Logical Expressiveness of Graph Neural Networks

DIG seminar

Mikaël Monet

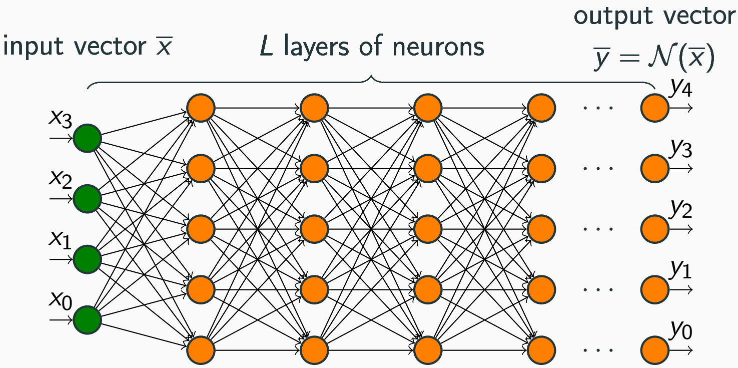
March 12th, 2020

Millennium Institute for Foundational Research on Data, Chile

Graph Neural Networks (GNNs)

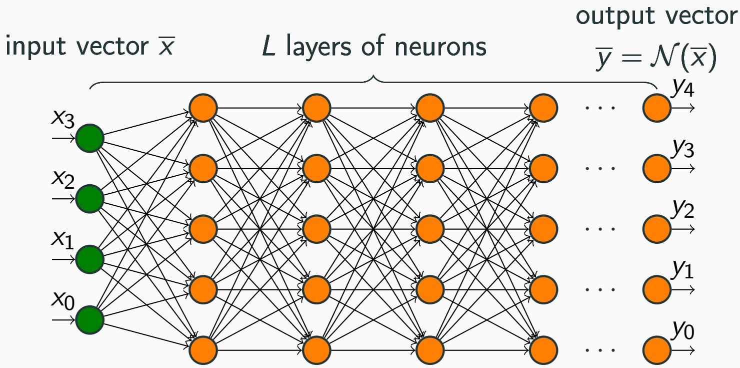
- **With:** Pablo Barceló, Egor Kostylev, Jorge Pérez, Juan Reutter, Juan Pablo Silva
- Graph Neural Networks (GNNs) [Merkwirth and Lengauer, 2005, Scarselli et al., 2009]: a class of NN architectures that has recently become popular to deal with structured data
  - → Goal: understand what they are, and their theoretical properties

Neural Networks (NNs)



A fully connected neural network $\mathcal{N}$.

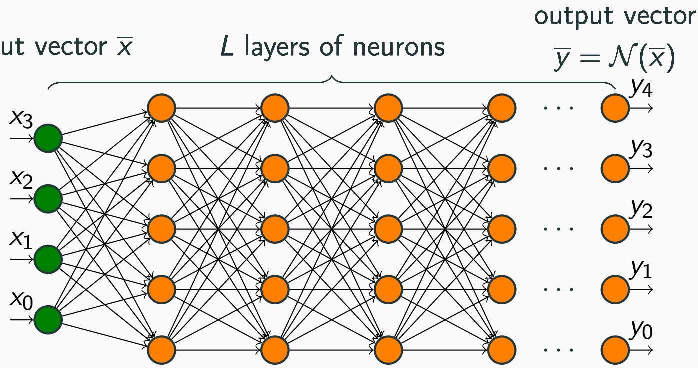
Neural Networks (NNs)



A fully connected neural network $\mathcal{N}$.

- Weight $w_{n' \to n}$ between two consecutive neurons

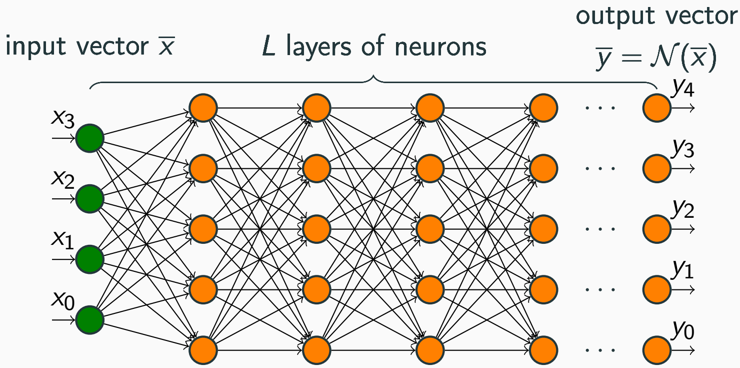
Neural Networks (NNs)



A fully connected neural network $\mathcal{N}$.

- Weight $w_{n' \to n}$ between two consecutive neurons
- Compute left to right $\lambda(n) := f(\sum w_{n' \to n} \times \lambda(n'))$

Neural Networks (NNs)



A fully connected neural network $\mathcal{N}$.

- Weight $w_{n' \to n}$ between two consecutive 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, clustering, regression, etc.)

Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where g is what you want to learn

Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where g is what you want to learn

$\rightarrow$ use backpropagation algorithms

Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where g is what you want to learn

$\rightarrow$ use backpropagation algorithms

- **Problem**: for fully connected NNs, when a layer has many neurons there are a lot of weights...

Finding the weights

- Goal: find the weights that "solve" your problem

$\rightarrow$ minimize $\mathrm{Dist}(\mathcal{N}(\overline{x}), g(\overline{x}))$, where g is what you want to learn

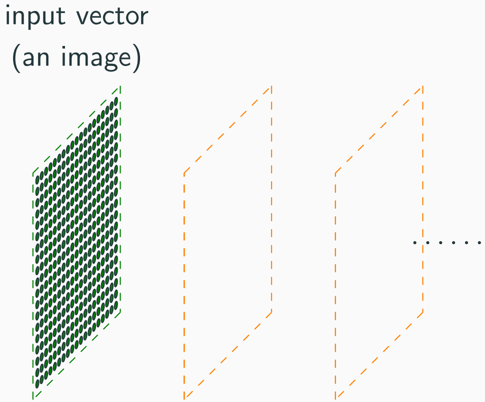
$\rightarrow$ use backpropagation algorithms

- **Problem**: for fully connected NNs, when a layer has many neurons there are a lot of weights...

