

# Graphs in Machine Learning

Daniele Calandriello

*DeepMind Paris, France*

Collaborators: Achraf Azize,  
Michal Valko

Based on material by: Petar Veličković

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



# Revisiting SL on graphs

modern problems require modern solutions

# Supervised learning on graphs

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

# Supervised learning on graphs

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

More perspectives today:

- many graphs, graphs are the *data*

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

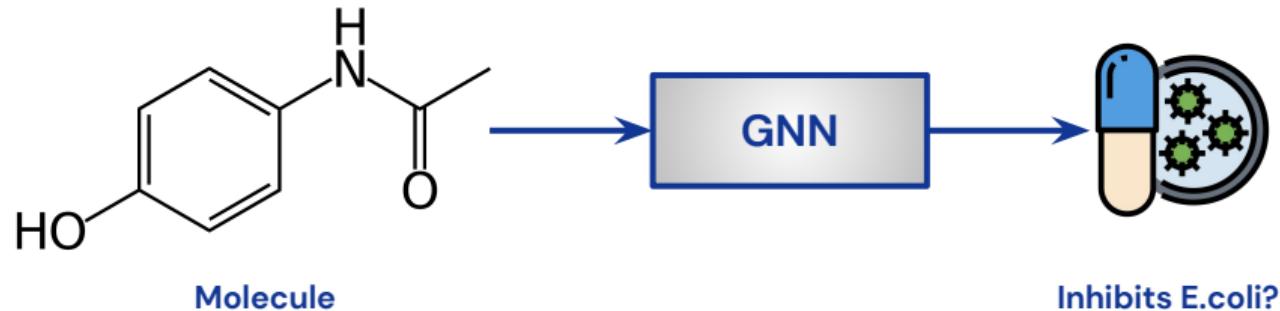
# GNNs for molecule classification

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

# GNNs for molecule classification

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

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



## Attempt naive drug discovery

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

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

Build large dataset of candidates  $\rightarrow$  sample with confidence logic  
We take the one that is the most confident.  
Can do additional filtering on the answer

## Attempt naive drug discovery

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

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

## Attempt naive drug discovery

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

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

## Attempt naive drug discovery

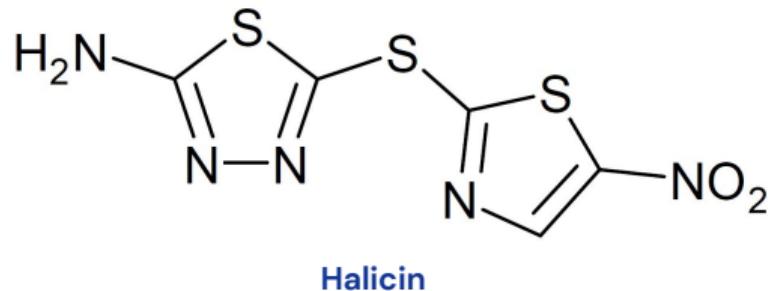
Once trained, the model can be applied to any molecule.

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

## Attempt naive drug discovery

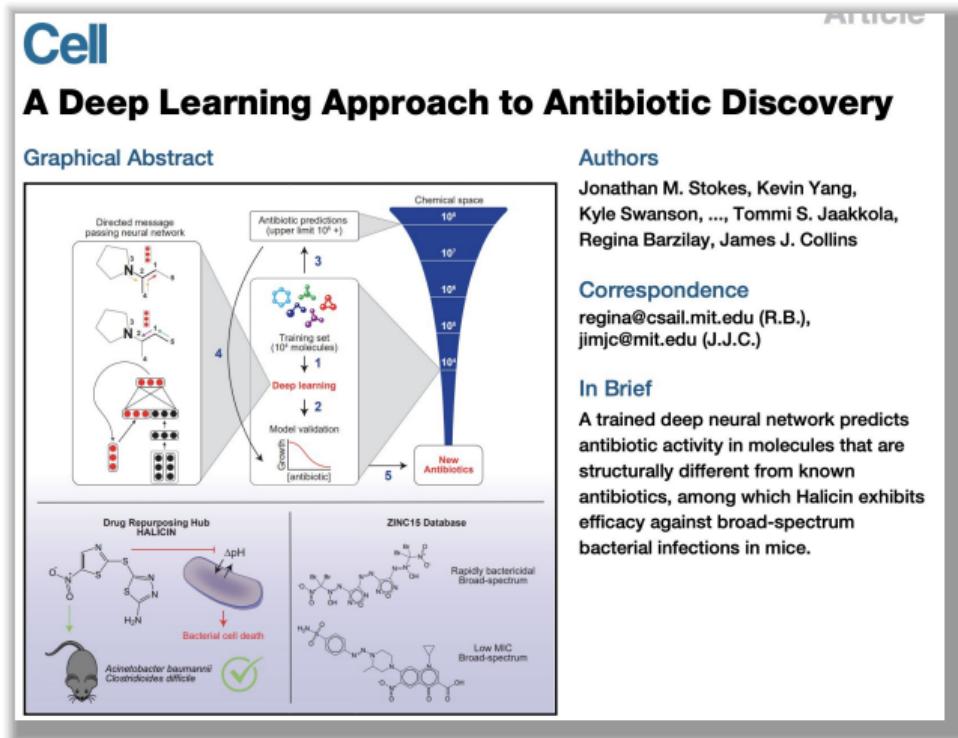
Once trained, the model can be applied to any molecule.

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



## 6. become famous

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



(Stokes et al., Cell'20)

## 6. become famous

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

The image shows a composite screenshot of two scientific publications. At the top, the 'Cell' logo is visible above the 'nature' logo. To the right of the logos are buttons for 'Subscribe' and 'Cover'. Below the logos, the text 'news · 20 FEBRUARY 2020' is displayed. The main title of the article is 'Powerful antibiotics discovered using AI'. A sub-headline reads: 'Machine learning spots molecules that work even against 'untreatable' strains of bacteria.' In the bottom left corner, the citation '(Stokes et al., Cell'20)' is shown. The bottom right corner contains a chemical structure of a molecule labeled 'Broad-spectrum'.

Cell

nature

Subscribe

Cover

news · 20 FEBRUARY 2020

# Powerful antibiotics discovered using AI

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

(Stokes et al., Cell'20)

Acinetobacter baumannii  
Clostridioides difficile

Broad-spectrum

## 6. become famous

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

The screenshot shows a BBC News article. At the top, there's a navigation bar with the BBC logo, a 'Sign in' button, and categories: News, Sport, Reel, Worklife, Travel, Future. Below that is a red header with 'NEWS' in large white letters. A blue sidebar on the right has 'Subscribe' and some names: Kevin Yang, Ni S. Jaakkola, J. Collins. The main content area features a blue banner with 'BBC WORKLIFE' and the text 'Our new guide for getting ahead'. The main headline is 'Scientists discover powerful antibiotic using AI'. Below it, a sub-headline reads 'Machine bacteria. (S' and 'trains of'. At the bottom left is a timestamp '(S) 21 February 2020'. On the bottom right is a 'Share' button.

ARTICLE

covery

oscribe

Kevin Yang,  
Ni S. Jaakkola,  
J. Collins

B.,

etwork predicts  
ecules that are  
m known  
ch Halicin exhibits  
spectrum  
ice.

NEWS · 2

Power

Machine bacteria. (S

trains of

BBC WORKLIFE

Our new guide for getting ahead

# Scientists discover powerful antibiotic using AI

© 21 February 2020

Share

## 6. become famous

Arguably the most popularised s

 Sign in

# NEWS

Home | Video | World | UK | Business



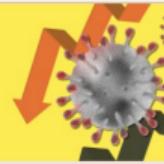
## Scientists discover powerful antibiotic using AI

(S

© 21 February 2020

Share

**CORONAVIRUS BUSINESS UPDATE**  
Get 30 days' complimentary access to our Coronavirus Business Update newsletter



Artificial intelligence

Robotics      'Death of the office' homeworking claims exaggerated      Anti-social robots harr increase social distanc

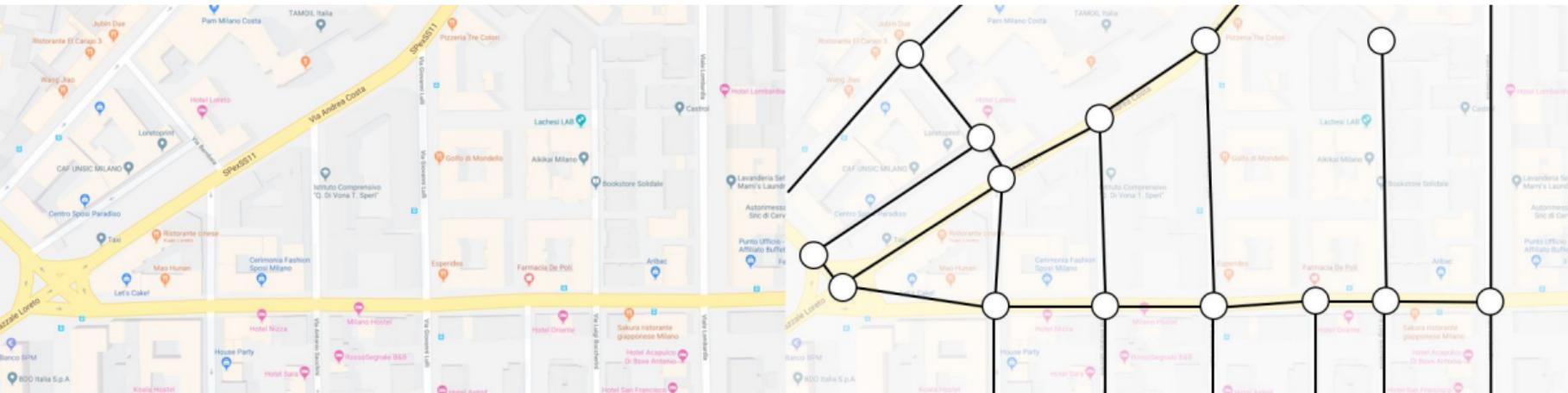
AI discovers antibiotics to treat drug-resistant diseases

