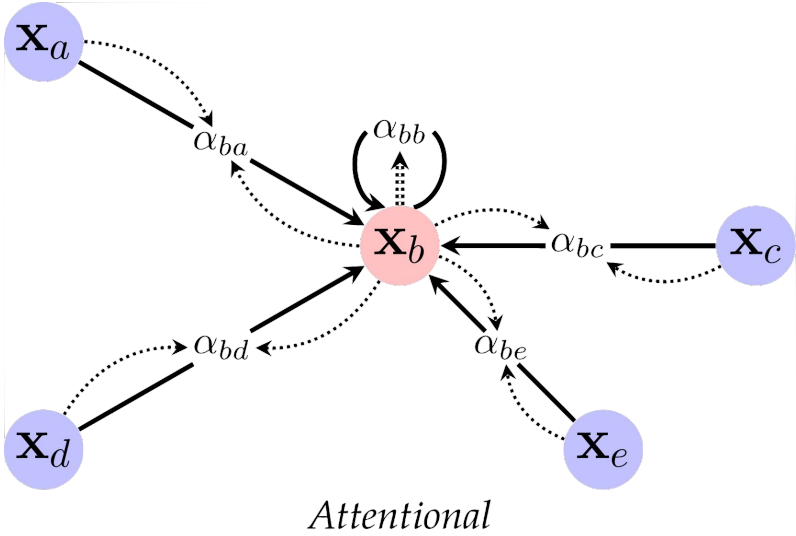
Attentional GNN

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

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

- Attention weight computed as $a_{ij} = a(\mathbf{x}_i, \mathbf{x}_j)$
  - MoNet (Monti *et al.*, CVPR'17)
  - GAT (Veličković *et al.*, ICLR'18)
  - GaAN (Zhang *et al.*, UAI'18)

- Useful as "middle ground" w.r.t. **capacity** and **scale**
  - Edges need not encode homophily
  - But still compute *scalar* value in each edge



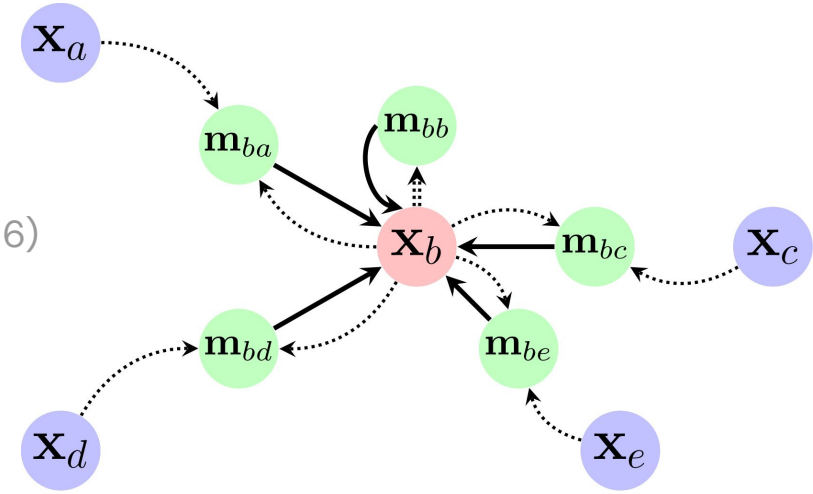
Attentional

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)



Message-passing

- Most **generic** GNN layer
  - May have *scalability* or *learnability* issues
  - Ideal for *computational chemistry*, *reasoning* and *simulation*

DeepMind

6 Perspectives on GNNs

Towards the bleeding edge

- Now I will present and state **three** areas full of *open problems*, which can be easily expressed using the tools we've built up so far
- This can enable you to make immediate contributions to the area.
- If you've read up on graph machine learning before, there's a good chance you will have seen at least some of these.
- Happy to chat about these (or related) open problems at any point :)

DeepMind

I Latent Graph Inference

What to do when there is **no** graph?

- So far, we've assumed something (seemingly) very innocent: that *the **graph** is given to us!*

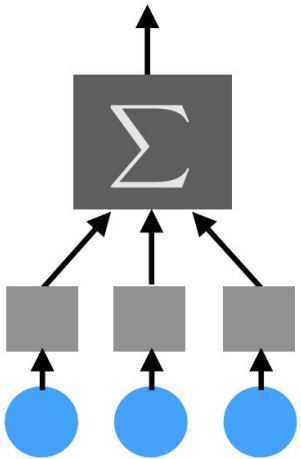
- In practice, the given graph may often be *suboptimal* for the task we're solving
  - For various connectivity querying on graphs, maintaining the right set of edges can make a difference between **linear-time** and (amortised) **constant-time** complexity!

- Taken to the extreme: *what to do when there is **no** graph?*
  - Assume we're given a node feature matrix, but no adjacency
  - We've seen one "solution" for this already...

Option 1: Assume *no* edges

Deep Sets (Zaheer *et al.*, NeurIPS'17)

$$\mathcal{N}_i = \{i\}$$



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

No edges (*set*)

Option 2: Assume *all* edges

$$\mathcal{N}_i = \mathcal{V}$$

Let the GNN decide which edges matter!

Using conv-GNNs no longer makes sense.

If we use **attentional** GNNs we recover:

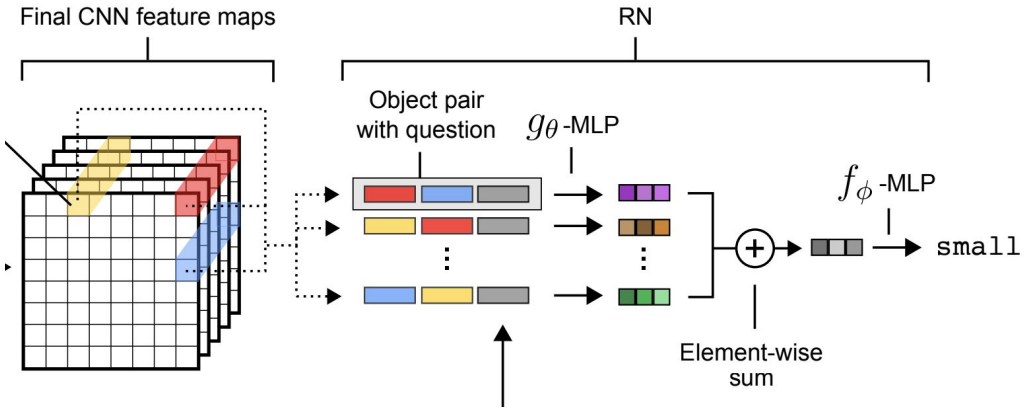
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{V}} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

Does this look *familiar*?

