



RWTHAACHEN
UNIVERSITY

The Descriptive Complexity of Graph Neural Networks

Martin Grohe

Theorem

For every unary query on graphs, the following are equivalent.

- (1) Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.*
- (2) Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.*
- (3) Q is in TC^0 .*

equivariant Boolean function
on the vertices of graphs

Theorem

For every unary query on graphs, the following are equivalent.

- (1) Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.
- (2) Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.
- (3) Q is in TC^0 .

deep learning architecture for graphs:
learned distributed algorithm

Main Theorem

equivariant Boolean function
on the vertices of graphs

Theorem

For every unary query on graphs, the following are equivalent.

- (1) Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.
- (2) Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.
- (3) Q is in TC^0 .

deep learning architecture for graphs:
learned distributed algorithm

Main Theorem

equivariant Boolean function
on the vertices of graphs

Theorem

For every unary query on graphs, the following are equivalent.

- (1) Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.
- (2) Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.
- (3) Q is in TC^0 .

2-variable first-order logic
with counting

deep learning architecture for graphs:
learned distributed algorithm

Main Theorem

equivariant Boolean function
on the vertices of graphs

Theorem

For every unary query on graphs, the following are equivalent.

- (1) Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.
- (2) Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.
- (3) Q is in TC^0 .

2-variable first-order logic
with counting

add non-uniformity
to logic

deep learning architecture for graphs:
learned distributed algorithm

Main Theorem

equivariant Boolean function
on the vertices of graphs

Theorem

For every unary query on graphs, the following are equivalent.

- (1) Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.
- (2) Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.
- (3) Q is in TC^0 .

2-variable first-order logic
with counting

Boolean functions computable by
non-uniform polynomial-size bounded-depth
family of circuits with threshold gates

add non-uniformity
to logic

Theorem

For every unary query on graphs, the following are equivalent.

- (1) Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.*
- (2) Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.*
- (3) Q is in TC^0 .*

Theorem

For every unary query on graphs, the following are equivalent.

- (1) *Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.*
- (2) *Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.*
- (3) *Q is in TC^0 .*

Remarks

- ▶ Equivalence between (2) and (3) is known (Barrington, Immerman, Straubing 1997)

Theorem

For every unary query on graphs, the following are equivalent.

- (1) *Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation.*
- (2) *Q is expressible in order-invariant $\text{FO}^2 + \text{C}$ with built-in relations.*
- (3) *Q is in TC^0 .*

Remarks

- ▶ Equivalence between (2) and (3) is known (Barrington, Immerman, Straubing 1997)
- ▶ Computational model in (1) involves arbitrary real numbers, transcendental functions such as logistic function (“sigmoid”), and randomisation.

Graph Neural Networks

- ▶ Graph neural networks (GNNs) are deep learning architectures for machine learning problems on graphs.

Graph Neural Networks

- ▶ **Graph neural networks (GNNs)** are deep learning architectures for machine learning problems on graphs.
- ▶ GNNs have a wide range of applications, for example, in computational biology, chemical engineering, physics, etc.

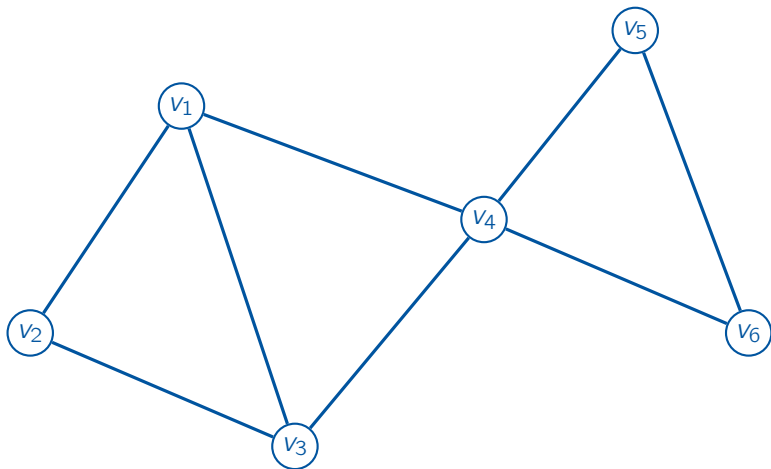
Graph Neural Networks

- ▶ **Graph neural networks (GNNs)** are deep learning architectures for machine learning problems on graphs.
- ▶ GNNs have a wide range of applications, for example, in computational biology, chemical engineering, physics, etc.
- ▶ By now, there is a large variety of GNN architectures. We consider the core GNN architecture known as **message passing graph neural network** or **aggregate-combine graph neural network**.