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

trains of

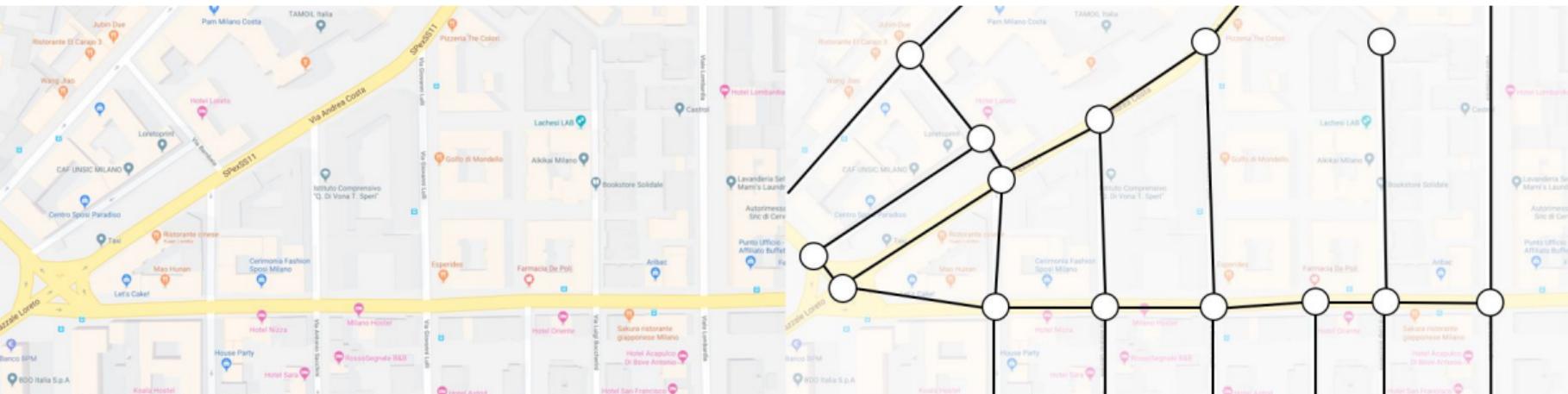
# Transportation sub-graph classification

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



# Transportation sub-graph classification

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



Classify sub-graph properties:

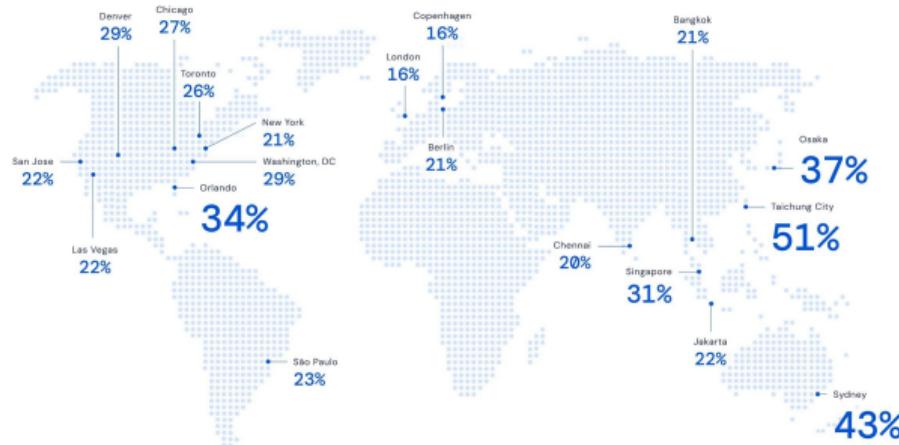
↳ Very livable neighborhood? Dangerous intersection? Traffic trap?

↳ not good → not good..

# Transportation sub-graph regression

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

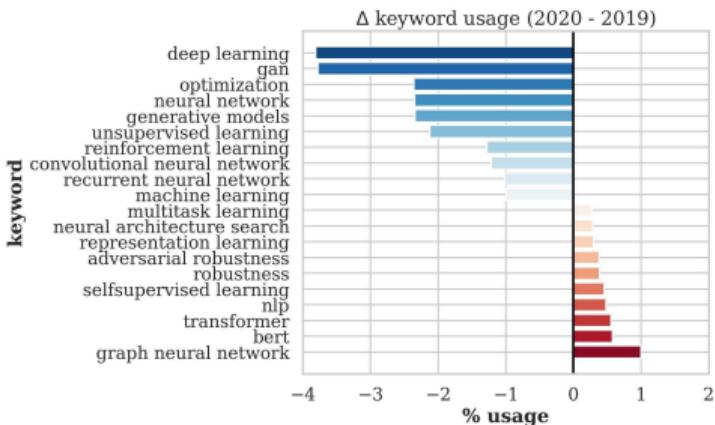
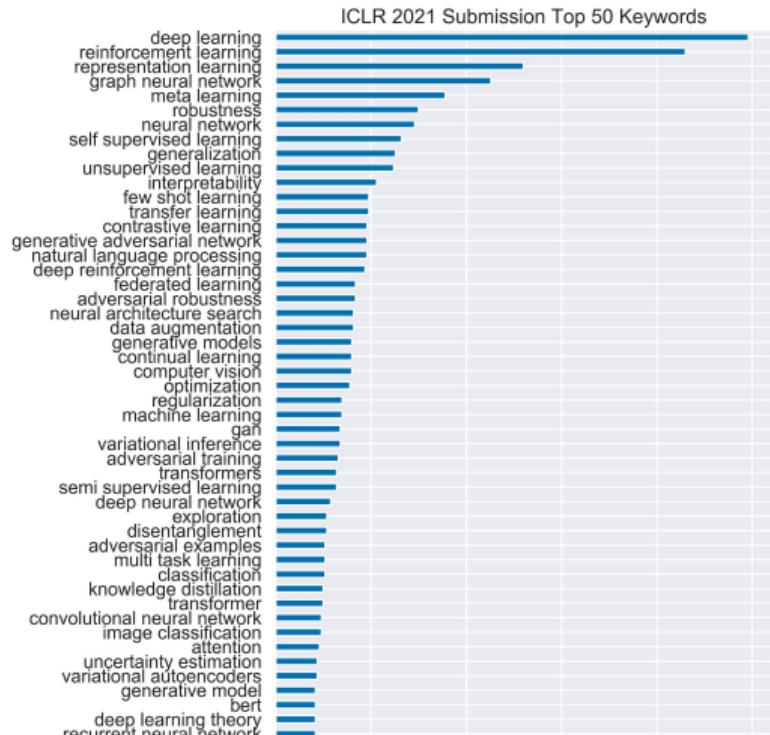
↳ DeepMind's ETA Prediction using GNNs in Google Maps



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

# GNN's imangenet moment

In AGC we've seen data to make work even  
easier for GNN: recently dataset emerge



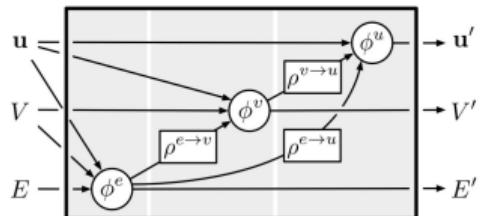
# Rich ecosystem of libraries



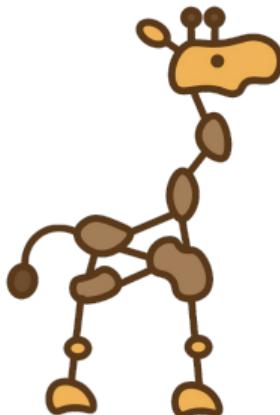
[github.com/rusty1s/pytorch\\_geometric](https://github.com/rusty1s/pytorch_geometric)



[graphneural.network](https://graphneural.network)



[github.com/deepmind/graph\\_nets](https://github.com/deepmind/graph_nets)



[github.com/deepmind/jraph](https://github.com/deepmind/jraph)

## Rich ecosystem of datasets



[ogb.stanford.edu](http://ogb.stanford.edu)



PyTorch  
geometric

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



TUDataset

[graphlearning.io](http://graphlearning.io)

## Benchmarking Graph Neural Networks

[github.com/graphdeeplearning/benchmarking-gnns](https://github.com/graphdeeplearning/benchmarking-gnns)

## More references

Main  
Material

Petar Veličković's site (<https://petar-v.com>), this class' material is based on

↳ Theoretical Foundations of Graph Neural Networks.

Everything is Connected: Graph Neural Networks from the Ground Up.

This twitter thread for more GNN resources

[https://twitter.com/PetarV\\_93/status/1306689702020382720](https://twitter.com/PetarV_93/status/1306689702020382720)

Will Hamilton's graph representation learning textbook (esp. Chapter 7)

[https://www.cs.mcgill.ca/~wlh/grl\\_book/](https://www.cs.mcgill.ca/~wlh/grl_book/)

Aleksa Gordić's GitHub repository on GATs

<https://github.com/gordicaleksa/pytorch-GAT>

# **Set neural networks**

from first principles

## Function invariance in networks

$$\begin{matrix} \begin{matrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \text{x1} & \text{x0} & \text{x1} & & & & \\ \hline 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \text{x0} & \text{x1} & \text{x0} & & & & \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \text{x1} & \text{x0} & \text{x1} & & & & \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \text{x0} & \text{x1} & \text{x0} & & & & \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \text{x0} & \text{x1} & \text{x0} & & & & \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \text{x1} & \text{x0} & \text{x1} & & & & \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{matrix} & * & \begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix} & = & \begin{matrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{matrix} \end{matrix}$$

$\mathbf{I}$                      $\mathbf{K}$                      $\mathbf{I} * \mathbf{K}$

## Function invariance in networks

$$\begin{matrix} \begin{matrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{matrix} & \times & \begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix} & = & \begin{matrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{matrix} \end{matrix}$$

The diagram illustrates the convolution operation between two matrices, I and K, resulting in the matrix I \* K.