Interaction Nets (Battaglia *et al.*, NeurIPS'16)

Relational Nets (Santoro *et al.*, NeurIPS'17)

Transformers (Vaswani *et al.*, NeurIPS'17)



All edges (*fully-connected graph*)

A note on Transformers

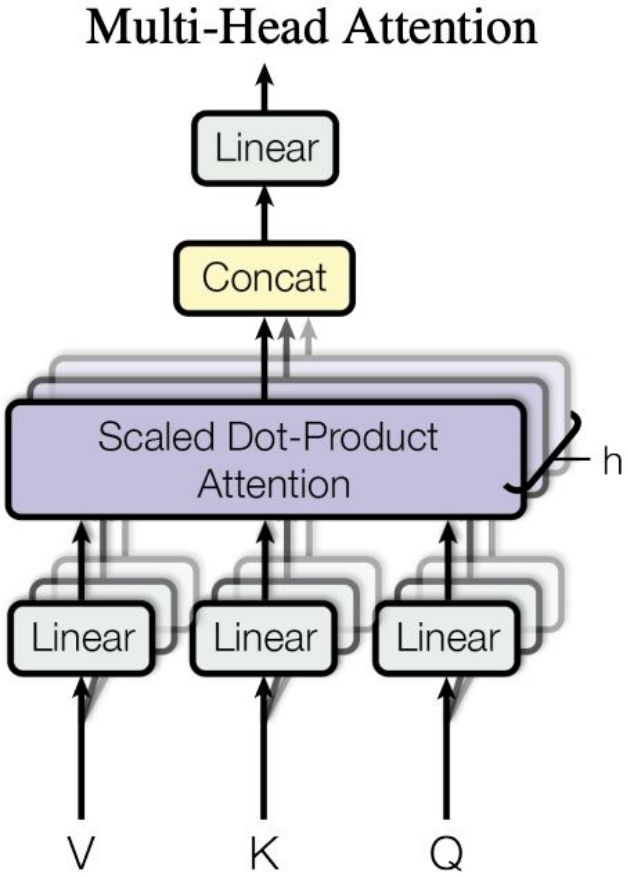
Transformers **are** Graph Neural Networks!

- Fully-connected graph
- Attentional flavour

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

Attention can be seen as inferring **soft adjacency**.

See Joshi (The Gradient; 2020).

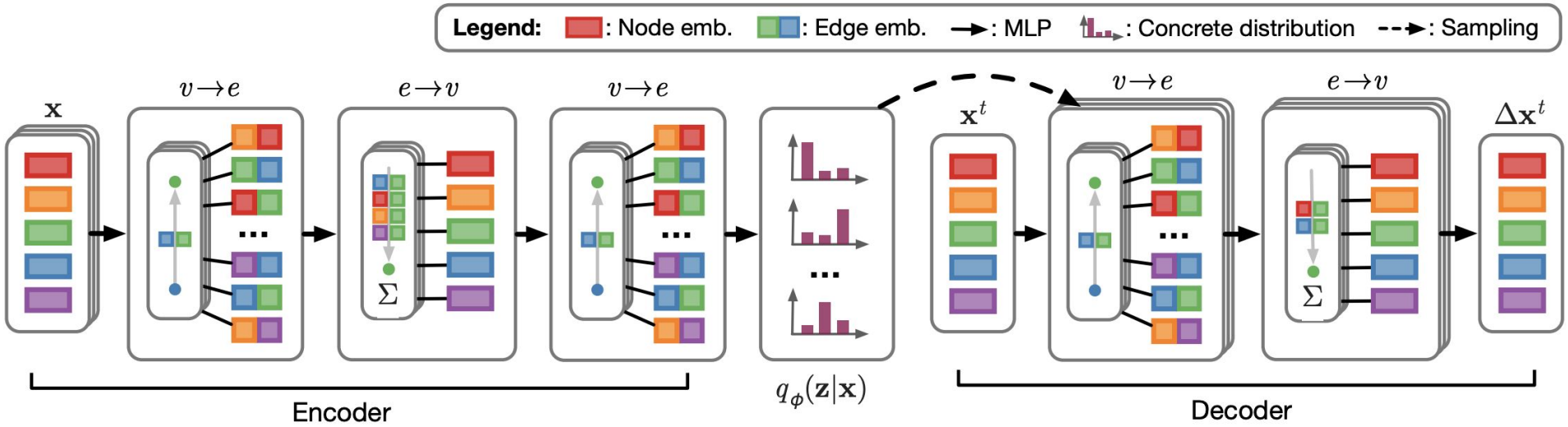


The “truth” likely lies *in between*

- Empty graph ignores a potential **wealth** of information
- Full graph can be hard to scale (*quadratic* complexity, large neighbourhoods)
- Ideally, we want to **infer** the adjacency matrix **A**, then use it as **edges** for a GNN!
- Commonly termed *“latent graph inference”*.
- Choosing edges is a **discrete decision** -- inherently hard to backpropagate!

Option 3a: *Infer* edges to use (*variational*)

Neural Relational Inference (Kipf, Fetaya *et al.*, ICML'18)



Specify **prior** over edges, use encoder to derive **posterior**, sample to infer graph (~VAE setup)

k-NN graphs

- The NRI approach above gives a way to reason about the graph structure probabilistically
  - But it still requires running a fully-connected GNN!

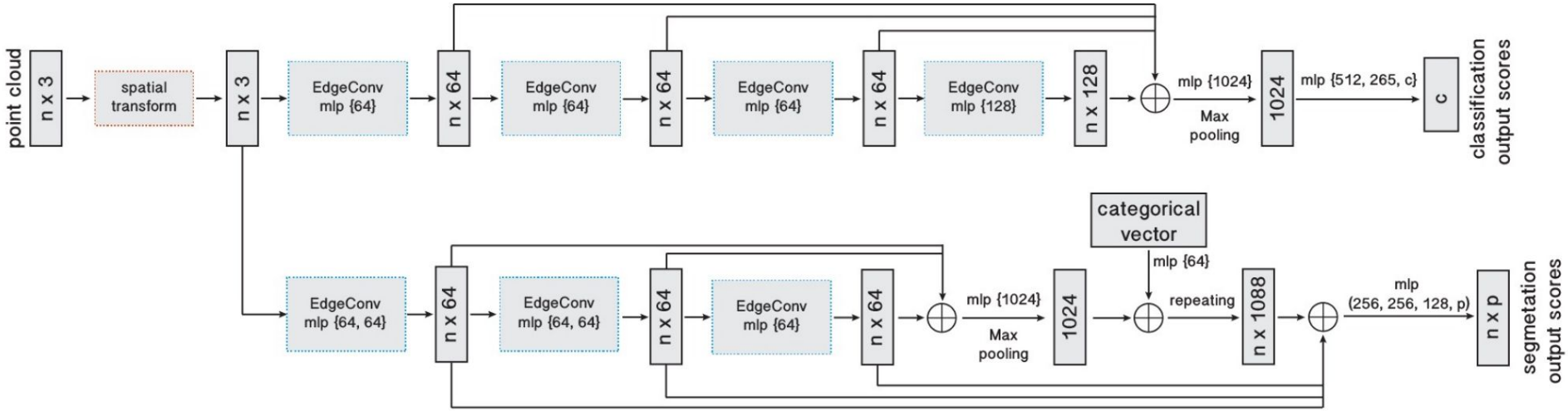
- Ideally, we want to infer a sparse graph **without ever** doing a dense GNN
  - (at least at inference time…)