$\rightarrow$ example: input is a 250×250 pixels image, and we want to build a fully connected NN with 500 neurons per layer

$\rightarrow$ between the first two layers we have $250 \times 250 \times 500 = \mathbf{31,250,000}$ weights

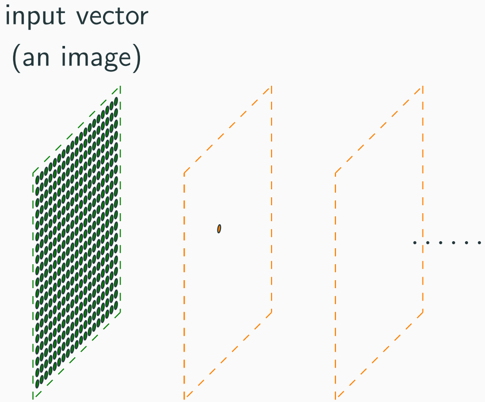
Convolutional Neural Networks



A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

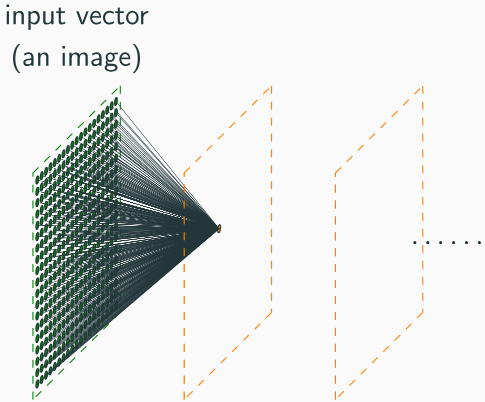
Convolutional Neural Networks



A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

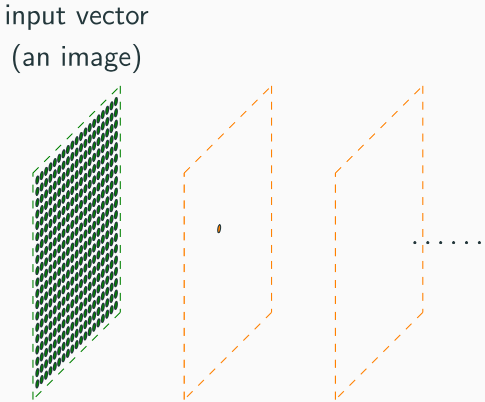
Convolutional Neural Networks



A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

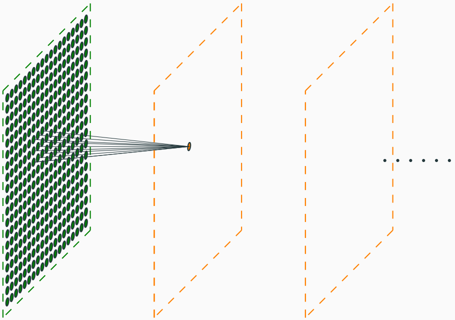
Convolutional Neural Networks



A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

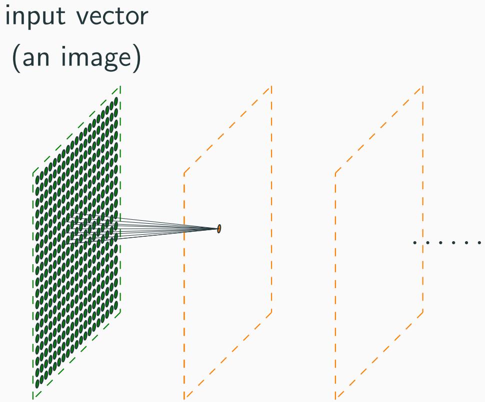
Convolutional Neural Networks



A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

Convolutional Neural Networks

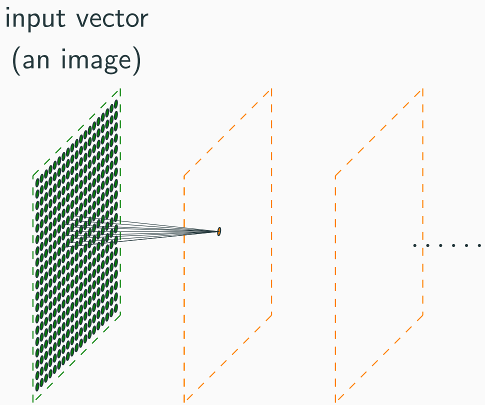


A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

→ fewer weights to learn (e.g, $500 * 9 = 4,500$ for the first layer)

Convolutional Neural Networks



A convolutional neural network.

- **Idea**: use the structure of the data (here, a grid)

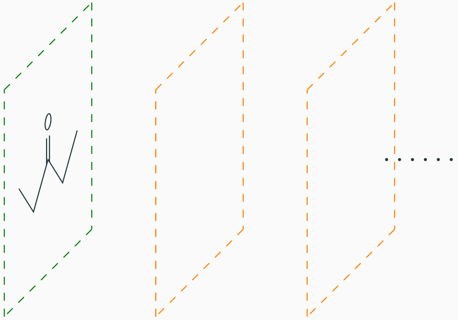
$\rightarrow$ fewer weights to learn (e.g, $500 * 9 = 4,500$ for the first layer)

$\rightarrow$ other advantage: recognize patterns that are local

Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [Duvenaud et al., 2015])



A (convolutional) graph neural network.

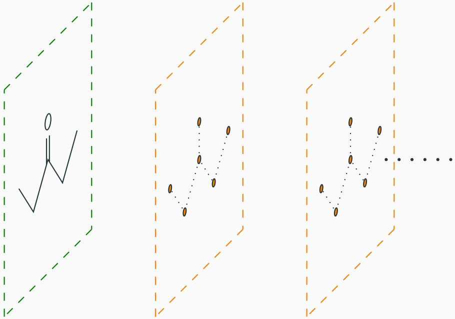
- **Idea**: use the structure of the data

$\rightarrow$ GNNs generalize this idea to allow *any* graph as input

Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [Duvenaud et al., 2015])



A (convolutional) graph neural network.