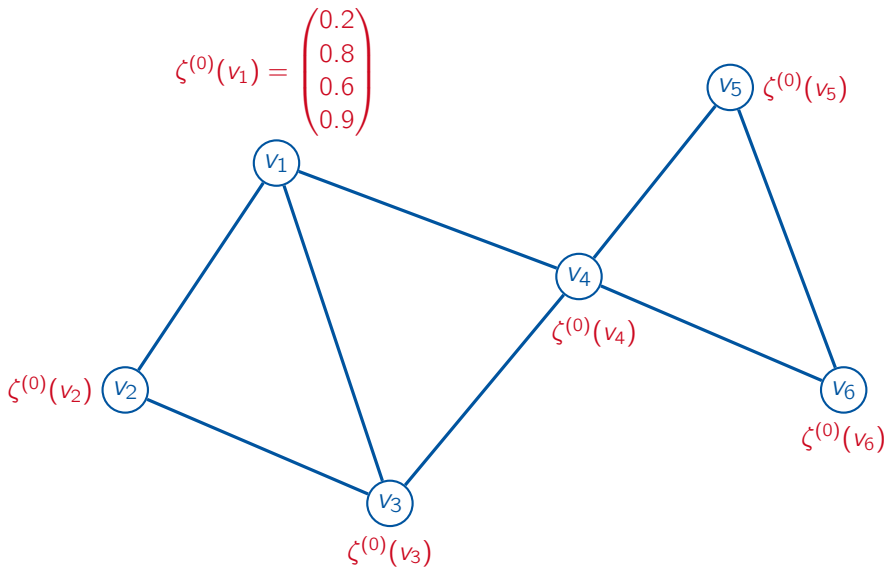
Graph Neural Networks

- ▶ **Graph neural networks (GNNs)** are deep learning architectures for machine learning problems on graphs.
- ▶ GNNs have a wide range of applications, for example, in computational biology, chemical engineering, physics, etc.
- ▶ By now, there is a large variety of GNN architectures. We consider the core GNN architecture known as **message passing graph neural network** or **aggregate-combine graph neural network**.
- ▶ This talk is about **expressivity**: which functions are computable by a GNN?

