RWTH AACHEN UNIVERSITY

The Descriptive Complexity of Graph Neural Networks

Martin Grohe

Graph Neural Networks

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.
- They can be viewed as a generalisation of convolutional neural networks from a rigid grid structure to more flexibly structured data.

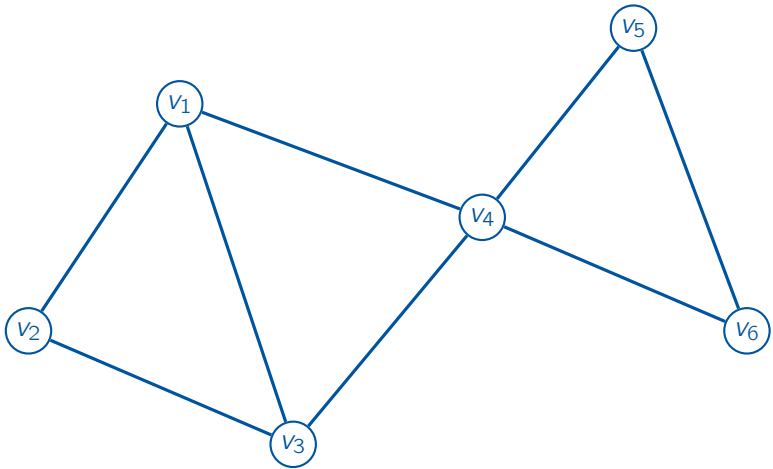
Graph Neural Networks

- Graph neural networks (GNNs) are deep learning architectures for machine learning problems on graphs.
- They can be viewed as a generalisation of convolutional neural networks from a rigid grid structure to more flexibly structured data.
- GNNs have a wide range of applications, for example, in computational biology, chemical engineering, physics, etc.

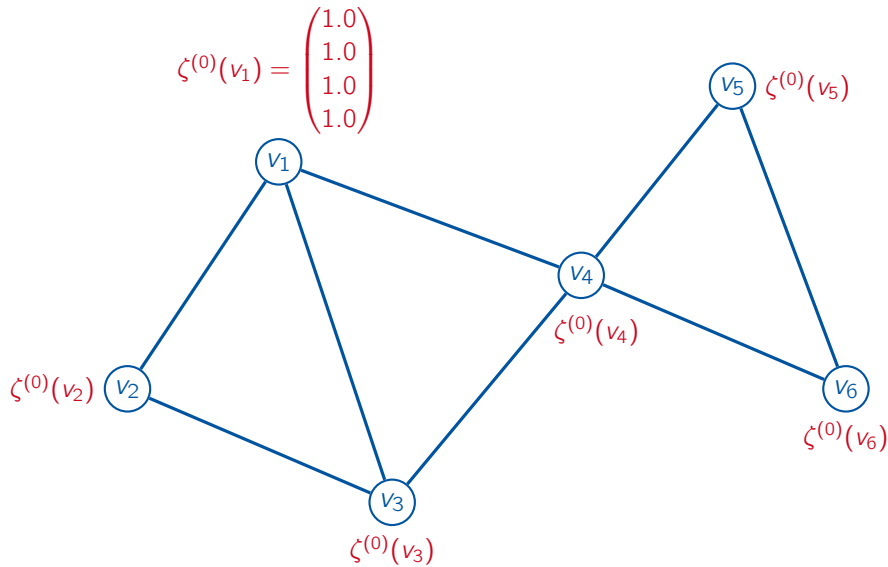
Graph Neural Networks

- Graph neural networks (GNNs) are deep learning architectures for machine learning problems on graphs.
- They can be viewed as a generalisation of convolutional neural networks from a rigid grid structure to more flexibly structured data.
- 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.
  In this talk, we focus on the core GNN architecture known as message passing graph neural network or aggregate-combine graph neural network.