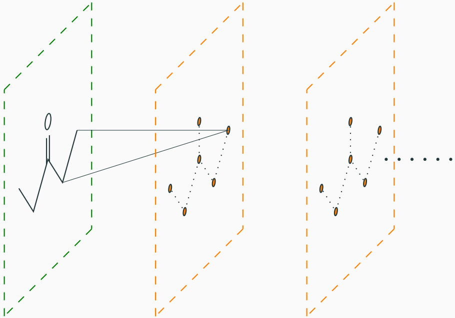
- **Idea**: use the structure of the data

$\rightarrow$ GNNs generalize this idea to allow *any* graph as input

Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [Duvenaud et al., 2015])



A (convolutional) graph neural network.

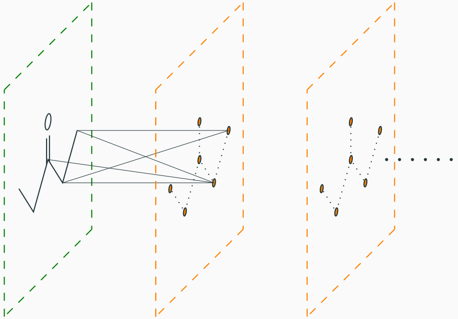
- **Idea**: use the structure of the data

$\rightarrow$ GNNs generalize this idea to allow *any* graph as input

Graph Neural Networks (GNNs)

input vector
(a molecule)

output:
is it poisonous? (e.g., [Duvenaud et al., 2015])

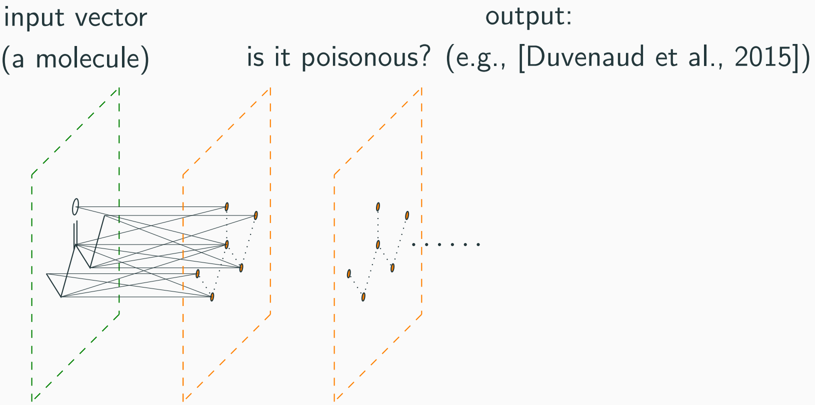


A (convolutional) graph neural network.

- **Idea**: use the structure of the data

$\rightarrow$ GNNs generalize this idea to allow *any* graph as input

Graph Neural Networks (GNNs)

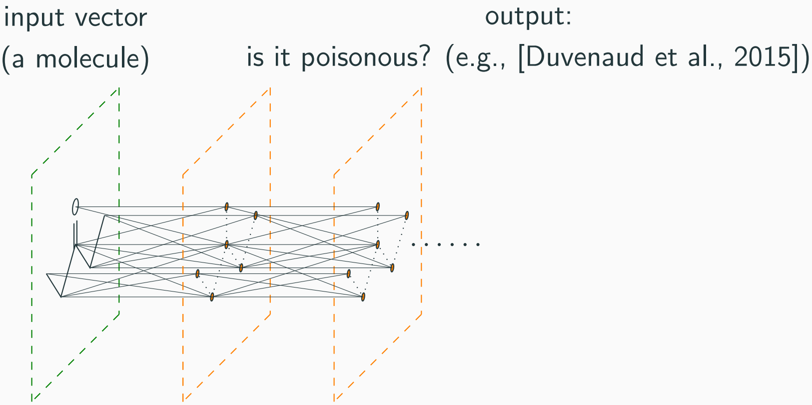


A (convolutional) graph neural network.

- **Idea**: use the structure of the data

$\rightarrow$ GNNs generalize this idea to allow *any* graph as input

Graph Neural Networks (GNNs)



A (convolutional) graph neural network.

- **Idea**: use the structure of the data

$\rightarrow$ GNNs generalize this idea to allow *any* graph as input

Question: what can we do with graph neural networks? (from a *theoretical* perspective)

GNNs: formalisation

- Simple, undirected, node-labeled graph $G = (V, E, \lambda)$, where $\lambda : V \to \mathbb{R}^d$

GNNs: formalisation

- Simple, undirected, node-labeled graph $G = (V, E, \lambda)$, where $\lambda : V \to \mathbb{R}^d$
- Run of a GNN with L layers on G: iteratively compute $\boldsymbol{x}_u^{(i)} \in \mathbb{R}^d$ for $0 \leq i \leq L$ as follows:

GNNs: formalisation

- Simple, undirected, node-labeled graph $G = (V, E, \lambda)$, where $\lambda : V \to \mathbb{R}^d$
- Run of a GNN with L layers on G: iteratively compute $\boldsymbol{x}_u^{(i)} \in \mathbb{R}^d$ for $0 \leq i \leq L$ as follows:

$\rightarrow$ $\boldsymbol{x}_u^{(0)} := \lambda(u)$

GNNs: formalisation

- Simple, undirected, node-labeled graph $G = (V, E, \lambda)$, where $\lambda : V \to \mathbb{R}^d$
- Run of a GNN with L layers on G: iteratively compute $\boldsymbol{x}_u^{(i)} \in \mathbb{R}^d$ for $0 \leq i \leq L$ as follows:

$\rightarrow$ $\boldsymbol{x}_u^{(0)} := \lambda(u)$

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

- Where the $\mathrm{AGG}^{(i)}$ are called *aggregation functions* and the $\mathrm{COMB}^{(i)}$ *combination functions*