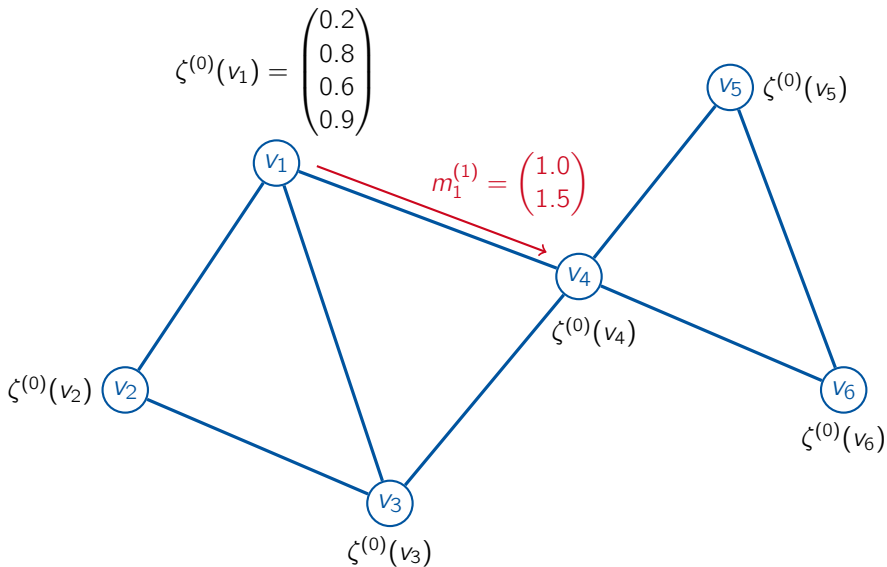
Graph Neural Networks



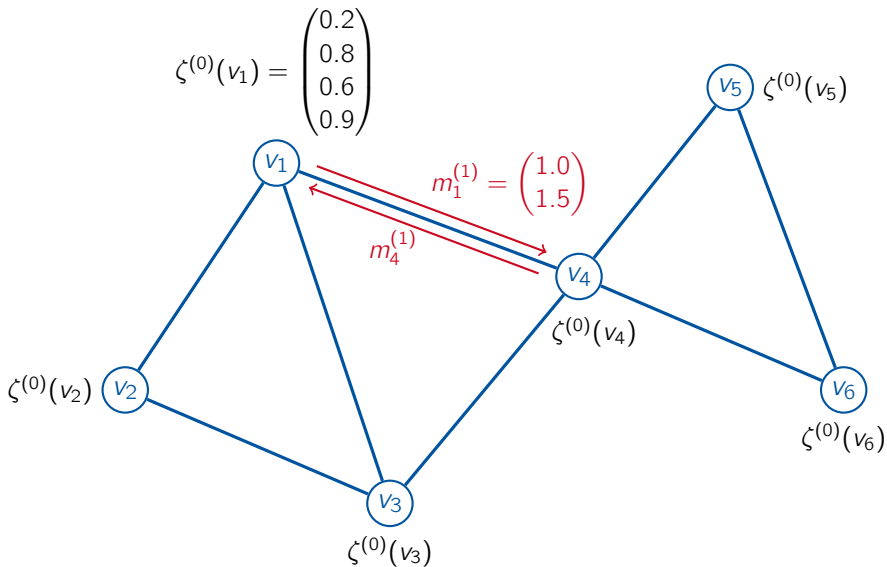
Graph Neural Networks



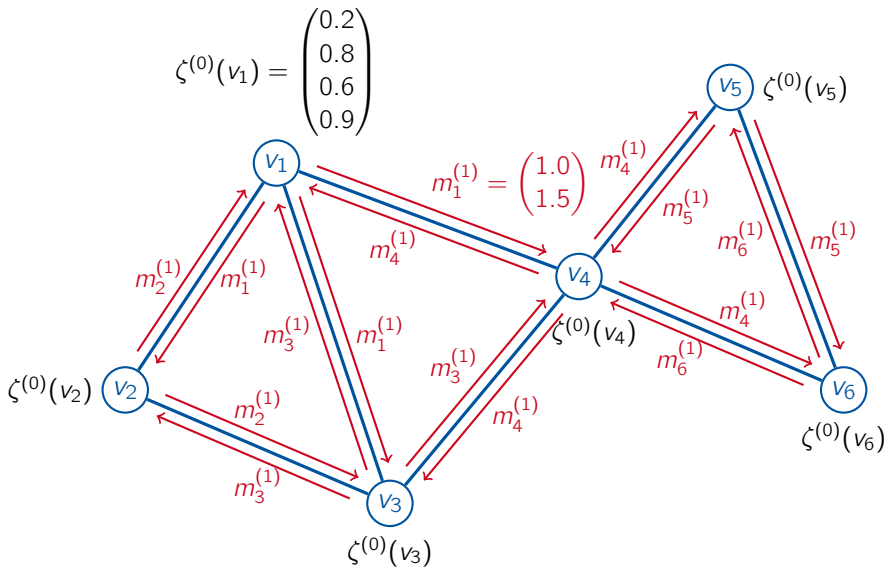
Graph Neural Networks



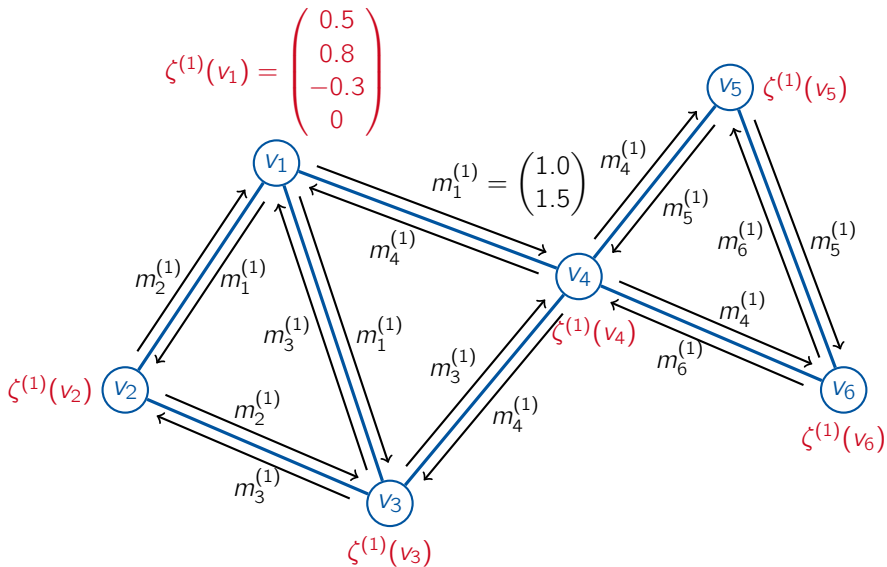
Graph Neural Networks



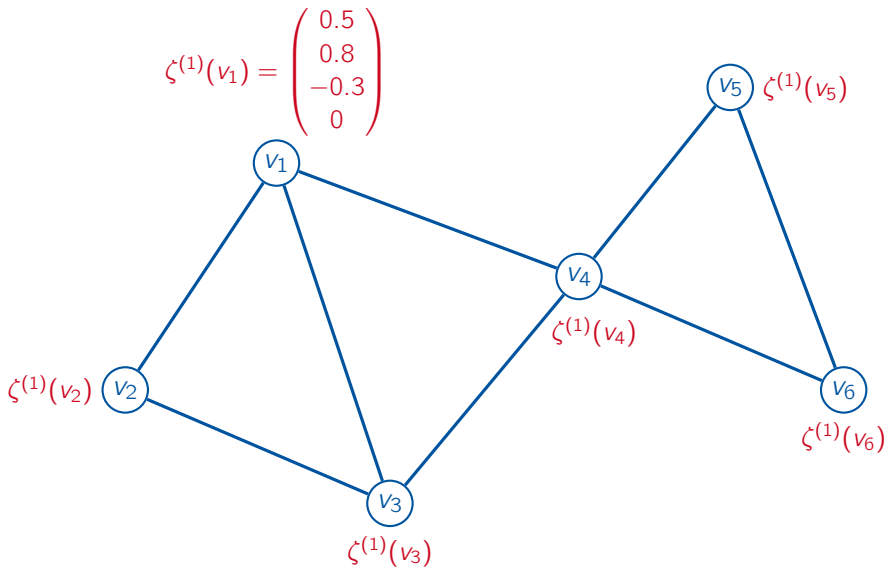
Graph Neural Networks



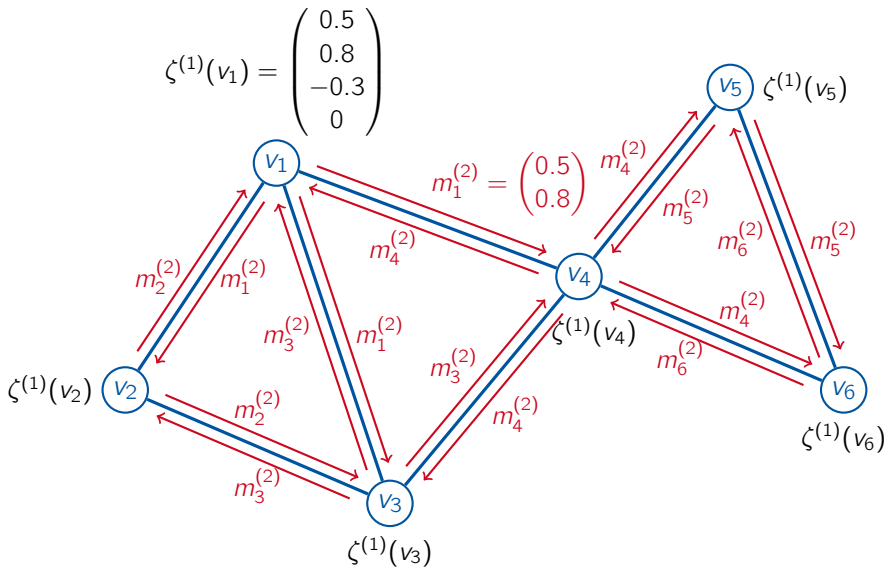
Graph Neural Networks



