Foundations of Graph Neural Networks
(A Logician's View)

Egor V. Kostylev
University of Oslo
egork@ifi.uio.no

Open lectures for PhD students in computer science
Otwarte wykłady dla doktorantów informatyki

21-23 May 2025

Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw

Graph Love

We, logic-minded people, love graphs:

- graphs are a very nice and neat data model, useful both in theory and practice

Graph Love

We, logic-minded people, love graphs:

- graphs are a very nice and neat data model, useful both in theory and practice

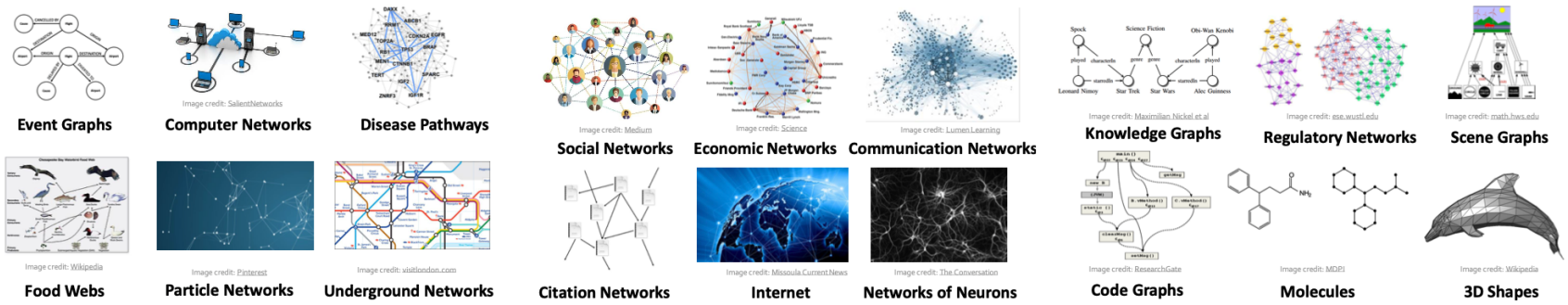
Machine-Learning people also love graphs in recent years:

- they understood that usual NNs are not good enough for (abundant) problems where objects are naturally graph-structured
- and invented Graph Neural Networks (GNNs) [SGT+08]

Graphs are Everywhere

Now days, graph learning with GNNs is one of hot topics in machine-learning community:

- hundreds of papers every year
- applied to many problems (where it should and should not)



Source Images: Machine Learning with Graphs course from Jure Leskovec `cs224w.stanford.edu`

Goals of This Lecture

To understand what are GNNs

- it is actually a rich family of formalisms, not a single thing
- new GNN architectures pop up every week

To understand what GNNs have in common with classic CS formalisms (isomorphism tests, bisimulations, logics)

- Surprisingly, a lot, even beyond just a basic idea that they both do something with graphs
- This is currently quite an active research area, with several deep and interesting papers appearing every year

Goals of This Lecture

To understand what are GNNs

- it is actually a rich family of formalisms, not a single thing
- new GNN architectures pop up every week

To understand what GNNs have in common with classic CS formalisms (isomorphism tests, bisimulations, logics)

- Surprisingly, a lot, even beyond just a basic idea that they both do something with graphs
- This is currently quite an active research area, with several deep and interesting papers appearing every year
- Unfortunately, the area is quite a big hype: many more not-so-good papers also appear, and it may be difficult to find the gems

To see how this can help us learn useful things about GNNs

- what they can and cannot do (for various notions of expressibility)

My Assumptions about You

I assume that you know First Order Logic and basics of ML&NNs

My Assumptions about You

I assume that you know First Order Logic and basics of ML&NNs

Knowledge of Modal/Description Logics and (U)CQs is also desirable, but I will introduce them in terms of FO

Knowledge of GNNs is also welcome, but if you know too much, my lecture may be boring for you

The Plan

1. Graphs and Graph Learning
2. Distinguishing Power of GNNs
   (vs. WL-test)
3. (Uniform) Expressive Power
   (vs. bisimulation and logic)
4. Non-uniform Expressive and Approximation Power,
   other considerations

The Plan

1. Graphs and Graph Learning — Wednesday
2. Distinguishing Power of GNNs (vs. WL-test) — ∼Wednesday
3. (Uniform) Expressive Power (vs. bisimulation and logic) — ∼Thursday
4. Non-uniform Expressive and Approximation Power, other considerations — ∼Friday

The Plan

1. Graphs and Graph Learning
2. Distinguishing Power of GNNs
   (vs. WL-test) ∼Wednesday
3. (Uniform) Expressive Power
   (vs. bisimulation and logic) ∼Thursday
4. Non-uniform Expressive and Approximation Power,
   other considerations ∼Friday

+ tutorials on Thursday and Friday

1. Preliminaries: Graphs and Graph Learning

Graphs

Definition

A *graph* $G = (V, E, \lambda)$ with dimension $d \in \mathbb{N}$ has

- a set V of nodes
- a set E of undirected edges—that is, unordered pairs of (distinct) elements in V
- a node labelling $\lambda : V \to \mathbb{R}^d$

Observations:

- undirected
- edges are not labelled (and of one type)
- simple (no self-loops, repeated edges, etc)

This is for simplicity (and customary in ML):
nearly all of the results we discuss generalise to other graphs

Graph Learning: Graph Embeddings

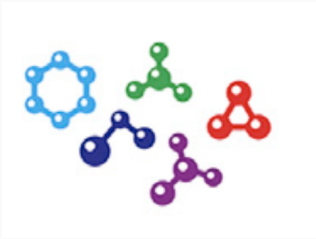
Graph learning is about learning (partially) unknown graph embeddings:

- Let $\mathcal{G}$ be the class of all graphs (of some given dimension d)
- Let $\mathbb{Y}$ be an output space
- A graph embedding is a function of the form

$$\xi : \mathcal{G} \to \mathbb{Y}$$

Example: Prediction of chemical/medical property of molecules

$$\xi : \quad \to \{YES, NO\}$$



Note: embedding are invariant under permutations of the nodes and edges—that is, under representation of the graphs (or isomorphisms)

Graph Learning: Node Embeddings

Also, graph learning is about learning partially unknown node embeddings:

- Let $\mathcal{G}$ be the class of all graphs (of some given dimension d)
- Let $\mathcal{GV}$ be the class of all pairs (G, v), where
  $$G = (V, E, \lambda) \in \mathcal{G} \text{ and } v \in V$$
- Let $\mathbb{Y}$ be an output space
- A node embedding is a function of the form
  $$\xi : \mathcal{GV} \to \mathbb{Y}$$

Example: prediction of the relevance of a paper in citation network

- Given one (or several) networks,
  predict a number in $[0, 1]$ for every paper

Graph Learning: Node Embeddings

Also, graph learning is about learning partially unknown node embeddings:

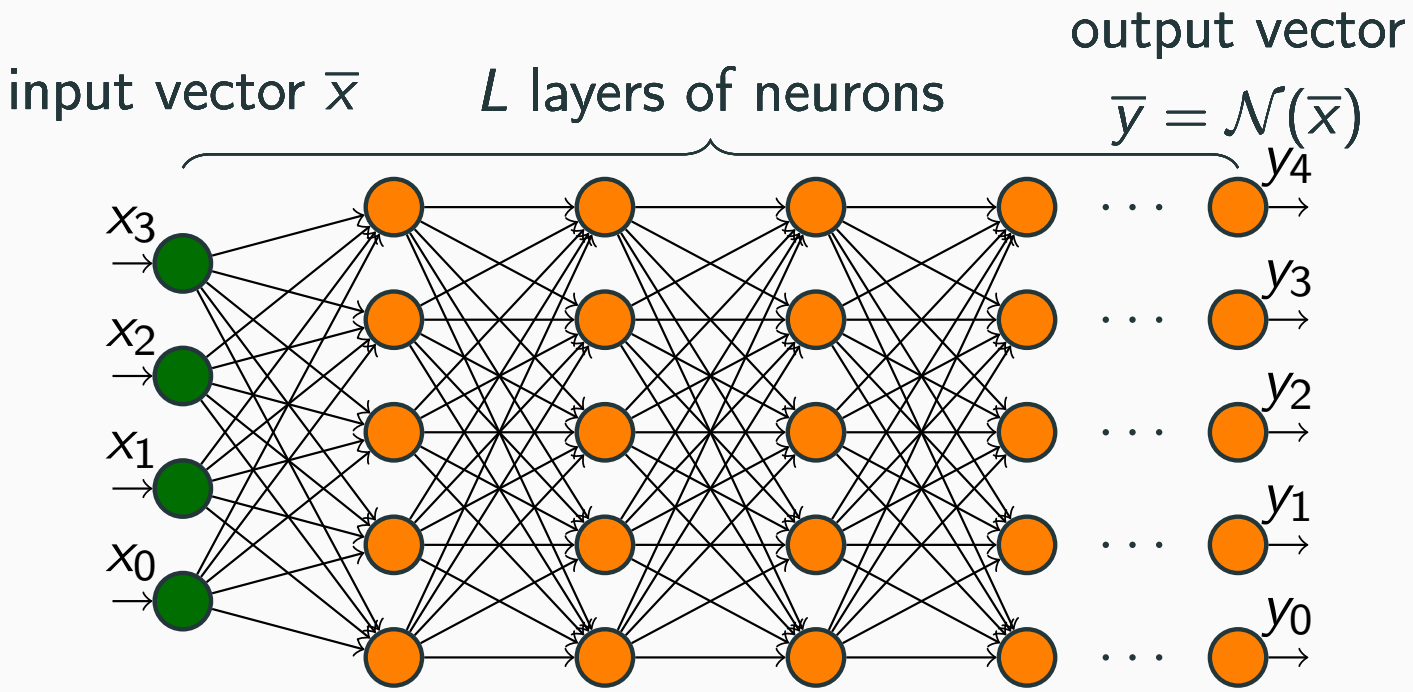
- Let $\mathcal{G}$ be the class of all graphs (of some given dimension d)
- Let $\mathcal{GV}$ be the class of all pairs (G, v), where
  $$G = (V, E, \lambda) \in \mathcal{G} \text{ and } v \in V$$
- Let $\mathbb{Y}$ be an output space
- A node embedding is a function of the form
  $$\xi : \mathcal{GV} \to \mathbb{Y}$$

Example: prediction of the relevance of a paper in citation network

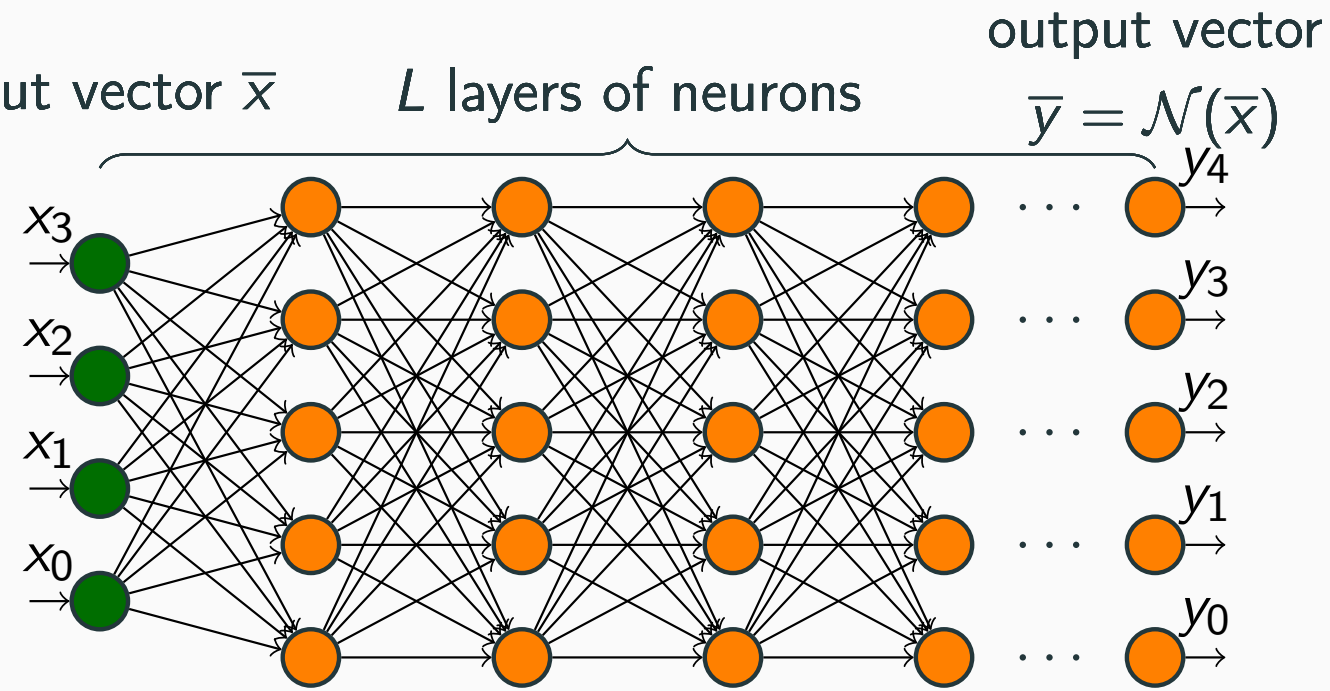
- Given one (or several) networks,
  predict a number in $[0, 1]$ for every paper

We mostly concentrate on node embeddings in this lecture,
but results usually transfer to graph level

Recall: Usual Deep Neural Networks (NNs)

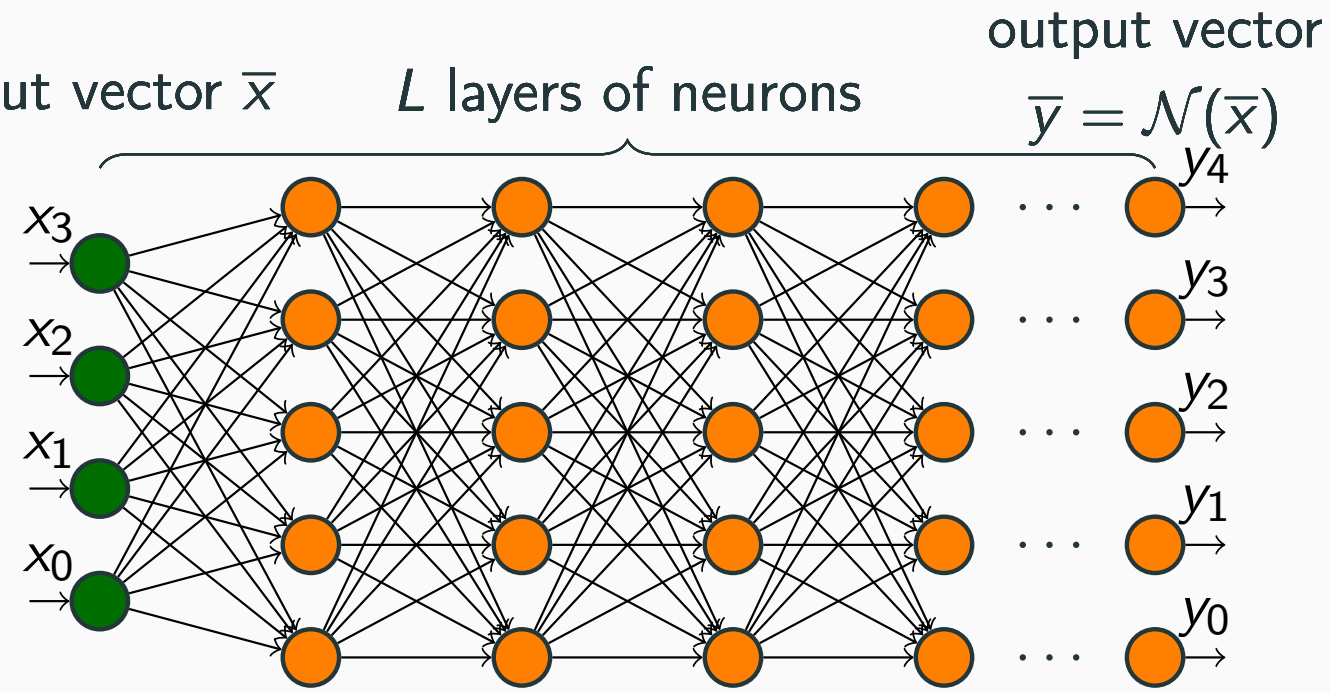


Recall: Usual Deep Neural Networks (NNs)