Matrix I (Input) is a 7x7 matrix with values:

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

Matrix K (Kernel) is a 3x3 matrix with values:

1	0	1
0	1	0
1	0	1

The result of the convolution I \* K is a 5x5 matrix with values:

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

Detailed description: The diagram shows the convolution process. Matrix I is a 7x7 grid. A 3x3 subgrid in the top-left corner of I is highlighted in red and multiplied by matrix K. Matrix K is shown below I. The result of this multiplication is a 3x3 matrix. This result is then convolved with the remaining part of I (a 5x5 grid). The result of this second convolution is a 5x5 matrix labeled I \* K. The value 4 in the top-left cell of I \* K is highlighted in green, indicating it is the result of the first convolution step. Dotted lines connect the highlighted subgrid in I to the 3x3 kernel K, and another set of dotted lines connects the result of that multiplication to the corresponding cell in I \* K.

## Function invariance in networks

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \quad \begin{matrix} * \\ \mathbf{K} \end{matrix} \quad \begin{array}{|c|c|c|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 1 & 0 & 1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|c|c|} \hline 1 & 4 & 3 & 4 & 1 \\ \hline 1 & 2 & 4 & 3 & 3 \\ \hline 1 & 2 & 3 & 4 & 1 \\ \hline 1 & 3 & 3 & 1 & 1 \\ \hline 3 & 3 & 1 & 1 & 0 \\ \hline \end{array}$$

The diagram illustrates the convolution operation  $I * K$ . The input matrix  $I$  is a 7x7 grid of binary values. A 3x3 kernel  $K$  is applied to  $I$  to produce the output matrix  $I * K$ . The result is a 5x5 matrix where each value is the sum of the products of the corresponding elements in the kernel and the receptive field it covers in the input. The kernel  $K$  is highlighted with a blue border. The output value 3 at position (3,3) in the result matrix is highlighted with a green box.

## Function invariance in networks

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \quad I$$
$$\begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 1 & 0 & 1 \\ \hline \end{array} \quad K$$
$$\begin{array}{|c|c|c|c|c|} \hline 1 & 4 & 3 & 4 & 1 \\ \hline 1 & 2 & 4 & 3 & 3 \\ \hline 1 & 2 & 3 & 4 & 1 \\ \hline 1 & 3 & 3 & 1 & 1 \\ \hline 3 & 3 & 1 & 1 & 0 \\ \hline \end{array} \quad I * K$$

The diagram illustrates the convolution operation between a 3x3 input matrix  $I$  and a 3x3 kernel  $K$ . The input  $I$  has values 0 or 1. The kernel  $K$  has values 1, 0, 1 in its rows. The result  $I * K$  is a 5x5 matrix where each value is the sum of products of corresponding elements from a 3x3 receptive field in  $I$  and the kernel  $K$ . The result matrix  $I * K$  is shown with a green highlighted cell at index (4,4) containing the value 4.

## Function invariance in networks

The diagram illustrates the convolution operation  $I * K = I * K$ . It shows a 7x7 input matrix  $I$  and a 3x3 kernel matrix  $K$ , with their element-wise product  $I * K$  shown on the right.

**Input Matrix  $I$ :**

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

**Kernel Matrix  $K$ :**

1	0	1
0	1	0
1	0	1

**Output Matrix  $I * K$ :**

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

The diagram highlights specific elements in red and green to show how the kernel  $K$  slides across the input  $I$  to produce the output  $I * K$ .

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

## Set invariance

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

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

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

## Set invariance

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

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

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

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

## Set invariance

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

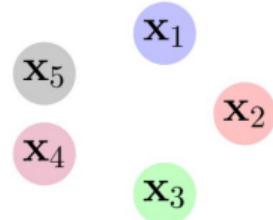
$$\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top$$

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

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

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

## Set invariance



## Set invariance

$$f \left( \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} \right) = y$$

## Set invariance

invariance on set permutation.

$$f \left( \begin{matrix} x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \end{matrix} \right) = y = f \left( \begin{matrix} x_2 \\ x_5 \\ x_1 \\ x_4 \\ x_3 \end{matrix} \right)$$

## Permutation invariance

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

## Permutation invariance

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

To explain permutations to the GPU we can use linear algebra

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

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

## Permutation invariance

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

To explain permutations to the GPU we can use linear algebra

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

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

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

## Permutation invariance

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

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

## Permutation invariance

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

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

One very generic form is the Deep Sets model

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

where  $\phi$  and  $\psi$  are (learnable) functions (e.g. ReLU and MLPs).

## Permutation invariance

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

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

One very generic form is the Deep Sets model

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

where  $\phi$  and  $\psi$  are (learnable) functions (e.g. ReLU and MLPs).

What if I am trying to classify individual nodes?

## Permutation invariance

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

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

One very generic form is the Deep Sets model

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

where  $\phi$  and  $\psi$  are (learnable) functions (e.g. ReLU and MLPs).

What if I am trying to classify individual nodes?

↳ We also need functions that don't change the node order!

## Permutation equivariance

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

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

## Permutation equivariance

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

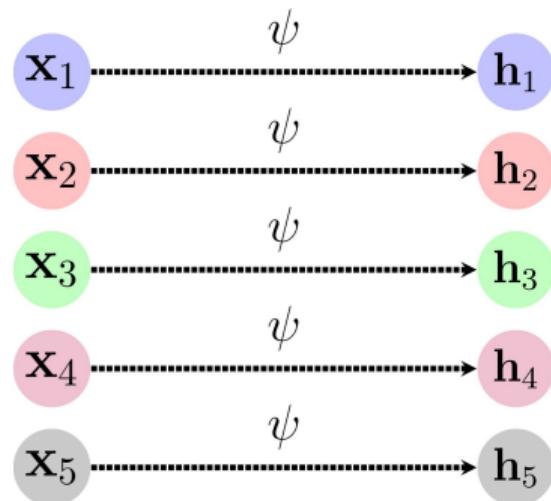
$$f_{\text{equ}}(\mathbf{P}\mathbf{X}) = \mathbf{P}f_{\text{equ}}(\mathbf{X}).$$

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

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

where  $\psi$  is any function.

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



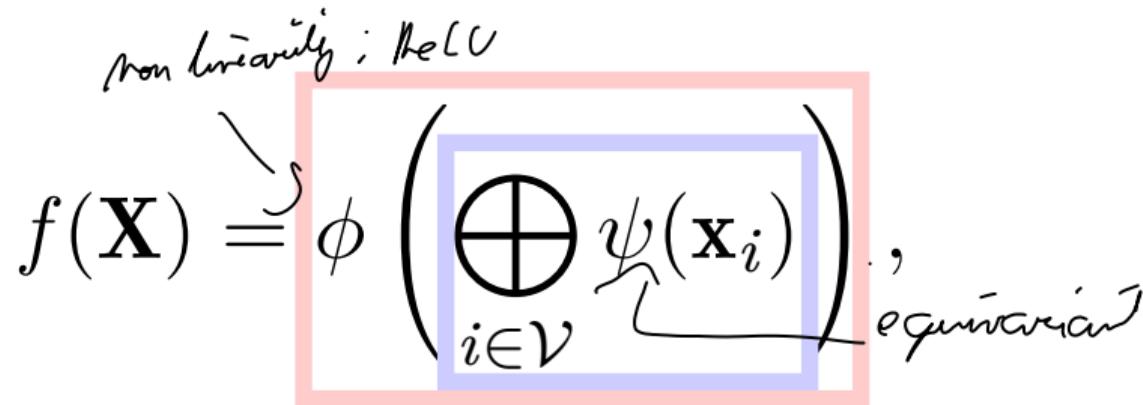
## One-layer set neural network

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

*non-linearity; ReLU*

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

*equivariant*



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

# Graph neural networks

from first principles

## From set to graphs

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

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

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

## From set to graphs

In theory: invariance to graph isomorphism!

$$f \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \right) = y = f \left( \begin{array}{c} x_2 \\ x_4 \\ x_5 \\ x_1 \\ x_3 \end{array} \right)$$

$$f \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \right) = y = f \left( \begin{array}{c} x_2 \\ x_4 \\ x_5 \\ x_1 \\ x_3 \end{array} \right)$$

## From set to graphs

In theory: invariance to graph isomorphism!

$$f \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \right) = y = f \left( \begin{array}{c} x_2 \\ x_4 \\ x_5 \\ x_1 \\ x_3 \end{array} \right)$$

$$f \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \right) = y = f \left( \begin{array}{c} x_2 \\ x_4 \\ x_5 \\ x_1 \\ x_3 \end{array} \right)$$

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

## Permutation {in, equi}variance on graphs

We need to appropriately permute both rows and columns of  $\mathbf{A}$ :

↳ this amounts to applying the permutation matrix  $\mathbf{P}$  on both sides  $\mathbf{P}\mathbf{A}\mathbf{P}^T$

## Permutation {in, equi}variance on graphs

We need to appropriately permute both rows and columns of  $\mathbf{A}$ :

↳ this amounts to applying the permutation matrix  $\mathbf{P}$  on both sides  $\mathbf{P}\mathbf{A}\mathbf{P}^T$

Invariance:

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

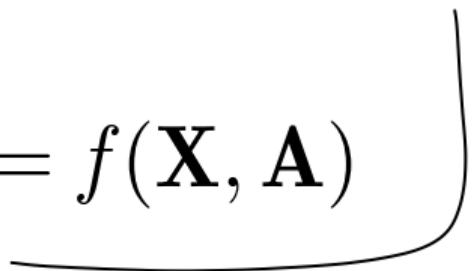
## Permutation {in, equi}variance on graphs

We need to appropriately permute both rows and columns of  $\mathbf{A}$ :

↳ this amounts to applying the permutation matrix  $\mathbf{P}$  on both sides  $\mathbf{P}\mathbf{A}\mathbf{P}^T$

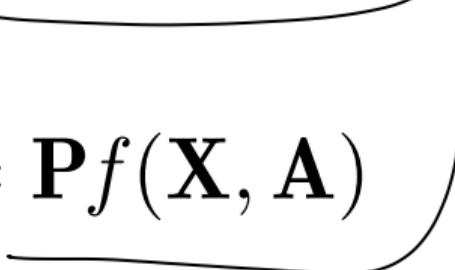
Invariance:

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



Equivariance:

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



## More desirable properties: locality

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

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

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

## More desirable properties: locality

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

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

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

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

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

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

## A recipe for graph neural networks

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

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

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

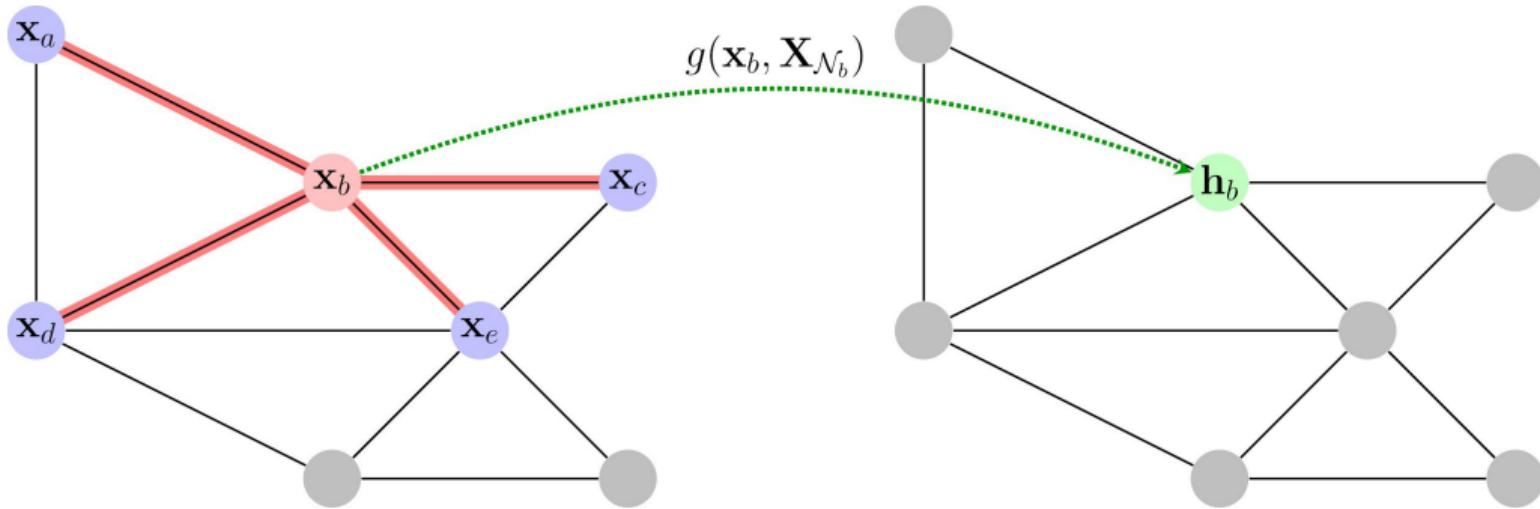
## A recipe for graph neural networks

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

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

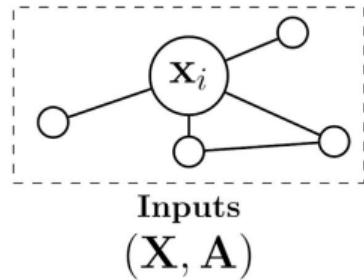
To ensure **equivariance**  $g$  must not depend on the order of the vertices in  $\mathbf{X}_{\mathcal{N}_i}$   
↳  $g$  should be permutation **invariant!**

# A recipe for graph neural networks, visualised

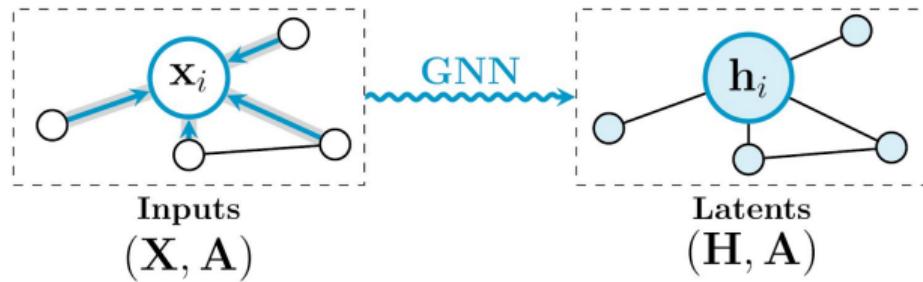


$$\mathbf{X}_{\mathcal{N}_b} = \{\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}\}$$

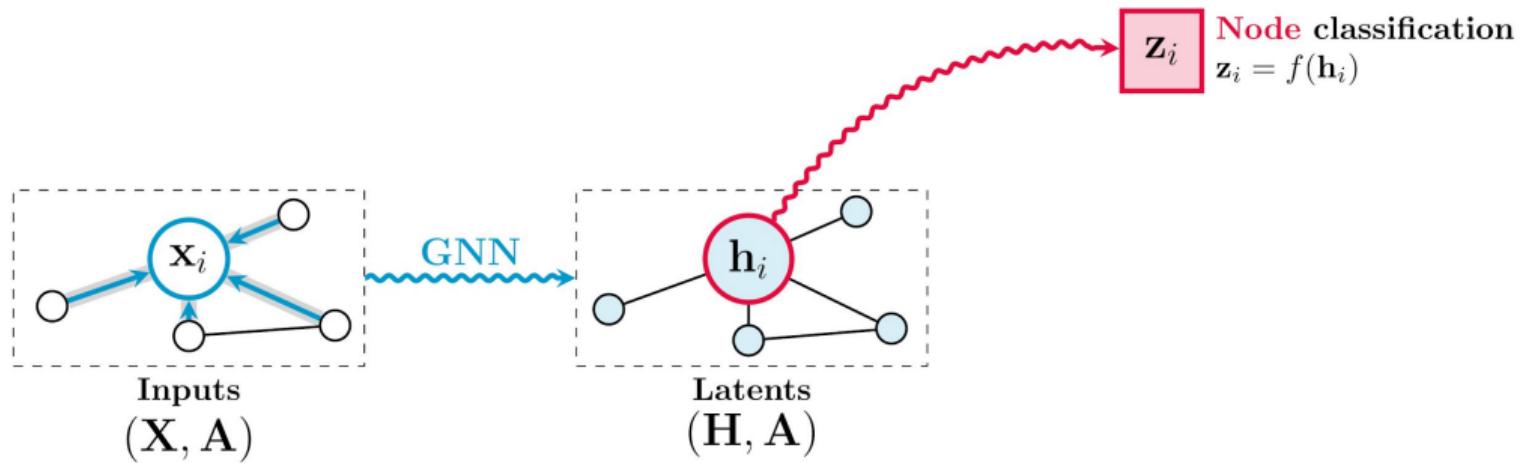
# General blueprint for learning on graphs



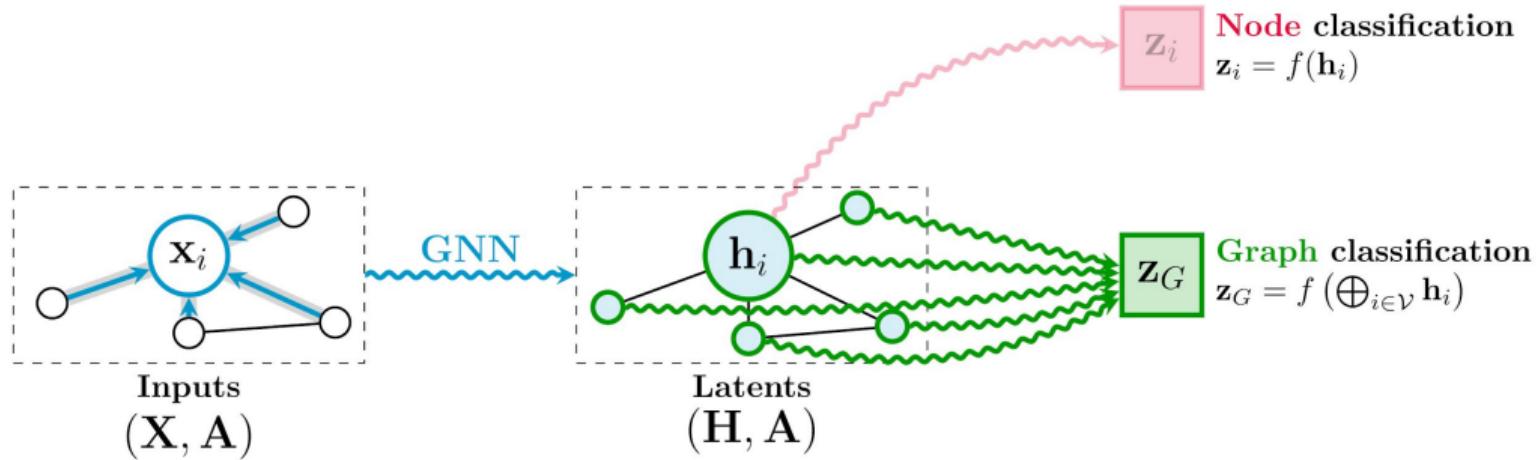
# General blueprint for learning on graphs



