Karlsruhe Institute of Technology

Geometric Deep Learning

8. Graphs and Sets II*

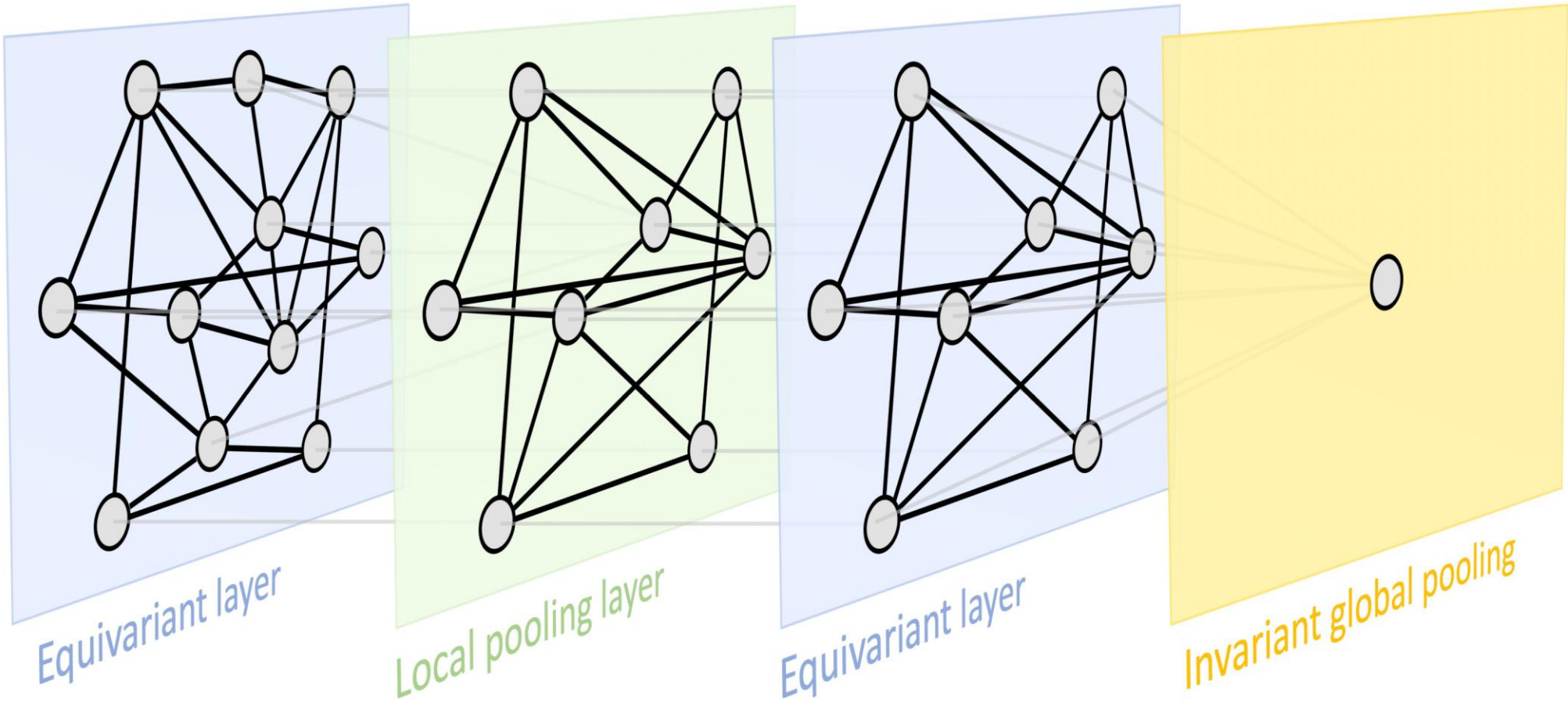
Jun.-Prof. Dr. Jan Stühmer

*Based on the slides by Petar Veličković, AMMI 2022

Karlsruhe Institute of Technology

Based on the slides by Petar Veličković, AMMI 2022
Available at geometricdeeplearning.com

Recap: Blueprint of Geometric Deep Learning



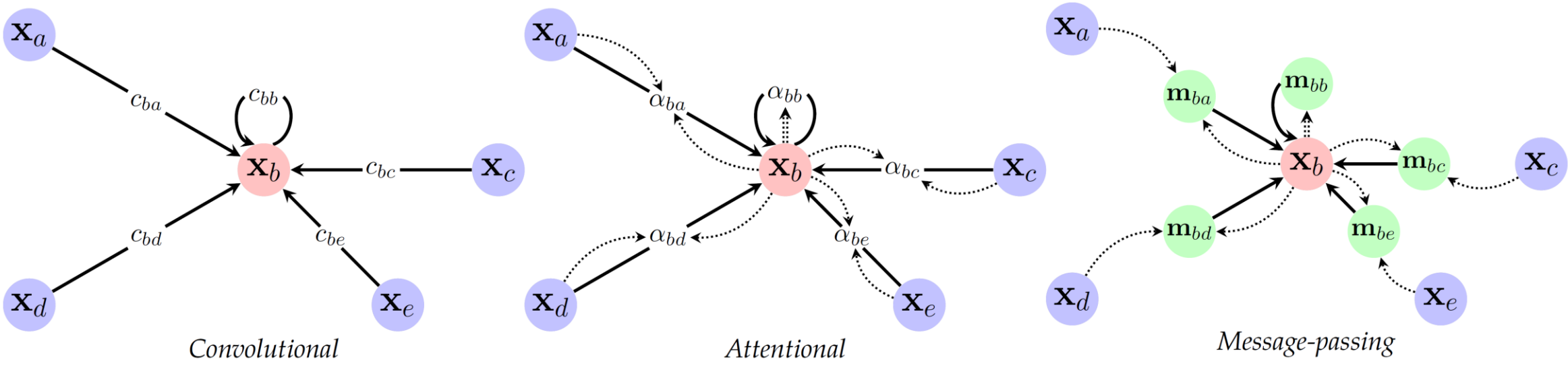
Recap: Building blocks of Geometric Deep Learning

Let Ω and Ω' be domains, $\mathfrak{G}$ a symmetry group over Ω.
Write $\Omega' \subseteq \Omega$ if Ω' can be considered a compact version of Ω.