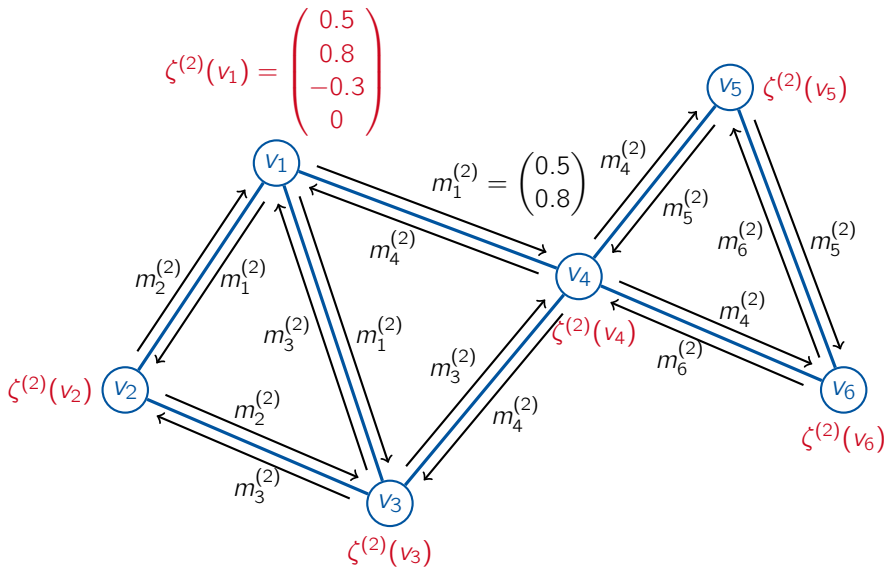
Graph Neural Networks



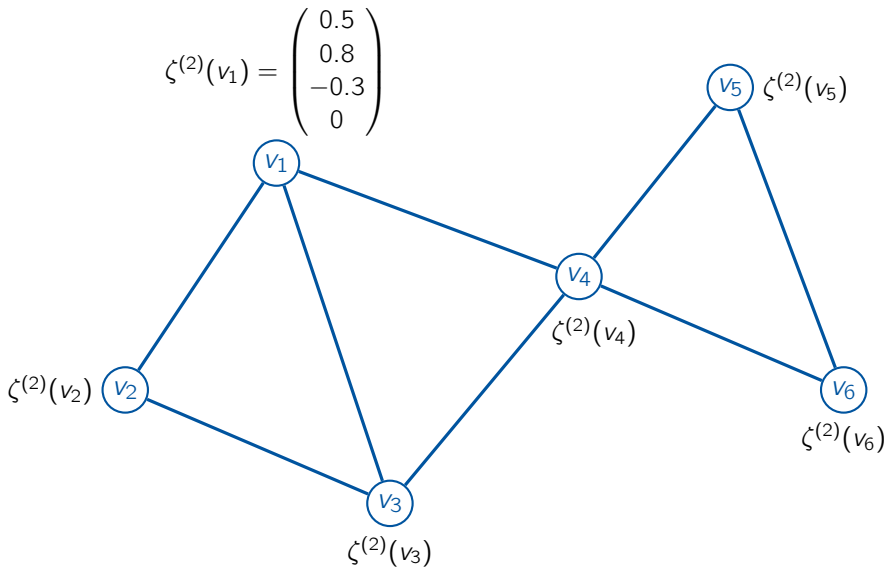
Graph Neural Networks



Graph Neural Networks



Graph Neural Networks



Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^k$ randomly initialised (say, with uniform distribution on $[0, 1]$).

Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^k$ randomly initialised (say, with uniform distribution on $[0, 1]$).

Aggregation: $\alpha^{(t)}(v) := \text{agg}_t(\{\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\})$.

Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^k$ randomly initialised (say, with uniform distribution on $[0, 1]$).

Aggregation: $\alpha^{(t)}(v) := \text{agg}_t(\{\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\})$.