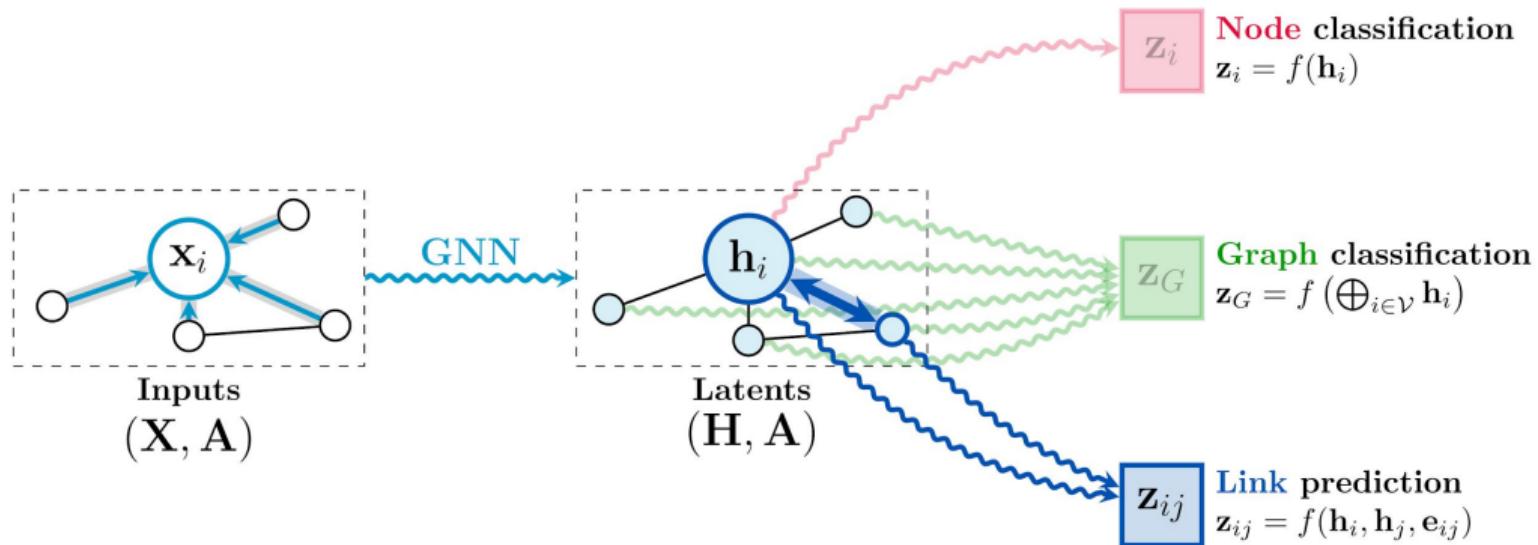
# General blueprint for learning on graphs



# General blueprint for learning on graphs



# General blueprint for learning on graphs



# Message passing on graphs

three flavours for all tastes

## What's in a GNN layer?

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

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

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

## What's in a GNN layer?

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

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

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

Defining a GNN layer  $\iff$  Defining  $g$  function

## What's in a GNN layer?

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

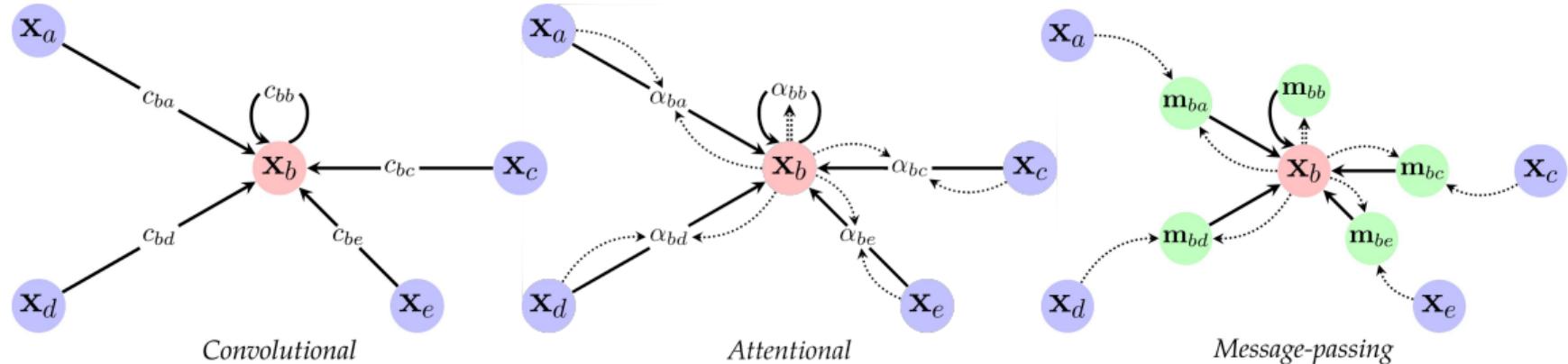
$$f(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} \text{---} & g(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & \text{---} \\ \text{---} & g(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & \text{---} \\ & \vdots & \\ \text{---} & g(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & \text{---} \end{bmatrix}$$

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

Defining a GNN layer  $\iff$  Defining  $g$  function

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

# The three “flavours” of GNN layers



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

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

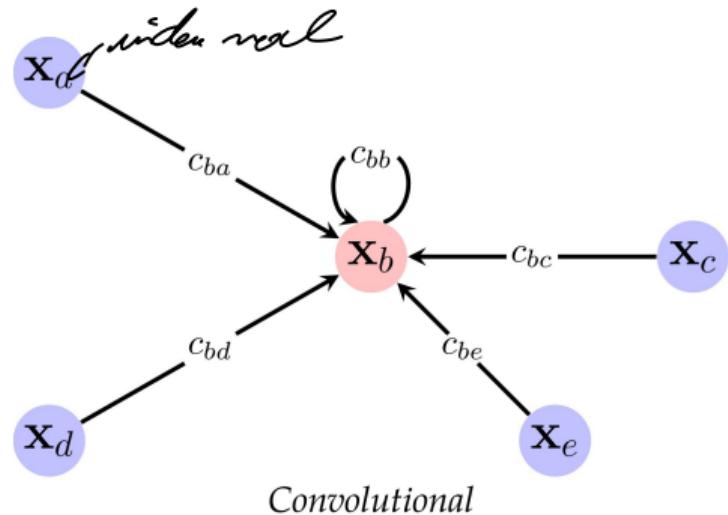
$$\mathbf{h}_i = \phi \left( \mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j) \right)$$

# Convolutional GNNs

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

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

*weights from graph index matrix*  
*MLP:  $\mathbb{R}^d \rightarrow \mathbb{R}^d$*



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**

# Attentional GNNs

Features of neighbours aggregated with implicit weights (via attention)

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

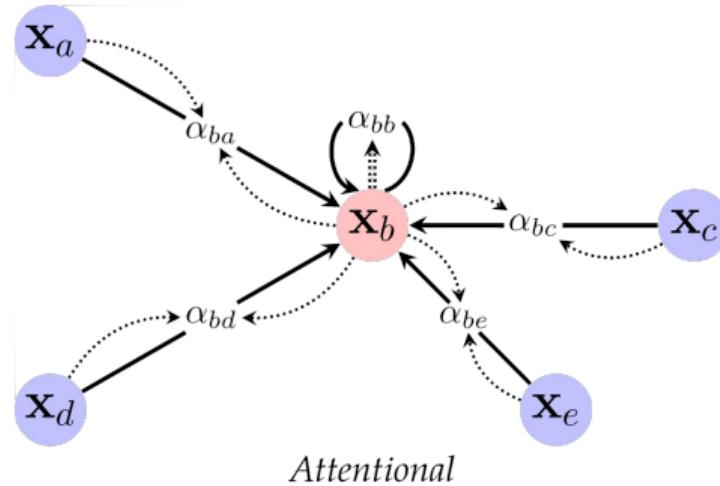
*weights between nodes are parameters of (can be an M(p))*

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

↳ MoNet (Monti et al., CVPR'17)

GAT (Veličković et al., ICLR'18)

GaAN (Zhang et al., UAI'18)



Useful as “middle ground” w.r.t. capacity and scale

↳ Edges still scalars but need not encode homophily

# Message-Passing GNNs

Compute arbitrary vectors (“messages”) to be sent across edges

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

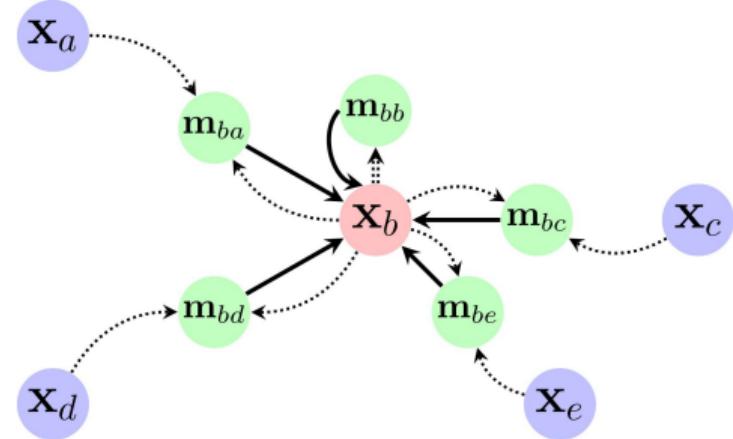
*arbitrarily encoding*

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

↳ Interaction Networks (Battaglia et al., NeurIPS'16)

