Logical Approach to Graph Neural Networks

Flavio Ferrarotti

Software Competence Center Hagenberg, Asustria

June 8, 2022

In this Talk

Which functions on graphs can be expressed by Graph Neural Networks?

In this Talk

Which functions on graphs can be expressed by Graph Neural Networks?

Plan:

1. Not so short intro to the core model Graph Neural Network computation.

In this Talk

Which functions on graphs can be expressed by Graph Neural Networks?

Plan:

1. Not so short intro to the core model Graph Neural Network computation.
2. Digression 1: Colour Refinement Algorithm.

In this Talk

Which functions on graphs can be expressed by Graph Neural Networks?

Plan:

1. Not so short intro to the core model Graph Neural Network computation.
2. Digression 1: Colour Refinement Algorithm.
3. Digression 2: Finite Variable Counting Logics.

In this Talk

Which functions on graphs can be expressed by Graph Neural Networks?

Plan:

1. Not so short intro to the core model Graph Neural Network computation.
2. Digression 1: Colour Refinement Algorithm.
3. Digression 2: Finite Variable Counting Logics.
4. Expressive power of Graph Neural Networks w.r.t distinguishing graphs or their nodes.

In this Talk

Which functions on graphs can be expressed by Graph Neural Networks?

Plan:

1. Not so short intro to the core model Graph Neural Network computation.
2. Digression 1: Colour Refinement Algorithm.
3. Digression 2: Finite Variable Counting Logics.
4. Expressive power of Graph Neural Networks w.r.t distinguishing graphs or their nodes.
5. Digression 3: Expressiveness, database queries and logics.

In this Talk

Which functions on graphs can be expressed by Graph Neural Networks?

Plan:

1. Not so short intro to the core model Graph Neural Network computation.
2. Digression 1: Colour Refinement Algorithm.
3. Digression 2: Finite Variable Counting Logics.
4. Expressive power of Graph Neural Networks w.r.t distinguishing graphs or their nodes.
5. Digression 3: Expressiveness, database queries and logics.
6. Expressiveness of Graph Neural Networks.

Graph Neural Networks (GNNs)

- Deep learning architectures for graph structured data.

Graph Neural Networks (GNNs)

- Deep learning architectures for graph structured data.
- Wide range of applications, e.g., computer vision, natural language processing, social network analysis, knowledge graphs, chemistry, and recommendation systems.

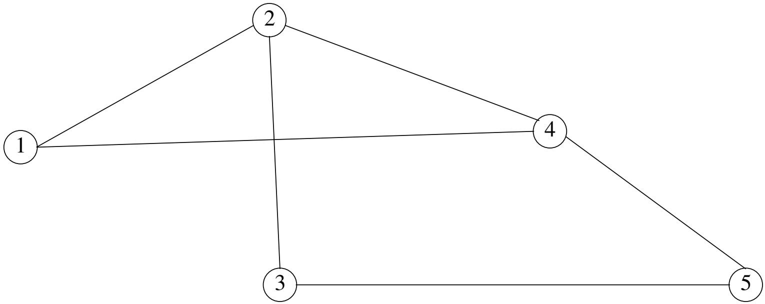
Graph Neural Networks (GNNs)

- Deep learning architectures for graph structured data.
- Wide range of applications, e.g., computer vision, natural language processing, social network analysis, knowledge graphs, chemistry, and recommendation systems.
- We consider the core GNN architecture, known as Message Passing GNN or aggregate-combine GNN.