- The current workhorse of such approaches is the ***k*-nearest neighbour** (*k*-NN) graph
  - Each node stores features $\mathbf{h}_i$
  - Connect it only to its *k* nearest neighbours in **h** (e.g. based on Euclidean distance).

- Open problem: Can we *(should we)* do better?

Option 3b: *Infer* edges to use (*no learning*)

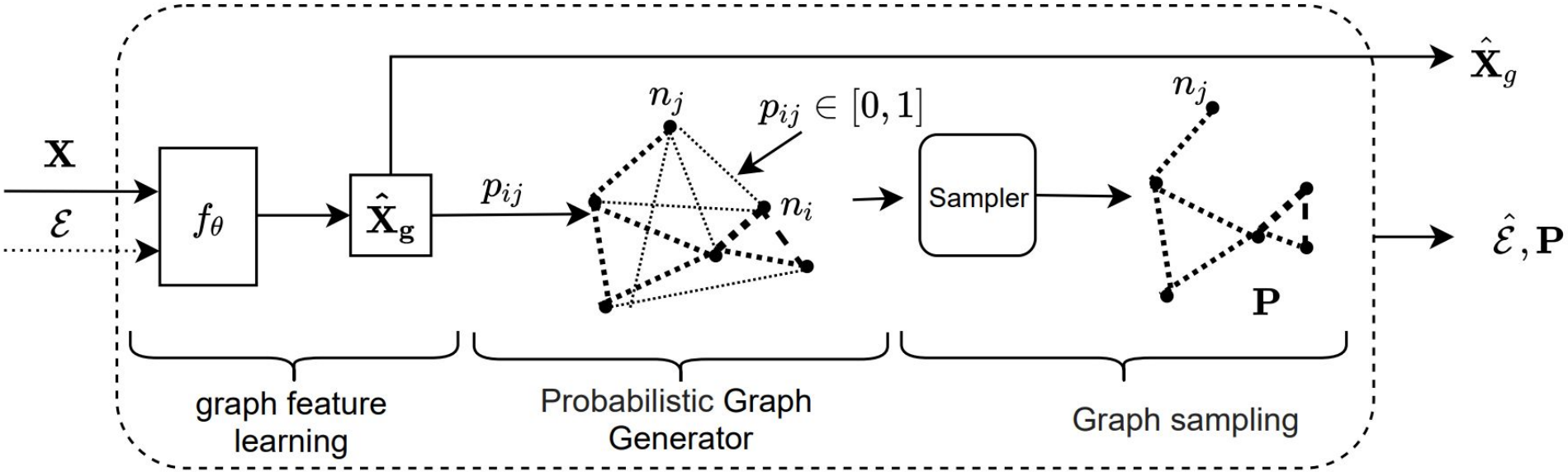
Dynamic Graph CNN (Wang *et al.*, 2018)



Literally take k nearest neighbours to decide edges at every layer, without any new parameters

Option 3c: *Infer* edges to use (*reinforcement learning*)

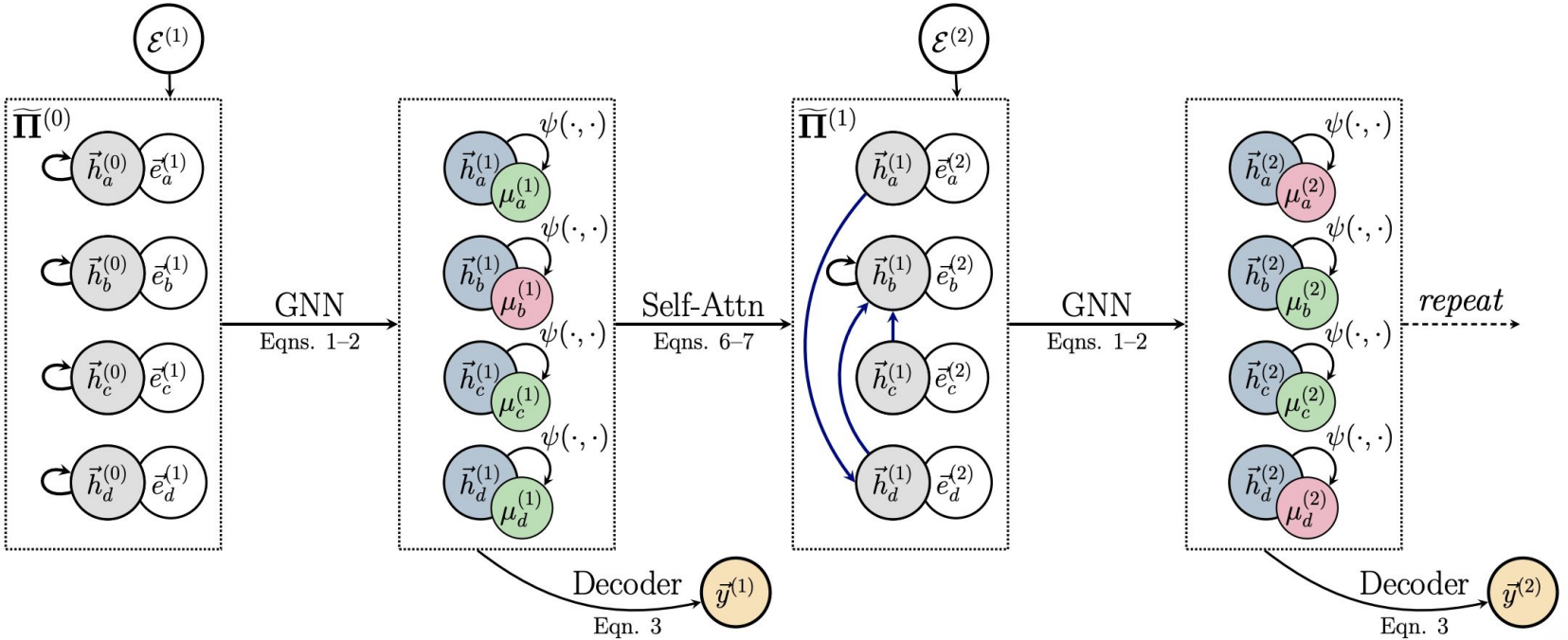
Differentiable Graph Module (Kazi *et al.*, 2020)



Sample (probabilistically) a **k-NN graph**, optimise **downstream performance** as RL reward

Option 3d: *Infer* edges to use (*supervised learning*)

Pointer Graph Networks (Veličković *et al.*, NeurIPS'20)



Directly **supervise** the **k-NN graph** using ground-truth knowledge (e.g. **data structure rollouts**)

DeepMind

II 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*

- We will assess the **power** of GNNs by *which graphs they are able to* ***distinguish***.