MPNN (Gilmer et al., ICML'17)

GraphNets (Battaglia et al., 2018)



Most generic GNN layer

↳ May have scalability or learnability issues

Ideal for computational chemistry, reasoning and simulation

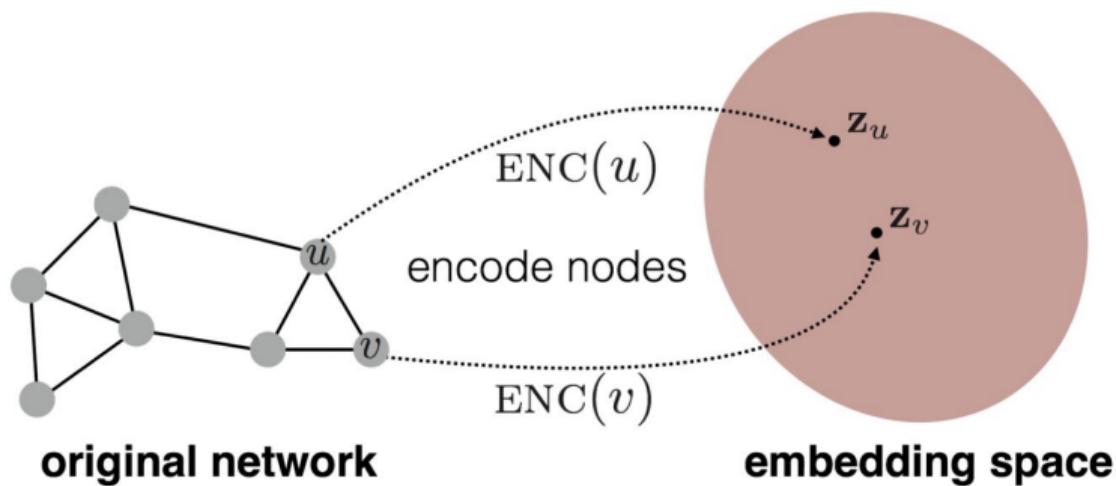
## From ad-hoc node embeddings to GNNs

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

# From ad-hoc node embeddings to GNNs

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

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



## Node embedding as edge presence classification

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

## Node embedding as edge presence classification

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

↳ Can use standard binary cross-entropy loss!

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

## Node embedding as edge presence classification

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

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

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

## Node embedding as edge presence classification

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

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

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

## Node embedding as edge presence classification

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

↳ Can use standard binary cross-entropy loss!

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

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

↳ DeepWalk (Perozzi et al., KDD'14)

node2vec (Grover & Leskovec, KDD'16)

LINE (Tang et al., WWW'15)

## Node embedding as edge presence classification

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

↳ Can use standard binary cross-entropy loss!

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

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

↳ DeepWalk (Perozzi et al., KDD'14)

node2vec (Grover & Leskovec, KDD'16)

LINE (Tang et al., WWW'15)

Not the only time GNNs and NLP's path overlapped

## A note on transformers

Transformers are Graph Neural Networks!

- ↳ Fully-connected graph
- Attentional flavour

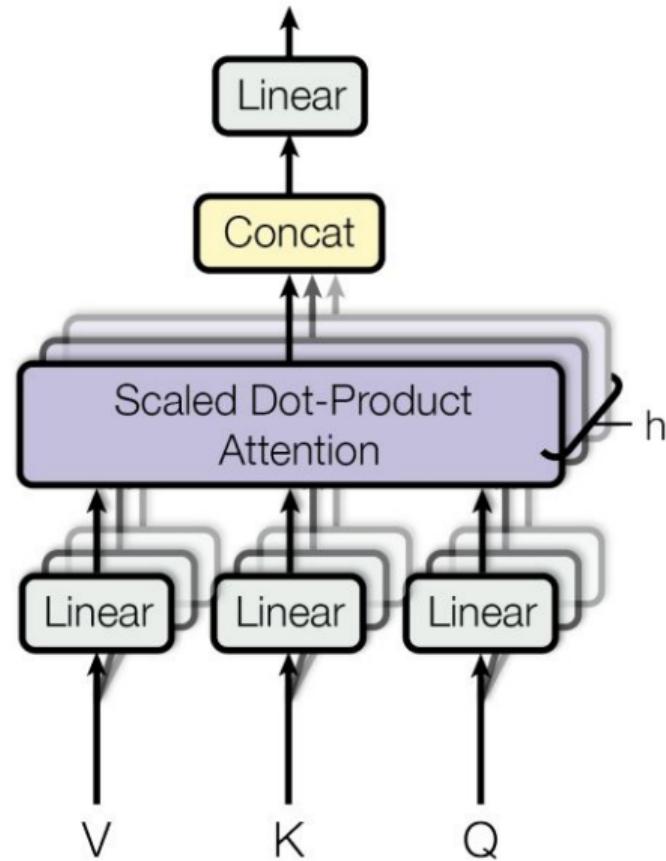
The sequential structural information is injected through the positional embeddings.

Dropping them yields a fully-connected GAT model.

Attention can be seen as inferring soft adjacency.

See Joshi (The Gradient; 2020).

### Multi-Head Attention



# The wild west of heuristics

34v5 [cs.LG] 24 Nov 2021

## Do Transformers Really Perform Bad for Graph Representation?

Chengxuan Ying<sup>1\*</sup>, Tianle Cai<sup>2</sup>, Shengjie Luo<sup>3\*</sup>,

Shuxin Zheng<sup>4†</sup>, Guolin Ke<sup>4</sup>, Di He<sup>4†</sup>, Yanming Shen<sup>1</sup>, Tie-Yan Liu<sup>4</sup>

<sup>1</sup>Dalian University of Technology    <sup>2</sup>Princeton University

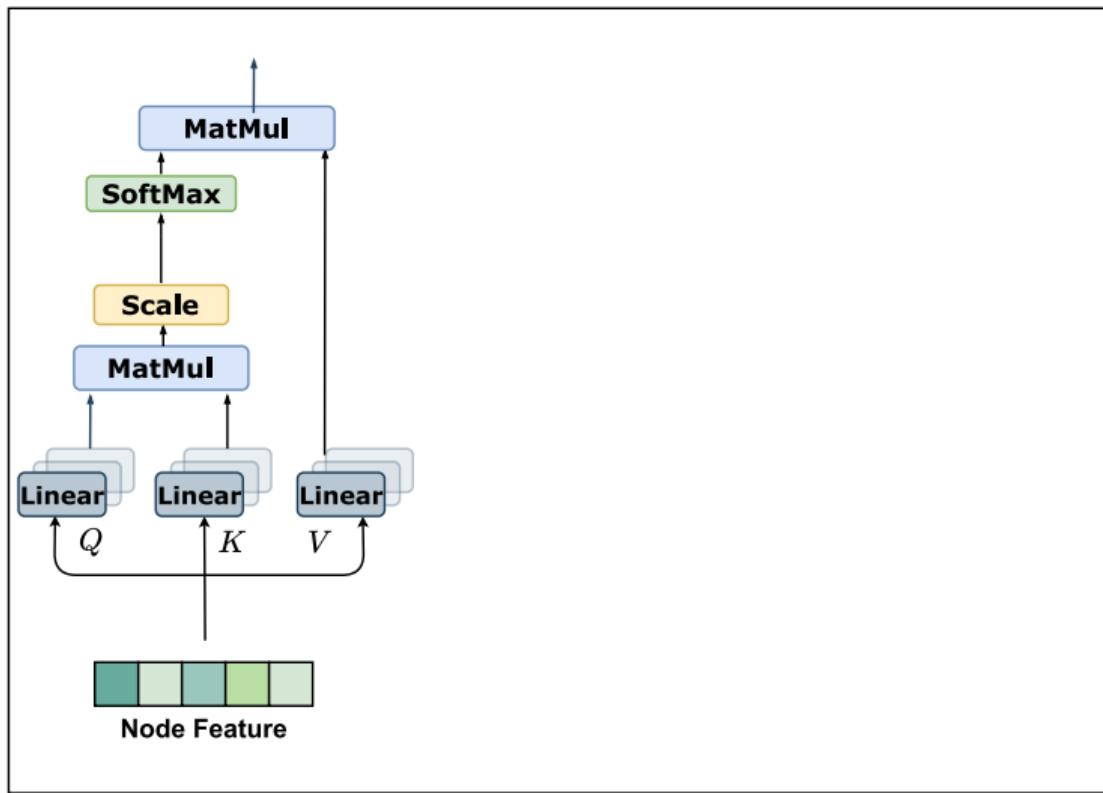
<sup>3</sup>Peking University    <sup>4</sup>Microsoft Research Asia

yingchengsyuan@gmail.com, tianle.cai@princeton.edu, luosj@stu.pku.edu.cn  
{shuz, guoke, dihe, tieliu}@microsoft.com, shen@dlut.edu.cn