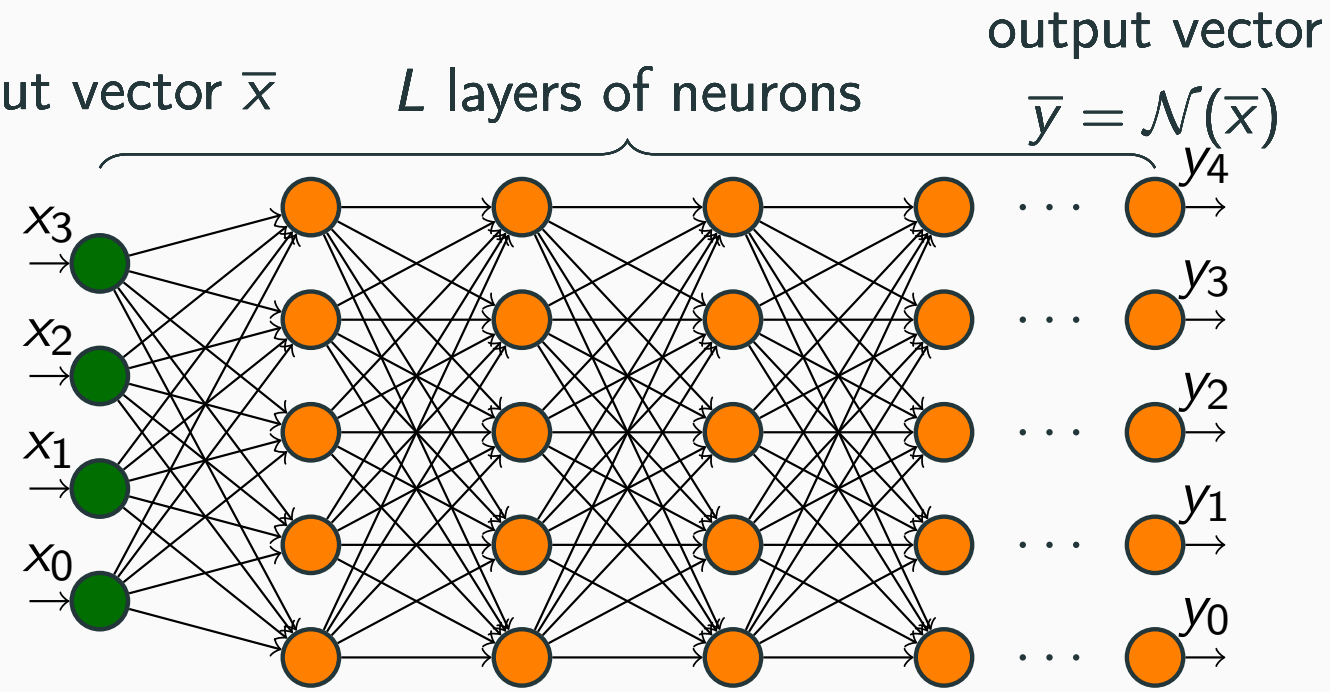
– NN $\mathcal{N}$ with weight $w_{n' \to n}$ for each two connected neurons

Recall: Usual Deep Neural Networks (NNs)



- NN $\mathcal{N}$ with weight $w_{n' \to n}$ for each two connected neurons
- Compute left to right $\lambda(n) := f(\sum w_{n' \to n} \times \lambda(n'))$

Recall: Usual Deep Neural Networks (NNs)



- NN $\mathcal{N}$ with weight $w_{n' \to n}$ for each two connected neurons
- Compute left to right $\lambda(n) := f(\sum w_{n' \to n} \times \lambda(n'))$
- Goal: Find the weights that 'solve' your problem (classification, regression, etc.)
  - minimise $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where g is what you want to learn
  - use backpropagation algorithms

Can We Use Them for Graph Learning?

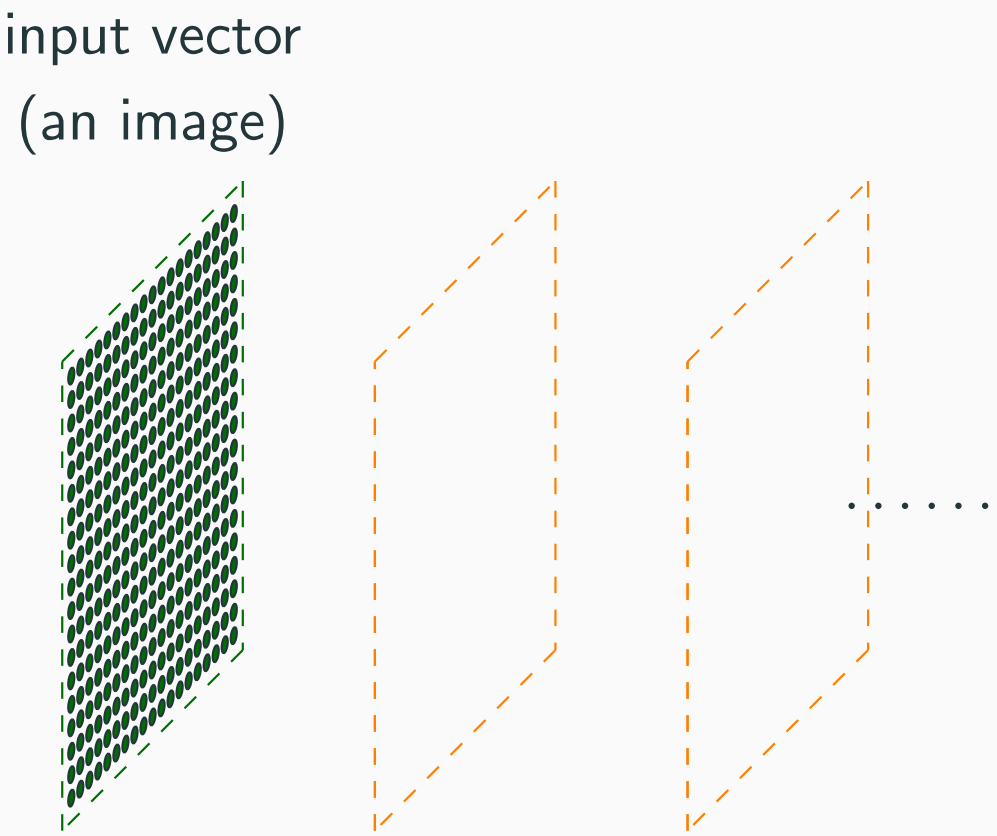
- Problem 1: for fully connected NNs, when a layer has many neurons, there are a lot of weights
  - Example: input is a 250×250 grid graph, and we want to build a fully connected NN with 500 neurons per layer
  - Between the first two layers we have $250 \times 250 \times 500 = 31,250,000$ weights

Can We Use Them for Graph Learning?

- Problem 1: for fully connected NNs, when a layer has many neurons, there are a lot of weights
  - Example: input is a 250×250 grid graph, and we want to build a fully connected NN with 500 neurons per layer
  - Between the first two layers we have $250 \times 250 \times 500 = 31,250,000$ weights

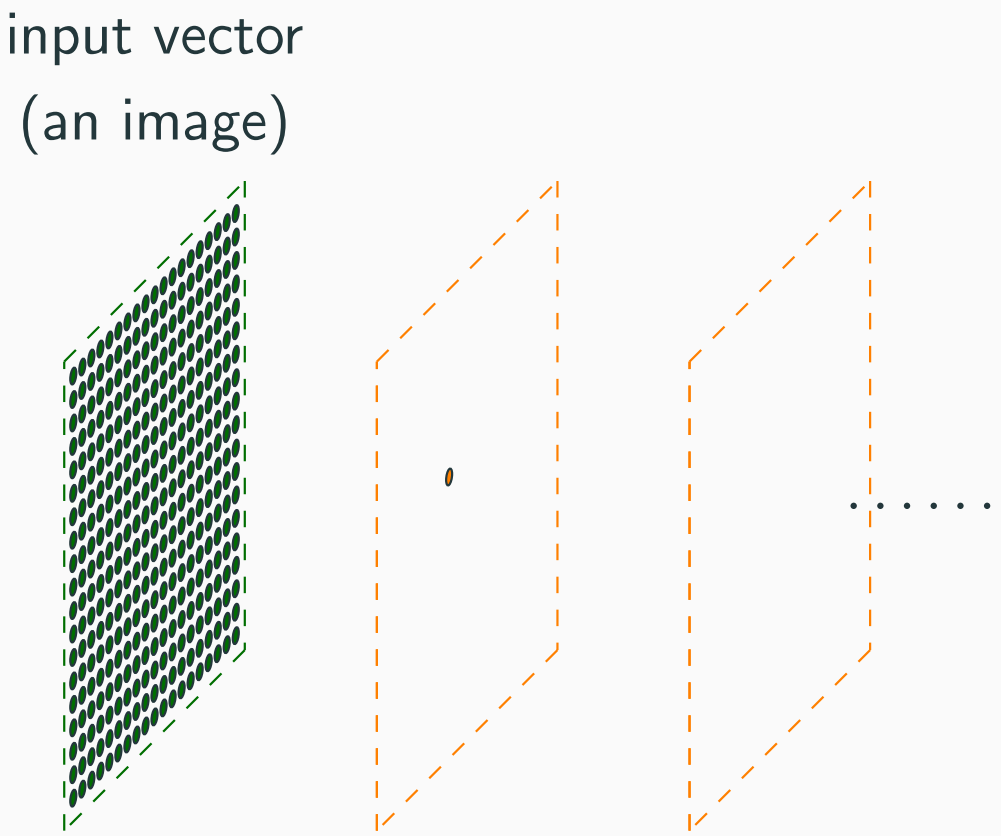
- Problem 2: The size of the input for each NN is fixed, but
  - Graph learning does not assume any restriction on the size of the input graph: a model trained on small graphs should be applicable to graphs of arbitrary big size
  - We can, of course, impose a hard bound on the size of the graphs, but....

Solution for Grids: Convolutional Neural Networks



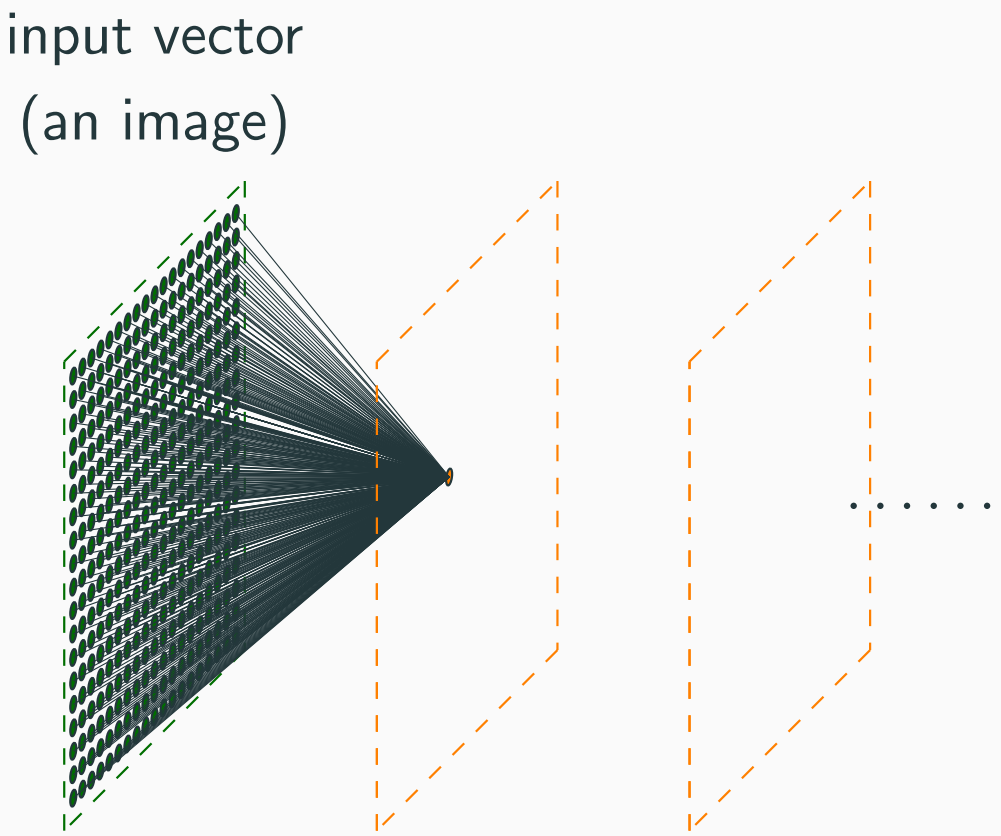
Idea: CNNs use the structure of the data (here, the grid)

Solution for Grids: Convolutional Neural Networks



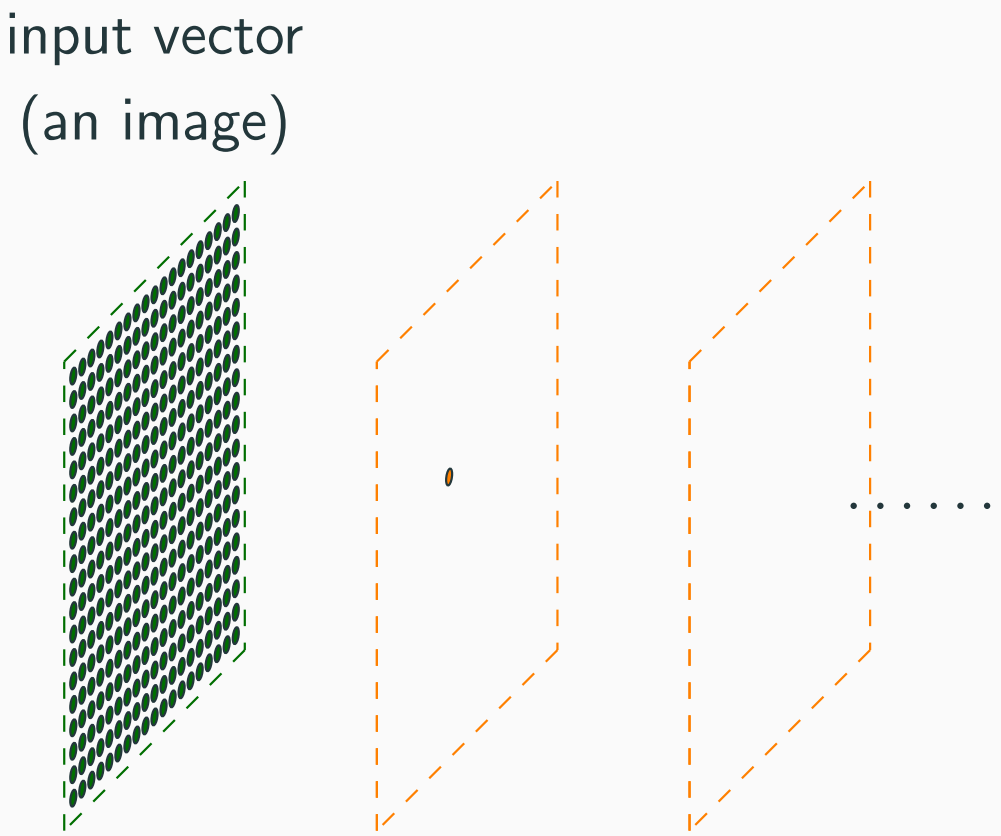
Idea: CNNs use the structure of the data (here, the grid)

Solution for Grids: Convolutional Neural Networks



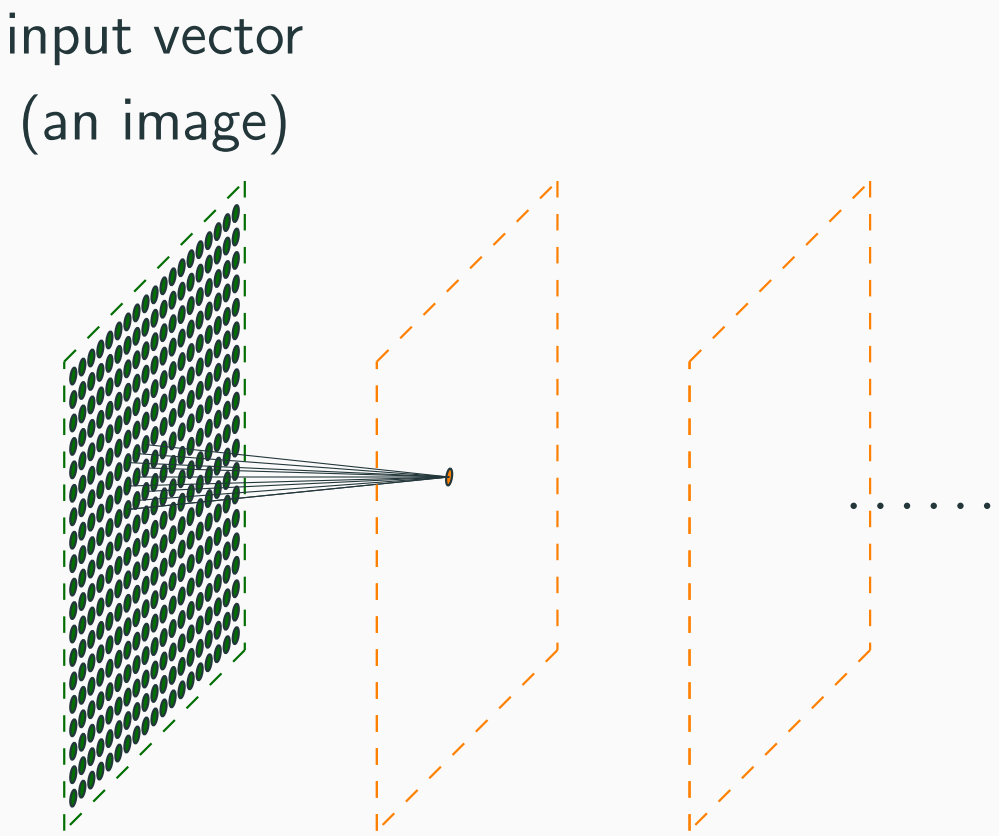
Idea: CNNs use the structure of the data (here, the grid)