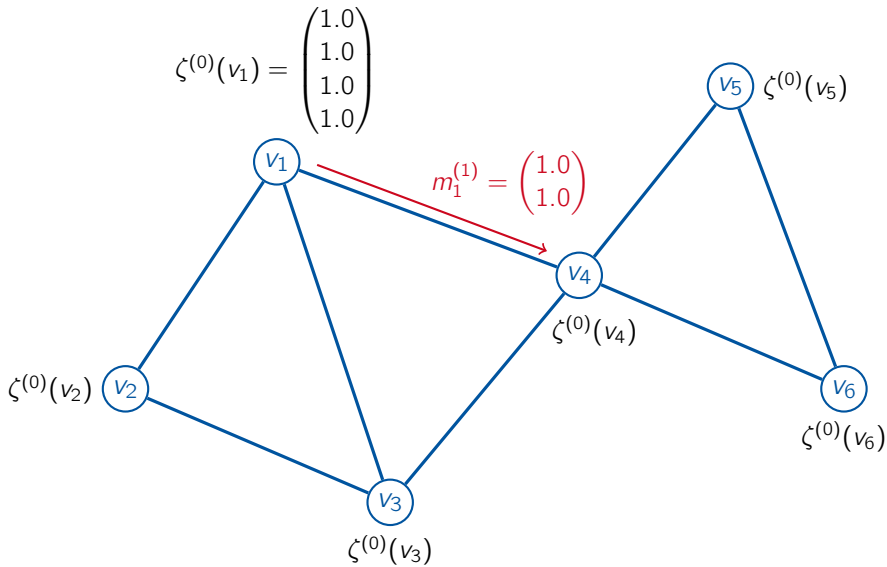
Graph Neural Networks



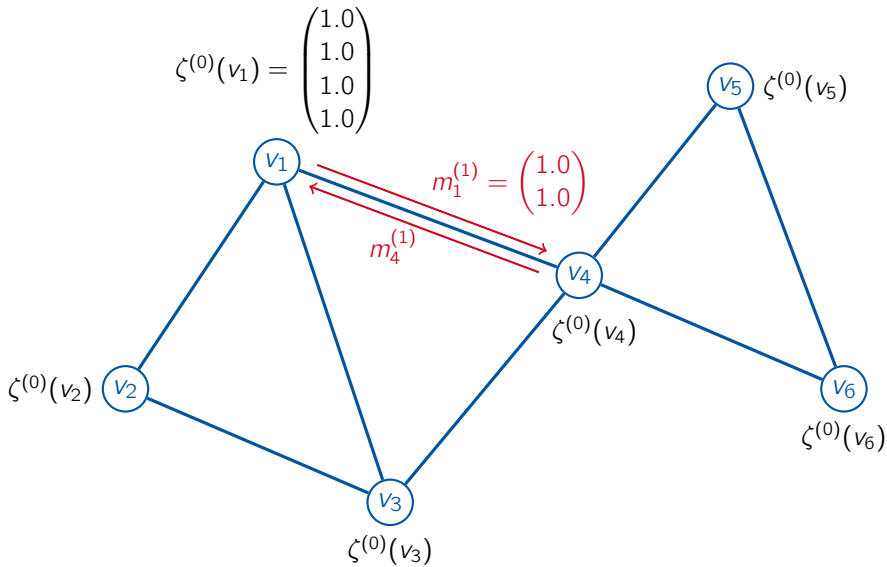
Graph Neural Networks



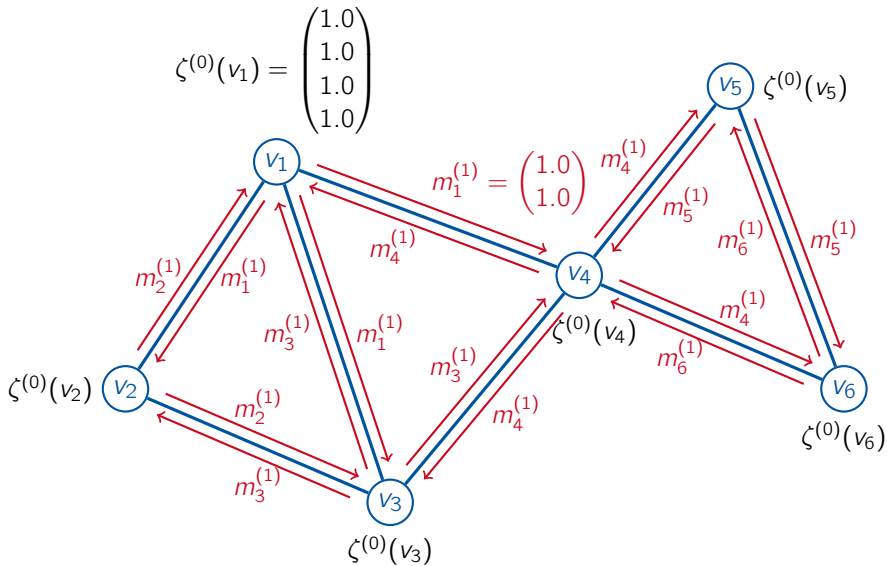
Graph Neural Networks