GNNs: formalisation

- Simple, undirected, node-labeled graph $G = (V, E, \lambda)$, where $\lambda : V \to \mathbb{R}^d$
- Run of a GNN with L layers on G: iteratively compute $\boldsymbol{x}_u^{(i)} \in \mathbb{R}^d$ for $0 \leq i \leq L$ as follows:

$\to$ $\boldsymbol{x}_u^{(0)} := \lambda(u)$

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

- Where the $\mathrm{AGG}^{(i)}$ are called *aggregation functions* and the $\mathrm{COMB}^{(i)}$ *combination functions*
- Let us call such a GNN an aggregate-combine GNN (AC-GNN)

Link with Weisfeiler-Lehman

- Recently, [Morris et al., 2019, Xu et al., 2019] established a link with the *Weisfeiler-Lehman (WL) isomorphism test*

Link with Weisfeiler-Lehman

- Recently, [Morris et al., 2019, Xu et al., 2019] established a link with the *Weisfeiler-Lehman (WL) isomorphism test*

$\rightarrow$ A **heuristic** to determine if two graphs are isomorphic (also called color refinement)

Link with Weisfeiler-Lehman

- Recently, [Morris et al., 2019, Xu et al., 2019] established a link with the *Weisfeiler-Lehman (WL) isomorphism test*

→ A **heuristic** to determine if two graphs are isomorphic (also called color refinement)

  1. Start from two graphs, with all nodes having the same color

Link with Weisfeiler-Lehman

- Recently, [Morris et al., 2019, Xu et al., 2019] established a link with the *Weisfeiler-Lehman (WL) isomorphism test*

→ A **heuristic** to determine if two graphs are isomorphic (also called color refinement)

1. Start from two graphs, with all nodes having the same color
2. At the next step, two nodes v, v' of the same color are assigned different colors if there is a color c such that v and v' have a different number of neighbors with color c

Link with Weisfeiler-Lehman

- Recently, [Morris et al., 2019, Xu et al., 2019] established a link with the *Weisfeiler-Lehman (WL) isomorphism test*

→ A **heuristic** to determine if two graphs are isomorphic (also called color refinement)

1. Start from two graphs, with all nodes having the same color
2. At the next step, two nodes v, v' of the same color are assigned different colors if there is a color c such that v and v' have a different number of neighbors with color c
3. Iterate step 2 until the coloring is stable (the partition of the nodes into colors does not change)

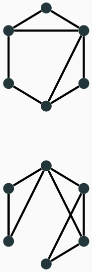
Link with Weisfeiler-Lehman

- Recently, [Morris et al., 2019, Xu et al., 2019] established a link with the *Weisfeiler-Lehman (WL) isomorphism test*

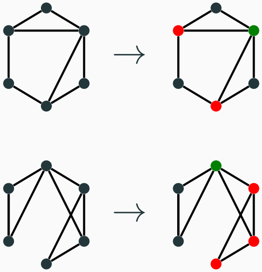
→ A **heuristic** to determine if two graphs are isomorphic (also called color refinement)

1. Start from two graphs, with all nodes having the same color
2. At the next step, two nodes v, v' of the same color are assigned different colors if there is a color c such that v and v' have a different number of neighbors with color c
3. Iterate step 2 until the coloring is stable (the partition of the nodes into colors does not change)
4. If the two graphs have the same multiset of colors, **accept**, else **reject**

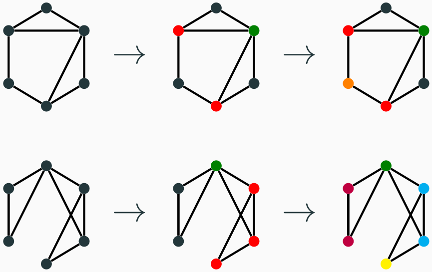
Weisfeiler-Lehman: example 1



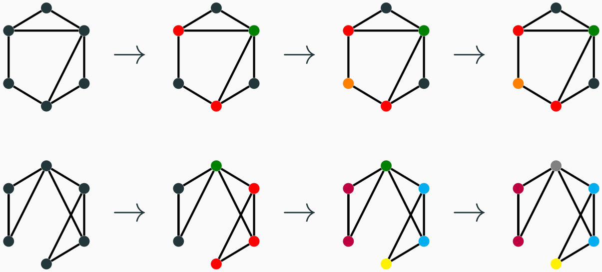
Weisfeiler-Lehman: example 1



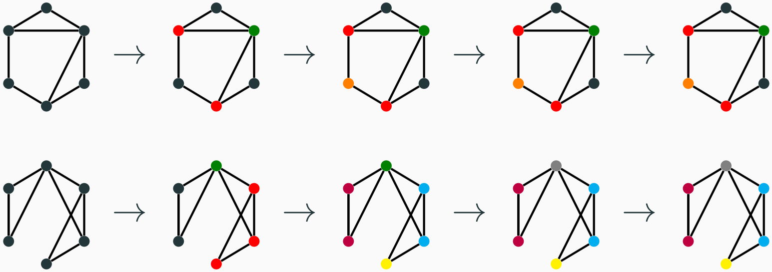
Weisfeiler-Lehman: example 1



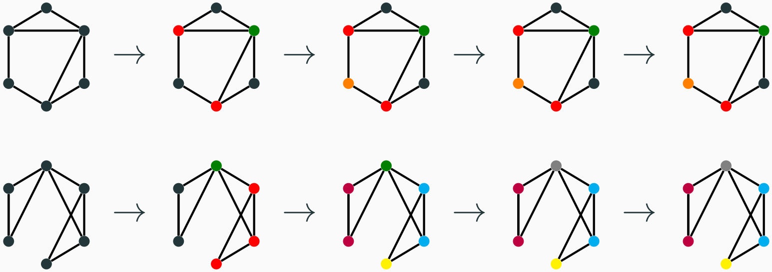
Weisfeiler-Lehman: example 1



Weisfeiler-Lehman: example 1



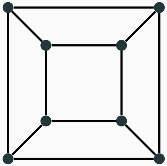
Weisfeiler-Lehman: example 1



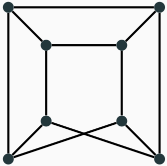
$$\{\{\textcolor{red}{\bullet}, \textcolor{red}{\bullet}, \textcolor{orange}{\bullet}, \bullet, \bullet, \textcolor{green}{\bullet}\}\} \neq \{\{\textcolor{purple}{\bullet}, \textcolor{purple}{\bullet}, \textcolor{gray}{\bullet}, \textcolor{cyan}{\bullet}, \textcolor{cyan}{\bullet}, \textcolor{yellow}{\bullet}\}\}$$