Solution for Grids: Convolutional Neural Networks



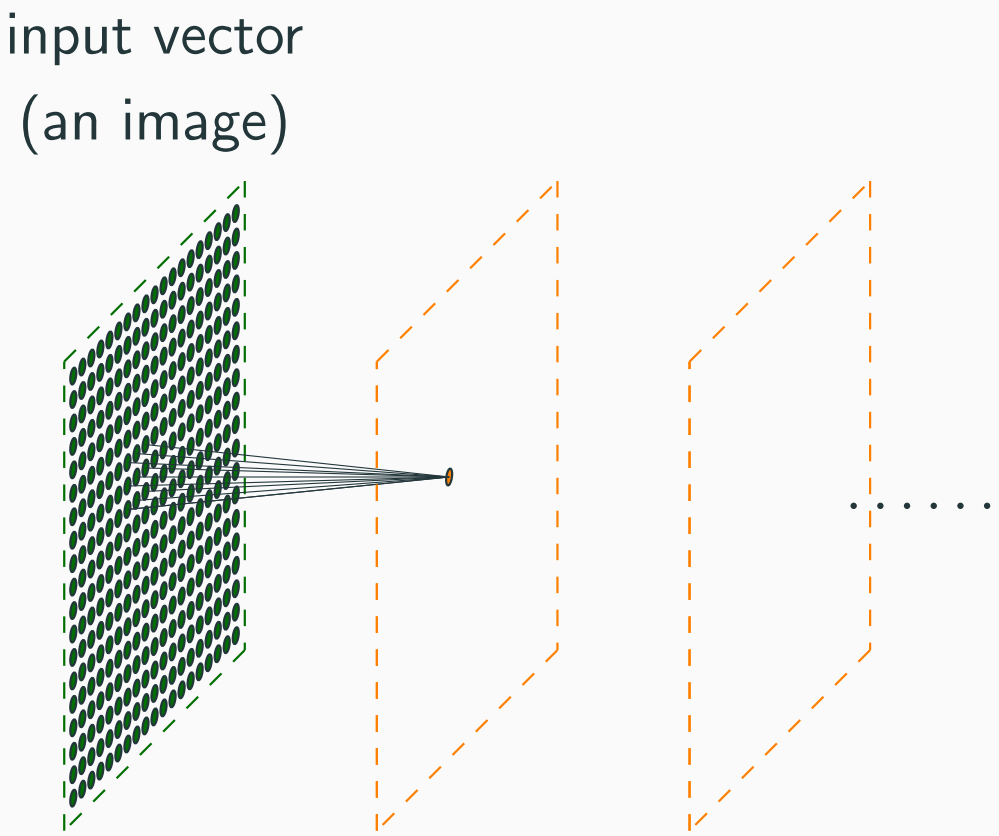
Idea: CNNs use the structure of the data (here, the grid)

Solution for Grids: Convolutional Neural Networks



Idea: CNNs use the structure of the data (here, the grid)

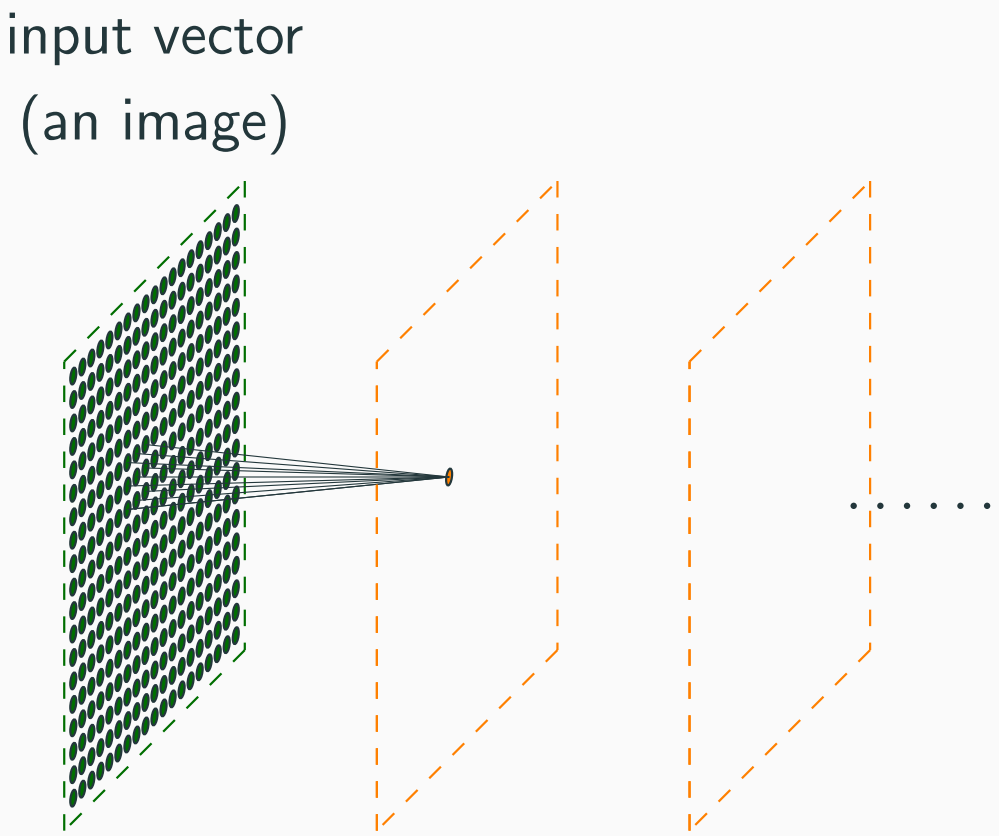
Solution for Grids: Convolutional Neural Networks



Idea: CNNs use the structure of the data (here, the grid)

– fewer weights to learn (e.g, only 9 for the first layer)

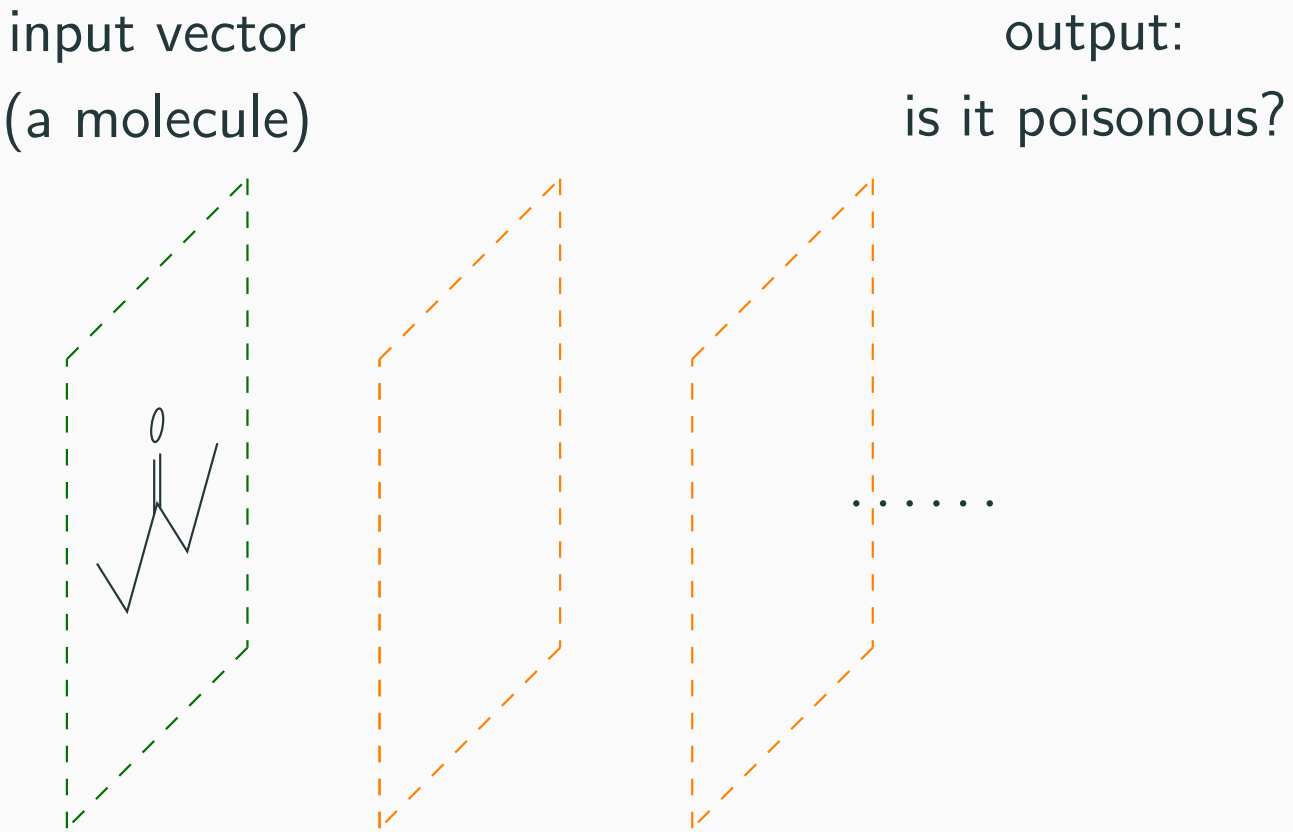
Solution for Grids: Convolutional Neural Networks



Idea: CNNs use the structure of the data (here, the grid)

- fewer weights to learn (e.g, only 9 for the first layer)
- other advantage: recognise patterns that are local

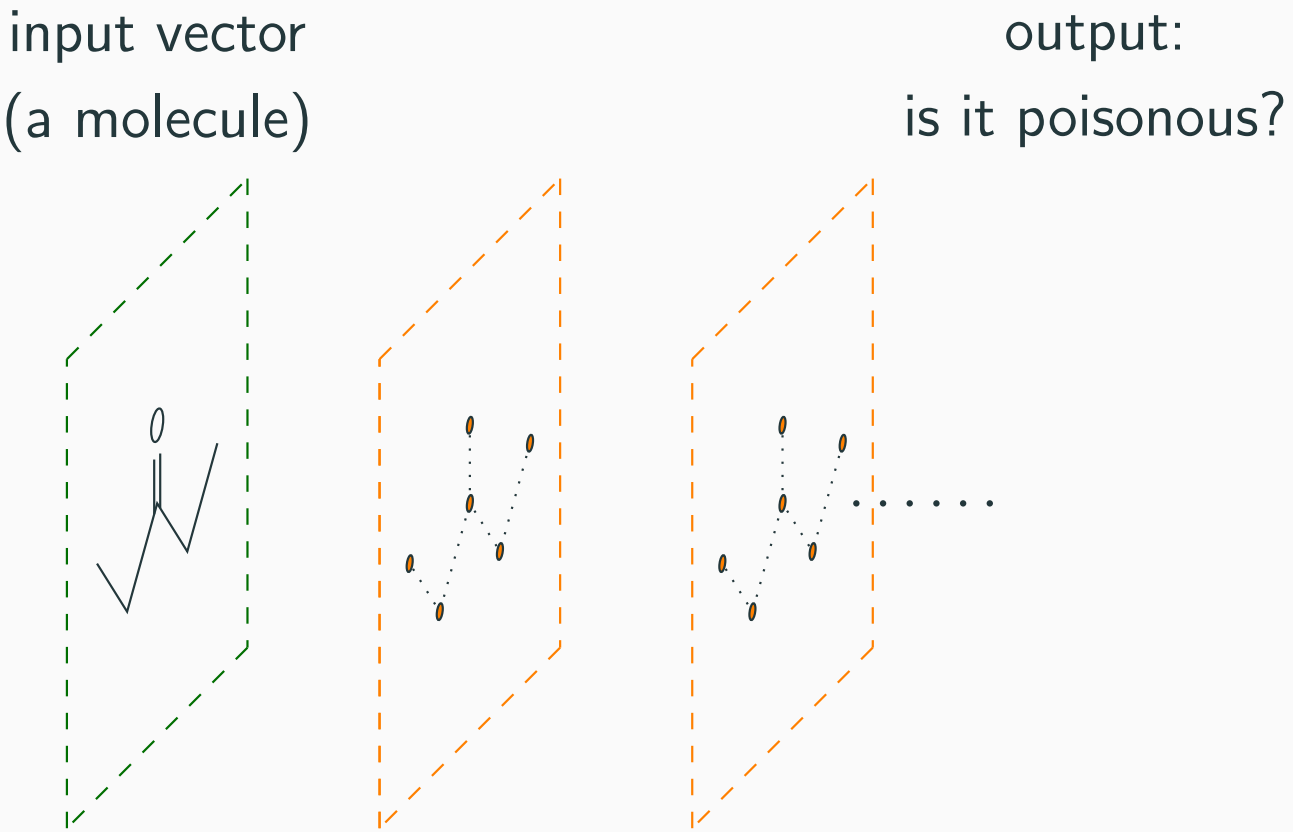
Generalisation to All Graphs: Graph Neural Networks (GNNs)



Idea: use the structure of the data

- GNNs generalise this idea to allow any graph as input

Generalisation to All Graphs: Graph Neural Networks (GNNs)



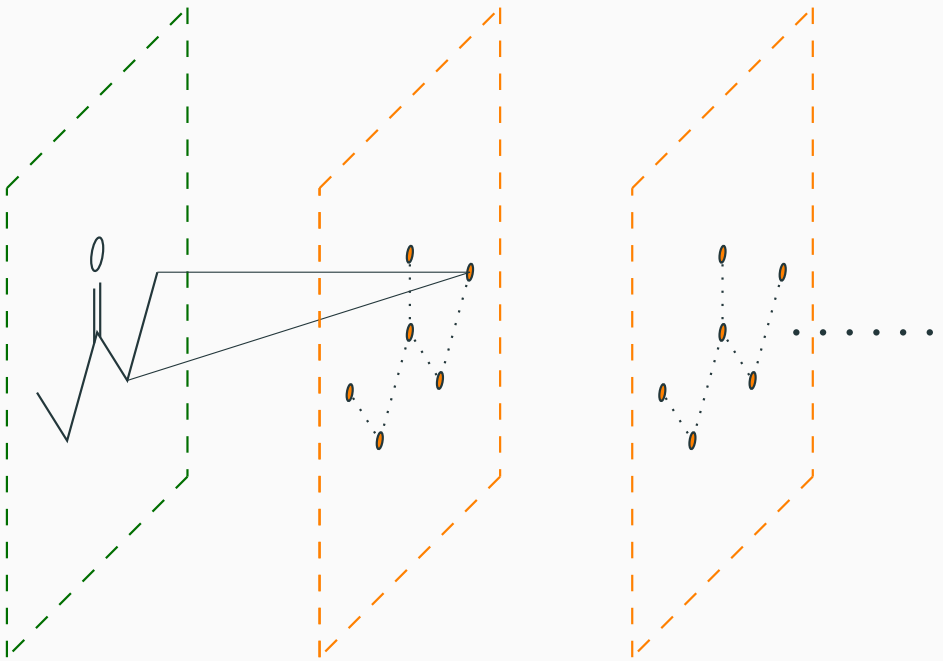
Idea: use the structure of the data

- GNNs generalise this idea to allow any graph as input

Generalisation to All Graphs: Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous?