We define the following building blocks:

- *Linear $\mathfrak{G}$-equivariant layer* $B: \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega', \mathcal{C}')$, satisfying $B(\mathfrak{g}.x) = \mathfrak{g}.B(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$.
- *Nonlinearity* $\sigma: \mathcal{C} \to \mathcal{C}'$ applied element-wise as $\left(\boldsymbol{\sigma}(x)\right)(u): \sigma(x(u))$.
- *Local pooling (coarsening)* $P: \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{X}(\Omega', \mathcal{C})$, such that $\Omega' \subseteq \Omega$.
- *$\mathfrak{G}$-invariant layer (global pooling)* $A: \mathcal{X}(\Omega, \mathcal{C}) \to \mathcal{Y}$, satisfying $A(\mathfrak{g}.x) = A(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, \mathcal{C})$.

Recap: The three “flavors” of GNN layers



[Bronstein, Bruna, Cohen, Veličković, 2021]

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

Graph Networks

General attributed graphs

- Graphs can convey rich information at all granularities
- We previously ignored non-node data for simplicity
- Now we will assume the following possible set of features:
  - *Node* features, $\mathbf{x}_u \in \mathbb{R}^k$ *(e.g. atom type, charge, nb. of hydrogen bonds)*
  - *Edge* features, $\mathbf{x}_{uv} \in \mathbb{R}^l$ *(e.g. bond type, is in a ring?)*
  - *Graph* features $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^m$ *(e.g. molecular weight, fingerprints)*
- Analogously defining latents $\mathbf{h}_u$, $\mathbf{h}_{uv}$, $\mathbf{h}_{\mathcal{G}}$
- It is possible to extend this furthers (e.g. *hypergraphs*)
  but such input can usually be represented as an instance of the above

Graph Networks

- We will now define a general blueprint for a spatial GNN, which generalizes all the flavors referenced before
- We use the Graph Network (Battaglia et al., 2018) as our basis
  This is because it operates over generic attributed graphs
- Dataflow:
  - Update edge features (using graph + relevant nodes)
  - Update node features (using updated relevant edges + graph)
  - Update graph features (using updated nodes + edges)
- and extensive usage of *skip connections!*

Graph Networks

- Update edge features (using graph + relevant nodes)

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

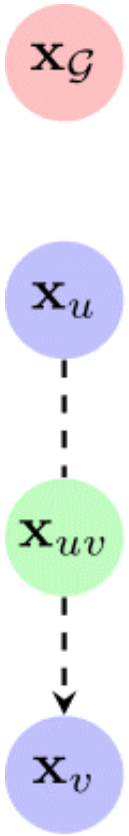
- Update node features (using updated relevant edges + graph)

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

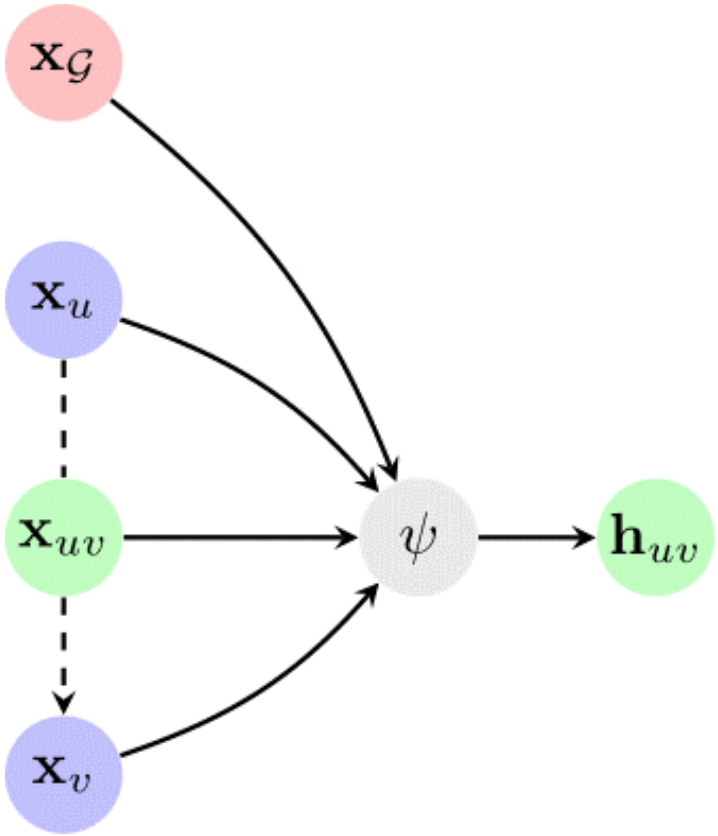
- Update graph features (using updated nodes + edges)