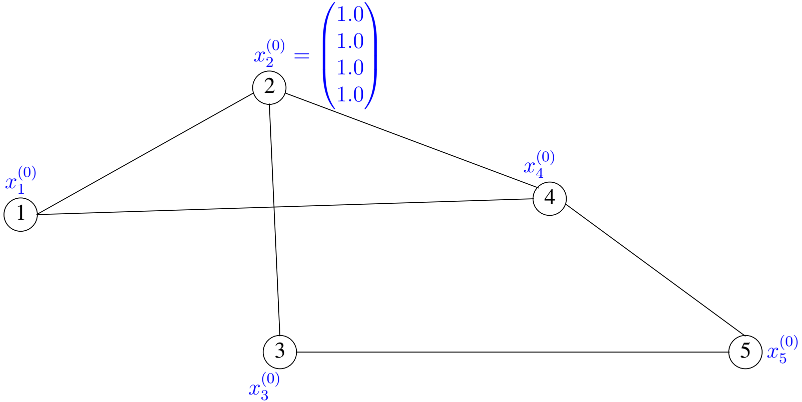
Graph Neural Networks (GNNs)

- Deep learning architectures for graph structured data.
- Wide range of applications, e.g., computer vision, natural language processing, social network analysis, knowledge graphs, chemistry, and recommendation systems.
- We consider the core GNN architecture, known as Message Passing GNN or aggregate-combine GNN.
- There are many other variants, but they are mostly based in this core architecture.

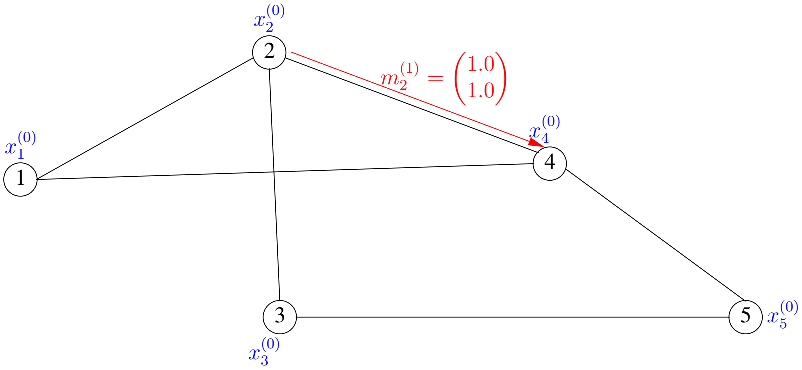
GNN



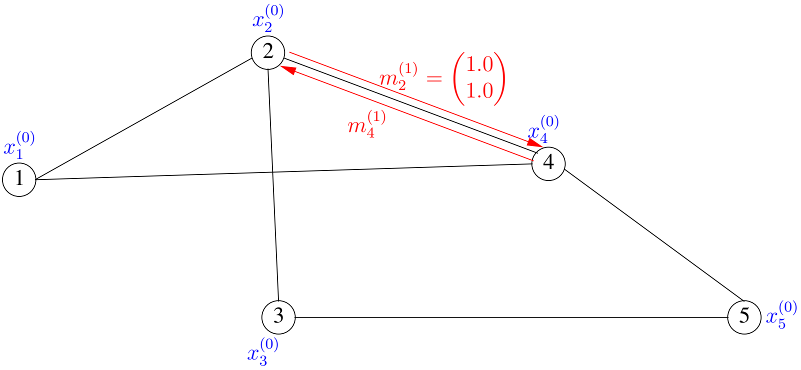
GNN


$x_2^{(0)} = \begin{pmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{pmatrix}$
$x_1^{(0)}$
$x_3^{(0)}$
$x_4^{(0)}$
$x_5^{(0)}$
1
2
3
4
5