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

Weisfeiler-Lehman: example 2

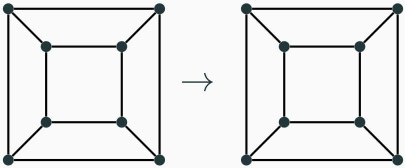


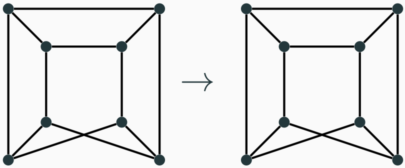
$\rightarrow$



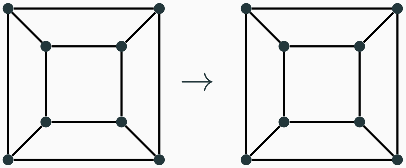
$\rightarrow$

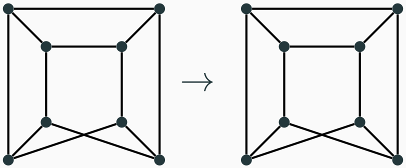
Weisfeiler-Lehman: example 2





Weisfeiler-Lehman: example 2





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

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

Link between AC-GNNs and Weisfeiler-Lehman

Weisfeiler-Lehman works like this:

- $\mathrm{WL}_{\mathrm{u}}^{(0)} \coloneqq \lambda(\mathrm{u})$
- $\mathrm{WL}_{\mathrm{u}}^{(\mathrm{i}+1)} \coloneqq \mathrm{HASH}^{(\mathrm{i}+1)}(\mathrm{WL}_{\mathrm{u}}^{(\mathrm{i})}, \{\{\mathrm{WL}_{\mathrm{v}}^{(\mathrm{i})} \mid \mathrm{v} \in \mathcal{N}_{\mathrm{G}}(\mathrm{u})\}\})$

Link between AC-GNNs and Weisfeiler-Lehman

Weisfeiler-Lehman works like this:

- $\mathrm{WL}_{\mathrm{u}}^{(0)} \coloneqq \lambda(\mathrm{u})$
- $\mathrm{WL}_{\mathrm{u}}^{(\mathrm{i+1})} \coloneqq \mathrm{HASH}^{(\mathrm{i+1})}(\mathrm{WL}_{\mathrm{u}}^{(\mathrm{i})}, \{\{\mathrm{WL}_{\mathrm{v}}^{(\mathrm{i})} \mid \mathrm{v} \in \mathcal{N}_{\mathrm{G}}(\mathrm{u})\}\})$

Aggregate-combine GNNs work like this:

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

Link between AC-GNNs and Weisfeiler-Lehman

Weisfeiler-Lehman works like this:

- $\mathrm{WL}_{\mathrm{u}}^{(0)} \coloneqq \lambda(\mathrm{u})$
- $\mathrm{WL}_{\mathrm{u}}^{(\mathrm{i}+1)} \coloneqq \mathrm{HASH}^{(\mathrm{i}+1)}(\mathrm{WL}_{\mathrm{u}}^{(\mathrm{i})}, \{\{\mathrm{WL}_{\mathrm{v}}^{(\mathrm{i})} \mid \mathrm{v} \in \mathcal{N}_{\mathrm{G}}(\mathrm{u})\}\})$

Aggregate-combine GNNs work like this:

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

$\rightarrow$ WL works exactly like an AC-GNNs with **injective** aggregation and combination functions

Link between AC-GNNs and Weisfeiler-Lehman

Weisfeiler-Lehman works like this:

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

Aggregate-combine GNNs work like this:

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

$\rightarrow$ WL works exactly like an AC-GNNs with **injective** aggregation and combination functions

Corollary ([Morris et al., 2019, Xu et al., 2019])

If WL assigns the same value to two nodes at round i, then any AC-GNN will also assign the same value to these two nodes at round i

Binary classifiers GNNs

Corollary ([Morris et al., 2019, Xu et al., 2019])

If WL assigns the same value to two nodes at round i, then any AC-GNN will also assign the same value to these two nodes at round i

- Is this all there is to say?
- Binary node classifier GNN: the final feature of every node is 0 or 1

Binary classifiers GNNs

Corollary ([Morris et al., 2019, Xu et al., 2019])

If WL assigns the same value to two nodes at round i, then any AC-GNN will also assign the same value to these two nodes at round i

- Is this all there is to say?
- Binary node classifier GNN: the final feature of every node is 0 or 1

$\rightarrow$ What are the binary node classifiers that a GNN can learn?

Binary classifiers GNNs

Corollary ([Morris et al., 2019, Xu et al., 2019])

If WL assigns the same value to two nodes at round i, then any AC-GNN will also assign the same value to these two nodes at round i

- Is this all there is to say?
- Binary node classifier GNN: the final feature of every node is 0 or 1

$\rightarrow$ What are the binary node classifiers that a GNN can learn?

- For instance, **logical classifiers**?