Weisfeiler-Lehman Test

- Simple but powerful way of distinguishing: pass **random hashes** of **sums** along the edges
  - Iterate until hashes don't change.
  - *"Possibly isomorphic"* if hash histograms are the same.

УДК 519.1

ПРИВЕДЕНИЕ ГРАФА К КАНОНИЧЕСКОМУ ВИДУ И ВОЗНИКАЮЩАЯ ПРИ ЭТОМ АЛГЕБРА

Б. Ю. ВЕЙСФЕЙЛЕР, А. А. ЛЕМАН

Рассматривается алгоритм приведения заданного конечного мультиграфа Γ к каноническому виду. В процессе такого приведения возникает новый инвариант графа — алгебра $\mathfrak{A}(\Gamma)$. Изучение свойств алгебры $\mathfrak{A}(\Gamma)$ оказывается полезным при решении некоторых задач теории графов.

Выдвигается и обсуждается некоторые предположения относительно связи между свойствами алгебры $\mathfrak{A}(\Gamma)$ и группой автоморфизмов графа $Aut(\Gamma)$. Построен пример неориентированного графа Γ, алгебра $\mathfrak{A}(\Gamma)$ которого совпадает с групповой алгеброй некоторой некоммутативной группы.

An algorithm is considered, reducing the specified finite multigraph Γ to canonical form. In the course of this reduction, a new invariant of the graph is generated — algebra $\mathfrak{A}(\Gamma)$. Study of the properties of the algebra $\mathfrak{A}(\Gamma)$ proves helpful in solving a number of graph-theoretic problems. Some propositions concerning the relationships between the properties of the algebra $\mathfrak{A}(\Gamma)$ and the graph's automorphism group $Aut(\Gamma)$ are discussed. An example of non-oriented graph Γ is constructed whose algebra $\mathfrak{A}(\Gamma)$ coincides with the group algebra of a non-commutative group.

1. Рассмотрим произвольный конечный граф Γ и его матрицу смежности $A(\Gamma)=\|a_{ij}\|$: здесь a_{ij} — число ребер, исходящих из i-й вершины графа в j-ую; $i, j=1, 2, \ldots, n$. В случае неориентированного графа полагаем $a_{ij}=a_{ji}$. Каноническим видом графа мы будем называть его матрицу смежности при канонической нумерации вершин, т. е. при таком частичном упорядочении множества вершин, при котором из того, что $a \neq b$ неравенство следует, что существует автоморфизм графа, переводящий вершину a в b и сохраняющий отношение смежности.

В п. 5, 7 описан процесс приведения графа к каноническому виду, состоящий в поэтапном перенумеровании строк и столбцов матрицы $A(\Gamma)$, который, грубо говоря, сводится к следующему.

Рассмотрим для простоты неориентированный граф без кратных связей. Сначала каждой вершине графа сопоставим характеристический вектор, единственная компонента которого равна числу соседей данной вершины. Затем разобьем вершины на классы, так чтобы вершины с одинаковым характеристическим вектором принадлежали одному классу; классы при этом упорядочим в соответствии с естественным порядком в множестве характеристических векторов. Далее, каждой вершине сопоставим характеристический вектор $v_1=(l, v_{11}, v_{12}, \ldots)$, где v_{1k} — число соседей k-го класса у j-ой вершины, l — номер класса, к которому принадлежит i-ая вершина. Теперь снова разобьем вершины на классы в соответствии с новыми характеристическими векторами, упорядоченными лексикографически, и т. д. Заметим, что если вершины a и b на некотором шаге принадлежали разным классам и было выполнено условие $a<b$, то и в дальнейшем это условие всегда будет выполнено. Отсюда следует, что описанный процесс обрывается на таком, мы через α шагов — и либо все вершины относятся к разным классам (т. е. построена каноническая нумерация), либо дальнейшего разбиения на классы не происходит.

В случае, если Γ — ориентированный мультиграф, возьмем в качестве характеристического вектора v_i упорядоченную i-ую строку матрицы $A(\Gamma)$ (считая при этом, что диагональный элемент предшествует всем остальным). Вместо различных элементов a_{ij} введем различные независимые переменные $x_0, x_1, \ldots$, упорядочив их в соответствии с порядком следования a_{ij}. Полученную таким способом матрицу обозначим $X=X(\Gamma)$. При очередном разбиении вершин на классы к одному классу, как и прежде, отнесем вершины с одинаковым характеристическим вектором; при этом k-ая компонента вектора v_i есть по определению сумма элементов i-ой строки матрицы $X(\Gamma)$, соответствующих вершинам k-го класса (предыдущего разбиения). Матрица $X(\Gamma)$ разбивается таким образом на блоки, в каждом из которых мы можем вместо своих независимых переменных и т. д. (точное определение этих операций см. п. 6), совершив α_0, β_0.

Заметим, что описанное до сих пор процедуре аналогична методам, изложенным в [1] и [2].

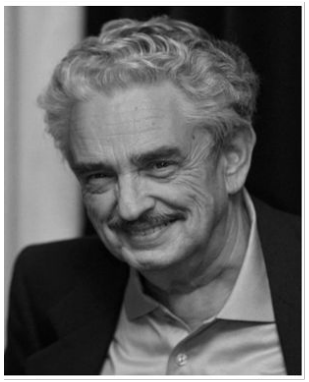
Для дальнейшего разбиения вершин на классы рассмотрим элемент a_{12} матрицы $U=X \cdot X'$, где X' — матрица, получения из X заменой переменных $x_0, x_1, \ldots$ переменными $x_1', x_2', \ldots$, причем все переменные $x_0, x_0, \ldots, x_1, x_2', \ldots$ независимы. Элемент a_{12} является многочленом второй степени от $x_1, x_2, \ldots, x_1', x_2', \ldots$. Если теперь обозначить различные многочлены различными новыми независимыми переменными, то к полученной матрице снова можно будет применить все описанные выше операции и т. д., до тех пор, пока и этот процесс не обрывается (см. п. 6, операция $\alpha_0, \beta_0, \beta_0$).