GNN

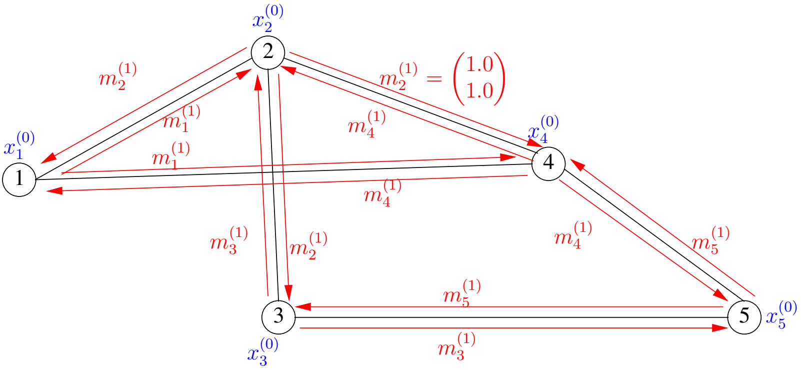


GNN



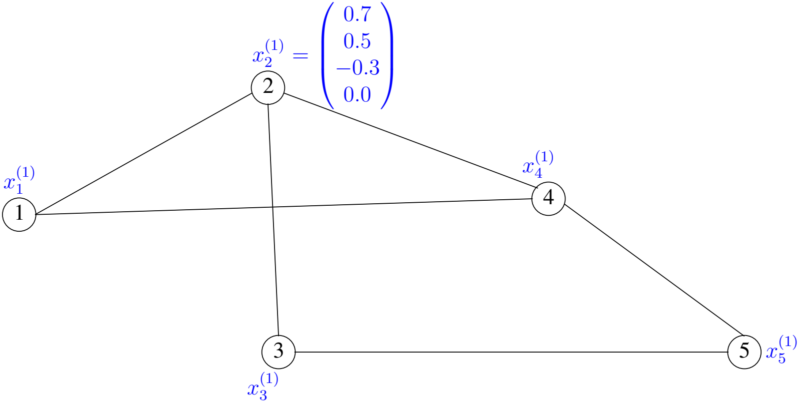
$x_2^{(0)}$
2
$m_2^{(1)} = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}$
$m_4^{(1)}$
$x_4^{(0)}$
4
$x_1^{(0)}$
1
3
$x_3^{(0)}$
5
$x_5^{(0)}$

GNN

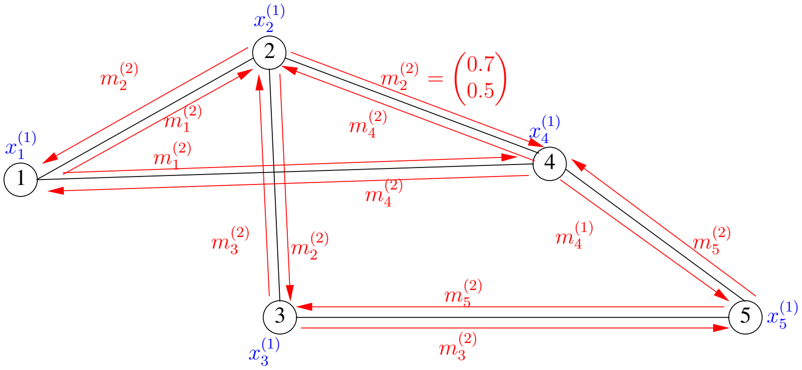


$x_2^{(0)}$
2
$m_2^{(1)}$
$m_2^{(1)} = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}$
$m_1^{(1)}$
$m_4^{(1)}$
$x_1^{(0)}$
$x_4^{(0)}$
1
$m_1^{(1)}$
4
$m_4^{(1)}$
$m_3^{(1)}$
$m_2^{(1)}$
$m_4^{(1)}$
$m_5^{(1)}$
$m_5^{(1)}$
3
5
$x_5^{(0)}$
$x_3^{(0)}$
$m_3^{(1)}$