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

Graph Networks, visualized

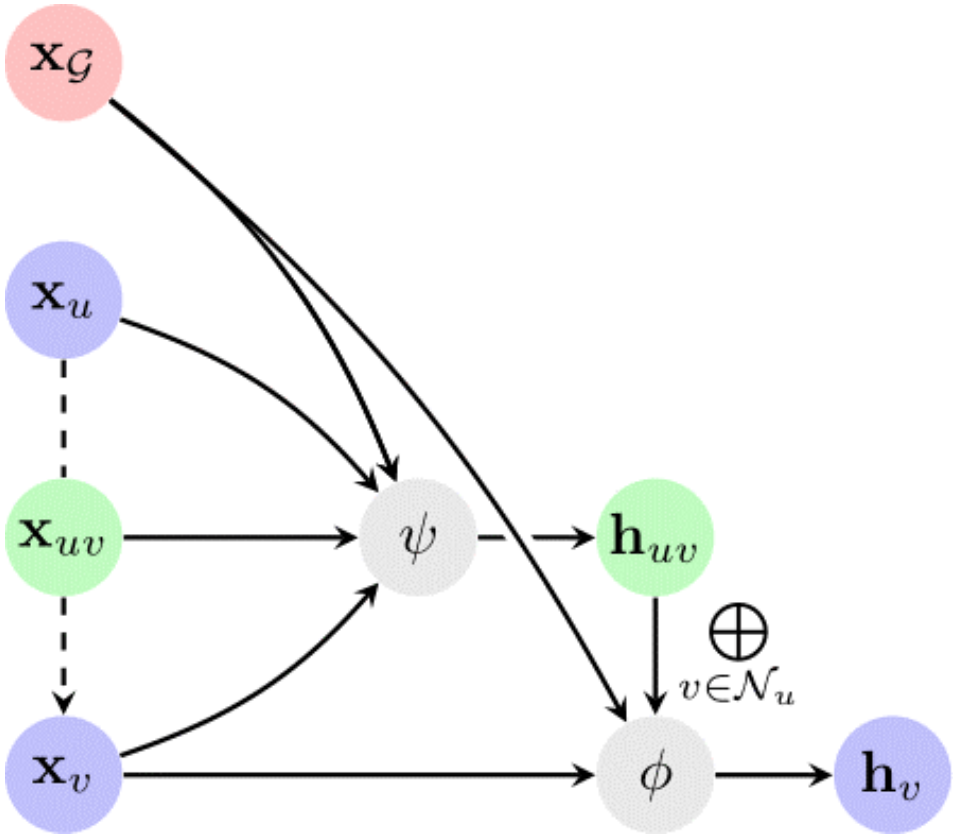


Graph Networks, visualized



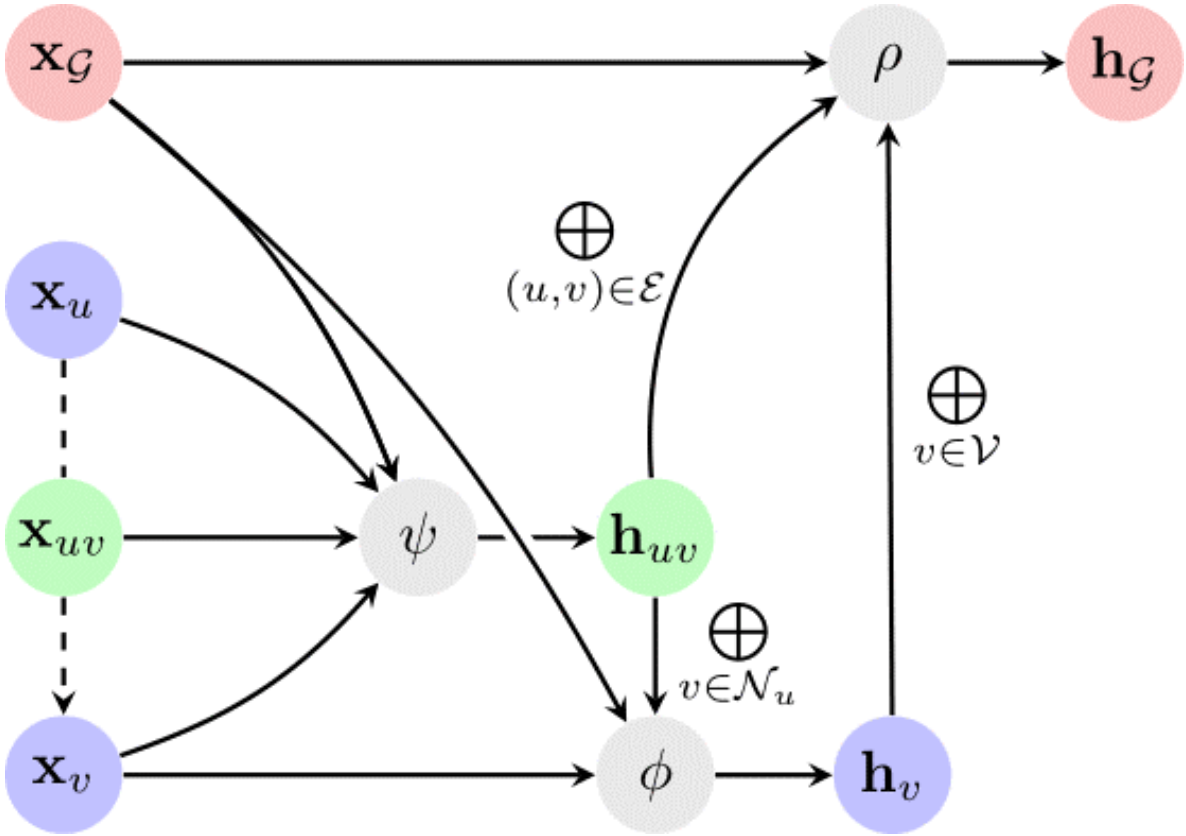
$$\mathbf{h}_{uv} = \psi\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_{uv}, \mathbf{x}_{\mathcal{G}}\right)$$