2. Геометрически введение независимых переменных и матрицы $U=X \cdot X'$ означает следующее. Если вместо с ребрами данного графа Γ рассматривать ребра дополнительного графа $\bar{\Gamma}$, то на первом же шаге малой процедуры этим ребрам будут соответствовать наборами равных переменных — т. е. произойдет раскраска ребер данного и дополнительного графов в разные цвета. В дальнейшем каждое новое введение дополнительных переменных вводит новую раскраску ребер. (причем 1) ребра, окрашенные на каком-то шаге по разному, в дальнейшем также будут окрашены по-разному; 2) разбиение вершин на классы производится в соответствии с количеством ребер каждого цвета, исходящих из вершины.

Известно, далее, что элемент $a_{ij} \in A$, $A=A(\Gamma)$, где Γ — неориентированный граф без кратных связей, равен числу путей длины 2, ведущих из i-й вершины в j-ую. Аналогично, коэффициент при $x_0 x_k'$ в многочлене $a_{ij} \in U=X \cdot X'$ равен числу путей, ведущих из i-й вершины в j-ую по ребрам сначала k-го, а затем l-го цвета.

3). Дальнейшая процедура приведения графа к каноническому виду использует применение описанных выше операций к матрице, полученной из $X(\Gamma)$ вычеркиванием некоторого ее столбца и соответствующей строки. Если для матрица порядка $k \cdot n-1$ приведения к каноническому виду определено, то возникает новая возможность для разбиения вершин на классы: старшей считается та вершина, после вычеркивание которой получается лексикографически старший канонический вид оставшегося графа (см. операция $\alpha_2, \alpha_k, \beta_1, \beta_k$ п. 7). Очевидно, что и такое разбиение на некотором шаге обрывается. Доказано, что любые две вершины a и b, отнесенные к одному классу при последнем разбиении, эквивалентны, т. е. существует автоморфизм графа Γ, переводящий a в b с сохранением отношения смежности.

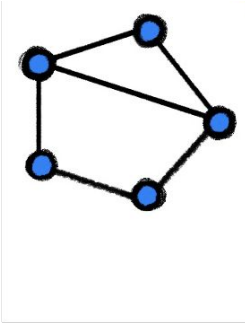
Рассмотрим снова такую матрицу $X=X(\Gamma)$, что в матрице $U=X \cdot X'$ на месте одинаковых переменных матрицы X стоят одинаковые многочлены. Матрица $X(\Gamma)$ является тогда общей точкой некоторой матричной алгебры $\mathfrak{A}(\Gamma)$, т. е. если взять кольцо K (например, кольцо целых чисел Z или поле рациональных чисел Q), то множество матриц, получения подстановкой в X вместо ее переменных элементов кольца K образует алгебру $\mathfrak{A}_K(\Gamma)=\mathfrak{A}(\Gamma) \otimes K$. Алгебра $\mathfrak{A}(\Gamma)$ является, очевидно, инвариантом графа. Некоторые соотношения между



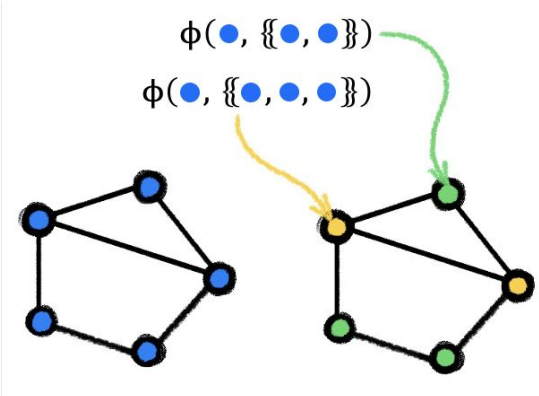
A. Lehman

B. Weisfeiler

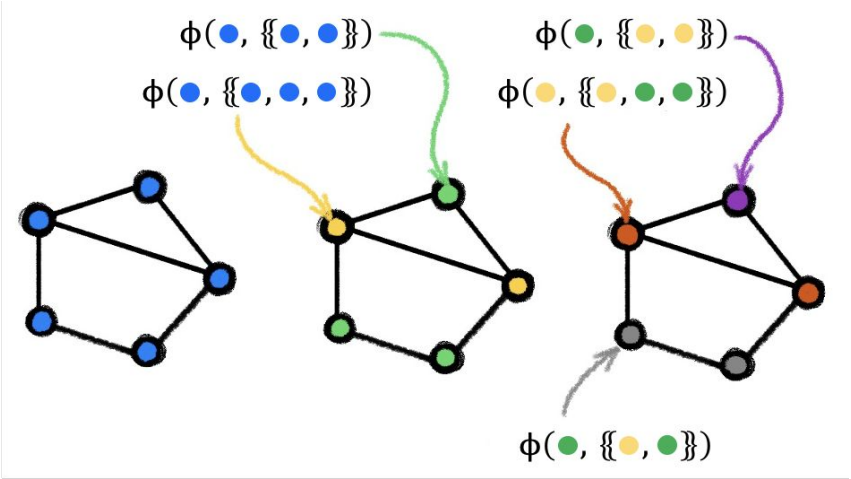
Let’s run the WL Test!



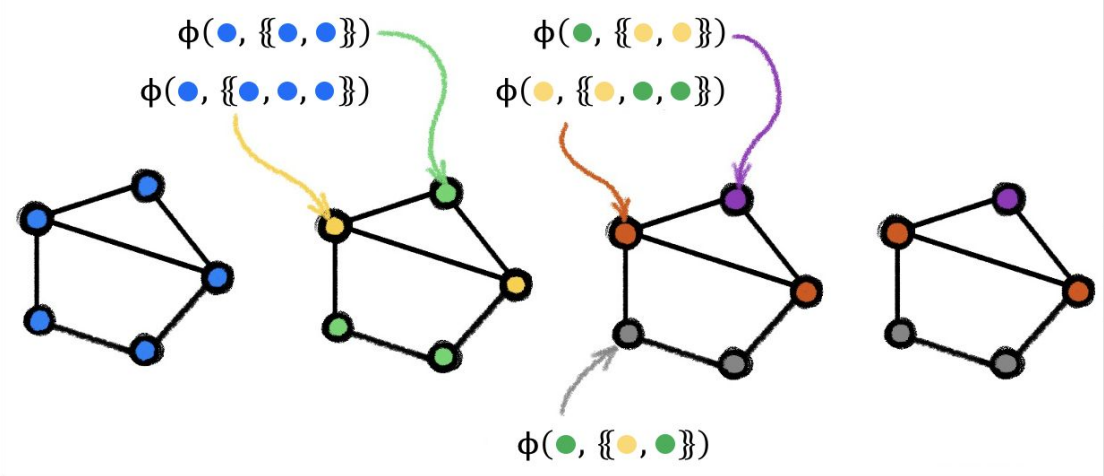
Let's run the WL Test!