GNN

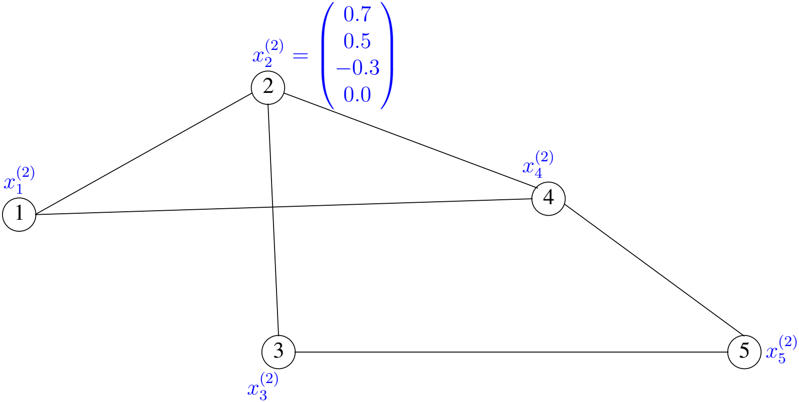

$x_2^{(1)} = \begin{pmatrix} 0.7 \\ 0.5 \\ -0.3 \\ 0.0 \end{pmatrix}$
$x_1^{(1)}$
$x_3^{(1)}$
$x_4^{(1)}$
$x_5^{(1)}$
1
2
3
4
5

GNN



$x_2^{(1)}$
2
$m_2^{(2)}$
$m_2^{(2)} = \begin{pmatrix} 0.7 \\ 0.5 \end{pmatrix}$
$m_1^{(2)}$
$m_4^{(2)}$
$x_4^{(1)}$
$x_1^{(1)}$
$m_1^{(2)}$
1
4
$m_4^{(2)}$
$m_4^{(1)}$
$m_3^{(2)}$
$m_2^{(2)}$
$m_5^{(2)}$
$m_5^{(2)}$
3
5
$x_5^{(1)}$
$x_3^{(1)}$
$m_3^{(2)}$

GNN



$x_2^{(2)} = \begin{pmatrix} 0.7 \\ 0.5 \\ -0.3 \\ 0.0 \end{pmatrix}$
2
$x_1^{(2)}$
1
$x_4^{(2)}$
4
3
$x_3^{(2)}$
5
$x_5^{(2)}$

Computation of GNN

A GNN $\mathcal{N}$ with d layers maps a vertex-labelled graph G with finite node set V to a sequence of functions

$$\zeta^{(t)} : V \to \mathbb{R}^p \quad \text{for } t = 0, \ldots, d$$

Computation of GNN

A GNN $\mathcal{N}$ with d layers maps a vertex-labelled graph G with finite node set V to a sequence of functions

$$\zeta^{(t)} : V \to \mathbb{R}^p \quad \text{for } t = 0, \dots, d$$

- Initialize: $\zeta^{(0)}(v)$ encodes node label of v.

Computation of GNN

A GNN $\mathcal{N}$ with d layers maps a vertex-labelled graph G with finite node set V to a sequence of functions

$$\zeta^{(t)} : V \to \mathbb{R}^p \quad \text{for } t = 0, \ldots, d$$

- Initialize: $\zeta^{(0)}(v)$ encodes node label of v.
- Aggregate: $\alpha^{(t)}(v) := \mathrm{agg}_t(\{\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\})$
  - Symmetric function, e.g., sum, mean, max.
  - Either applied directly to "states" $\zeta^{(t-1)}(w)$ or to some (learned) linear function of these states.

Computation of GNN

A GNN $\mathcal{N}$ with d layers maps a vertex-labelled graph G with finite node set V to a sequence of functions

$$\zeta^{(t)} : V \to \mathbb{R}^p \quad \text{for } t = 0, \dots, d$$

- Initialize: $\zeta^{(0)}(v)$ encodes node label of v.
- Aggregate: $\alpha^{(t)}(v) := \mathrm{agg}_t(\{\!\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\!\})$
  - Symmetric function, e.g., sum, mean, max.
  - Either applied directly to "states" $\zeta^{(t-1)}(w)$ or to some (learned) linear function of these states.
- Combine: $\zeta^{(t)}(v) := \mathrm{comb}_t(\zeta^{(t-1)}(v), \alpha^{(t)}(v))$
  - Computed by a FNN or by some more sophisticated neural network architecture.