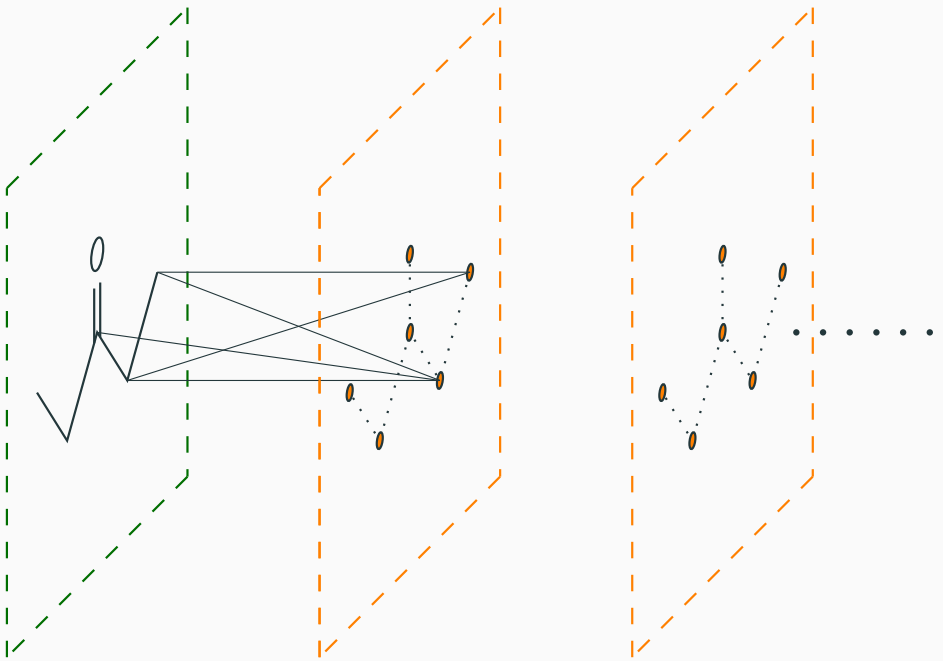
Idea: use the structure of the data

– GNNs generalise this idea to allow any graph as input

Generalisation to All Graphs: Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous?



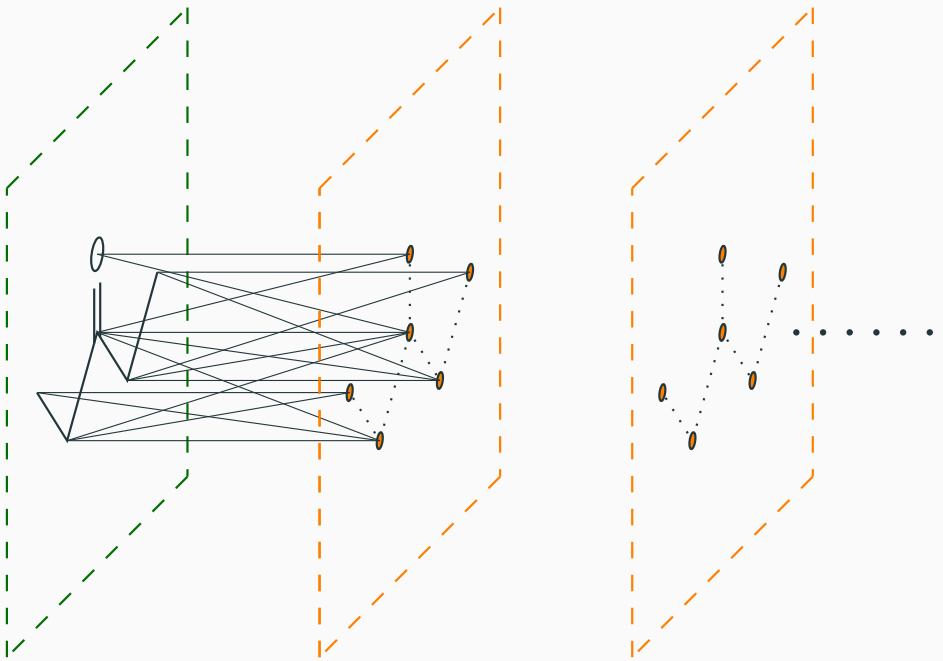
Idea: use the structure of the data

– GNNs generalise this idea to allow any graph as input

Generalisation to All Graphs: Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous?



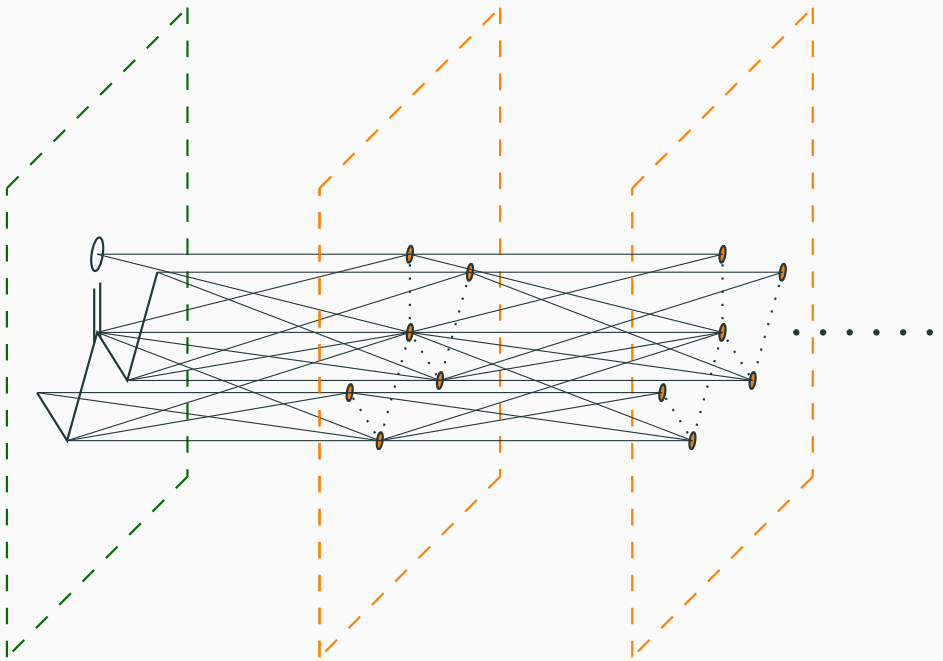
Idea: use the structure of the data

– GNNs generalise this idea to allow any graph as input

Generalisation to All Graphs: Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous?



Idea: use the structure of the data

– GNNs generalise this idea to allow any graph as input

Node-Level AC-GNNs: Framework Formalisation

Definition ([BKM+20, GSR+17])

An aggregate-combine (node-level) GNN (AC-GNN) with L layers, input dimension d, (hidden) dimensions $d_0 = d, d_1, \ldots, d_L$, and output space $\mathbb{Y}$ is two families of functions and another function:

- aggregation $\mathrm{AGG}^{(\ell)} : \mathbb{N}^{\mathbb{R}^{d_{\ell-1}}} \to \mathbb{R}^{d_{\ell-1}}$, for $\ell = 1, \ldots, L$
- combination $\mathrm{COMB}^{(\ell)} : \mathbb{R}^{2d_{\ell-1}} \to \mathbb{R}^{d_\ell}$, for $\ell = 1, \ldots, L$
- output $\mathrm{OUT} : \mathbb{R}^{d_L} \to \mathbb{Y}$

Note: in case of $\mathbb{Y} = \mathbb{R}^{d_L}$ for $d_L = 1$ (e.g., regression) we can take OUT as identity and not mention it

AC-GNN Application

1. Input: a graph $G = (V, E, \lambda)$ of dimension d
   (undirected, simple, node-labelled)

AC-GNN Application

1. Input: a graph $G = (V, E, \lambda)$ of dimension d
   (undirected, simple, node-labelled)
2. Run of a GNN with L layers on G:
   - initialise $\boldsymbol{x}_v^{(0)} := \lambda(v)$ for every $v \in V$
   - iteratively compute $\boldsymbol{x}_v^{(\ell)} \in \mathbb{R}^{d_\ell}$ for each v and ℓ as follows, where $\mathcal{N}_G(v)$ is the set of neighbours of v in G:

$$\boldsymbol{x}_v^{(\ell)} := \mathrm{COMB}^{(\ell)}(\boldsymbol{x}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\}))$$

3. Output: $\mathrm{OUT}(\boldsymbol{x}_v^{(L)})$ for each $v \in V$

AC-GNN Application

1. Input: a graph $G = (V, E, \lambda)$ of dimension d
   (undirected, simple, node-labelled)
2. Run of a GNN with L layers on G:
   - initialise $\boldsymbol{x}_v^{(0)} := \lambda(v)$ for every $v \in V$
   - iteratively compute $\boldsymbol{x}_v^{(\ell)} \in \mathbb{R}^{d_\ell}$ for each v and ℓ as follows, where $\mathcal{N}_G(v)$ is the set of neighbours of v in G:

$$\boldsymbol{x}_v^{(\ell)} := \mathrm{COMB}^{(\ell)}(\boldsymbol{x}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\}))$$

3. Output: $\mathrm{OUT}(\boldsymbol{x}_v^{(L)})$ for each $v \in V$

Thus, AC-GNNs realise node embeddings

Graph-Level AC-GNNs

The same, except that OUT aggregates final labels from all nodes:

Definition

An graph-level AC-GNN ... is two families of functions and one single function

- aggregation $\mathrm{AGG}^{(\ell)}$ ..., combination $\mathrm{COMB}^{(\ell)}$ ...
- output $\mathrm{READOUT} : \mathbb{N}^{\mathbb{R}^{d_L}} \to \mathbb{Y}$

Graph-Level AC-GNNs

The same, except that OUT aggregates final labels from all nodes:

Definition

An graph-level AC-GNN ... is two families of functions and one single function

- aggregation $\mathrm{AGG}^{(\ell)}$ ..., combination $\mathrm{COMB}^{(\ell)}$ ...
- output $\mathrm{READOUT} : \mathbb{N}^{\mathbb{R}^{d_L}} \to \mathbb{Y}$

Application: again the same, except

1–2. ...

3. Output: $\mathrm{READOUT}(\{\{\boldsymbol{x}_v^{(L)} \mid v \in V\}\})$

Thus, graph-level AC-GNNs realise graph embeddings

Graph-Level AC-GNNs

The same, except that OUT aggregates final labels from all nodes:

Definition

An graph-level AC-GNN ... is two families of functions and one single function

- aggregation $\mathrm{AGG}^{(\ell)}$ ..., combination $\mathrm{COMB}^{(\ell)}$ ...
- output $\mathrm{READOUT} : \mathbb{N}^{\mathbb{R}^{d_L}} \to \mathbb{Y}$

Application: again the same, except

1–2. ...

3. Output: $\mathrm{READOUT}(\{\{\boldsymbol{x}_v^{(L)} \mid v \in V\}\})$

Thus, graph-level AC-GNNs realise graph embeddings

We concentrate on node-level GNNs,
but nearly all results transfer to the graph level

Where is Something Neural? Max-Sum GNNs

In real GNNs, functions $\mathrm{AGG}^{(\ell)}$ and/or $\mathrm{COMB}^{(\ell)}$ depend on matrices and/or vectors of (usually trainable) parameters and include non-linearity

Example: Basic Sum-Plus GNN with sigmoid

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\Sigma_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

where

- $\boldsymbol{A}^{(\ell)}$ and $\boldsymbol{C}^{(\ell)}$ are matrices of appropriate dimensions
- $\boldsymbol{b}^{(\ell)}$ is a vector of appropriate size
- *Sigmoid* is sigmoid function (applied element-wise to vectors)

Where is Something Neural? Max-Sum GNNs

In real GNNs, functions $\mathrm{AGG}^{(\ell)}$ and/or $\mathrm{COMB}^{(\ell)}$ depend on matrices and/or vectors of (usually trainable) parameters and include non-linearity

Example: Basic Sum-Plus GNN with sigmoid

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\textstyle\sum_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

where

- $\boldsymbol{A}^{(\ell)}$ and $\boldsymbol{C}^{(\ell)}$ are matrices of appropriate dimensions
- $\boldsymbol{b}^{(\ell)}$ is a vector of appropriate size
- *Sigmoid* is sigmoid function (applied element-wise to vectors)

It is an AC-GNN: $\mathrm{AGG}^{(\ell)}$ is the sum and $\mathrm{COMB}^{(\ell)}$ is the rest

Abuse of terminology: it is actually a family of AC-GNNs, or GNN architecture

More GNN Architectures

In

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\sum_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

we can have

- average, max, etc. instead of sum
- vector concatenation $||$, etc. instead of $+$
- ReLU, etc. instead of sigmoid: $\mathit{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{otherwise} \end{cases}$

More GNN Architectures

In

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\Sigma_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

we can have

- average, max, etc. instead of sum
- vector concatenation $||$, etc. instead of $+$
- ReLU, etc. instead of sigmoid: $\mathit{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{otherwise} \end{cases}$

(a version of) Graph Convolutional Networks (GCNs) [SKB+18]:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{ReLU}\left(\boldsymbol{A}^{(\ell)}\left(\mathrm{avg}_{u\in\mathcal{N}_G(v)\cup\{v\}}\boldsymbol{x}_u^{(\ell-1)}\right)\right)$$

GraphSAGE [HYL17]:

$$\boldsymbol{x}_v^{(\ell)} := \boldsymbol{A}^{(\ell)}\left(\boldsymbol{x}_v^{(\ell-1)} \;||\; \left(\max_{u\in\mathcal{N}_G(v)} \mathit{ReLU}\left(\boldsymbol{C}^{(\ell)}\boldsymbol{x}_u^{(\ell-1)}\right)\right)\right)$$

(many-many more . . .)

Matrix Form

Such GNNs are often written in matrix form.
For example:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\Sigma_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

is the same as

$$\boldsymbol{X}^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{X}^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\boldsymbol{G}\boldsymbol{X}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\right)$$

for $\boldsymbol{G}$ the adjacency matrix of G

Beyond Aggregate-Combine GNNs

Now, there are many advanced GNN architectures that do not fall into the AC-GNN framework

Many of them appeared as attempts to address the limitations of AC-GNNs discovered in the work I will present today

Example 1: node embedding 'node is reachable from a node with label 1' is not expressible by any AC-GNN

because each AC-GNN has a fixed number of layers

- Recurrent GNNs [PTCK24] can, in principle, learn reachability

Beyond Aggregate-Combine GNNs

Now, there are many advanced GNN architectures that do not fall into the AC-GNN framework

Many of them appeared as attempts to address the limitations of AC-GNNs discovered in the work I will present today

Example 2: node embedding 'there is a red node somewhere in the graph that is neither me nor my neighbour' is not expressible by any AC-GNN, and recursion does not help

because AC-GNNs (even with recursion) cannot look into other connected components

- GNNs with global readout [BKM$^+$20] can, in principle, check global information (we will have a look)

Beyond Aggregate-Combine GNNs

Now, there are many advanced GNN architectures that do not fall into the AC-GNN framework

Many of them appeared as attempts to address the limitations of AC-GNNs discovered in the work I will present today

Example 3: graph embedding 'graph has a cycle of length 5' is also not expressible by any (graph-level) AC-GNN

- 5-WL GNNs [MRF$^+$19] can, in principle, check for a 5-cycle

We will nearly ultimately concentrate on AC-GNNs, and discuss more powerful architectures briefly at the end

Expressivity Notions for GNNs

So, we intuitively see that different GNN architectures can do some things and cannot do other. What does it mean, formally?

Expressivity Notions for GNNs

So, we intuitively see that different GNN architectures can do some things and cannot do other. What does it mean, formally?

We next consider several related notions for GNNs,
all sometimes called expressive power:

- distinguishing power
- (uniform) expressive power
- non-uniform expressive power
- (non-uniform) approximation power
- (+ learnability)

Answers to these questions may reveal:

- what graph information is used by embedding methods
- which embeddings could—in principle—be learned
- whether more powerful embedding methods may be needed for the application at hand

Before We Proceed

‘Weaker expressivity’ does not mean ‘worse’:

it is just more a-priori knowledge!

2. Distinguishing Power of GNNs

Distinguishing Power

Definition

A class $\mathcal{C}_2$ of node embeddings has as *strong distinguishing (or separating) power* as a class $\mathcal{C}_1$ if, for every G, v, G', v',

$\xi_1(G, v) \neq \xi_1(G', v')$ for some $\xi_1 \in \mathcal{C}_1$ implies

$\xi_2(G, v) \neq \xi_2(G', v')$ for some $\xi_2 \in \mathcal{C}_2$.

Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the *same distinguishing power* if both directions hold.

Note: One of $\mathcal{C}_1$, $\mathcal{C}_2$ may be singleton

The notion transfers to formalisms realising embeddings, such as AC-GNNs.

Distinguishing Power

Definition

A class $\mathcal{C}_2$ of node embeddings has as *strong distinguishing (or separating) power* as a class $\mathcal{C}_1$ if, for every G, v, G', v',

$\xi_1(G, v) \neq \xi_1(G', v')$ for some $\xi_1 \in \mathcal{C}_1$ implies
$\xi_2(G, v) \neq \xi_2(G', v')$ for some $\xi_2 \in \mathcal{C}_2$.

Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the *same distinguishing power* if both directions hold.

Note: One of $\mathcal{C}_1$, $\mathcal{C}_2$ may be singleton

The notion transfers to formalisms realising embeddings, such as AC-GNNs.

Our first (and famous) result:

Theorem ([MRF+19, XHLJ19])

AC-GNNs have the same distinguishing power as the stable colouring of the Weisfeiler-Leman (WL) test.

Also, we will have a more tricky result: GINs are also such

Weisfeiler-Leman Graph Isomorphism Test: Properties

Theorem ([MRF+19, XHLJ19])

AC-GNNs have the same distinguishing power as the stable colouring of the Weisfeiler-Leman (WL) test.

- WL test is a classic formalism, so it is a good reference point for distinguishability
- For example, it has the same distinguishing power as Graded Modal Logic (node level, we will come back to it) and sentences FOC^2 (graph level)

Weisfeiler-Leman Graph Isomorphism Test: Properties

Theorem ([MRF+19, XHLJ19])

AC-GNNs have the same distinguishing power as the stable colouring of the Weisfeiler-Leman (WL) test.

- WL test is a classic formalism, so it is a good reference point for distinguishability
- For example, it has the same distinguishing power as Graded Modal Logic (node level, we will come back to it) and sentences FOC^2 (graph level)

It is a complete, but not sound test for isomorphism of two graphs:

- if it says NO, then the graphs are not isomorphic
- if it says YES, then it may be isomorphic or not
  (but the latter is only in special cases rare in practice)

Weisfeiler-Leman Graph Isomorphism Test: Properties

Theorem ([MRF+19, XHLJ19])

AC-GNNs have the same distinguishing power as the stable colouring of the Weisfeiler-Leman (WL) test.

- WL test is a classic formalism, so it is a good reference point for distinguishability
- For example, it has the same distinguishing power as Graded Modal Logic (node level, we will come back to it) and sentences FOC^2 (graph level)

It is a complete, but not sound test for isomorphism of two graphs:

- if it says NO, then the graphs are not isomorphic
- if it says YES, then it may be isomorphic or not
  (but the latter is only in special cases rare in practice)

Useful, because complexity of checking isomorphism of two graphs is probably hard (a big open problem in CS):

- no polynomial algorithm is known, but no one can prove NP-hardness

Weisfeiler-Leman Graph Isomorphism Test: Algorithm

Weisfeiler-Leman (WL) graph isomorphism test [WL68]

(also called colour refinement)

1. Input: two graphs with coloured nodes

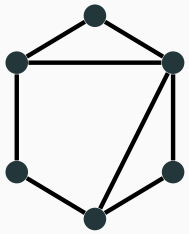
   (or, in our case nodes with embeddings)

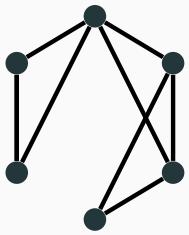
Weisfeiler-Leman Graph Isomorphism Test: Algorithm

Weisfeiler-Leman (WL) graph isomorphism test [WL68]
(also called colour refinement)

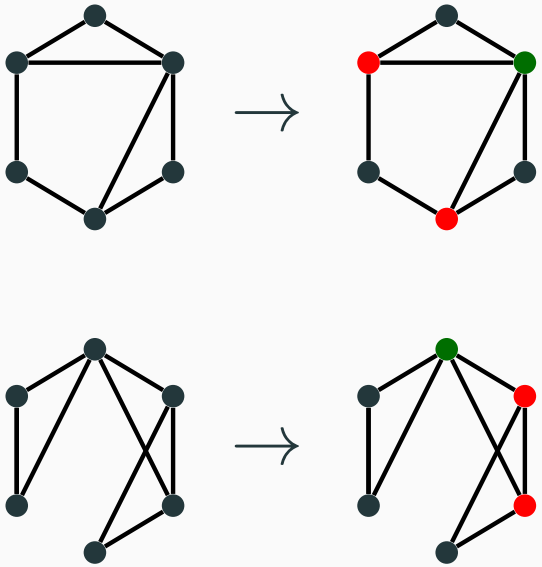
1. Input: two graphs with coloured nodes
   (or, in our case nodes with embeddings)
2. Iterate the following until the colouring is stable
   (i.e., the partition of the nodes into colours does not change):
   - two nodes v, v' are assigned the same colour iff they have same colour and same multisets of colours of neighbours
3. Accept if the two graphs have the same multiset of colours,
   Reject otherwise

Weisfeiler-Leman: Example 1

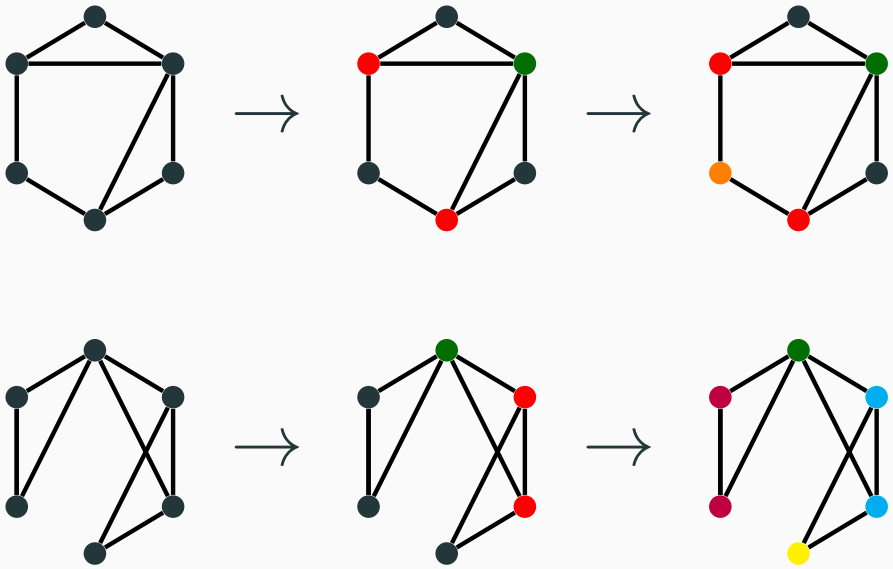




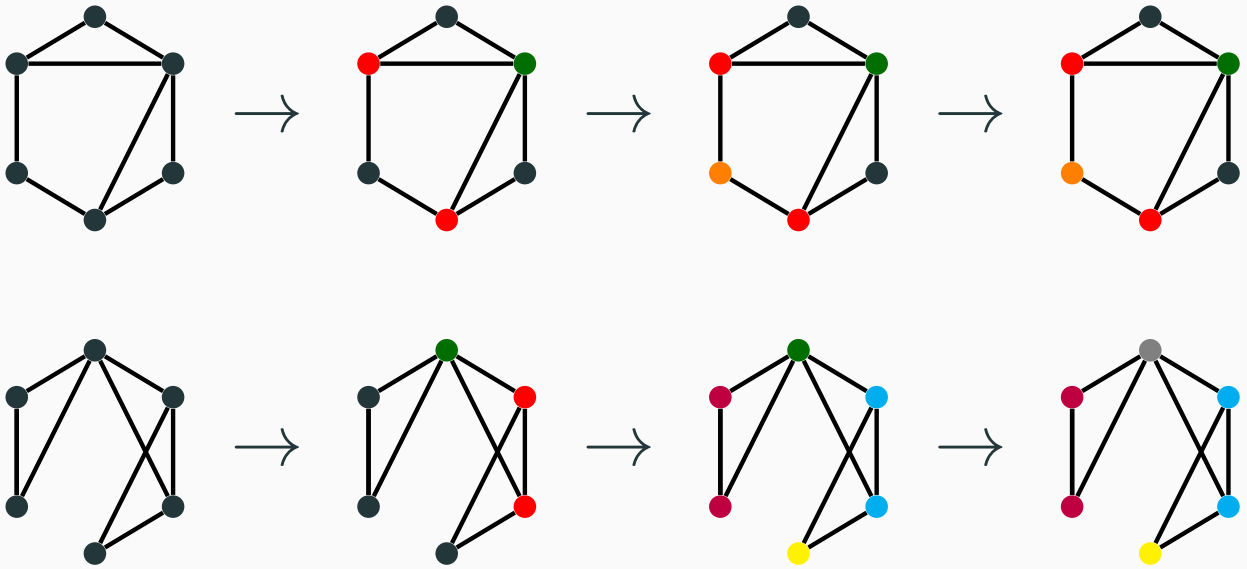
Weisfeiler-Leman: Example 1



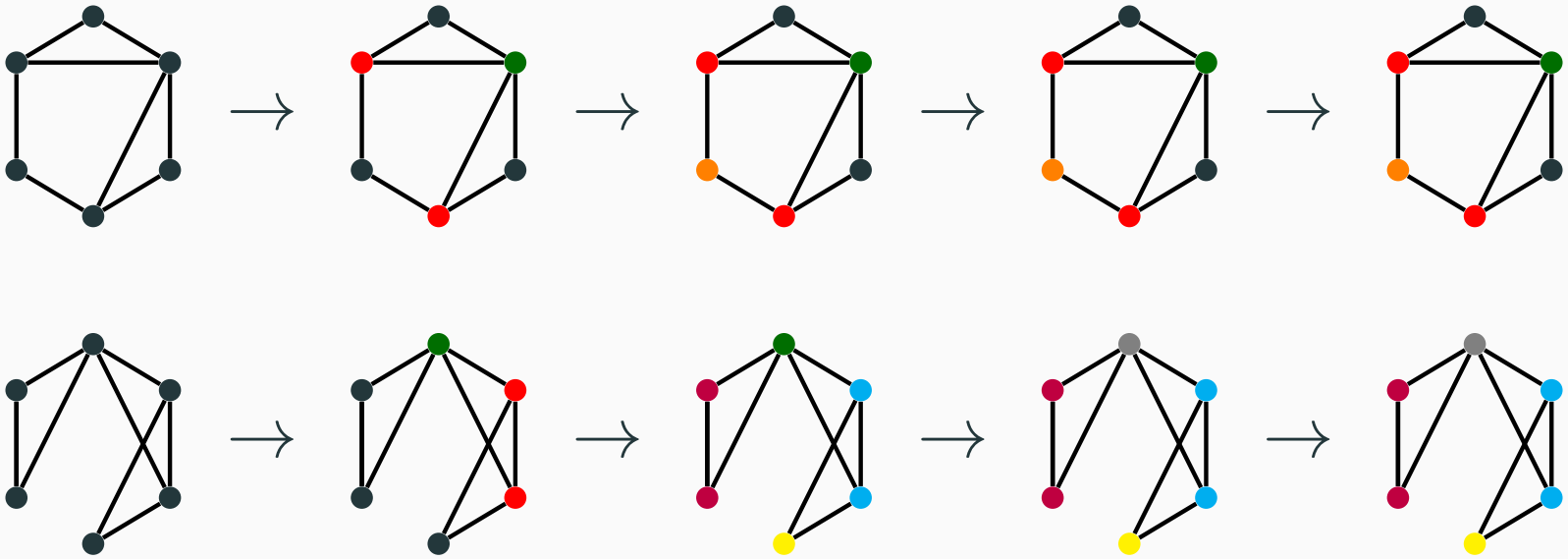
Weisfeiler-Leman: Example 1



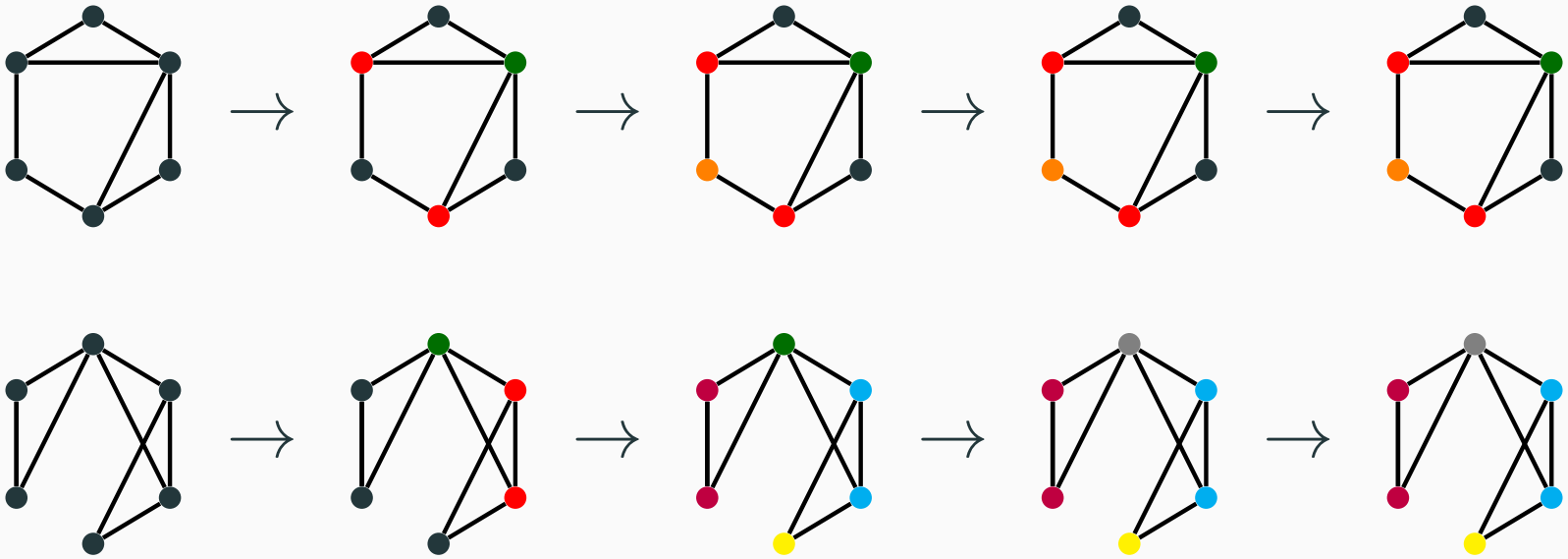
Weisfeiler-Leman: Example 1



Weisfeiler-Leman: Example 1



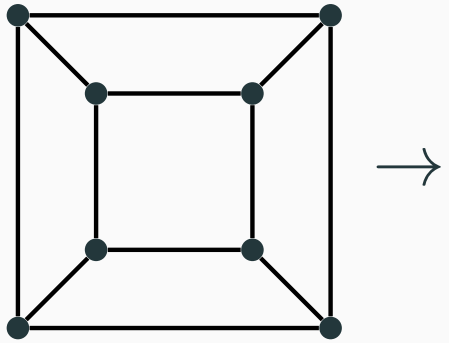
Weisfeiler-Leman: Example 1

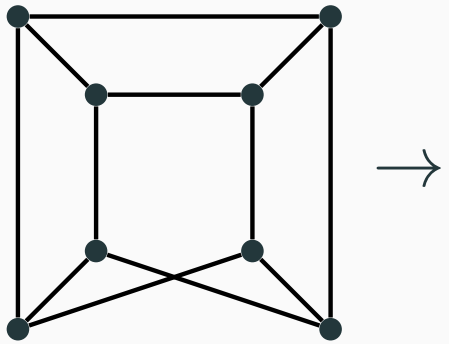


$\{\{\bullet, \bullet, \bullet, \bullet, \bullet, \bullet\}\} \neq \{\{\bullet, \bullet, \bullet, \bullet, \bullet, \bullet\}\}$