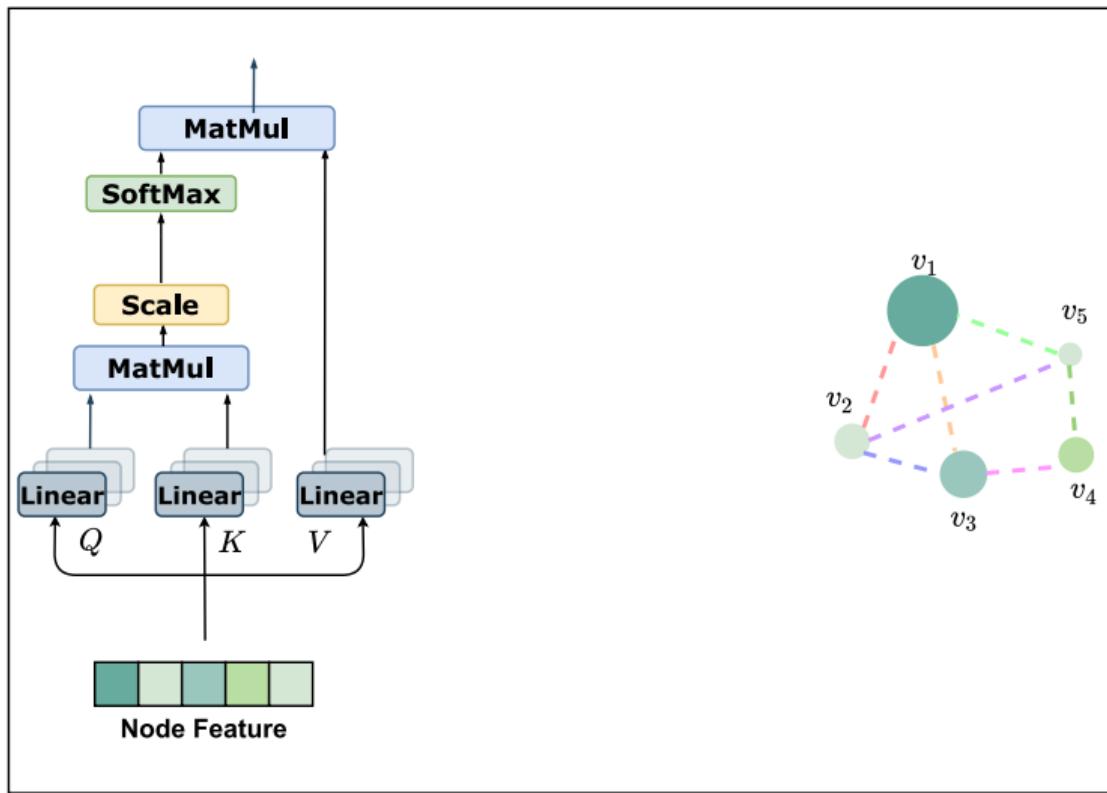
### Abstract

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

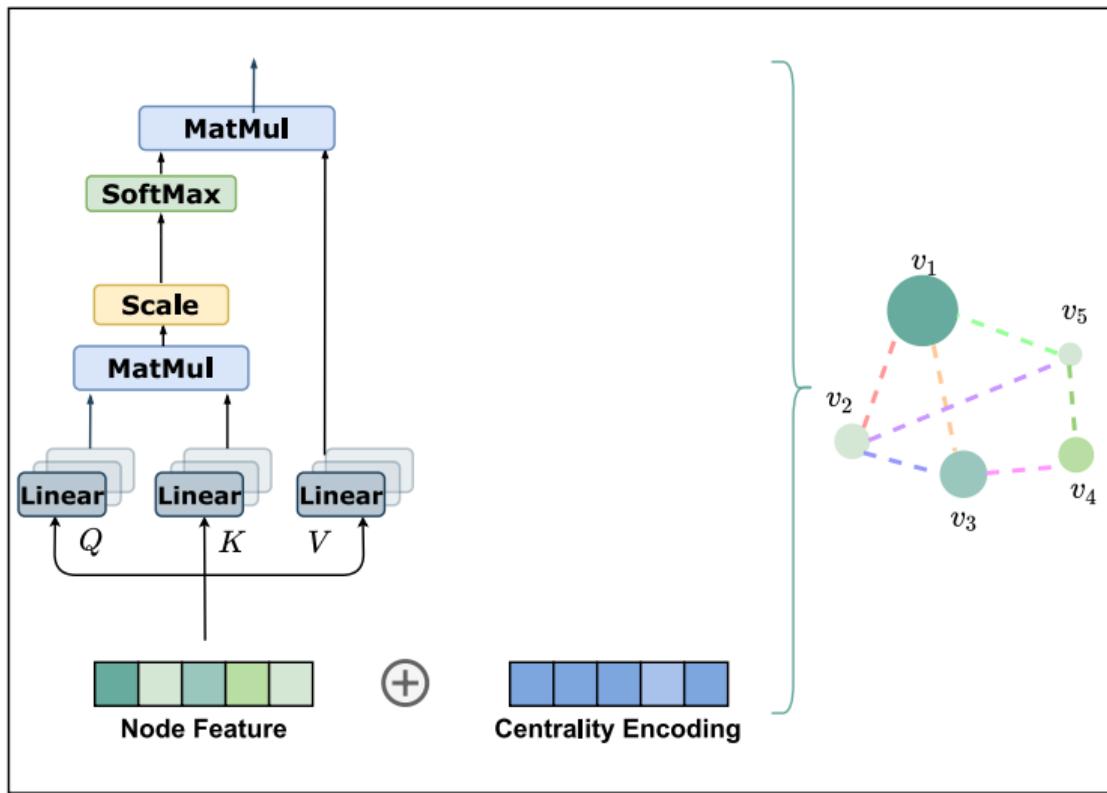
# From Transformer to (a) Graphformer



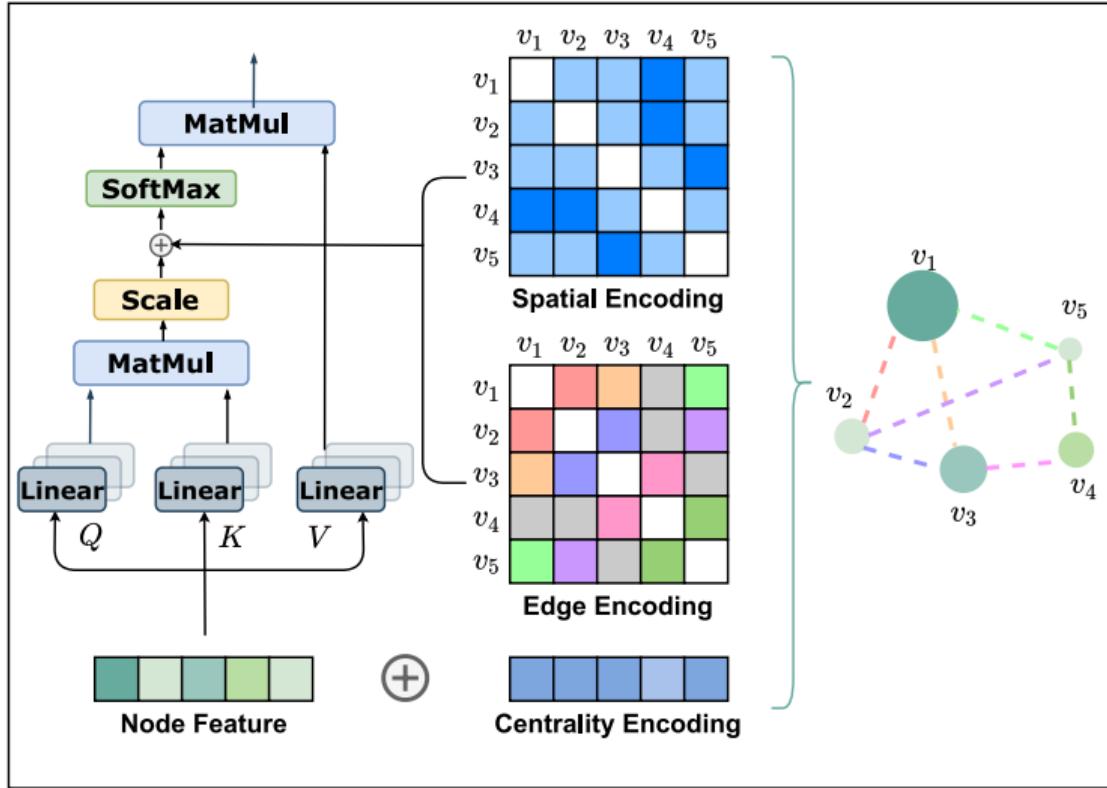
# From Transformer to (a) Graphformer



# From Transformer to (a) Graphformer



# From Transformer to (a) Graphformer



## From Transformer to (a) Graphformer: modifying the input

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

## From Transformer to (a) Graphformer: modifying the attention

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

## From Transformer to (a) Graphformer: modifying the attention

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

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

## From Transformer to (a) Graphformer: modifying the attention

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

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

## From Transformer to (a) Graphformer: modifying the attention

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

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

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

## From Transformer to (a) Graphformer: modifying the attention

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

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

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

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

## From Transformer to (a) Graphformer: modifying the attention

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

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

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

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

↳  $\text{SP} = (e_1, e_2, \dots, e_N)$  is (one of) the shortest path from  $v_i$  to  $v_j$ ,

$\mathbf{x}_{e_n}$  the feature of the  $n$ -th edge  $e_n$  in  $\text{SP}_{ij}$ ,

$\mathbf{w}_n^E \in \mathbb{R}^{d_E}$  the learned  $n$ -th weight embedding,

## From Transformer to (a) Graphformer: why?

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

# From Transformer to (a) Graphformer: why?

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

In practice: wins on OGB!

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

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

# Wrapping up

more applications and open questions

# Bleeding edge applications

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

 Pinterest Engineering [Follow](#)

Aug 15, 2018 · 8 min read

 **amazon | science** 

PUBLICATION

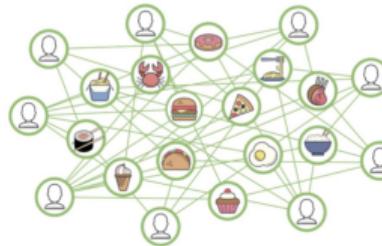
### P-Companion: A principled framework for diversified complementary product recommendation

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, 2015



# Bleeding edge applications

The image shows two news articles side-by-side. The left article is from **nature** magazine, featuring a research paper titled "A graph placement methodology for fast chip design". The right article is from **BBC**, specifically the **TECH** section, about Google using AI to design AI chips.