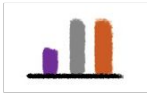


Let's run the WL Test!

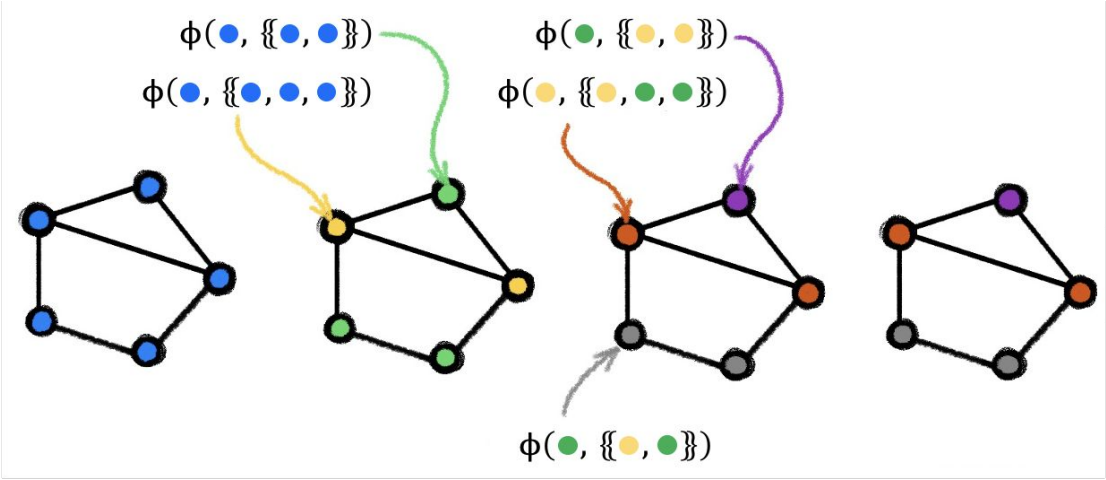


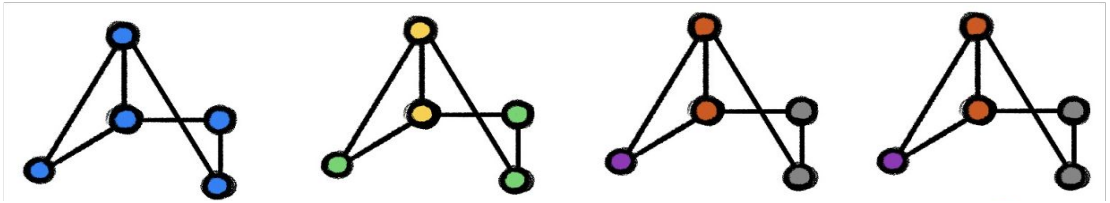
Let's run the WL Test!





Let's run the WL Test!





Weisfeiler-Lehman Test

- Connection to conv-GNNs spotted very early; e.g. by GCN (Kipf & Welling, ICLR'17)
- **Untrained** GNNs can hence work very well!
  - Untrained ~ random hash
- The test does **fail** at times, however:

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

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

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

$t \leftarrow 0$;

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*;

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

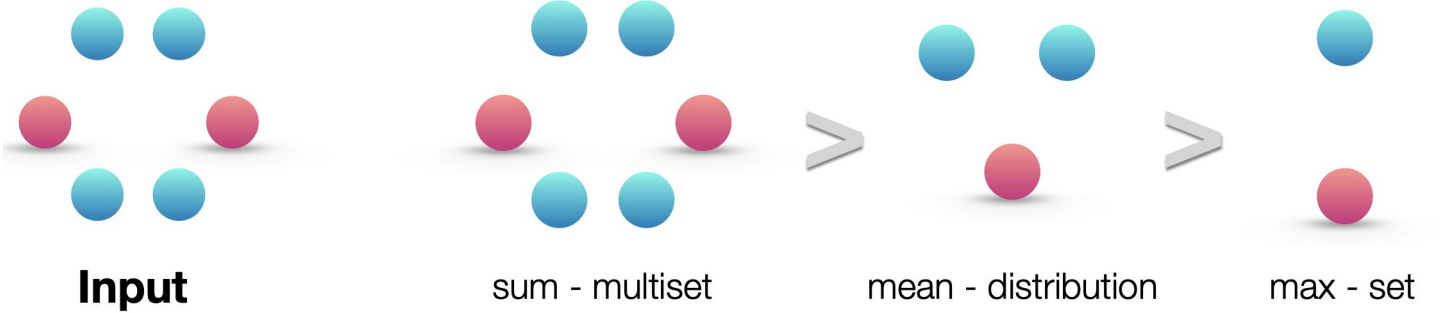
GNNs are **no more powerful** than 1-WL

- Over ***discrete*** *features*, GNNs can only be **as powerful** as the 1-WL test described before!

GNNs are **no more powerful** than 1-WL

- Over ***discrete*** *features*, GNNs can only be **as powerful** as the 1-WL test described before!

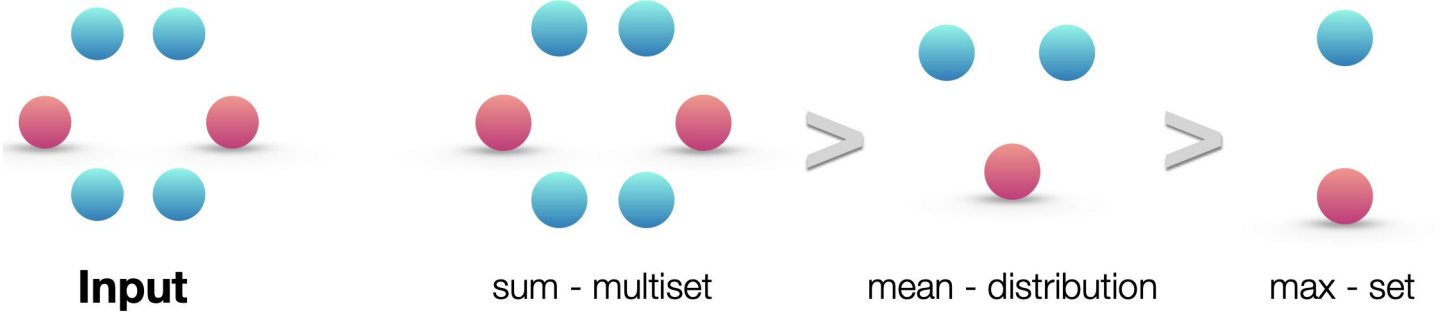
- One important condition for maximal power is an ***injective*** aggregator (e.g. **sum**)



GNNs are **no more powerful** than 1-WL

- Over ***discrete*** *features*, GNNs can only be **as powerful** as the 1-WL test described before!