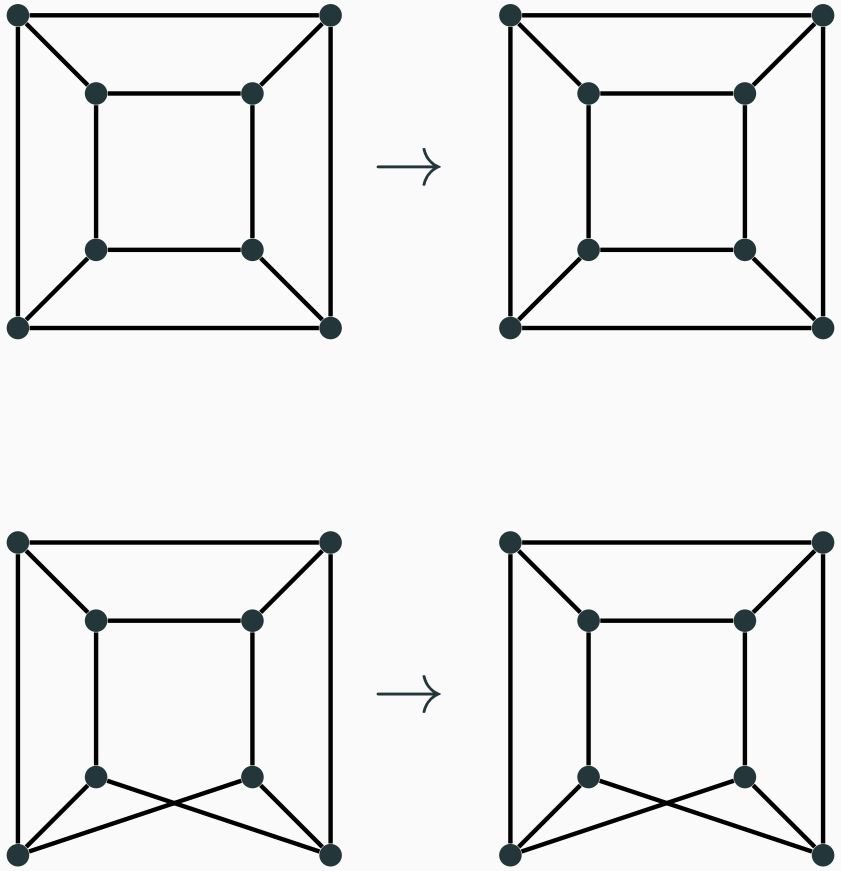
$\rightarrow$ **reject** (and this is correct)

Weisfeiler-Leman: Example 2

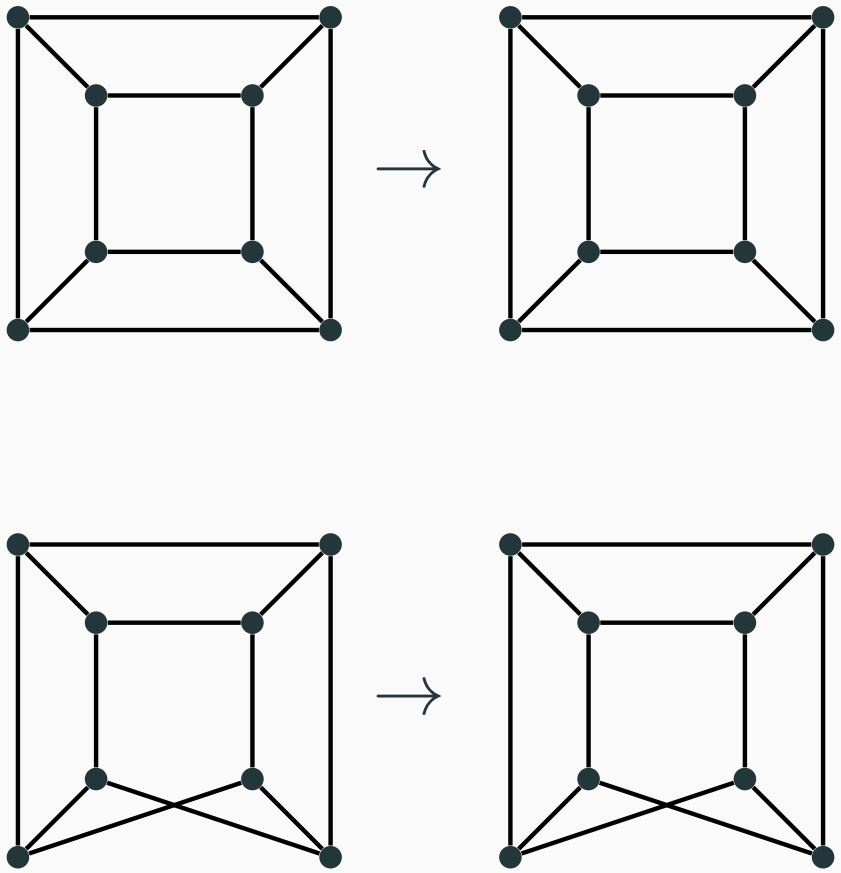




Weisfeiler-Leman: Example 2



Weisfeiler-Leman: Example 2



$$\{\{\bullet,\bullet,\bullet,\bullet,\bullet,\bullet,\bullet,\bullet,\}\} = \{\{\bullet,\bullet,\bullet,\bullet,\bullet,\bullet,\bullet,\bullet,\}\}$$

$\rightarrow$ **accept** (but this is incorrect!)

AC-GNNs and Weisfeiler-Leman

Weisfeiler-Leman for each node v works as

- $\mathrm{WL}_v^{(0)} := \lambda(v)$
- $\mathrm{WL}_v^{(\ell)} := \mathrm{INJ}^{(\ell)}(\mathrm{WL}_v^{(\ell-1)}, \{\{\mathrm{WL}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\})$

AC-GNNs and Weisfeiler-Leman

Weisfeiler-Leman for each node v works as

- $\mathrm{WL}_v^{(0)} := \lambda(v)$
- $\mathrm{WL}_v^{(\ell)} := \mathrm{INJ}^{(\ell)}(\mathrm{WL}_v^{(\ell-1)}, \{\{\mathrm{WL}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\})$

AC-GNNs for each node v work as

- $\boldsymbol{x}_v^{(0)} := \lambda(v)$
- $\boldsymbol{x}_v^{(\ell)} := \mathrm{COMB}^{(\ell)}(\boldsymbol{x}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\}))$

AC-GNNs and Weisfeiler-Leman

Weisfeiler-Leman for each node v works as

- $\mathrm{WL}_v^{(0)} := \lambda(v)$
- $\mathrm{WL}_v^{(\ell)} := \mathrm{INJ}^{(\ell)}(\mathrm{WL}_v^{(\ell-1)}, \{\{\mathrm{WL}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\})$

AC-GNNs for each node v work as

- $\boldsymbol{x}_v^{(0)} := \lambda(v)$
- $\boldsymbol{x}_v^{(\ell)} := \mathrm{COMB}^{(\ell)}(\boldsymbol{x}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\}))$

We observe:

- WL works exactly as an AC-GNN with injective aggregation and combination functions
- If WL assigns the same value to two nodes at round ℓ, then every AC-GNN also assigns the same value to these two nodes at layer ℓ

AC-GNNs and Weisfeiler-Leman

We observe:

- WL works exactly as an AC-GNN with injective aggregation and combination functions
- If WL assigns the same value to two nodes at round ℓ, then every AC-GNN also assigns the same value to these two nodes at layer ℓ

We can say that WL algorithm realises a node embedding (or a class of, but all with the same distinguishing power)

Theorem ([MRF+19, XHLJ19])

AC-GNNs and the stable colouring of the WL algorithm have the same distinguishing power.

For the 'GNN simulates WL' direction, we just need to choose the number of layers big enough so that WL stabilises on the two given graphs (recall that it is ok to have a separate GNN for each two of graph-node pairs).

One Question Left

Recall that we rely on the fact:
WL works as an AC-GNN with injective $\mathrm{AGG}^{(\ell)}$ and $\mathrm{COMB}^{(\ell)}$

Such functions obviously exist, but can they be really 'neural',
something like in our Sum-Plus GNNs?

We can try $\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} \;||\; \boldsymbol{C}^{(\ell)}\left(\sum_{u\in\mathcal{N}_G(v)} f\left(\boldsymbol{x}_u^{(\ell-1)}\right)\right)\right)$
for some smart f so that $\sum_{x\in X} f(x)$ is always injective

One Question Left

Recall that we rely on the fact:
WL works as an AC-GNN with injective $\mathrm{AGG}^{(\ell)}$ and $\mathrm{COMB}^{(\ell)}$

Such functions obviously exist, but can they be really 'neural',
something like in our Sum-Plus GNNs?

We can try $\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} \;||\; \boldsymbol{C}^{(\ell)}\left(\sum_{u\in\mathcal{N}_G(v)} f\left(\boldsymbol{x}_u^{(\ell-1)}\right)\right)\right)$
for some smart f so that $\sum_{x\in X} f(x)$ is always injective

- Unfortunately, such an f working for arbitrarily big X does not exist

One Question Left

Recall that we rely on the fact:
WL works as an AC-GNN with injective $\mathrm{AGG}^{(\ell)}$ and $\mathrm{COMB}^{(\ell)}$

Such functions obviously exist, but can they be really 'neural', something like in our Sum-Plus GNNs?

We can try $\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} \,||\, \boldsymbol{C}^{(\ell)}\left(\sum_{u\in\mathcal{N}_G(v)} f\left(\boldsymbol{x}_u^{(\ell-1)}\right)\right)\right)$ for some smart f so that $\sum_{x\in X} f(x)$ is always injective

- Unfortunately, such an f working for arbitrarily big X does not exist
- Luckily, we can use the same trick:
  we do not need arbitrarily big X, only up to the number of nodes in the input graphs, which ensures that $\mathrm{AGG}^{(\ell)}$ is injective for these graphs (and not all graphs)
- such f exists (constructed using a straightforward generalisation of so-called deep sets)

One Question Left

Recall that we rely on the fact:
WL works as an AC-GNN with injective $\mathrm{AGG}^{(\ell)}$ and $\mathrm{COMB}^{(\ell)}$

Such functions obviously exist, but can they be really 'neural', something like in our Sum-Plus GNNs?

We can try $\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} \,||\, \boldsymbol{C}^{(\ell)}(\Sigma_{u\in\mathcal{N}_G(v)} f(\boldsymbol{x}_u^{(\ell-1)}))\right)$
for some smart f so that $\Sigma_{x\in X} f(x)$ is always injective

- Unfortunately, such an f working for arbitrarily big X does not exist
- Luckily, we can use the same trick:
  we do not need arbitrarily big X, only up to the number of nodes in the input graphs, which ensures that $\mathrm{AGG}^{(\ell)}$ is injective for these graphs (and not all graphs)
- such f exists (constructed using a straightforward generalisation of so-called deep sets)
- this is a variation of so-called GINs [XHLJ19] (Graph Isomorphism Networks, misleading name)

Is Distinguishable Power Enough?

1. WL realises a node embedding
2. Each AC-GNN realises a node embedding
3. AC-GNNs have the same distinguishing power as WL

Question: Can we say that there is an AC-GNN that gives the same answer for every node in every graph as WL?

Is Distinguishable Power Enough?

1. WL realises a node embedding
2. Each AC-GNN realises a node embedding
3. AC-GNNs have the same distinguishing power as WL

Question: Can we say that there is an AC-GNN that gives the same answer for every node in every graph as WL?

Actually, no.

A simple argument is that each AC-GNN has a fixed number of layers, but the number of iterations of WL may be arbitrarily large, depending on the graph.

A Glimpse Beyond AC-GNNs: ACR-GNNs

Definition ([BKM+20])

An aggregate-combine-readout (node-level) GNN (ACR-GNN) with L layers, input dimension d, (hidden) dimensions $d_0 = d, d_1, \ldots, d_L$, and output space $\mathbb{Y}$ is three families of functions and another function:

- aggregation $\mathrm{AGG}^{(\ell)} : \mathbb{N}^{\mathbb{R}^{d_{\ell-1}}} \to \mathbb{R}^{d_{\ell-1}}$, for $\ell = 1, \ldots, L$
- readout $\mathrm{READOUT}^{(\ell)} : \mathbb{N}^{\mathbb{R}^{d_{\ell-1}}} \to \mathbb{R}^{d_{\ell-1}}$, for $\ell = 1, \ldots, L$
- combination $\mathrm{COMB}^{(\ell)} : \mathbb{R}^{3d_{\ell-1}} \to \mathbb{R}^{d_\ell}$, for $\ell = 1, \ldots, L$
- output $\mathrm{OUT} : \mathbb{R}^{d_L} \to \mathbb{Y}$

Node update:

$$\boldsymbol{x}_v^{(\ell)} := \mathrm{COMB}^{(\ell)}(\boldsymbol{x}_v^{(\ell-1)}, \mathrm{AGG}^{(\ell)}(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in \mathcal{N}_G(v)\}\}, \mathrm{READOUT}^{(\ell)}(\{\{\boldsymbol{x}_u^{(\ell-1)} \mid u \in V\}\}))$$

AC-GNN Application

'Folklore' Weisfeiler-Leman graph isomorphism test

1. Input: two graphs with coloured nodes
2. Iterate the following until the colouring is stable:
   - two nodes v, v' are assigned the same colour iff they have same colour, same multisets of colours of neighbours, and same multisets of colours of non-neighbours
3. Accept if and only if the graphs have the same multiset of colours

Theorem ([MRF+19, XHLJ19])

ACR-GNNs and the stable colouring of the 'folklore' WL algorithm have the same distinguishing power.

3. Expressive Power of GNNs

Expressive Power

Definition

A class $\mathcal{C}_2$ of node embeddings has as *strong (uniform) expressive power* as a class $\mathcal{C}_1$ if $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the *same expressive power* if both directions hold.

This notion again transfers directly to formalisms realising embeddings, when it becomes more interesting

- We start with expressivity of AC-GNNs in terms of a special kind of bisimulation
  (which is a modification of the distinguishability result)
- Then, finally, we look at connections to logic

3.1 Expressivity via Bisimulation

Bisimulation

Definition

Let $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$ be graphs. Relation $\rho \subseteq V_1 \times V_2$ is a *bisimulation* if for all $(v_1, v_2) \in \varrho$

- $\lambda_1(v_1) = \lambda_2(v_2)$
- for every $v_1' \in \mathcal{N}_G(v_1)$ there is $v_2' \in \mathcal{N}_G(v_2)$ with $(v_1', v_2') \in \varrho$
- for every $v_2' \in \mathcal{N}_G(v_2)$ there is $v_1' \in \mathcal{N}_G(v_1)$ with $(v_1', v_2') \in \varrho$

Observations:

- G_1 and G_2 may be the same
- a union of two bisimulations is also a bisimulation, so we can talk about the maximal bisimulation

This is a classic notion that appears in many areas of CS: verification, databases, modal logics, etc.

Counting Bisimulation

Definition

Let $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$ be graphs. Relation $\rho \subseteq V_1 \times V_2$ is a *counting bisimulation* if for all $(v_1, v_2) \in \varrho$

- $\lambda_1(v_1) = \lambda_2(v_2)$,
- for every $v_1' \in \mathcal{N}_G(v_1)$ there is a distinct $v_2' \in \mathcal{N}_G(v_2)$ with $(v_1', v_2') \in \varrho$
- for every $v_2' \in \mathcal{N}_G(v_2)$ there is a distinct $v_1' \in \mathcal{N}_G(v_1)$ with $(v_1', v_2') \in \varrho$

In other words, the number of neighbours in each counting bisimulation class should now be the same

Example: the same-colouring relation of the result of WL is the maximal counting bisimulation

Partial Counting Bisimulation

Definition

Let $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$ be graphs. Relation $\rho_\ell \subseteq V_1 \times V_2$ is a *ℓ-partial counting bisimulation*, for $\ell \geq 0$, if for all $(v_1, v_2) \in \varrho_\ell$

- $\lambda_1(v_1) = \lambda_2(v_2)$,
- if $\ell > 0$, then for every $v_1' \in \mathcal{N}_G(v_1)$ there is a distinct $v_2' \in \mathcal{N}_G(v_2)$ with $(v_1', v_2') \in \varrho_{\ell-1}$
- if $\ell > 0$, then for every $v_2' \in \mathcal{N}_G(v_2)$ there is a distinct $v_1' \in \mathcal{N}_G(v_1)$ with $(v_1', v_2') \in \varrho_{\ell-1}$

In other words, we now care 'how far we look'

Example: the same-colouring relation of the intermediate result of WL after ℓ rounds is the maximal ℓ-partial counting bisimulation

Bisimulation-Invariant Node Embedings and AC-GNNs

Definition

A node embedding ξ is *(ℓ-partial, counting) bisimulation invariant* if $\xi(G, v) = \xi(G', v')$ for every pair (v, v') in the maximal (ℓ-partial, counting) bisimulation

In other words, it cannot separate bisimilar nodes

Bisimulation-Invariant Node Embedings and AC-GNNs

Definition

A node embedding ξ is *(ℓ-partial, counting) bisimulation invariant* if $\xi(G, v) = \xi(G', v')$ for every pair (v, v') in the maximal (ℓ-partial, counting) bisimulation

In other words, it cannot separate bisimilar nodes

We have:

- Every AC-GNN with L layers realises an L-partial counting bisimulation invariant embedding: both care about tree-unravellings only
- Every L-partial counting bisimulation invariant embedding is realised by an AC-GNN with L layers: take injective aggregate and combine, make the final output function do all the job

Theorem ([PTCK24])

AC-GNNs is equivalent in expressive power to the class of all partial counting bisimulation-invariant embeddings

Bisimulation-Invariant Node Embedings and AC-GNNs

Theorem ([PTCK24])

AC-GNNs is equivalent in expressive power to the class of the ℓ-partial counting bisimulation-invariant embeddings for all ℓ.