Computation of GNN

A GNN $\mathcal{N}$ with d layers maps a vertex-labelled graph G with finite node set V to a sequence of functions

$$\zeta^{(t)} : V \to \mathbb{R}^p \quad \text{for } t = 0, \dots, d$$

- Initialize: $\zeta^{(0)}(v)$ encodes node label of v.
- Aggregate: $\alpha^{(t)}(v) := \mathrm{agg}_t(\{\!\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\!\})$
  - Symmetric function, e.g., sum, mean, max.
  - Either applied directly to "states" $\zeta^{(t-1)}(w)$ or to some (learned) linear function of these states.
- Combine: $\zeta^{(t)}(v) := \mathrm{comb}_t(\zeta^{(t-1)}(v), \alpha^{(t)}(v))$
  - Computed by a FNN or by some more sophisticated neural network architecture.

Functions Computed by GNNs

$$\zeta^{(t)}(v) := \mathrm{comb}_t(\zeta^{(t-1)}(v), \mathrm{agg}_t(\{\!\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\!\}))$$

Functions Computed by GNNs

$$\zeta^{(t)}(v) := \mathrm{comb}_t(\zeta^{(t-1)}(v), \mathrm{agg}_t(\{\!\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\!\}))$$

To compute a node level function F that maps each graph G to a mapping $F(G) : V \to \mathbb{R}^q$, we apply a readout function ro s.t.

$$F(G)(v) = \mathrm{ro}\big(\zeta^{(d)}(v)\big) \qquad (\mathrm{ro} \text{ is normally computed by a FNN})$$

Functions Computed by GNNs

$$\zeta^{(t)}(v) := \mathrm{comb}_t(\zeta^{(t-1)}(v), \mathrm{agg}_t(\{\!\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\!\}))$$

To compute a node level function F that maps each graph G to a mapping $F(G) : V \to \mathbb{R}^q$, we apply a readout function ro s.t.

$$F(G)(v) = \mathrm{ro}\big(\zeta^{(d)}(v)\big) \qquad (\mathrm{ro} \text{ is normally computed by a FNN})$$

To compute a graph level function f that maps graphs to $\mathbb{R}^q$, we apply an aggregate readout function aggro s.t.

$$f(G) = \mathrm{aggro}\big(\{\!\{\zeta^{(d)}(v) \mid v \in V\}\!\}\big)$$

(aggro is normally summation followed a FNN)

Properties of Functions Computed by GNNs

Invariance of Graph Level Functions If f is a graph level function computed by a GNN and graphs G and H are isomorphic, then

$$f(G) = f(H)$$

Properties of Functions Computed by GNNs

Invariance of Graph Level Functions If f is a graph level function computed by a GNN and graphs G and H are isomorphic, then

$$f(G) = f(H)$$

Equivariance of Node Level Functions If f is a node level function computed by a GNN and h is an isomorphism from a graph G to a graph H, then for all node v of G we get

$$F(G)(v) = F(H)(h(v))$$

Recurrent GNNs

So far... we defined GNNs with fixed number d of layers, each layer t with its own functions agg_t and comb_t.

Recurrent GNNs

So far... we defined GNNs with fixed number d of layers, each layer t with its own functions agg_t and comb_t.

Recurrent GNN determine the number d of iterations at runtime and use instead single agg and comb functions.

Recurrent GNNs

So far... we defined GNNs with fixed number d of layers, each layer t with its own functions agg_t and comb_t.

Recurrent GNN determine the number d of iterations at runtime and use instead single agg and comb functions.

The number of iterations might depend for instance on the size of the input graph or the evolution of the sequence $(\zeta^{(t)})_{t\geq 0}$.

Recurrent GNNs

So far... we defined GNNs with fixed number d of layers, each layer t with its own functions agg_t and comb_t.

Recurrent GNN determine the number d of iterations at runtime and use instead single agg and comb functions.

The number of iterations might depend for instance on the size of the input graph or the evolution of the sequence $(\zeta^{(t)})_{t\geq 0}$.

No convergence is required, we can arbitrary decide when to stop.

Colour Refinement ($\sim$ 1-dim. Weisfeiler-Leman) Alg.