- ▶ symmetric function like **sum**, **mean**, **max** applied either directly to “states” $\zeta^{(t-1)}(w)$ of neighbours w of v or to some (learned) function of these states

Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^k$ randomly initialised (say, with uniform distribution on $[0, 1]$).

Aggregation: $\alpha^{(t)}(v) := \text{agg}_t(\{\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\})$.

- ▶ symmetric function like **sum**, **mean**, **max** applied either directly to “states” $\zeta^{(t-1)}(w)$ of neighbours w of v or to some (learned) function of these states

Global readout: $\gamma^{(t)} := \text{agg}'_t(\{\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\})$.

Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^k$ randomly initialised (say, with uniform distribution on $[0, 1]$).

Aggregation: $\alpha^{(t)}(v) := \text{agg}_t(\{\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\})$.

- ▶ symmetric function like **sum**, **mean**, **max** applied either directly to “states” $\zeta^{(t-1)}(w)$ of neighbours w of v or to some (learned) function of these states

Global readout: $\gamma^{(t)} := \text{agg}'_t(\{\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\})$.

Combination: $\zeta^{(t)}(v) := \text{comb}_t(\zeta^{(t-1)}(v), \alpha^{(t)}(v), \gamma^{(t)})$

Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^k$ randomly initialised (say, with uniform distribution on $[0, 1]$).

Aggregation: $\alpha^{(t)}(v) := \text{agg}_t(\{\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\})$.

- ▶ symmetric function like **sum**, **mean**, **max** applied either directly to “states” $\zeta^{(t-1)}(w)$ of neighbours w of v or to some (learned) function of these states

Global readout: $\gamma^{(t)} := \text{agg}'_t(\{\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\})$.

Combination: $\zeta^{(t)}(v) := \text{comb}_t(\zeta^{(t-1)}(v), \alpha^{(t)}(v), \gamma^{(t)})$

- ▶ function computed by a feedforward neural network

Computation of GNNs

GNN N with d layers maps graph G to sequence of functions

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^k$ randomly initialised (say, with uniform distribution on $[0, 1]$).

Aggregation: $\alpha^{(t)}(v) := \text{agg}_t(\{\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\})$.

- ▶ symmetric function like **sum**, **mean**, **max** applied either directly to “states” $\zeta^{(t-1)}(w)$ of neighbours w of v or to some (learned) function of these states

Global readout: $\gamma^{(t)} := \text{agg}'_t(\{\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\})$.

Combination: $\zeta^{(t)}(v) := \text{comb}_t(\zeta^{(t-1)}(v), \alpha^{(t)}(v), \gamma^{(t)})$

- ▶ function computed by a feedforward neural network
- ▶ weights of neural network are learned from data

Functions Computed by GNNs

To compute a function F that maps each graph G to a mapping $F(G) : V(G) \rightarrow \mathbb{R}^q$, we apply a **readout function** ro such that

$$F(G, v) \approx \text{ro}(\zeta^{(d)}(v)).$$

Functions Computed by GNNs

To compute a function F that maps each graph G to a mapping $F(G) : V(G) \rightarrow \mathbb{R}^q$, we apply a **readout function** ro such that

$$F(G, v) \approx \text{ro}(\zeta^{(d)}(v)).$$

Typically, ro is computed by a feedforward neural network.

Functions Computed by GNNs

To compute a function F that maps each graph G to a mapping $F(G) : V(G) \rightarrow \mathbb{R}^q$, we apply a **readout function** ro such that

$$F(G, v) \approx \text{ro}(\zeta^{(d)}(v)).$$

Typically, ro is computed by a feedforward neural network.

Equivariance

Let F be the function computed by a GNN. Then for all isomorphic graphs G, H , all isomorphisms h from G to H , and all vertices $v \in V(G)$:

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

Functions Computed by GNNs

To compute a function F that maps each graph G to a mapping $F(G) : V(G) \rightarrow \mathbb{R}^q$, we apply a **readout function** ro such that

$$F(G, v) \approx \text{ro}(\zeta^{(d)}(v)).$$

Typically, ro is computed by a feedforward neural network.

Equivariance

Let F be the function computed by a GNN. Then for all isomorphic graphs G, H , all isomorphisms h from G to H , and all vertices $v \in V(G)$:

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

We are mainly interested in **unary queries**, that is, equivariant Boolean functions on the vertices.