Graph Networks, visualized



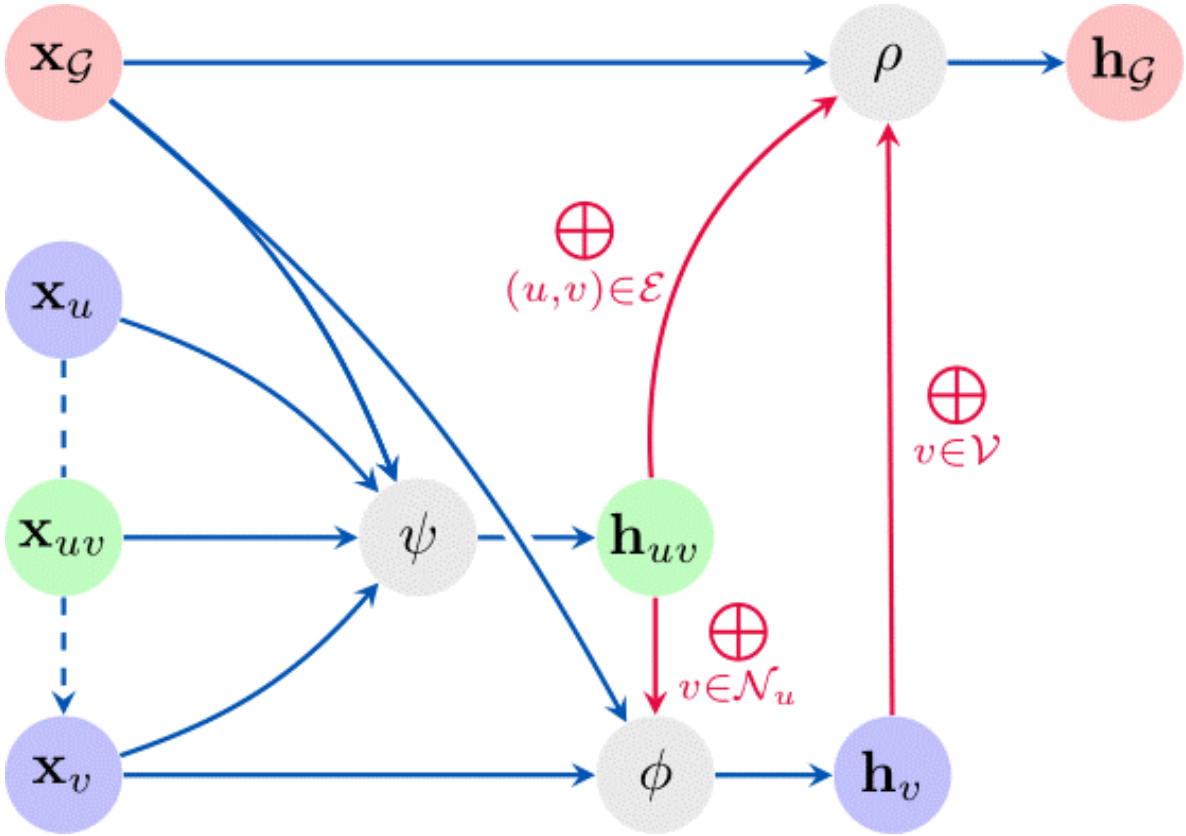
$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{u \in \mathcal{N}_v} \mathbf{h}_{vu}, \mathbf{x}_{\mathcal{G}}\right)$$

Graph Networks, visualized



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

Graph Networks, visualized



Equivariant and invariant layers

Latent Graph Inference

- So far, we've the "ground-truth" graph to be given
- However, this it not always the case
- This leads to laten graph inference, a key emerging area of graph representation learning (and an area of high interest to *science*)
- Further, studying this area will reveal surprising connections between state-of-the-art models and graph neural networks

In the absence of a graph …

- Even when the input graph is absolutely correct, it does not mean it is **optimal** for the task at hand
- Hence, *rewiring* might be a good idea

- Taken to the extreme, *what to do when there is no graph?*
  (but we assume an underlying graph may exist)

- Assume we are given a node feature matrix, $\mathbf{X}$, but no adjacency

- We have seen a solution for this already …

Option 1: Empty edge set

- Assume the graph is completely disconnected
  Let $\mathbf{A} = \mathbf{I}$, or, equivalently, $\mathcal{N}_u = \{u\}$ for all nodes u

- Each element is processed independently:

$$\mathbf{h}_\mathrm{u} = \psi(\mathbf{x}_u)$$

- This, as we have seen, results in a permutation equivariant function over the set

- **Deep Sets** [Zaheer *et al.*, Neurips 2017] implements such an architecture

Option 2: Complete graph

- Assume the graph is *fully connected*
  Let $\mathbf{A} = \mathbf{1}\mathbf{1}^{\mathrm{T}}$, or, equivalently, $\mathcal{N}_u = \mathcal{V}$ for all nodes u

- If we do not model any information about the edges, i.e. we do not have access to any coefficients depending on the vertices interacting along the edges, any convolutional GNN is **equivalent** to Deep Sets:

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

- This is because the edges are indistinguishable, and the complete edge set of the graph permutation invariant

Option 3: Complete graph (with attention)

- Assume the graph is *fully connected*
  Let $\mathbf{A} = \mathbf{1}\mathbf{1}^{\mathbf{T}}$, or, equivalently, $\mathcal{N}_u = \mathcal{V}$ for all nodes u