Link between WL and first-order logic

- There is a link between the WL test and first-order logic with 2 variables and counting (FOC_2)

Link between WL and first-order logic

- There is a link between the WL test and first-order logic with 2 variables and counting (FOC_2)

$\rightarrow$ example: $\varphi(x) = \exists^{\geq 5} y(E(x,y) \vee \exists^{\geq 2} x(\neg E(y,x) \wedge C(x)))$

Link between WL and first-order logic

- There is a link between the WL test and first-order logic with 2 variables and counting (FOC_2)

$\rightarrow$ example: $\varphi(x) = \exists^{\geq 5} y(E(x,y) \vee \exists^{\geq 2} x(\neg E(y,x) \wedge C(x)))$

Theorem ([Cai et al., 1992])

We have $\mathrm{WL}_u^{(i)} = \mathrm{WL}_v^{(i)}$ *if and only if* u *and* v *agree on all* FOC_2 *unary formulas of quantifier depth* $\leq i$ *in* G

Link between WL and first-order logic

- There is a link between the WL test and first-order logic with 2 variables and counting (FOC_2)

$\rightarrow$ example: $\varphi(x) = \exists^{\geq 5} y(E(x,y) \vee \exists^{\geq 2} x(\neg E(y,x) \wedge C(x)))$

Theorem ([Cai et al., 1992])

We have $\mathsf{WL}_u^{(i)} = \mathsf{WL}_v^{(i)}$ if and only if u and v agree on all FOC_2 unary formulas of quantifier depth $\leq i$ in G

- Given these connections, we ask: let $\varphi(x)$ be a unary FOC_2 formula. Can we "capture" it with an AC-GNN?
  - (capture: after some number L of layers, we have $\boldsymbol{x}_u^{(L)} = 1$ if $(G,u) \models \varphi(x)$ and $\boldsymbol{x}_u^{(L)} = 0$ if $(G,u) \not\models \varphi(x)$)

Link between WL and first-order logic

- There is a link between the WL test and first-order logic with 2 variables and counting (FOC_2)

$\rightarrow$ example: $\varphi(x) = \exists^{\geq 5} y(E(x, y) \vee \exists^{\geq 2} x(\neg E(y, x) \wedge C(x)))$

Theorem ([Cai et al., 1992])

We have $\mathsf{WL}_u^{(i)} = \mathsf{WL}_v^{(i)}$ if and only if u and v agree on all FOC_2 unary formulas of quantifier depth $\leq i$ in G

- Given these connections, we ask: let $\varphi(x)$ be a unary FOC_2 formula. Can we "capture" it with an AC-GNN?
  - (capture: after some number L of layers, we have $\boldsymbol{x}_u^{(L)} = 1$ if $(G, u) \models \varphi(x)$ and $\boldsymbol{x}_u^{(L)} = 0$ if $(G, u) \not\models \varphi(x)$)

$\rightarrow$ We answer this!

AC-GNNs for FOC_2: graded modal logic

- **Observation**: there are FOC_2 unary formulas that we cannot capture with any AC-GNN
  - $\rightarrow$ $\varphi(x) = \mathrm{Blue}(x) \wedge \exists y\, \mathrm{Red}(y)$

AC-GNNs for FOC_2: graded modal logic

- **Observation**: there are FOC_2 unary formulas that we cannot capture with any AC-GNN
  - $\rightarrow$ $\varphi(x) = \mathrm{Blue}(x) \wedge \exists y\, \mathrm{Red}(y)$

    G_1 : • •, G_2 : • •

AC-GNNs for FOC_2: graded modal logic

- **Observation**: there are FOC_2 unary formulas that we cannot capture with any AC-GNN
  → $\varphi(x) = \mathrm{Blue}(x) \wedge \exists y\, \mathrm{Red}(y)$
  G_1 : • •, G_2 : • •
- What are the FOC_2 formulas that can be captured by an AC-GNN?

AC-GNNs for FOC_2: graded modal logic

- **Observation**: there are FOC_2 unary formulas that we cannot capture with any AC-GNN
  - $\rightarrow$ $\varphi(x) = \mathrm{Blue}(x) \wedge \exists y\, \mathrm{Red}(y)$

    G_1 : $\bullet$ $\bullet$, G_2 : $\bullet$ $\bullet$
- What are the FOC_2 formulas that can be captured by an AC-GNN?

$\rightarrow$ Graded modal logic [de Rijke, 2000]: syntactical fragment of FOC_2 in which quantifiers are only of the form $\exists^{\geq N} y\, (E(x,y) \wedge \varphi'(y))$

(Also called $\mathcal{ALCQ}$ in description logics)

AC-GNNs for FOC_2: graded modal logic

- **Observation**: there are FOC_2 unary formulas that we cannot capture with any AC-GNN
  $\rightarrow$ $\varphi(x) = \mathrm{Blue}(x) \wedge \exists y\, \mathrm{Red}(y)$
  G_1 : $\color{blue}\bullet$ $\color{red}\bullet$, G_2 : $\color{blue}\bullet$ $\color{blue}\bullet$
- What are the FOC_2 formulas that can be captured by an AC-GNN?

$\rightarrow$ Graded modal logic [de Rijke, 2000]: syntactical fragment of FOC_2 in which quantifiers are only of the form $\exists^{\geq N} y\, (E(x,y) \wedge \varphi'(y))$
(Also called $\mathcal{ALCQ}$ in description logics)

Theorem

Let φ be a unary FOC formula. If φ is equivalent to a graded modal logic formula, then φ can be captured by an AC-GNN, otherwise it cannot.

Positive result: building simple GNNs

- We say that a GNN is simple if we update according to

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

  where f is the truncated ReLU (zero if ≤ 0, one if ≥ 1, identity in between)

Positive result: building simple GNNs

- We say that a GNN is simple if we update according to

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

  where f is the truncated ReLU (zero if ≤ 0, one if ≥ 1, identity in between)
- **Idea**: the feature vectors $\boldsymbol{x}_u^{(i)}$ of each node have one component $\boldsymbol{x}_u^{(i)}(\varphi') \in \{0, 1\}$ for each subformula φ' of φ

Positive result: building simple GNNs

- We say that a GNN is simple if we update according to

$$\mathbf{x}_u^{(i+1)} := f\left(\mathbf{C}^{(i)}\mathbf{x}_u^{(i)} + \mathbf{A}^{(i)}\left(\sum_{v\in\mathcal{N}_G(u)} \mathbf{x}_v^{(i)}\right) + \mathbf{b}^{(i)}\right),$$

  where f is the truncated ReLU (zero if ≤ 0, one if ≥ 1, identity in between)
- **Idea**: the feature vectors $\mathbf{x}_u^{(i)}$ of each node have one component $\mathbf{x}_u^{(i)}(\varphi') \in \{0,1\}$ for each subformula φ' of φ
  - $\mathbf{x}_u^{(i+1)}(\varphi_1 \wedge \varphi_2) = f(\mathbf{x}_u^{(i)}(\varphi_1) + \mathbf{x}_u^{(i)}(\varphi_2) - 1)$