The Logical Expressiveness of GNNs

Theorem (Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, G. 2019, Xu, Hu, Leskovec, Jegelka 2019)

Two graphs are distinguishable by a GNN if and only if they are distinguishable in the logic C^2 (or by the 1-dimensional Weisfeiler-Leman algorithm).

The Logical Expressiveness of GNNs

Theorem (Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, G. 2019, Xu, Hu, Leskovec, Jegelka 2019)

Two graphs are distinguishable by a GNN if and only if they are distinguishable in the logic C^2 (or by the 1-dimensional Weisfeiler-Leman algorithm).

In their basic form, GNNs compute functions on the vertices of graphs. We are mainly interested in Boolean functions, that is, **unary queries**.

The Logical Expressiveness of GNNs

Theorem (Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, G. 2019, Xu, Hu, Leskovec, Jegelka 2019)

Two graphs are distinguishable by a GNN if and only if they are distinguishable in the logic C^2 (or by the 1-dimensional Weisfeiler-Leman algorithm).

In their basic form, GNNs compute functions on the vertices of graphs. We are mainly interested in Boolean functions, that is, **unary queries**.

Theorem (Barceló, Kostylev, Monet, Pérez, Reutter, Silva 2019)

Every unary query expressible in the logic C^2 is computable by a GNN.

Circuits with Threshold Gates

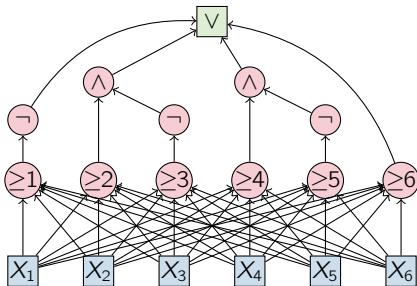
We consider Boolean circuits with **threshold gates**. For all $t \in \mathbb{N}$, a **t -threshold gate** evaluates to 1 if at least t of its inputs are 1.

Circuits with Threshold Gates

We consider Boolean circuits with **threshold gates**. For all $t \in \mathbb{N}$, a **t -threshold gate** evaluates to 1 if at least t of its inputs are 1.

Example

The following threshold circuit evaluates to 1 if an even number of input bits is 1.

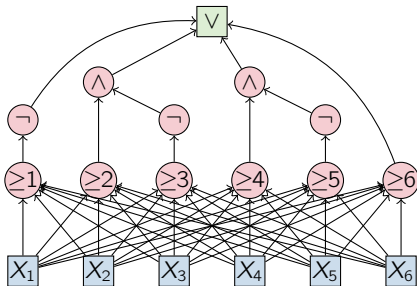


Circuits with Threshold Gates

We consider Boolean circuits with **threshold gates**. For all $t \in \mathbb{N}$, a **t -threshold gate** evaluates to 1 if at least t of its inputs are 1.

Example

The following threshold circuit evaluates to 1 if an even number of input bits is 1.



TC^0 is the class of all languages in $\{0, 1\}^*$ decidable by a polynomial-size, bounded-depth family of threshold circuits.

First-Order Logic with Counting

- ▶ **FO+C** is first-order logic with counting in a 2-sorted framework with number variables ranging over \mathbb{N} and arithmetic.

First-Order Logic with Counting

- ▶ **FO+C** is first-order logic with counting in a 2-sorted framework with number variables ranging over \mathbb{N} and arithmetic.
- ▶ **Difference between C and FO+C**: both allow counting, but C only has numerical constants k in formulas $\exists^{\geq k} x \dots$, whereas FO+C has numerical variables.

First-Order Logic with Counting

- ▶ **FO+C** is first-order logic with counting in a 2-sorted framework with number variables ranging over \mathbb{N} and arithmetic.
- ▶ **Difference between C and FO+C**: both allow counting, but C only has numerical constants k in formulas $\exists^{\geq k} x \dots$, whereas FO+C has numerical variables.

Theorem (Barrington, Immerman, Straubing 1997)

A language $L \subseteq \{0, 1\}^$ is in (nonuniform) TC^0 if and only if it is definable in FO+C with built-in relations.*

Lemma (Forward Direction of Main Theorem)

*Let Q be a unary query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with arbitrary real weights and *rpl approximable* activation functions).*

Then Q is expressible in $\text{FO}^2 + \text{C}$ with built-in relations.