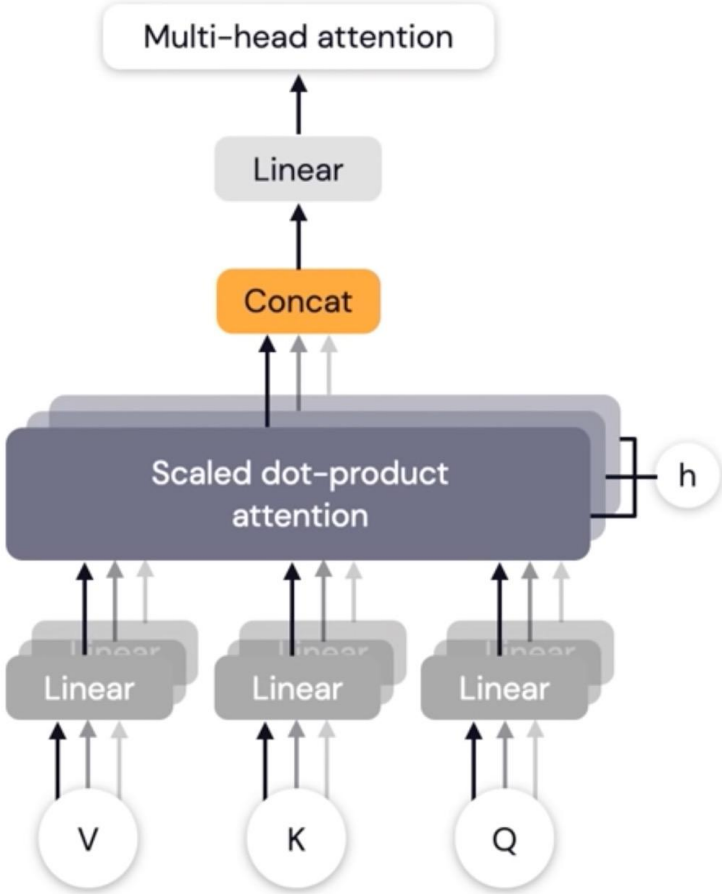
- This motivates the use of a more expressive GNN flavor, the attentional,

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

  which yields the *self attention* operator, the core of the Transformer architecture [Vaswani *et al.* Neurips 2017]

A Note on Transformers

- Transformers **are** Graph Neural Networks:
  - Fully-connected graph
  - Attentional “flavor”
- The sequential structural information is injected through the **positional embeddings**
- Dropping them yields a fully-connected Graph Attention Network (GAT)
- Attention amounts to inferring **soft adjacency**



Option 4: Complete graph with message passing

- When nodes interact in more complex ways (e.g. physics, relational reasoning), the message passing flavor may be suitable
- For physics modelling: Interaction Nets [Battaglia *et al.*, NeurIPS'16] treat every interacting body as a node in a fully connected graph
- For visual reasoning: Relational Networks [Santoro *et al.*, NeurIPS'17] treat every pixel representation (after CNN as feature extractor) as a node in a fully connected graph

Empty edge set vs Fully connected graph

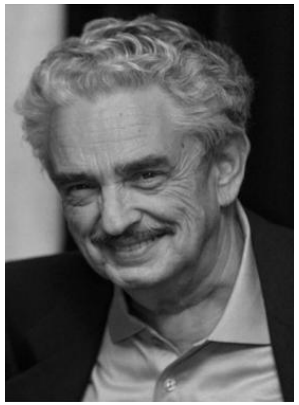
- Empty graph ignores a potential wealth of information
- Full graph can be hard to scale (quadratic space, large neighborhoods)
- Ideally, want to infer the adjacency A, commonly termed “latent graph inference”
- Five recently proposed prominent ways to do so:
  - Variational [Kipf, Fetaya *et al.*, ICML’18] NRI
  - No learning [Wang *et al.*, ACM TOG’18] DGCNN
  - Reinforcement Learning [Kazi *et al.*, 2020] DGM
  - Supervised [Veličković *et al.*, NeurIPS’20] PGN
  - Self-supervised [Fatemi *et al.*, NeurIPS’21] SLAPS

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*
- Can the GNN **distinguish** two *non*-isomorphic graphs? $\mathbf{h}_{\mathcal{G}_1} \neq \mathbf{h}_{\mathcal{G}_1}$
  If not, any kind of task discriminating them is hopeless
- Assess power of GNNs by which graphs they are able to distinguish.
- To formalise this, we will study a popular graph isomorphism test

Weisfeiler-Lehman Test

- Simple but powerful way of distinguishing: pass random hashes of sums along edges
- Iterate until hashes don’t change
  “Possibly isomorphic” if histograms are same



A. Lehman

B. Weisfeiler

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.

Выдвигаются и обсуждаются 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, ведущих 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 $a \neq b$ несравнимы следует, что существует автоморфизм графа, переводящий вершину a в b и сохраняющий отношение смежности.

В п. 6, 7 описан процесс приведения графа к каноническому виду, состоящий в поэтапном переупорядочении 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 $v_i=(i, v_{i1}, v_{i2}, \ldots)$, где v_{ik} — число соседей k-го класса у i-ой вершины, i — 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 не позже, чем через n 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 $x_1, x_2, \ldots$, упорядочив их в соответствии с порядком среди 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 ввести свои независимые переменные и т. д. (точное определение этих операций см. п. 6, операции α_2, β_2).

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

Для дальнейшего разбиения вершин на классы рассмотрим элемент u_{ij} матрицы $U=X\cdot X'$, где X' — матрица, полученная из X заменой переменных $x_1, x_2, \ldots$ переменными $x'_1, x'_2, \ldots$, причем все переменные $x_1, x_2, \ldots, x'_1, x'_2, \ldots$ независимы. Элемент u_{ij} 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 применять все описанные выше операции и т. д., до тех пор, пока и этот процесс не оборвется (см. п. 6, операции α_3, β_3, β_4).

2. Геометрически введение независимых переменных и матрицы $U=X\cdot X'$ означает следующее. Если наряду с ребрами данного графа Γ рассматривать ребра дополнительного графа $\bar{\Gamma}$, то на первом же шаге нашей процедуры этим ребрам будут соответствовать наперед разные переменные — 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 $a_{ij} \in A^k$, $A=A(\Gamma)$, где Γ — неориентированный граф без кратных связей, равен числу путей длины 2, ведущих из i-й вершины в j-ую. Аналогично, коэффициент при $x_k x_l$ в многочлене $u_{ij} \in U=X\cdot X'$ равен числу путей, ведущих из i-й вершины в j-ую по ребрам сначала k-го, а затем l-го цвета.