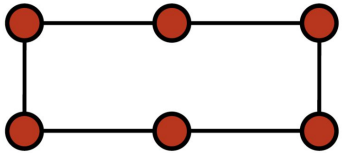
- One important condition for maximal power is an ***injective*** aggregator (e.g. **sum**)

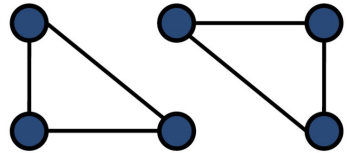


- Graph isomorphism network (**GIN**; Xu *et al.*, ICLR'19) proposes a simple, maximally-expressive GNN, following this principle:

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

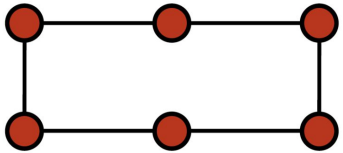
Higher-order GNNs

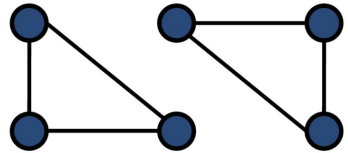




- We can make GNNs stronger by analysing **failure cases** of 1-WL!
  - Very active area, with many open problems!

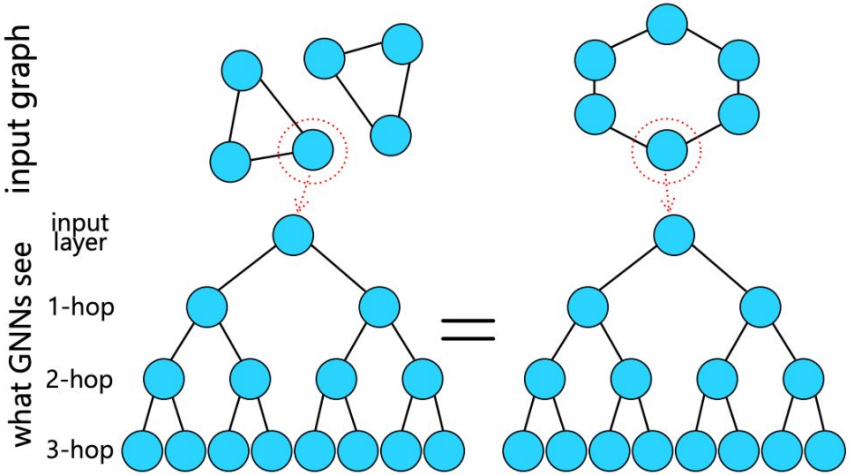
Higher-order GNNs



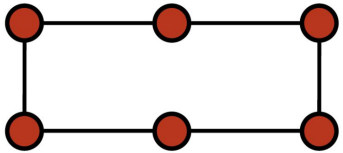


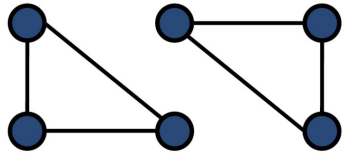
- We can make GNNs stronger by analysing **failure cases** of 1-WL!

- For example, just like 1-WL, GNNs cannot detect **closed triangles**
  - This is because, from a GNN's perspective, all nodes look the same!
  - Can you think of a simple fix?



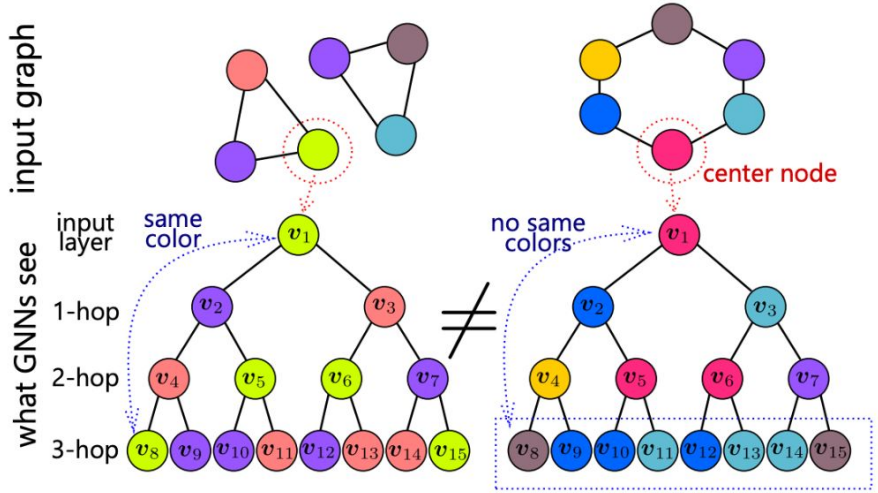
Higher-order GNNs



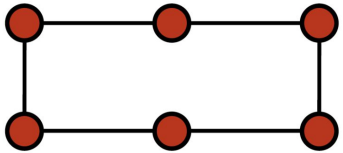


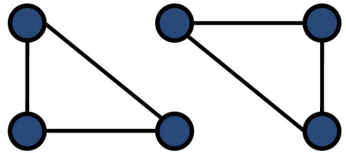
- We can make GNNs stronger by analysing **failure cases** of 1-WL!

- For example, just like 1-WL, GNNs cannot detect **closed triangles**
  - Augment nodes with **randomised** features (Sato *et al.*, SDM'21)
  - Now a node can "see itself" k hops away!



Higher-order GNNs





- We can make GNNs stronger by analysing **failure cases** of 1-WL!