Positive result: building simple GNNs

- We say that a GNN is simple if we update according to

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

  where f is the truncated ReLU (zero if ≤ 0, one if ≥ 1, identity in between)
- **Idea**: the feature vectors $\boldsymbol{x}_u^{(i)}$ of each node have one component $\boldsymbol{x}_u^{(i)}(\varphi') \in \{0, 1\}$ for each subformula φ' of φ
  - $\boldsymbol{x}_u^{(i+1)}(\varphi_1 \wedge \varphi_2) = f(\boldsymbol{x}_u^{(i)}(\varphi_1) + \boldsymbol{x}_u^{(i)}(\varphi_2) - 1)$
  - $\boldsymbol{x}_u^{(i+1)}(\neg\varphi') = f(-\boldsymbol{x}_u^{(i)}(\varphi') + 1)$

Positive result: building simple GNNs

- We say that a GNN is simple if we update according to

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

  where f is the truncated ReLU (zero if ≤ 0, one if ≥ 1, identity in between)
- **Idea**: the feature vectors $\boldsymbol{x}_u^{(i)}$ of each node have one component $\boldsymbol{x}_u^{(i)}(\varphi') \in \{0, 1\}$ for each subformula φ' of φ
  - $\boldsymbol{x}_u^{(i+1)}(\varphi_1 \wedge \varphi_2) = f(\boldsymbol{x}_u^{(i)}(\varphi_1) + \boldsymbol{x}_u^{(i)}(\varphi_2) - 1)$
  - $\boldsymbol{x}_u^{(i+1)}(\neg\varphi') = f(-\boldsymbol{x}_u^{(i)}(\varphi') + 1)$
  - $\boldsymbol{x}_u^{(i+1)}(\exists^{\geq N} y\, E(x, y) \wedge \varphi') = f(\sum_{v \in \mathcal{N}_G(u)} \boldsymbol{x}_v^{(i)}(\varphi') - (N - 1))$

Positive result: building simple GNNs

- We say that a GNN is simple if we update according to

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

  where f is the truncated ReLU (zero if ≤ 0, one if ≥ 1, identity in between)

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

$\rightarrow$ After L layers, we will have $\boldsymbol{x}_u^{(L)}(\varphi) = 1$ iff $u \models \varphi(x)$

Negative result: Van Benthem/Rosen characterization of GML

- We use the following [Otto, 2019]: let φ be an FOC unary formula that is not equivalent to any GML formula. Then there exist a graph G and two nodes $u, v \in G$ such that $u \models \varphi$ and $v \not\models \varphi$ and such that for all $i \in \mathbb{N}$ we have $\mathsf{WL}_u^{(i)} = \mathsf{WL}_v^{(i)}$

Negative result: Van Benthem/Rosen characterization of GML

- We use the following [Otto, 2019]: let φ be an FOC unary formula that is not equivalent to any GML formula. Then there exist a graph G and two nodes $u, v \in G$ such that $u \models \varphi$ and $v \not\models \varphi$ and such that for all $i \in \mathbb{N}$ we have $\mathsf{WL}_u^{(i)} = \mathsf{WL}_v^{(i)}$

$\rightarrow$ By [Morris et al., 2019, Xu et al., 2019], any AC-GNN must have $\boldsymbol{x}_u^{(i)} = \boldsymbol{x}_v^{(i)}$ for all $i \in \mathbb{N}$, so it cannot capture φ

ACR-GNNs for FOC_2

- Can we extend AC-GNNs so that they are able to capture any FOC_2 unary formula?

→ Yes: add global computations in between every layer.

→ $\boldsymbol{x}_u^{(i+1)} := \mathrm{COMB}^{(i+1)}(\boldsymbol{x}_u^{(i)}, \mathrm{AGG}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in \mathcal{N}_G(u)\}\}), \mathrm{READ}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in G\}\}))$

ACR-GNNs for FOC_2

- Can we extend AC-GNNs so that they are able to capture any FOC_2 unary formula?

$\rightarrow$ Yes: add global computations in between every layer.

$\rightarrow$ $\boldsymbol{x}_u^{(i+1)} := \mathrm{COMB}^{(i+1)}(\boldsymbol{x}_u^{(i)}, \mathrm{AGG}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in \mathcal{N}_G(u)\}\}), \mathrm{READ}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in G\}\}))$

- Call that ACR-GNN, for aggregate-combine-readout GNNs

ACR-GNNs for FOC_2

- Can we extend AC-GNNs so that they are able to capture any FOC_2 unary formula?

$\rightarrow$ Yes: add global computations in between every layer.

$\rightarrow$ $\boldsymbol{x}_u^{(i+1)} := \mathrm{COMB}^{(i+1)}(\boldsymbol{x}_u^{(i)}, \mathrm{AGG}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in \mathcal{N}_G(u)\}\}), \mathrm{READ}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in G\}\}))$

- Call that ACR-GNN, for aggregate-combine-readout GNNs

Theorem

Each FOC_2 unary formula is captured by a simple ACR-GNN

ACR-GNNs for FOC_2

- Can we extend AC-GNNs so that they are able to capture any FOC_2 unary formula?

→ Yes: add global computations in between every layer.