1. Initialisation: All nodes get the same colour.

Colour Refinement ($\sim$ 1-dim. Weisfeiler-Leman) Alg.

1. Initialisation: All nodes get the same colour.
2. Refinement: Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.

Colour Refinement ($\sim$ 1-dim. Weisfeiler-Leman) Alg.

1. Initialisation: All nodes get the same colour.
2. Refinement: Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.
3. Termination: Step 2 (Refinement) is repeated until colouring stays stable.

Colour Refinement ($\sim$ 1-dim. Weisfeiler-Leman) Alg.

1. Initialisation: All nodes get the same colour.
2. Refinement: Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.
3. Termination: Step 2 (Refinement) is repeated until colouring stays stable.

- See illustration of colour refinement algorithm by Holger Dell (2020): `https://holgerdell.github.io/color-refinement/#seed=g61hy`

Colour Refinement ($\sim$ 1-dim. Weisfeiler-Leman) Alg.

1. Initialisation: All nodes get the same colour.
2. Refinement: Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.
3. Termination: Step 2 (Refinement) is repeated until colouring stays stable.

- See illustration of colour refinement algorithm by Holger Dell (2020): `https://holgerdell.github.io/color-refinement/#seed=g61hy`
- If some colour appears different number of times in the histograms of two graphs G and H, then the graphs are non-isomorphic.

Colour Refinement ($\sim$ 1-dim. Weisfeiler-Leman) Alg.

1. Initialisation: All nodes get the same colour.
2. Refinement: Two nodes v, w get different colours if there is some colour c such that v and w have different numbers of neighbours of colour c.
3. Termination: Step 2 (Refinement) is repeated until colouring stays stable.

- See illustration of colour refinement algorithm by Holger Dell (2020): `https://holgerdell.github.io/color-refinement/#seed=g61hy`
- If some colour appears different number of times in the histograms of two graphs G and H, then the graphs are non-isomorphic.
- Provides an efficient but incomplete isomorphism check.
  - $O((n+m)\log n)$ for n nodes and m edges.

Finite Variable Counting Logic

Syntactic extension C of first-order logic FO by counting quantifiers $\exists^{\geq p}$.

Finite Variable Counting Logic

Syntactic extension C of first-order logic FO by counting quantifiers $\exists^{\geq p}$.

Same expressiveness than FO,

$$\exists^{\geq p} x \varphi(x) \equiv \exists x_1 \ldots \exists x_p \Big(\bigwedge_{1 \leq i \leq j \leq p} x_i \neq x_j \wedge \bigwedge_{i=1,\ldots,p} \varphi(x_i) \Big)$$

Finite Variable Counting Logic

Syntactic extension C of first-order logic FO by counting quantifiers $\exists^{\geq p}$.

Same expressiveness than FO,

$$\exists^{\geq p} x \varphi(x) \equiv \exists x_1 \ldots \exists x_p \Big(\bigwedge_{1 \leq i \leq j \leq p} x_i \neq x_j \wedge \bigwedge_{i=1,\ldots,p} \varphi(x_i) \Big)$$

Translation from C to FO incurs an increase in the number of variables as well as the quantifier rank, e.g.,

$$\forall x \exists^{\geq d} y E(x,y) \quad \text{(uses 2 variables and quantifier rank 2)}$$

An equivalent FO formula needs at least $d+1$ variables and at least quantifier rank $d+1$.

Some Fragments Finite Variable Counting Logic

By C^k we denote the fragment of C consisting of all formulas with at most k variables.

Some Fragments Finite Variable Counting Logic

By C^k we denote the fragment of C consisting of all formulas with at most k variables.

The guarded fragment GC.
Quantifiers are restricted to range over the neighbours of the current nodes, i.e., to formulae of the form:

$$\exists^{\geq p} y(E(x,y) \wedge \psi)$$

where x and y are distinct variables and y appears as a free variable in ψ.

Some Fragments Finite Variable Counting Logic

By C^k we denote the fragment of C consisting of all formulas with at most k variables.