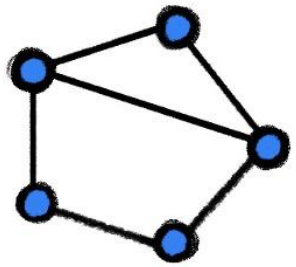
3). Дальнейшая процедура приведения графа к каноническому виду использует применение описанных выше операций к матрице, полученной из $X(\Gamma)$ вычеркиванием некоторого ее столбца и соответствующей строки. Если для матриц порядка $k \leqslant n-1$ приведение к каноническому виду определено, то возникает новая возможность для разбиения вершин на классы: старшей считается та вершина, после вычеркивания которой получается лексикографически старший канонический вид оставшегося графа (см. операция α_3, β_3, М. п. 7). Очевидно, что и такое разбиение на некотором шаге оборвется. 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 и сохраняющий 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 метричной алгебры $\mathfrak{A}(\Gamma)$, т. е. если задано кольцо K (например, кольцо целых чисел Z или поле рациональных чисел Q), то множество матриц, полученных 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

12 НТИ . СЕР. 2 . № 9 . 1968 . ИНФОРМАЦИОННЫЙ АНАЛИЗ

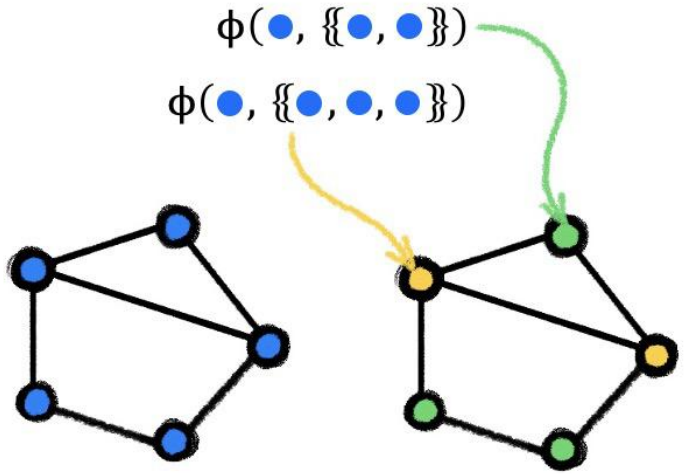
Weisfeiler, Lehman 1968; Portraits: Ihor Gorskiy