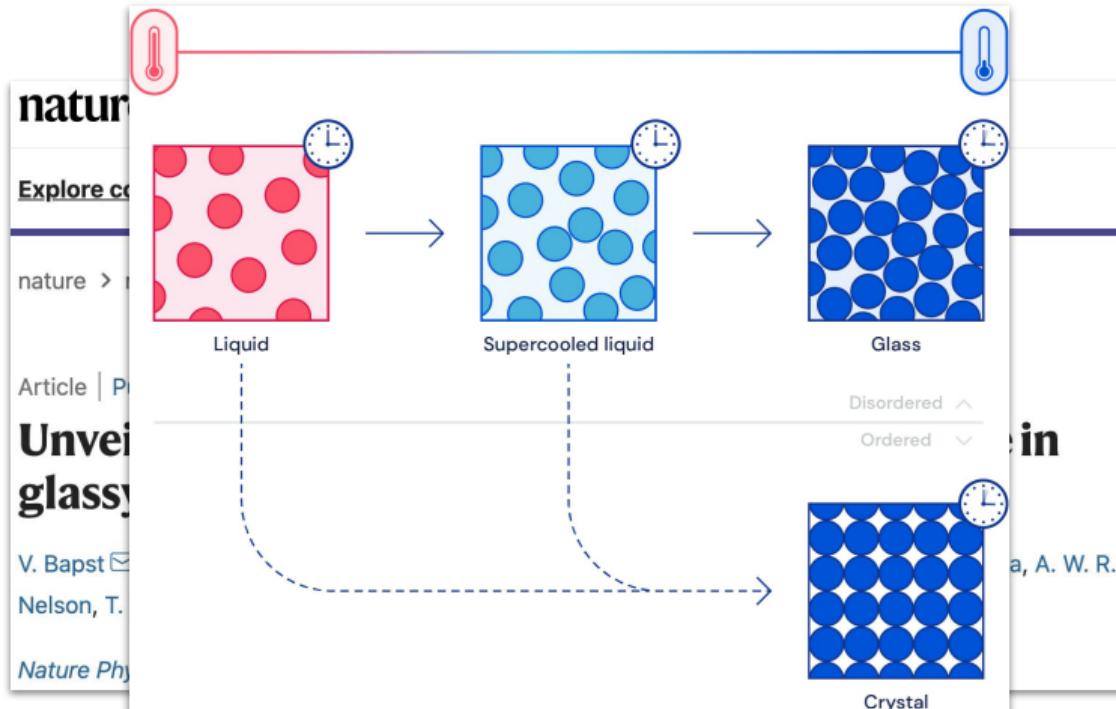
**nature** article details:

- Published: 09 June 2021
- Authors: 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

**BBC Tech article details:**

- Category: GOOGLE \ TECH \ ARTIFICIAL INTELLIGENCE
- Title: **Google is using AI to design its next generation of AI chips more quickly than humans can**
- Text: *Designs that take humans months can be matched or beaten by AI in six hours*
- Author: By James Vincent | Jun 10, 2021, 9:13am EDT

# Bleeding edge applications

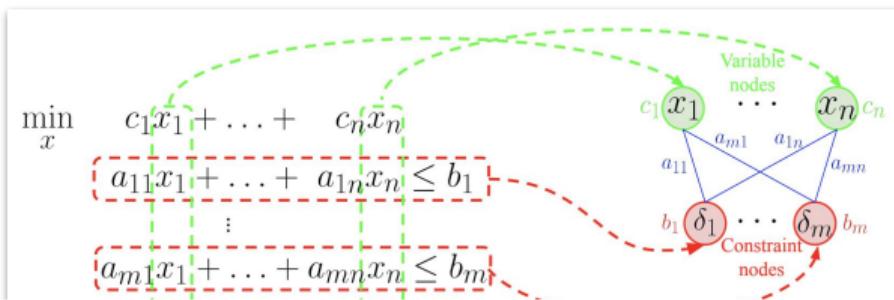


# Bleeding edge applications

## Solving Mixed Integer Programs Using Neural Networks

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

<sup>1</sup>DeepMind, <sup>2</sup>Google Research



## GNNs and beyond

GNNs are flexible blueprints based on few properties:

- ↳ locality, permutation invariance, permutation equivariance

## GNNs and beyond

GNNs are flexible blueprints based on few properties:

↳ locality, permutation invariance, permutation equivariance

Still remarkably powerful!

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

## GNNs and beyond

GNNs are flexible blueprints based on few properties:

↳ locality, permutation invariance, permutation equivariance

Still remarkably powerful!

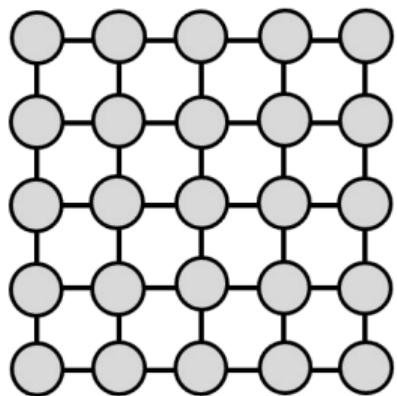
↳ generalization of many standard architectures (including CNNs and Transformers!)

Can we generalize it even further?

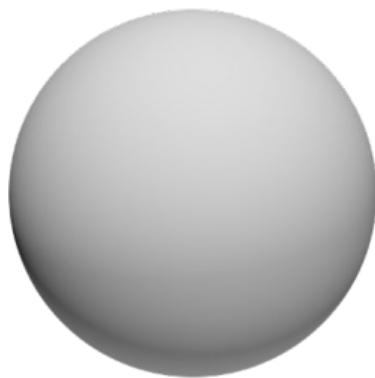
↳ full graph isomorphism

general class of geometric deep learning architectures

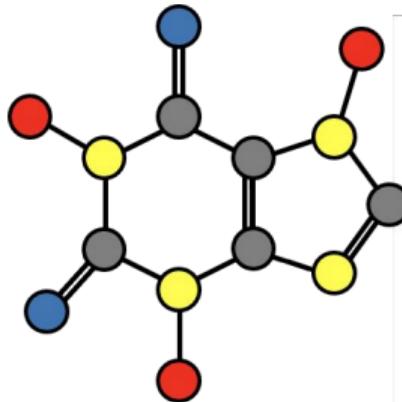
# Geometric deep learning



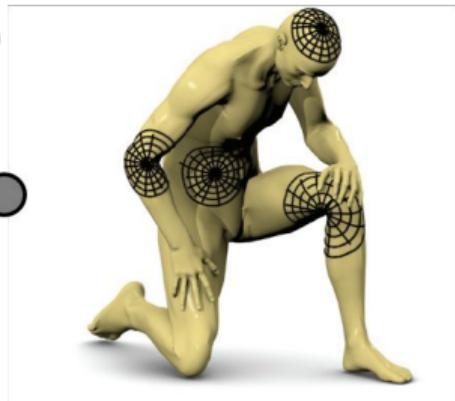
Images &  
Sequences



Homogeneous  
spaces

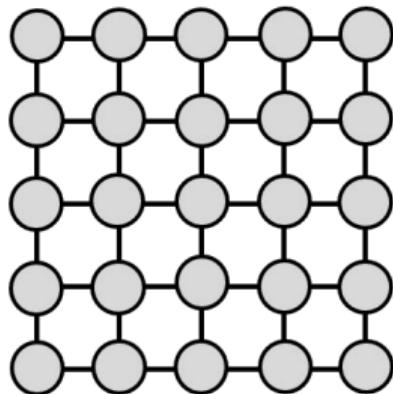


Graphs & Sets

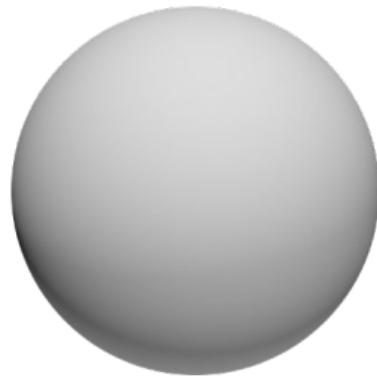


Manifolds, Meshes &  
Geometric graphs

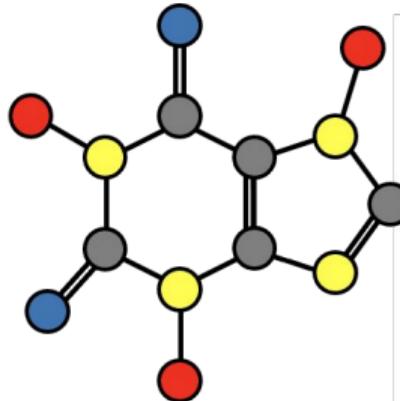
# The five “G”s of geometric deep learning



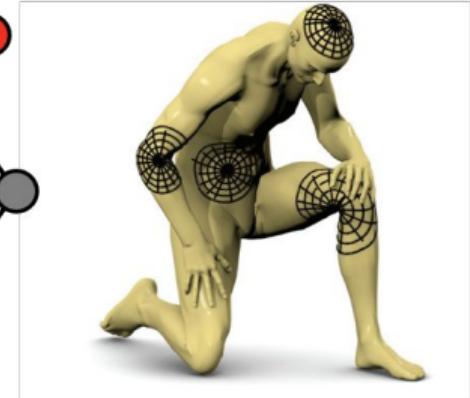
Grids



Groups



Graphs



Geodesics &  
Gauges

# The five “G”s of geometric deep learning

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

Credit to Michael Bronstein

*Daniele Calandriello*

dcalandriello@google.com

ENS Paris-Saclay, MVA 2022/2023

<https://sites.google.com/view/daniele-calandriello/>