The guarded fragment GC.
Quantifiers are restricted to range over the neighbours of the current nodes, i.e., to formulae of the form:

$$\exists^{\geq p} y(E(x,y) \wedge \psi)$$

where x and y are distinct variables and y appears as a free variable in ψ.

We are mainly interested in the 2-variable fragment GC^2, also known as graded modal logic.

Distinguishing Power of GNN

Theorem ([Immerman and Lander, 1990])

For all graph G, H, the following are equivalent:

- *Colour refinement does not distinguish G and H.*
- *G and H satisfy the same sentences of the logic C^2.*

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

For all graph G, H, the following are equivalent:

- *Colour refinement distinguishes G and H.*
- *G and H are distinguishable by a GNN, i.e., there is a graph level function f computed by a GNN N such that $f(G) \neq f(H)$.*

Expressiveness of a Logic

Which queries in a class of structures $\mathcal{C}$ are expressible in a logic $\mathcal{L}$?

Expressiveness of a Logic

Which queries in a class of structures $\mathcal{C}$ are expressible in a logic $\mathcal{L}$?

A *k-ary query Q on $\mathcal{C}$* is a function that maps each $\mathrm{A} \in \mathcal{C}$ to a relation $Q(\mathrm{A}) \subseteq A^k$ satisfying:

- $(a_1, \ldots, a_k) \in Q(\mathrm{A})$ iff $(f(a_1), \ldots, f(a_k)) \in Q(\mathrm{B})$ for every isomorphism f from A to B and every $(a_1, \ldots, a_k) \in A^k$.

Expressiveness of a Logic

Which queries in a class of structures $\mathcal{C}$ are expressible in a logic $\mathcal{L}$?

A *k-ary query Q on $\mathcal{C}$* is a function that maps each $\mathrm{A} \in \mathcal{C}$ to a relation $Q(\mathrm{A}) \subseteq A^k$ satisfying:

- $(a_1, \ldots, a_k) \in Q(\mathrm{A})$ iff $(f(a_1), \ldots, f(a_k)) \in Q(\mathrm{B})$ for every isomorphism f from A to B and every $(a_1, \ldots, a_k) \in A^k$.

Queries of arity $k = 0$ are known as Boolean queries.

Expressiveness of a Logic

Which queries in a class of structures $\mathcal{C}$ are expressible in a logic $\mathcal{L}$?

A *k-ary query Q on $\mathcal{C}$* is a function that maps each $\mathrm{A} \in \mathcal{C}$ to a relation $Q(\mathrm{A}) \subseteq A^k$ satisfying:

- $(a_1, \ldots, a_k) \in Q(\mathrm{A})$ iff $(f(a_1), \ldots, f(a_k)) \in Q(\mathrm{B})$ for every isomorphism f from A to B and every $(a_1, \ldots, a_k) \in A^k$.

Queries of arity $k = 0$ are known as Boolean queries.

Since there are only two 0-ary relations on a structure (true and false), a Boolean query Q is often identified with the class $\{\mathrm{A} \in \mathcal{C} \mid Q(\mathrm{A}) = \text{true}\}$.

Expressing Queries on Graphs

We say that a formula φ of a logic $\mathcal{L}$ *expresses* a k-ary query Q if for all graph G (in the considered class) and k-tuples $\bar{a}$ of nodes,

$$G \models \varphi(\bar{a}) \text{ iff } \bar{a} \in Q(G)$$

Expressing Queries on Graphs

We say that a formula φ of a logic $\mathcal{L}$ *expresses* a k-ary query Q if for all graph G (in the considered class) and k-tuples $\bar{a}$ of nodes,

$$G \models \varphi(\bar{a}) \text{ iff } \bar{a} \in Q(G)$$

Analogously, if a GNN $\mathcal{N}$ computes a node level function F, then it expresses a unary query.