Weisfeiler-Lehman Test



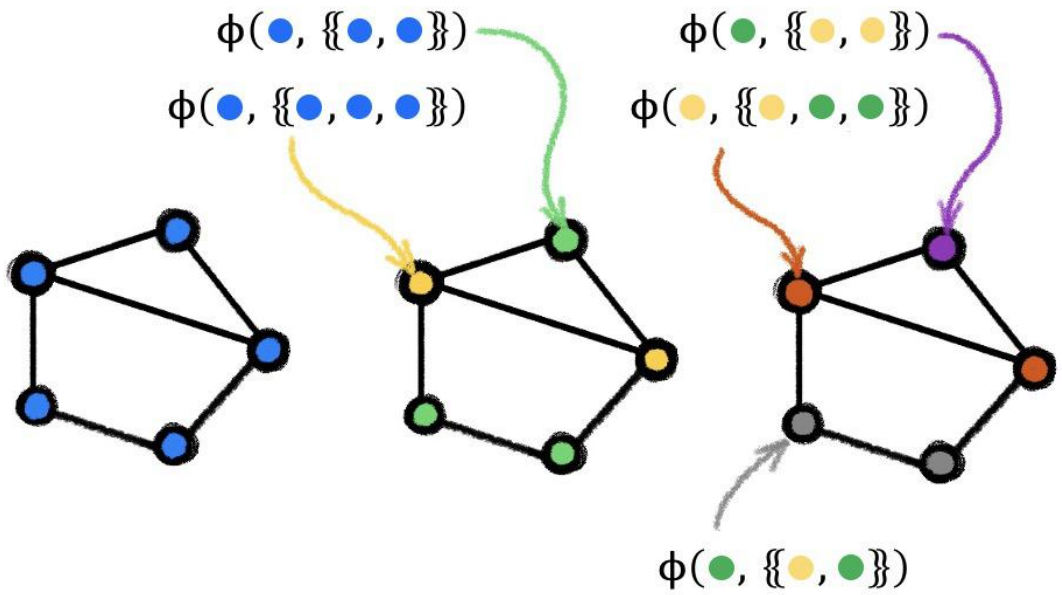
Weisfeiler, Lehman 1968

Weisfeiler-Lehman Test



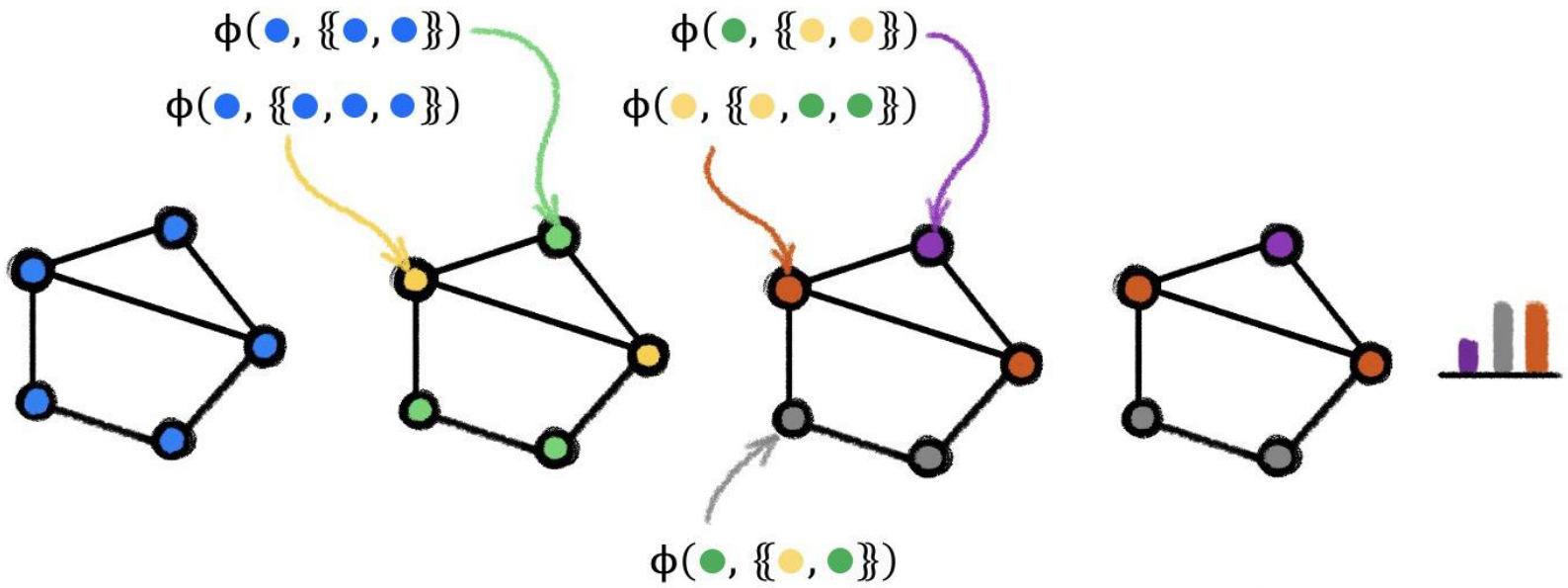
Weisfeiler, Lehman 1968

Weisfeiler-Lehman Test



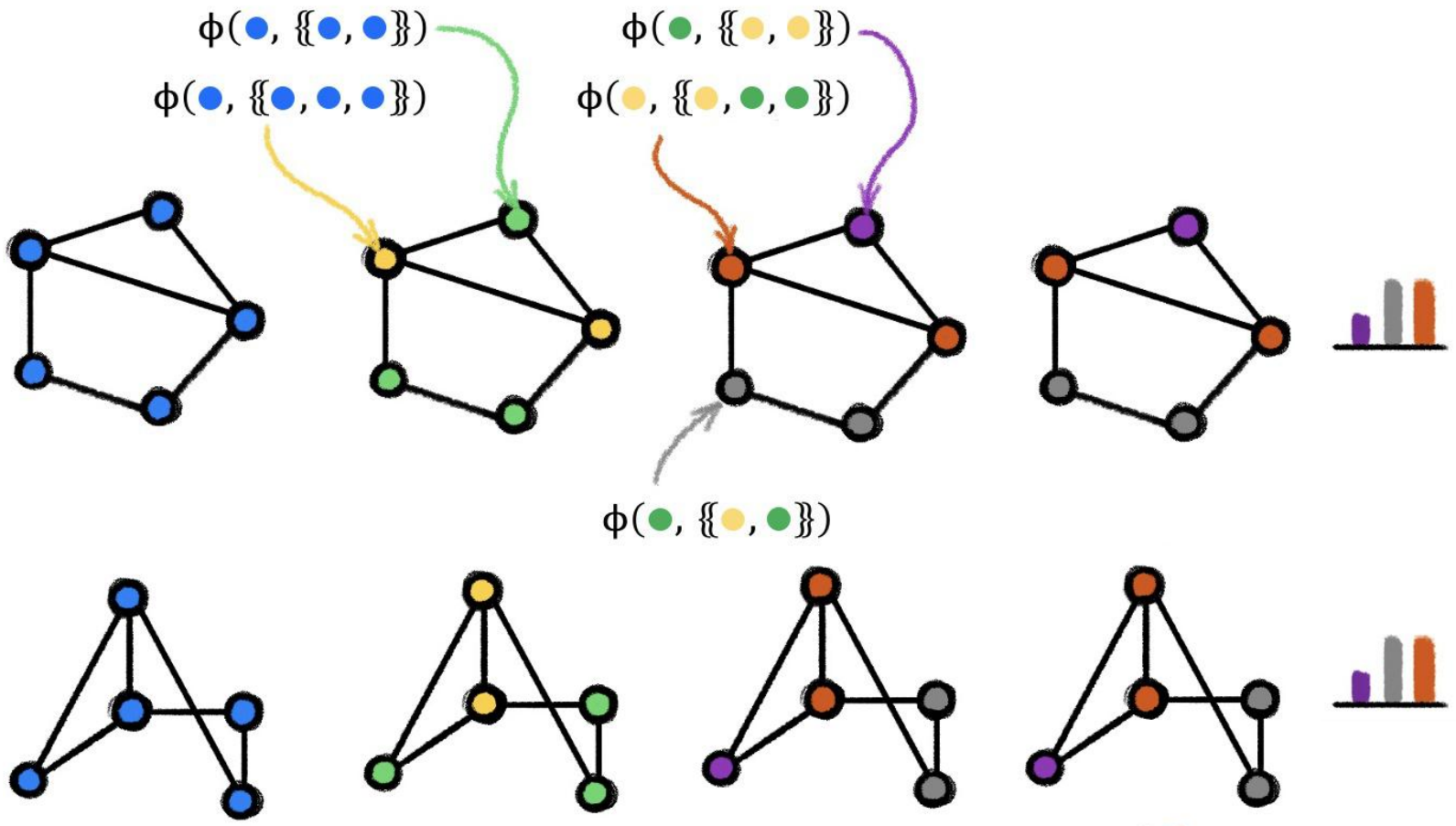
Weisfeiler, Lehman 1968

Weisfeiler-Lehman Test



Weisfeiler, Lehman 1968

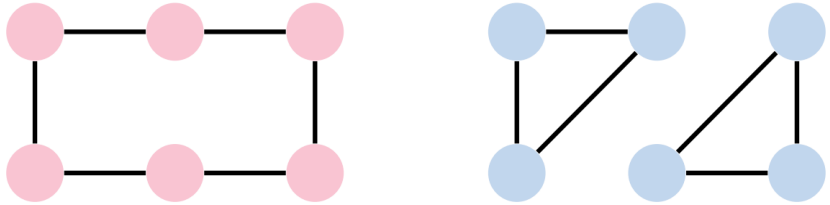
Weisfeiler-Lehman Test



Weisfeiler, Lehman 1968

Weisfeiler-Lehman Test

- The connection to GNNs was made already early, e.g. in GCN
- **Untrained** GNNs can work very well: Random features are equivalent to hash in Weisfeiler-Lehman test
- The test only provides a necessary but insufficient condition for graph isometry and can fail, e.g. for