→ $\boldsymbol{x}_u^{(i+1)} := \mathrm{COMB}^{(i+1)}(\boldsymbol{x}_u^{(i)}, \mathrm{AGG}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in \mathcal{N}_G(u)\}\}), \mathrm{READ}^{(i+1)}(\{\{\boldsymbol{x}_v^{(i)} \mid v \in G\}\}))$

- Call that ACR-GNN, for aggregate-combine-readout GNNs

Theorem

Each FOC_2 unary formula is captured by a simple ACR-GNN

→ Having readouts strictly increases the discriminative power of GNNs

Proofsketch

We use the following result of [Lutz et al., 2001]:

- Every FOC_2 formula φ can be rewritten as a FOC_2 formula φ' in which every unary subformula $\varphi''(x)$ starting with a quantifier is of one of the following form:
  - $\exists^{\geq N} y\, x = y \wedge \psi(y)$
  - $\exists^{\geq N} y\, E(x, y) \wedge \psi(y)$
  - $\exists^{\geq N} y\, \neg E(x, y) \wedge \psi(y)$
  - $\exists^{\geq N} y\, \neg E(x, y) \wedge x \neq y \wedge \psi(y)$
  - $\exists^{\geq N} y\, \psi(y)$

Proofsketch

We use the following result of [Lutz et al., 2001]:

- Every FOC_2 formula φ can be rewritten as a FOC_2 formula φ' in which every unary subformula $\varphi''(x)$ starting with a quantifier is of one of the following form:
  - $\exists^{\geq N} y\, x = y \wedge \psi(y)$
  - $\exists^{\geq N} y\, E(x, y) \wedge \psi(y)$
  - $\exists^{\geq N} y\, \neg E(x, y) \wedge \psi(y)$
  - $\exists^{\geq N} y\, \neg E(x, y) \wedge x \neq y \wedge \psi(y)$
  - $\exists^{\geq N} y\, \psi(y)$

We then build a simple ACR-GNN just like for AC-GNNs and GML, but, for instance:

- $\boldsymbol{x}_u^{(i+1)}(\exists^{\geq N} y\, \neg E(x, y) \wedge \psi(y)) = f(\sum_{v \in G} \boldsymbol{x}_v^{(i)}(\psi) - \sum_{v \in \mathcal{N}_G(u)} \boldsymbol{x}_v^{(i)}(\psi) - (N-1))$

Number of readouts

Theorem

Each FOC_2 *unary formula is captured by a simple ACR-GNN*

- How many readouts do we need? A fixed number? The quantifier depth of the formula?

Number of readouts

Theorem

Each FOC_2 unary formula is captured by a simple ACR-GNN

- How many readouts do we need? A fixed number? The quantifier depth of the formula?

→ We show that one final readout is enough (but the ACR-GNN is no longer simple)

Theorem

Each FOC_2 unary formula is captured by an ACR-GNN with one final readout

Conclusion

- We have seen the relationship between GNNs and WL
- We started to study the relationships between GNNs and logic
  - $\rightarrow$ "$\mathrm{GML} = \mathrm{FOC} \cap$ AC-GNNs $\subseteq$ simple AC-GNNs"
  - $\rightarrow$ "$\mathrm{FOC}_2 \subseteq$ simple ACR-GNNs"
  - $\rightarrow$ "$\mathrm{FOC}_2 \subseteq$ ACR-GNNs with only one final readout"

Conclusion

- We have seen the relationship between GNNs and WL
- We started to study the relationships between GNNs and logic
  - $\rightarrow$ "$\mathrm{GML} = \mathrm{FOC} \cap$ AC-GNNs $\subseteq$ simple AC-GNNs"
  - $\rightarrow$ "$\mathrm{FOC}_2 \subseteq$ simple ACR-GNNs"
  - $\rightarrow$ "$\mathrm{FOC}_2 \subseteq$ ACR-GNNs with only one final readout"

- Open: $\mathrm{FOC} \cap$ ACR-GNNs $= \mathrm{FOC}_2$?

Conclusion

- We have seen the relationship between GNNs and WL
- We started to study the relationships between GNNs and logic
  - $\rightarrow$ "$\mathrm{GML} = \mathrm{FOC} \cap$ AC-GNNs $\subseteq$ simple AC-GNNs"
  - $\rightarrow$ "$\mathrm{FOC}_2 \subseteq$ simple ACR-GNNs"
  - $\rightarrow$ "$\mathrm{FOC}_2 \subseteq$ ACR-GNNs with only one final readout"

- Open: $\mathrm{FOC} \cap$ ACR-GNNs $= \mathrm{FOC}_2$?

- Since then, GNNs have been compared to other known frameworks for local computations (message-passing, distributive local algorithms, etc). See, e.g., [Loukas, 2019, Sato et al., 2019]

Thanks for your attention!

Bibliography I

Cai, J.-Y., Fürer, M., and Immerman, N. (1992).
An optimal lower bound on the number of variables for graph identification.
Combinatorica, 12(4):389–410.

de Rijke, M. (2000).
A Note on graded modal logic.
Studia Logica, 64(2):271–283.

Bibliography II

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015).
Convolutional networks on graphs for learning molecular fingerprints.
In *Advances in neural information processing systems*, pages 2224–2232.

Loukas, A. (2019).
What graph neural networks cannot learn: depth vs width.
arXiv preprint arXiv:1907.03199.

Bibliography III

Lutz, C., Sattler, U., and Wolter, F. (2001).
Modal logic and the two-variable fragment.
In *Proceedings of the International Workshop on Computer Science Logic, CSL 2001, Paris, France, September 10–13, 2001*, pages 247–261. Springer.

Merkwirth, C. and Lengauer, T. (2005).
Automatic generation of complementary descriptors with molecular graph networks.
J. of Chemical Information and Modeling, 45(5):1159–1168.

Bibliography IV

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019).
Weisfeiler and Leman go neural: higher-order graph neural networks.
In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 – February 1, 2019*, pages 4602–4609.

Otto, M. (2019).
Graded modal logic and counting bisimulation.
`https://www2.mathematik.tu-darmstadt.de/~otto/papers/cml19.pdf`.

Bibliography V

Sato, R., Yamada, M., and Kashima, H. (2019).
Approximation ratios of graph neural networks for combinatorial problems.
In *Advances in Neural Information Processing Systems*, pages 4083–4092.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009).
The graph neural network model.
IEEE Trans. Neural Networks, 20(1):61–80.

Bibliography VI

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019).
How Powerful are graph neural networks?
In *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019.*