Expressing Queries on Graphs

We say that a formula φ of a logic $\mathcal{L}$ *expresses* a k-ary query Q if for all graph G (in the considered class) and k-tuples $\bar{a}$ of nodes,

$$G \models \varphi(\bar{a}) \text{ iff } \bar{a} \in Q(G)$$

Analogously, if a GNN $\mathcal{N}$ computes a node level function F, then it expresses a unary query.

More precisely, *$\mathcal{N}$ expresses the unary query Q* if there is an $\epsilon < 1/2$ such that for all graphs G and all nodes a,

$$F(G)(v) \geq 1 - \epsilon \text{ if } a \in Q(G) \quad \text{and}$$

$$F(G)(v) \leq \epsilon \text{ if } a \notin Q(G).$$

Expressing Queries on Graphs

We say that a formula φ of a logic $\mathcal{L}$ *expresses* a k-ary query Q if for all graph G (in the considered class) and k-tuples $\bar{a}$ of nodes,

$$G \models \varphi(\bar{a}) \text{ iff } \bar{a} \in Q(G)$$

Analogously, if a GNN $\mathcal{N}$ computes a node level function F, then it expresses a unary query.

More precisely, $\mathcal{N}$ *expresses the unary query* Q if there is an $\epsilon < 1/2$ such that for all graphs G and all nodes a,

$$F(G)(v) \geq 1 - \epsilon \text{ if } a \in Q(G) \quad \text{and}$$

$$F(G)(v) \leq \epsilon \text{ if } a \notin Q(G).$$

Similarly, if a GNN computes a graph level function, then it *expresses a Boolean query*

Expressiveness of GNNs

Theorem ([Barceló et al., 2020])

If Q is a unary query expressible in GC^2, then there is a GNN that expresses Q using a linearised sigmoid lsig as activation function.

Expressiveness of GNNs

Theorem ([Barceló et al., 2020])

If Q is a unary query expressible in GC^2, then there is a GNN that expresses Q using a linearised sigmoid lsig as activation function.

- It is not obvious how to adapt the result to other activation functions (beyond ReLU).

Expressiveness of GNNs

Theorem ([Barceló et al., 2020])

If Q is a unary query expressible in GC^2, then there is a GNN that expresses Q using a linearised sigmoid lsig as activation function.

- It is not obvious how to adapt the result to other activation functions (beyond ReLU).
- The converse inclusion does not (fully) hold.
  - E.g. a GNN can decide decide whether a node a has twice as many neighbours with a given label L_1 as it has with label L_2.

Expressiveness of GNNs

Theorem ([Barceló et al., 2020])

If Q is a unary query expressible in GC^2, then there is a GNN that expresses Q using a linearised sigmoid lsig as activation function.

- It is not obvious how to adapt the result to other activation functions (beyond ReLU).
- The converse inclusion does not (fully) hold.
  - E.g. a GNN can decide decide whether a node *a* has twice as many neighbours with a given label L_1 as it has with label L_2.

But there is a partial converse...

Theorem ([Barceló et al., 2020])

If Q is a unary query expressible by a GNN and also expressible in first-order logic, then Q is expressible in GC^2.

Final Consideration

- The described results provide clear evidence in favour of:
  - Pursuing a precise understanding of the expressive power of NNs through well established methods from logic.

Final Consideration

- The described results provide clear evidence in favour of:
  - Pursuing a precise understanding of the expressive power of NNs through well established methods from logic.
- Possible line of work (among others):
  - To explore the design space of GNNs by considering the impact in their expressive power of different features such as recurrence, activation functions (beyond ReLu) and alternative ways of assuring isomorphism invariance.

Thank you!!!

References I

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J. L., and Silva, J. P. (2020).
The logical expressiveness of graph neural networks.
In *ICLR*. OpenReview.net.

Immerman, N. and Lander, E. (1990).
Describing graphs: A first-order approach to graph canonization.
In Selman, A., editor, *Complexity Theory Retrospective*, pages 59–81. Springer-Verlag.

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 *AAAI*, pages 4602–4609. AAAI Press.

References II

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019).
How powerful are graph neural networks?
In *ICLR*. OpenReview.net.