Expressivity of GNNs

- For discrete node features, GNNs can only be **as powerful** as the Weisfeiler-Lehman test

- Condition for maximal expressivity is an *injective* aggregation (e.g. sum)
  The proof is similar to one used to show Deep Sets universality

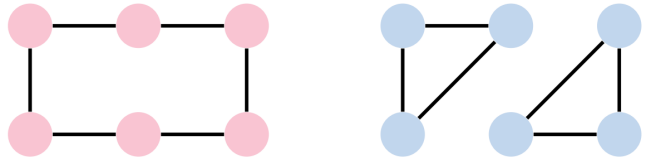
- Graph isomorphism networks [Xu *et al.* ICLR 2019] propose a simple maximally-expressive GNN

$$\mathbf{h}_v = \phi\left((1+\epsilon)\mathbf{h}_v + \sum_{u \in \mathcal{N}_v} \mathbf{h}_u\right)$$

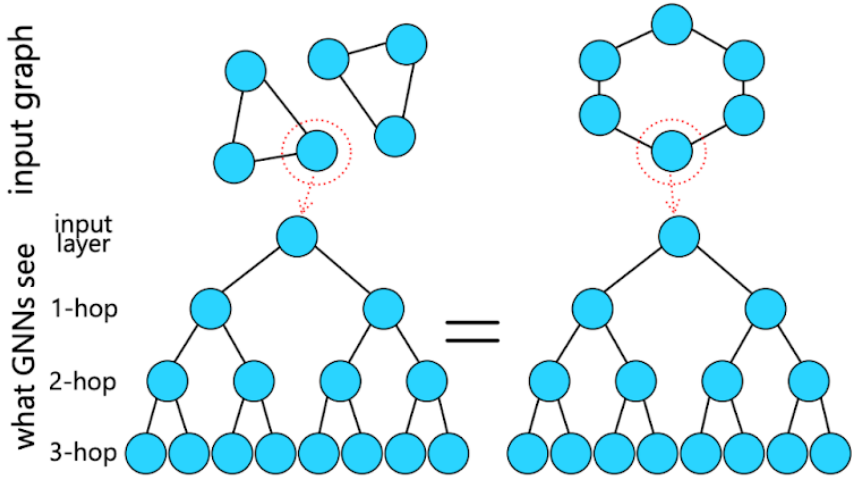
  The real valued coefficient ϵ results in a node-individual feature

Augmented Message Passing

- We can improve GNNs by analyzing the **failure cases** of Weisfeiler-Lehman



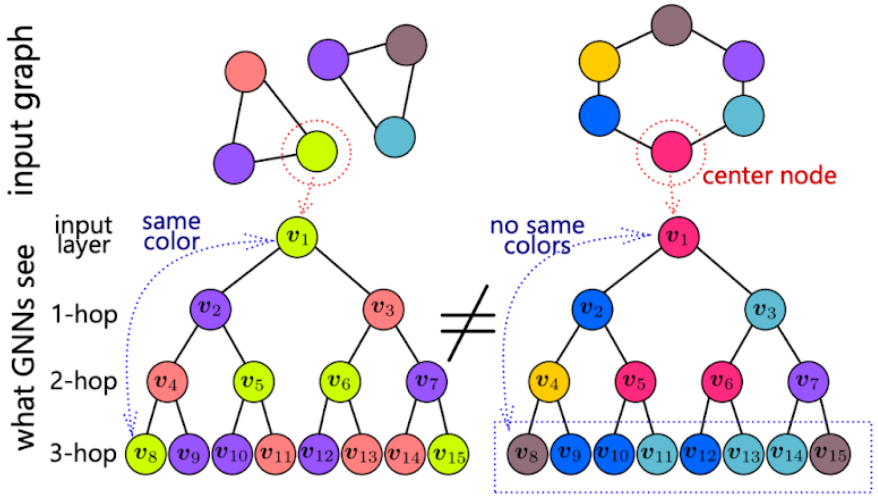
- For example, like Weisfeiler-Lehman, GNNs cannot detect closed triangles



Augmented Message Passing

- We can improve GNNs by analyzing the **failure cases** of Weisfeiler-Lehman

- Augment nodes with randomized features [Sato *et al.* 2021]
- Now a node can "see itself" k hops away!



(b) Random Features.

Augmented Message Passing

- We can improve GNNs by analyzing the **failure cases** of Weisfeiler-Lehman

- Augment nodes with randomized features [Sato *et al.* 2021]
  See also: RP-GNN [Murphy *et al.* ICML'19], P-GNN [You *et al.* ICML'19]

- These methods belong to *augmented message passing*
- They can be broadly categorized into three groups
  - Modifying **features** (as above)
  - Modifying the **message-passing rule**, e.g. DGN [Beaini *et al.* ICML'20]
  - Modifying the **graph** structure, e.g. 1-2-3-GNNs [Morris et al. AAAI'19]

What have we covered?

- A deep dive into graph neural networks (GNNs)
- The Graph Network architecture over generic attributed graphs
- How do *Transformers* and *Deep Sets* fit within the GNN flavors
- (Breaking) the limits of GNN expressive power: *graph isomorphism testing, augmented message passing*