Observations:

- Even if partial is unbounded here, it is essential: there are counting bisimulation invariant embeddings not realised by any AC-GNN
- Problem (or not): not constructive, there may not be 'real' GNN that does this
- But it is a good upper bound for all AC-GNNs

3.2 Expressivity via Logic: Graded Modal Logic Case

Comparison to Logic

Logic and GNNs have a lot in common,
both realise functions on graphs:

- Node-level GNNs realise node embedings—that is, functions from graph-node pairs to embeddings
- Unary logic (e.g., FOL) formulas realise functions from structure-element pairs to yes/no

Same correspondence for graph-level GNNs and logical sentences

Comparison to Logic

Logic and GNNs have a lot in common,
both realise functions on graphs:

- Node-level GNNs realise node embedings—that is, functions from graph-node pairs to embeddings
- Unary logic (e.g., FOL) formulas realise functions from structure-element pairs to yes/no

Same correspondence for graph-level GNNs and logical sentences

We have a very fine-grained landscape of logics in terms of expressive power, and it could be beneficial to leverage this knowledge for GNNs

Common Grounds: Inputs and Outputs of Logic and GNNs

We first need to unify the input and output spaces

Input:

- Every logical structure over the signature with one symmetric binary relation and a number of unary relations can be seen as a graph in the GNN sense: we just need to assign a dedicated position in the embedding vector to each unary predicate and do 1-0 encoding

Example: atoms $A(a), C(a)$ for the signature with unary predicates A, B, C can be seen as label $(1, 0, 1)$ of node a.

Common Grounds: Inputs and Outputs of Logic and GNNs

We first need to unify the input and output spaces

Input:

- Every logical structure over the signature with one symmetric binary relation and a number of unary relations can be seen as a graph in the GNN sense: we just need to assign a dedicated position in the embedding vector to each unary predicate and do 1-0 encoding

Example: atoms $A(a), C(a)$ for the signature with unary predicates A, B, C can be seen as label $(1, 0, 1)$ of node a.

Output:

- We can restrict to AC-GNNs with output space $\mathbb{Y} = \{0, 1\}$—that is, to hard binary node classification

Note: Real-life instances of AC-GNNs (GraphSAGE, GCN, etc.) can be converted to such by a 1-0 classification threshold function

Logical and GNN-Based Classifiers

Observations:

- The family of node classifiers realised by reasonable GNN architectures (all AC-GNNs, GCN, GraphSAGE) can express something way beyond any usual logic due to sophisticated aggregation over neighbours, and, especially, non-linearity (sigmoid, ReLU, etc.)
- There are results that find (very expressive) logics capturing, in terms of expressive powers, some of such architectures; we will come back to them
- But our first goal is to restrict ourselves to AC-GNN-based classifiers that are also in FOL

Logical and GNN-Based Classifiers

Observations:

- The family of node classifiers realised by reasonable GNN architectures (all AC-GNNs, GCN, GraphSAGE) can express something way beyond any usual logic due to sophisticated aggregation over neighbours, and, especially, non-linearity (sigmoid, ReLU, etc.)
- There are results that find (very expressive) logics capturing, in terms of expressive powers, some of such architectures; we will come back to them
- But our first goal is to restrict ourselves to AC-GNN-based classifiers that are also in FOL

Our next theorem:

Theorem ([BKM$^+$20])

A node classifier is realisable by both an AC-GNN and FOL formula if and only if it is realisable by a Graded Modal Logic formula.

Logical and GNN-Based Classifiers

Observations:

- The family of node classifiers realised by reasonable GNN architectures (all AC-GNNs, GCN, GraphSAGE) can express something way beyond any usual logic due to sophisticated aggregation over neighbours, and, especially, non-linearity (sigmoid, ReLU, etc.)
- There are results that find (very expressive) logics capturing, in terms of expressive powers, some of such architectures; we will come back to them
- But our first goal is to restrict ourselves to AC-GNN-based classifiers that are also in FOL

Our next theorem:

Theorem ([BKM$^+$20])

A node classifier is realisable by both an AC-GNN and FOL formula if and only if it is realisable by a Graded Modal Logic formula.

We will prove this for a neural subclass of AC-GNNs

Graded Modal Logic

GML has its own syntax and terminology:

$$\phi ::= p \mid \phi \wedge \phi \mid \neg\phi \mid \Diamond_n \phi$$

DL people may know it as $\mathcal{ALCQ}$ with one role name E,
with its own syntax and terminology:

$$C ::= A \mid C \sqcap C \mid \neg C \mid \exists^n E.C$$

However, we can see it as a fragment of unary FOL:

$$\varphi(x) ::= A(x) \mid \varphi(x) \wedge \varphi(x) \mid \neg\varphi(x) \mid \exists^{\geq n} y.\,(E(x,y) \wedge \varphi(y))$$

Notes:

- counting quantifiers $\exists^{\geq n}$ are a sugar in FOL: expressible via inequalities
- but with a cost of extra variables, and so FO^2 is different from FOC^2
- so, GML is inside FOC^2 but not in FO^2

Backward Direction of the Theorem: Building GNNs

Lemma

For every GML formula there is a binary AC-GNN node classifier realising the same embedding.

- Simple GNN classifier:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{trReLU}\left(\boldsymbol{A}\,\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}\right),$$

  where trReLU is the truncated ReLU:
  $\mathit{trReLU}(x) = \{0 \text{ if } x < 0;\ x \text{ if } 0 < x < 1;\ 1 \text{ otherwise}\}$

Backward Direction of the Theorem: Building GNNs

Lemma

For every GML formula there is a binary AC-GNN node classifier realising the same embedding.

- Simple GNN classifier:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{trReLU}\left(\boldsymbol{A}\,\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}\right),$$

  where trReLU is the truncated ReLU:
  $\mathit{trReLU}(x) = \{0 \text{ if } x < 0;\ x \text{ if } 0 < x < 1;\ 1 \text{ otherwise}\}$

- Idea: the feature vectors $\boldsymbol{x}_v^{(\ell)}$ of each node have one component $\boldsymbol{x}_v^{(\ell)}[\varphi'] \in \{0, 1\}$ for each subformula φ' of φ

Backward Direction of the Theorem: Building GNNs

Lemma

For every GML formula there is a binary AC-GNN node classifier realising the same embedding.

- Simple GNN classifier:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{trReLU}\left(\boldsymbol{A}\,\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}\right),$$

  where trReLU is the truncated ReLU:

  $\mathit{trReLU}(x) = \{0 \text{ if } x < 0;\ x \text{ if } 0 < x < 1;\ 1 \text{ otherwise}\}$

- Idea: the feature vectors $\boldsymbol{x}_v^{(\ell)}$ of each node have one component $\boldsymbol{x}_v^{(\ell)}[\varphi'] \in \{0,1\}$ for each subformula φ' of φ
  - $\boldsymbol{x}_v^{(\ell)}[\varphi_1 \wedge \varphi_2] = \mathit{trReLU}(\boldsymbol{x}_v^{(\ell-1)}[\varphi_1] + \boldsymbol{x}_v^{(\ell-1)}[\varphi_2] - 1)$

Backward Direction of the Theorem: Building GNNs

Lemma

For every GML formula there is a binary AC-GNN node classifier realising the same embedding.

- Simple GNN classifier:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{trReLU}\left(\boldsymbol{A}\,\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}\right),$$

  where trReLU is the truncated ReLU:

  $\mathit{trReLU}(x) = \{0 \text{ if } x < 0;\ x \text{ if } 0 < x < 1;\ 1 \text{ otherwise}\}$

- Idea: the feature vectors $\boldsymbol{x}_v^{(\ell)}$ of each node have one component $\boldsymbol{x}_v^{(\ell)}[\varphi'] \in \{0,1\}$ for each subformula φ' of φ
  - $\boldsymbol{x}_v^{(\ell)}[\varphi_1 \wedge \varphi_2] = \mathit{trReLU}(\boldsymbol{x}_v^{(\ell-1)}[\varphi_1] + \boldsymbol{x}_v^{(\ell-1)}[\varphi_2] - 1)$
  - $\boldsymbol{x}_v^{(\ell)}[\neg\varphi'] = \mathit{trReLU}(-\boldsymbol{x}_v^{(\ell-1)}[\varphi'] + 1)$

Backward Direction of the Theorem: Building GNNs

Lemma

For every GML formula there is a binary AC-GNN node classifier realising the same embedding.

- Simple GNN classifier:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{trReLU}\left(\boldsymbol{A}\,\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}\right),$$

  where trReLU is the truncated ReLU:

  $\mathit{trReLU}(x) = \{0 \text{ if } x < 0;\ x \text{ if } 0 < x < 1;\ 1 \text{ otherwise}\}$

- Idea: the feature vectors $\boldsymbol{x}_v^{(\ell)}$ of each node have one component $\boldsymbol{x}_v^{(\ell)}[\varphi'] \in \{0,1\}$ for each subformula φ' of φ
  - $\boldsymbol{x}_v^{(\ell)}[\varphi_1 \wedge \varphi_2] = \mathit{trReLU}(\boldsymbol{x}_v^{(\ell-1)}[\varphi_1] + \boldsymbol{x}_v^{(\ell-1)}[\varphi_2] - 1)$
  - $\boldsymbol{x}_v^{(\ell)}[\neg\varphi'] = \mathit{trReLU}(-\boldsymbol{x}_v^{(\ell-1)}[\varphi'] + 1)$
  - $\boldsymbol{x}_v^{(\ell)}[\exists^{\geq n} y\, E(x,y) \wedge \varphi'] = \mathit{trReLU}(\sum_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}[\varphi'] - (n-1))$

Backward Direction of the Theorem: Building GNNs

Lemma

For every GML formula there is a binary AC-GNN node classifier realising the same embedding.

- Simple GNN classifier:

$$\boldsymbol{x}_v^{(\ell)} := \mathit{trReLU}\left(\boldsymbol{A}\,\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}\left(\sum\nolimits_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}\right),$$

  where trReLU is the truncated ReLU:

  $\mathit{trReLU}(x) = \{0 \text{ if } x < 0;\ x \text{ if } 0 < x < 1;\ 1 \text{ otherwise}\}$

- Idea: the feature vectors $\boldsymbol{x}_v^{(\ell)}$ of each node have one component $\boldsymbol{x}_v^{(\ell)}[\varphi'] \in \{0,1\}$ for each subformula φ' of φ
  - $\boldsymbol{x}_v^{(\ell)}[\varphi_1 \wedge \varphi_2] = \mathit{trReLU}(\boldsymbol{x}_v^{(\ell-1)}[\varphi_1] + \boldsymbol{x}_v^{(\ell-1)}[\varphi_2] - 1)$
  - $\boldsymbol{x}_v^{(\ell)}[\neg\varphi'] = \mathit{trReLU}(-\boldsymbol{x}_v^{(\ell-1)}[\varphi'] + 1)$
  - $\boldsymbol{x}_v^{(\ell)}[\exists^{\geq n} y\, E(x,y) \wedge \varphi'] = \mathit{trReLU}(\sum_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}[\varphi'] - (n-1))$
- For L structure depth of φ, we have $\boldsymbol{x}_v^{(L)}[\varphi] = 1$ iff $v \models \varphi(x)$

Forward Direction: GML for FO-expressible AC-GNNs

Lemma

For every AC-GNN node classifier that is also in FO, there is a GML formula realising the same embedding.

We know:

- AC-GNN node classifiers are all partial counting bisimulation-invariant (i.e., binary embeddings)
- every partial counting bisimulation-invariant embedding is counting bisimulation-invariant
- Van Benthem-style (rather deep) theorem for GML: counting bisimulation-invariant fragment of FO is precisely GML [Ott19]

3.3 Expressivity via Logic: UCQs

Restricted AC-GNNs Equi-Expressible with Some Logic

Can we find a (preferably nice-looking) sub-class of AC-GNN classifiers that is equivalent in expressive power to a known logic?

- For GML (and plain Modal Logic) it may be a bit hard, even probably impossible for a reasonable notion of 'nice-looking' ...
- Something simpler?

Restricted AC-GNNs Equi-Expressible with Some Logic

Can we find a (preferably nice-looking) sub-class of AC-GNN classifiers that is equivalent in expressive power to a known logic?

- For GML (and plain Modal Logic) it may be a bit hard, even probably impossible for a reasonable notion of 'nice-looking' ...
- Something simpler? Tree-shaped UCQs!

Our next result:

Theorem ([TCCGMK23])

Monotonic Max GNNs have the same expressive power as unary tree-shaped UCQs.

Unary Tree-Shaped UCQs

Unary Tree-Shaped Conjunctive Query (tree-CQ):

- an existentially quantified conjunction of unary and binary atoms with 1 free variable and the shape of the tree with the variable as the root
- alternatively, a restriction of our FO fragment for GML:

$$\begin{array}{ll} \textit{GML}: & \phi(x) ::= A(x) \mid \phi(x) \wedge \phi(x) \mid \neg\phi(x) \mid \exists^{\geq \ell} y.\,(E(x,y) \wedge \phi(y)) \\ \textit{tree-CQs}: & \phi(x) ::= A(x) \mid \phi(x) \wedge \phi(x) \mid \exists y.\,(E(x,y) \wedge \phi(y)) \end{array}$$

Unary Tree-Shaped UCQs

Unary Tree-Shaped Conjunctive Query (tree-CQ):

- an existentially quantified conjunction of unary and binary atoms with 1 free variable and the shape of the tree with the variable as the root
- alternatively, a restriction of our FO fragment for GML:

$$\begin{aligned} &\textit{GML}: && \phi(x) ::= A(x) \mid \phi(x) \wedge \phi(x) \mid \neg\phi(x) \mid \exists^{\geq \ell} y.\,(E(x,y) \wedge \phi(y)) \\ &\textit{tree-CQs}: && \phi(x) ::= A(x) \mid \phi(x) \wedge \phi(x) \mid \exists y.\,(E(x,y) \wedge \phi(y)) \end{aligned}$$

A (unary) tree-shaped Union of CQs (tree-UCQ) is a disjunction (or union) of unary tree-CQs

Restricting GNNs

Each tree-UCQ Q is monotonic under homomorphisms:

- for every two graphs $G_1 = (V_1, E_1, \lambda_1)$ and $G_2 = (V_2, E_2, \lambda_2)$ with a function $h : V_1 \to V_2$ such that
  - $h(E_1) \subseteq h(E_2)$ (edges)
  - $\lambda_1(v) \leq \lambda_2(h(v))$ for every $v \in V_1$ (element-wise, nodes)

  we have that if $Q(v)$ is true then $Q(h(v))$ is also true for every $v \in V_1$.

How can we restrict AC-GNN classifiers so their embeddings are also monotonic under homomorphisms?

AC-GNNs are Not Monotonic Under Homomorphisms

Example: Consider a GNN with dimension 3 and 1 layer

$$\boldsymbol{x}_v^{(1)} := ReLU\left(\boldsymbol{A}\boldsymbol{x}_v^{(0)} + \boldsymbol{C}(\Sigma_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(0)}) + \boldsymbol{b}\right)$$

where

- $\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & -2 & 0 \end{pmatrix}$
- $\boldsymbol{C}$ and $\boldsymbol{b}$ consist only of zeroes
- the output classification function is threshold-1 of the 3rd element

For G with single node v with $\lambda(v) = (1 \quad 0 \quad 0)$, the result for v is 1

For G' with single node v with $\lambda(v) = (1 \quad 1 \quad 0)$, the result for v is 0

AC-GNNs are Not Monotonic Under Homomorphisms

Example: Consider a GNN with dimension 3 and 1 layer

$$\boldsymbol{x}_v^{(1)} := ReLU\left(\boldsymbol{A}\boldsymbol{x}_v^{(0)} + \boldsymbol{C}(\Sigma_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(0)}) + \boldsymbol{b}\right)$$

where

- $\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & -2 & 0 \end{pmatrix}$
- $\boldsymbol{C}$ and $\boldsymbol{b}$ consist only of zeroes
- the output classification function is threshold-1 of the 3rd element

For G with single node v with $\lambda(v) = (1 \quad 0 \quad 0)$, the result for v is 1

For G' with single node v with $\lambda(v) = (1 \quad 1 \quad 0)$, the result for v is 0

Negative matrices' weights is not the only problem:
sum can 'count', but UCQs cannot

Monotonic Max GNNs

A monotonic max GNN is

$$\boldsymbol{x}_v^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\max_{u\in\mathcal{N}_G(v)} \boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

such that

- all elements of matrices $\boldsymbol{A}^{(\ell)}$ and $\boldsymbol{C}^{(\ell)}$ are non-negative
- the activation function σ is monotonically increasing, unbounded, and has non-negative range
- the output classification function OUT is a threshold function for some element

Lemma

The embedding of a monotonic max GNN is monotonic under homomorphisms.