Lemma (Forward Direction of Main Theorem)

*Let Q be a unary query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with arbitrary real weights and *rpl approximable* activation functions).*

Then Q is expressible in $\text{FO}^2 + \text{C}$ with built-in relations.

Proof Ideas

- ▶ Simulate GNNs with rational weights and piecewise linear activations in $\text{FO}^2 + \text{C}$.

Lemma (Forward Direction of Main Theorem)

*Let Q be a unary query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with arbitrary real weights and *rpl approximable* activation functions).*

Then Q is expressible in $\text{FO}^2 + \text{C}$ with built-in relations.

Proof Ideas

- ▶ Simulate GNNs with rational weights and piecewise linear activations in $\text{FO}^2 + \text{C}$.
- ▶ Use built-in relations to simulate families of GNNs.

Lemma (Forward Direction of Main Theorem)

*Let Q be a unary query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with arbitrary real weights and *rpl approximable* activation functions).*

Then Q is expressible in $\text{FO}^2 + \text{C}$ with built-in relations.

Proof Ideas

- ▶ Simulate GNNs with rational weights and piecewise linear activations in $\text{FO}^2 + \text{C}$.
- ▶ Use built-in relations to simulate families of GNNs.
- ▶ Approximate families of arbitrary GNNs by families of rational-weight piecewise-linear GNNs.

Lemma (Forward Direction of Main Theorem)

*Let Q be a unary query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with arbitrary real weights and *rpl* approximable activation functions).*

Then Q is expressible in FO^2+C with built-in relations.

Proof Ideas

- ▶ Simulate GNNs with rational weights and piecewise linear activations in FO^2+C .
- ▶ Use built-in relations to simulate families of GNNs.
- ▶ Approximate families of arbitrary GNNs by families of rational-weight piecewise-linear GNNs.
- ▶ Trade randomness for non-uniformity.

Lemma (Backward Direction of Main Theorem)

Let Q be a unary query that is expressible in FO^2+C with built-in relations.

Then Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with rational weights, piecewise linear activation functions, and sum aggregation).

Lemma (Backward Direction of Main Theorem)

Let Q be a unary query that is expressible in FO^2+C with built-in relations.

Then Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with rational weights, piecewise linear activation functions, and sum aggregation).

Proof Ideas

- ▶ Transform FO^2+C -formula into a guarded (local) form.

Lemma (Backward Direction of Main Theorem)

Let Q be a unary query that is expressible in FO^2+C with built-in relations.

Then Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with rational weights, piecewise linear activation functions, and sum aggregation).

Proof Ideas

- ▶ Transform FO^2+C -formula into a guarded (local) form.
- ▶ Simulate guarded logic on graphs by message passing and arithmetic by feedforward neural network.

Lemma (Backward Direction of Main Theorem)

Let Q be a unary query that is expressible in FO^2+C with built-in relations.

Then Q is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation (with rational weights, piecewise linear activation functions, and sum aggregation).

Proof Ideas

- ▶ Transform FO^2+C -formula into a guarded (local) form.
- ▶ Simulate guarded logic on graphs by message passing and arithmetic by feedforward neural network.
- ▶ Random initialisation is used to obtain linear order.

Concluding Remarks

- ▶ We have a good understanding of the expressiveness of GNNs. Yet many interesting questions remain open, in particular regarding uniformity (expressiveness results across input sizes) and recurrent GNNs.

Concluding Remarks

- ▶ We have a good understanding of the expressiveness of GNNs. Yet many interesting questions remain open, in particular regarding uniformity (expressiveness results across input sizes) and recurrent GNNs.

For example:

Can all graph queries computable in polynomial time be expressed by a recurrent GNN with Random Initialisation?

Concluding Remarks

- ▶ We have a good understanding of the expressiveness of GNNs. Yet many interesting questions remain open, in particular regarding uniformity (expressiveness results across input sizes) and recurrent GNNs.

For example:

Can all graph queries computable in polynomial time be expressed by a recurrent GNN with Random Initialisation?

- ▶ Expressiveness results only tell half the story, because they ignore learning. We also have recent results on PAC learnability with GNNs (Morris, Geerts, G., ICML 2023).