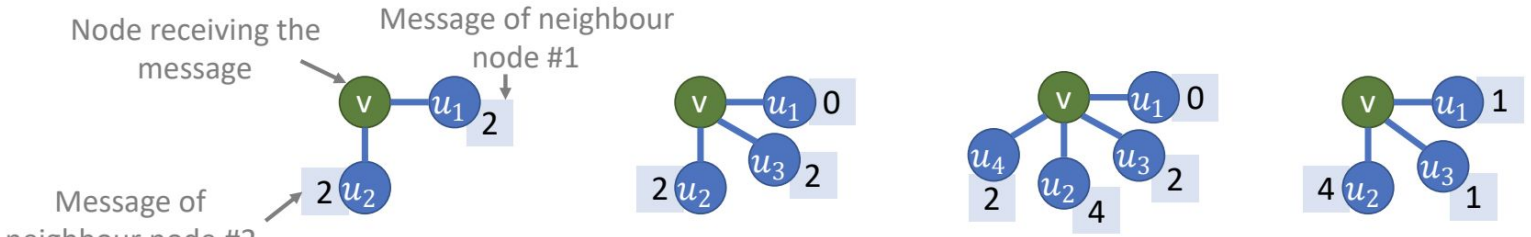
- For example, just like 1-WL, GNNs cannot detect **closed triangles**
  - Augment nodes with randomised/positional features (Sato *et al.*, SDM'21)
    - Explored by RP-GNN (Murphy *et al.*, ICML'19) and P-GNN (You *et al.*, ICML'19)
  - Can also literally **count** interesting subgraphs (Bouritsas *et al.*, 2020)

- Fixing "failure cases" of 1-WL yields many classes of *higher-order GNNs*

- They can broadly be categorised into three groups:
  - Modifying **features** (as above)
  - Modifying the **message passing rule**; e.g. DGN (Beaini, Passaro *et al.* (2020))
  - Modifying the **graph** structure; e.g. 1-2-3-GNNs (Morris *et al.*, AAAI'19)

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**



Simple aggregators that can differentiate graph 1 and 2:

Aggregators that fail:

Mean	Mean	Mean	Mean
Min	Min	Min	Min
Max	Max	Max	Max
STD	STD	STD	STD

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

III Geometric Deep Learning

Remark on **geometric** deep learning

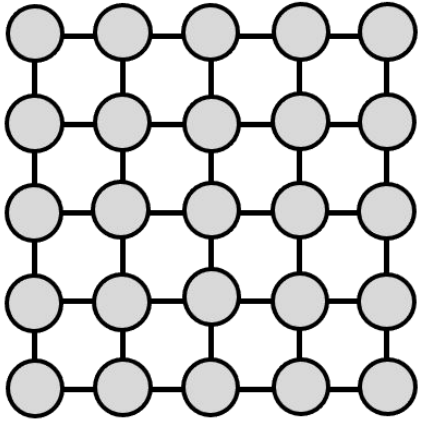
- We used the blueprint of *invariances* and *equivariances* to describe GNNs

- 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

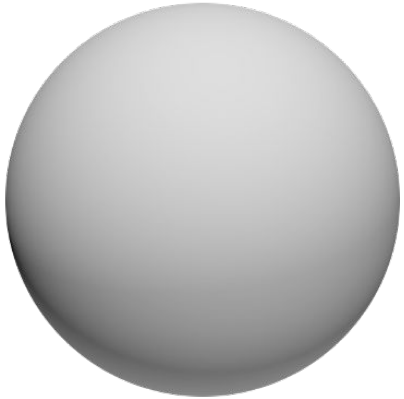
  we recover many standard architectures (including CNNs and Transformers!)

- But also a more general class of **geometric** deep learning architectures

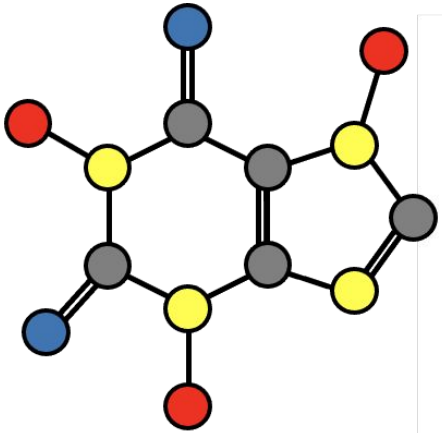
The “Five Gs” of geometric deep learning



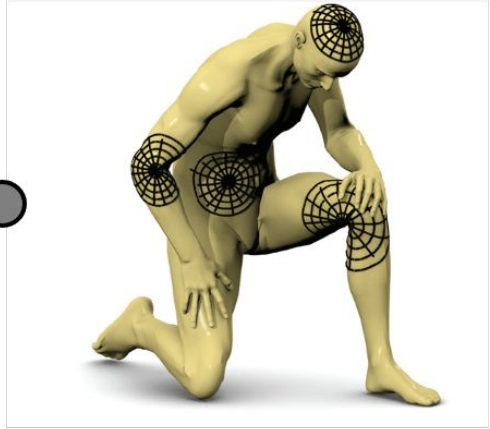
Grids



Groups



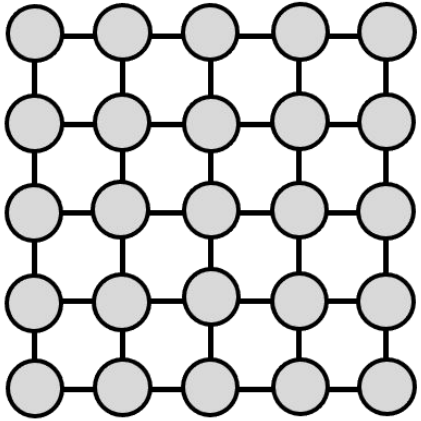
Graphs



Geodesics & Gauges

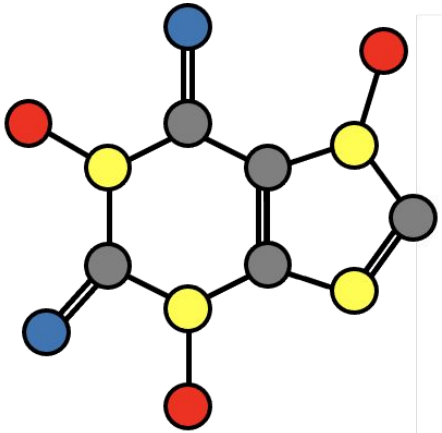
Credits to **Michael Bronstein**

The “Five Gs” of geometric deep learning

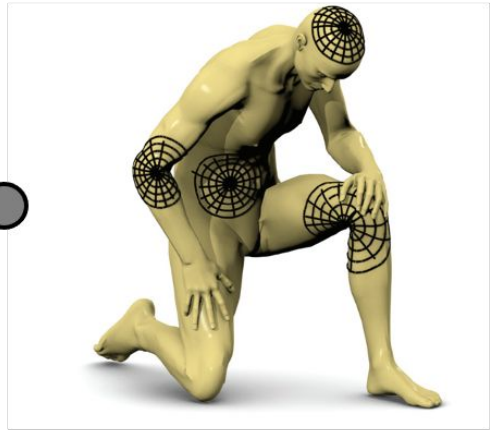


Images & Sequences

Homogeneous spaces



Graphs & Sets



Manifolds, Meshes & Geometric graphs

Credits to **Michael Bronstein**

Some architectures of interest

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

DeepMind

Thank you!

Questions?

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

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