So, we have a hope.

Backward Direction of the Theorem: Building GNNs

Lemma

For every unary tree-UCQ there is a monotonic max GNN node classifier realising the same embedding.

Can be proven with minor modifications of the analogous lemma for GML above:

- we have to use max instead of sum
- we cannot use trReLU, since it is bounded, so we have to use usual ReLU
- but we do not need to simulate negation and counting

Forward Direction: tree-UCQs for Monotonic Max GNNs

Lemma

For every Monotonic Max GNN node classifier there is a tree-UCQ realising the same embedding.

Main ideas:

1. We can check whether a tree-CQ is sound for a monotonic max GNN—that is, whether positive answer of the tree-CQ implies positive answer of the GNN for all graphs and nodes
2. We can restrict ourselves to tree-CQs of the depth at most the number of layers of the GNN, and there is only finite number of non-equivalent number of such tree-CQs
3. For every graph G and node v for which the GNN outputs 1 for (G, v), there is a sound tree-CQ that also outputs 1 for (G, v)

So, the union of all (non-equivalent) tree-CQs that are sound for the GNN will do the job

Forward Direction: Step 1

Lemma

For every Monotonic Max GNN node classifier there is a tree-UCQ realising the same embedding.

Main ideas:

1. we can check if a tree-CQ is sound for a monotonic max GNN—that is, whether positive answer of the tree-CQ implies positive answer of the GNN for all graphs and nodes

    - run the GNN on the body of the tree-CQ (as the graph) and see the result for the free variable:
      monotonicity under homomorphisms ensures the result

Forward Direction: Step 2

Lemma

For every Monotonic Max GNN node classifier there is a tree-UCQ realising the same embedding.

Main ideas:

2. we can restrict ourselves to tree-CQs of the depth at most the number of layers of the GNN, and there is only finite number of non-equivalent number of such tree-CQs
    - bounded depth and tree shapes ensure this

Forward Direction: Step 3

Lemma

For every Monotonic Max GNN node classifier there is a tree-UCQ realising the same embedding.

Main ideas:

3. for every graph G and node v for which the GNN outputs 1 for (G, v), there is a sound tree-CQ that also outputs 1 for (G, v)
   - run the GNN on G and convert the 'trace' to a tree-CQ

3.4 Expressivity via Logic: What Else?

Pushing the Result

Monotonic Max GNNs

- are nice: they are a trainable neural architecture with reasonable performance in practice
- but they are also a bit weak: tree-UCQs expressivity is not much really

Can we get such results for something more expressive?

Pushing the Result

Monotonic Max GNNs

- are nice: they are a trainable neural architecture with reasonable performance in practice
- but they are also a bit weak: tree-UCQs expressivity is not much really

Can we get such results for something more expressive?

It is also interesting enough if we can guarantee only one direction: a logic that is more expressive than a class of GNNs

- translation of GNNs to logic is useful,
  it can be seen as explanation, used to reason together with other logical knowledge, etc.

We will cover 3 such results, but without (more complicated) proofs

Something More 1: Monotonic Sum GNNs

A Monotonic Sum GNN is the same as monotonic max, except the aggregation:

$$\boldsymbol{x}_v^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\sum_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

such that

- all elements of matrices $\boldsymbol{A}^{(\ell)}$ and $\boldsymbol{C}^{(\ell)}$ are non-negative
- the activation function σ is monotonically increasing, unbounded, and has non-negative range
- the output classification function OUT is a threshold function for some element

We can guarantee:

Theorem ([TCCGMK23])

Monotonic Sum GNNs have less expressive power than unary tree-shaped UCQs with sibling inequalities.

Something More 1: Tree-UCQs with Sibling Inequalities

Tree-CQ with Sibling Inequalities:

- tree-CQ, but with possible inequalities between the tree siblings
- no nice grammar any more, it is not even in GML (sibling inequalities give us more than just counting)

A (unary) tree-shaped Union of CQs (tree-UCQ) with sibling inequalities is a disjunction (or union) of such tree-CQs.

We only need one lemma

Lemma

For every Monotonic Sum GNN node classifier there is a tree-UCQ with sibling inequalities realising the same embedding.

Something More 1: Key Lemma

Lemma

For every Monotonic Sum GNN node classifier there is a tree-UCQ with sibling inequalities realising the same embedding.

This is a much more complicated result than for max
It is also much less intuitive:

- the result of sum aggregation may be bigger and bigger when we aggregate with more and more neighbours
- a fixed number of inequalities has to take care of all this arbitrary large numbers
- the key observation here is that the GNN is monotonic, and so if we passed a threshold for a positive answer, then we cannot pass it back by adding more neighbours

Something More 2: Eventually Constant Activation

Sum-plus GNN with eventually constant activation is

$$\boldsymbol{x}_v^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\Sigma_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

where σ having

- t_{left} such that $\sigma(x) = \sigma(t_{left})$ for each $x < t_{left}$ and
- t_{right} such that $\sigma(x) = \sigma(t_{right})$ for each $x > t_{right}$

Note: trReLU and many more are such, but not Sigmoid or ReLU

We have a logic capturing this:

Theorem ([BLMT24])

Sum-plus GNNs with eventually constant activation have less expressive power than Local Modal Logic with Presburger Quantifiers.

Something More 2:
Local Modal Logic with Presburger Quantifiers

What is this logic (studied before)?

Something More 2:
Local Modal Logic with Presburger Quantifiers

What is this logic (studied before)?

An extension of GML with much more powerful counting

GML:

$$\phi(x) ::= A(x) \mid \phi(x) \wedge \phi(x) \mid \neg\phi(x) \mid \exists^{\geq \ell} y.\,(E(x,y) \wedge \phi(y))$$

Local Modal Logic with Presburger Quantifiers:

$$\phi(x) ::= \quad \cdots \quad \mid \quad \sum_{i=1}^{k} c_i \cdot \#_y E(x,y) \wedge \phi(y) \;\geq\; d$$

Lemma

For every Sum-plus GNNs with eventually constant activation there is a formula in Local Modal Logic with Presburger Quantifiers realising the same embedding.

Something More 3: GNNs with Piecewise-Linear Activation

Sum-plus GNN with piecewise-linear activation is

$$\boldsymbol{x}_v^{(\ell)} := \sigma\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\textstyle\sum_{u\in\mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

where σ being piece-wise linear (ReLU, trReLU, etc.) and matrix&vector weights are rational

GNN with piecewise-linear activation is a generalisation where we can also have avg, max, min, etc. instead of sum and concatenation, etc. instead of + (i.e., nearly everything).

We also have a (rather complex) logic capturing this:

Theorem ([Gro23])

GNNs with piecewise-linear activation have less expressive power than GFO+C.

I prefer not to go to the details.

3.5 Beyond Classifiers

Something About Numeric Output Domains

When comparing to logic, we concentrated on outputs $\mathbb{Y} = \{0, 1\}$.

Can we say something for, for example, $\mathbb{Y} = \mathcal{R}$ (i.e., regression)?

Something About Numeric Output Domains

When comparing to logic, we concentrated on outputs $\mathbb{Y} = \{0, 1\}$.

Can we say something for, for example, $\mathbb{Y} = \mathcal{R}$ (i.e., regression)?

Of course, we cannot compare to logic any more.

But we can compare different architectures between each other:

Something About Numeric Output Domains

When comparing to logic, we concentrated on outputs $\mathbb{Y} = \{0, 1\}$.

Can we say something for, for example, $\mathbb{Y} = \mathcal{R}$ (i.e., regression)?

Of course, we cannot compare to logic any more.

But we can compare different architectures between each other:

Theorem ([RTG23])

GNNs with

$$\boldsymbol{x}_v^{(\ell)} := \mathit{ReLU}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} + \boldsymbol{C}^{(\ell)}\left(\mathit{agg}_{u \in \mathcal{N}_G(v)}\boldsymbol{x}_u^{(\ell-1)}\right) + \boldsymbol{b}^{(\ell)}\right)$$

and $\mathbb{Y} = \mathbb{R}$ (i.e., identity output), for agg one of max, sum, avg are pairwise incomparable in expressive power.

4 Non-Uniform Expressive and Approximation Power

Restricting Graphs

As we see AC-GNNs have two conceptual limitations:

1. they cannot look further than their number of layers
2. they are 'blind' for cycles, and only see unravellings

Can we restrict the graphs so that AC-GNNs can express more than for all graphs?

Maybe, even express everything (i.e., to be universal)?

One trivial example:

Proposition

AC-GNNs are universal over trees of bounded depth.

Restricting Graphs

As we see AC-GNNs have two conceptual limitations:

1. they cannot look further than their number of layers
2. they are 'blind' for cycles, and only see unravellings

Can we restrict the graphs so that AC-GNNs can express more than for all graphs?

Maybe, even express everything (i.e., to be universal)?

One trivial example:

Proposition

AC-GNNs are universal over trees of bounded depth.

Something more interesting?

Non-Uniform Expressive Power

Definition

A class $\mathcal{C}_2$ of node embeddings has as *strong non-uniform expressive power* as class $\mathcal{C}_1$ if for every $n \in \mathbb{N}$ and every $\xi_1 \in \mathcal{C}_1$ there is $\xi_2 \in \mathcal{C}_2$ such that ξ_1 and ξ_2 agree on all nodes of all graphs with at most n nodes

Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the *same non-uniform expressive power* if both directions hold

Non-uniform expressive power:

- 'between' distinguishability, which allows its own 'simulator' ξ_2 for every two outputs of ξ_1, and uniform expressibility, which requires one single 'simulator' for all the cases
- can be seen as a restriction on graphs that overcomes Limitation 1 from previous slide
- may make sense from practical point of view: we may be interested in graphs of bounded size

An Easy Result: Graphs with Unique Node IDs

Recall AC-GNNs with

$$\boldsymbol{x}_v^{(\ell)} := \mathit{Sigmoid}\left(\boldsymbol{A}^{(\ell)}\boldsymbol{x}_v^{(\ell-1)} \;||\; \boldsymbol{C}^{(\ell)}\left(\textstyle\sum_{u\in\mathcal{N}_G(v)} f\left(\boldsymbol{x}_u^{(\ell-1)}\right)\right)\right)$$

with smart f such that $\sum_{x\in X} f(x)$ is injective for bounded X (i.e., for graphs of bounded size)

Theorem ([Lou20])

Such AC-GNNs with arbitrary output functions are non-uniform universal (i.e., can express any function) for the class of connected graphs with unique node labels

Proof ideas:

- injective aggregate and combine allows to unambiguously reconstruct the whole graph
- then, the output function can do any job we need

Improved Result: Graphs with Random Node Initialisations

Unique node labels (i.e., IDs) is a strict requirement

So, we can relax this and require uniqueness with high probability

This can be achieved by random node initialisation (RNI) where one component of the input graph labels is from a random distribution

This model is not covered in our formalisation (random input $\to$ random output), but we can relativise our setting, including expressive power with 'high probability'

Theorem ([ACGL21])

AC-GNNs as on the previous slide with RNI are non-uniform universal for the class of connected graphs

The proof is a relativisation of the previous one

Advanced Result: Non-Uniform Logic-Like Characterisation

There is a logic-like formalism that non-uniformly capture nearly all 'neural' AC-GNNs [Gro23]

Advanced Result: Non-Uniform Logic-Like Characterisation

There is a logic-like formalism that non-uniformly capture nearly all 'neural' AC-GNNs [Gro23]

I do not dare to present this result here.

(Non-Uniform) Approximation Power

Definition

A class $\mathcal{C}_2$ of node embeddings has as *strong (non-uniform) approximation power* as class $\mathcal{C}_1$ if for every $n \in \mathbb{N}$, every ϵ and every $\xi_1 \in \mathcal{C}_1$ there is $\xi_2^\epsilon \in \mathcal{C}_2$ such that embeddings of ξ_1 and ξ_2^ϵ ϵ-agree (additively) on all nodes of all graphs with at most n nodes

Classes $\mathcal{C}_1$ and $\mathcal{C}_2$ have the *same approximation power* if both directions hold

One result in this category:

Theorem ([GR22])

On compact sets of graphs (e.g., where initial labellings are bounded), sum-plus GNNs can approximate any continuous counting bisimulation invariant function (i.e., WL-bounded in terms of distinguishability).

5 Wrapping Up

Beyond AC-GNNs

Limitation in expressivity motivates people to invent:

- k-WL GNNs [MRF+19] consider k-tuples of nodes (pair, triples, etc.) instead of just nodes;
  can express the presence of a k-cycle
- Recurrent GNNs [PTCK24] have the same aggregate and combine functions for all layers, and iterate them for more than bounded number of times;
  can express reachability
- ...

All these generalisations (as well as some restrictions, e.g., GCNs) deserve our expressivity studies

What is next?

Open questions:

- Many questions to complete distinguishing, expressive, approximation power landscapes
  - For example, can we design a truly universal GNN-like architecture?
- Many logics from today are rather crazy; even if not, the formulas equivalent to GNNs are huge. What to do with this?
  - For example, design an algorithm that gives a reasonably constrained formula (by size, shape) that is as close as possible to a given GNN
- The fact that some GNN exists, does not mean at all that it is possible to train it with any reasonable training procedure. Can we describe 'trainable' GNN subclasses?
  - For example, can we guarantee that we can train GNNs realising embeddings of GML formulas with any reasonable quality?

Conclusion

Fascinating topic. But hard.

Conclusion

Fascinating topic. But hard.

PS: feel free to drop me an email (e.g., asking for references)

References i

[ACGL21] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz.
The surprising power of graph neural networks with random node initialization.
In *International Joint Conference on Artificial Intelligence*, volume 30, pages 2112–2118, 2021.

[BKM+20] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva.
The logical expressiveness of graph neural networks.
In *International Conference on Learning Representations*, volume 8, 2020.

References ii

[BLMT24] Michael Benedikt, Chia-Hsuan Lu, Boris Motik, and Tony Tan.
Decidability of graph neural networks via logical characterizations.
In *International Colloquium on Automata, Languages, and Programming*, volume 297 of *LIPIcs*, pages 127:1–127:20, 2024.

[GR22] Floris Geerts and Juan Reutter.
Expressiveness and approximation properties of graph neural networks.
In *International Conference on Learning Representations*, volume 10, 2022.

References iii

[Gro23] Martin Grohe.
The descriptive complexity of graph neural networks.
In *ACM/IEEE Symposium on Logic in Computer Science*, volume 38, pages 1–14. ACM, 2023.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl.
Neural message passing for quantum chemistry.
In *International Conference on Machine Learning*, pages 1263–1272, 2017.

References iv

[HYL17] William L. Hamilton, Zhitao Ying, and Jure Leskovec. **Inductive representation learning on large graphs.** In *Advances in Neural Information Processing Systems 30*, pages 1024–1034, 2017.

[Lou20] Andreas Loukas. **What graph neural networks cannot learn: Depth vs width.** In *International Conference on Learning Representations*, volume 8, 2020.

References v

[MRF+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe.
Weisfeiler and leman go neural: Higher-order graph neural networks.
In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[Ott19] Martin Otto.
Graded modal logic and counting bisimulation.
arXiv preprint arXiv:1910.00039, 2019.

References vi

[PTCK24] Maximilian Pflüger, David Tena Cucala, and Egor V. Kostylev.
Recurrent graph neural networks and their connections to bisimulation and logic.
In *AAAI Conference on Artificial Intelligence*, volume 38, 2024.

[RTG23] Eran Rosenbluth, Jan Tönshoff, and Martin Grohe.
Some might say all you need is sum.
In *International Joint Conference on Artificial Intelligence*, volume 32, pages 4172–4179. International Joint Conferences on Artificial Intelligence Organization, 2023.

[SGT+08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini.
The graph neural network model.
IEEE Transactions on Neural Networks, pages 61–80, 2008.

[SKB+18] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling.
Modeling relational data with graph convolutional networks.
In *European Semantic Web Conference*, volume 15, pages 593–607. Springer, 2018.

References viii

[TCCGMK23] David Tena Cucala, Bernardo Cuenca Grau, Boris Motik, and Egor V. Kostylev.
On the correspondence between monotonic max-sum gnns and datalog.
In *International Conference on Principles of Knowledge Representation and Reasoning*, volume 20, 2023.

[WL68] Boris Weisfeiler and Andrei Leman.
A reduction of a graph to a canonical form and an algebra arising during this reduction.
Nauchno-Technicheskaya Informatsia, 2(9):12–16, 1968.
Translated from Russian.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka.
How powerful are graph neural networks?
In *International Conference on Learning Representations*, volume 7, 2019.