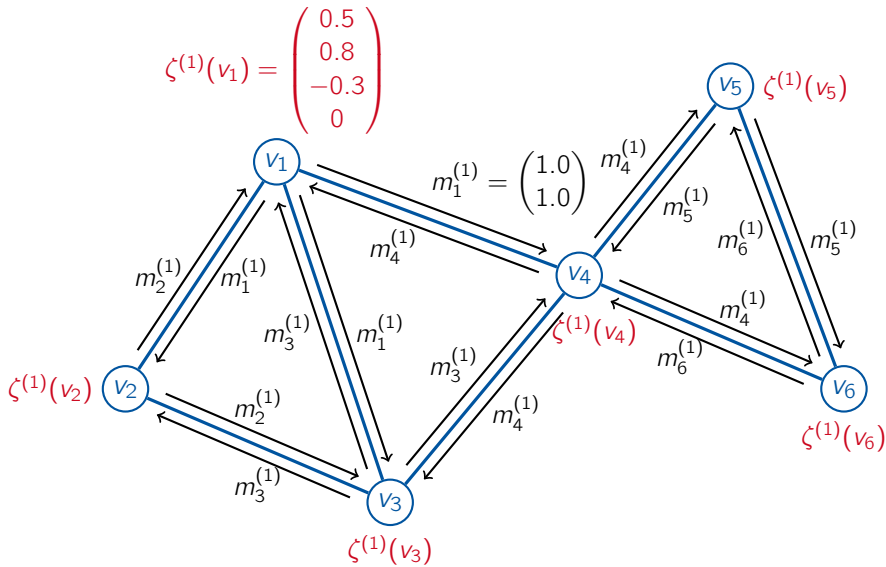
Graph Neural Networks



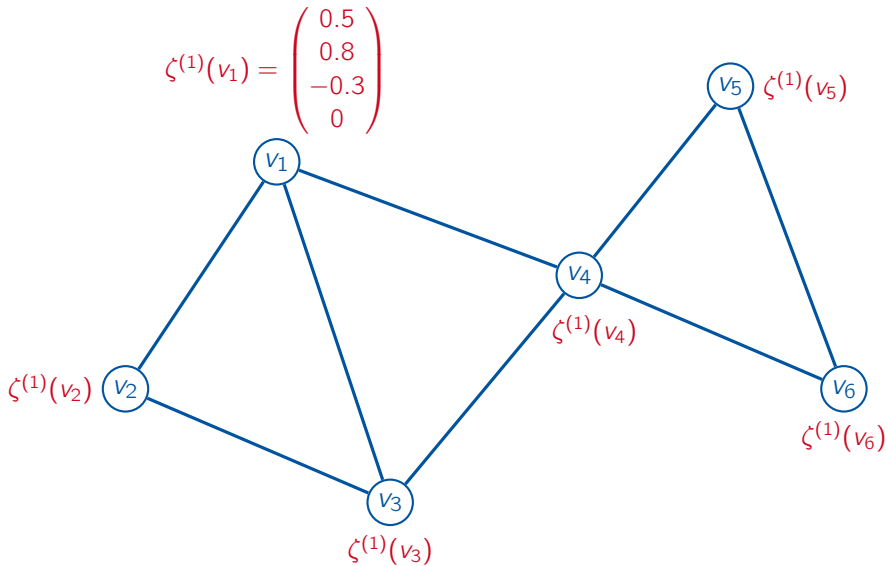
Graph Neural Networks



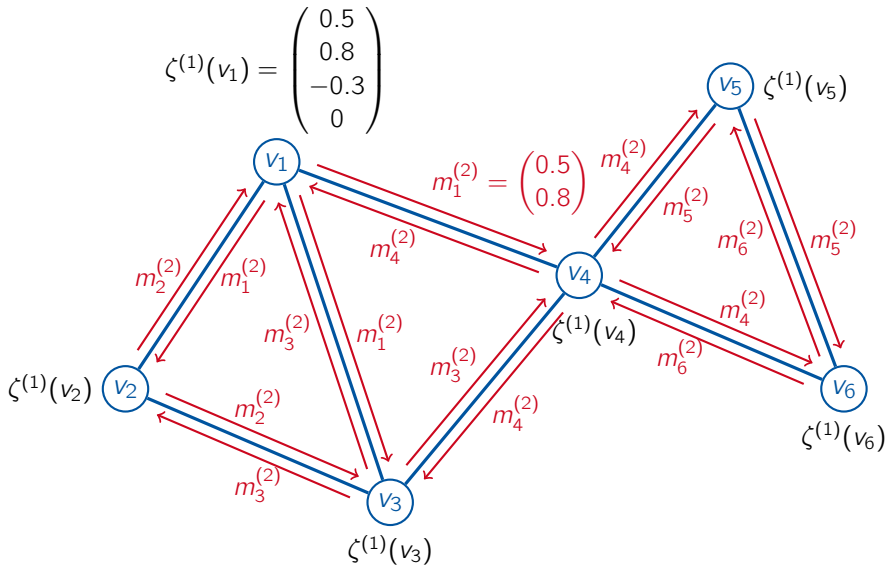
Graph Neural Networks



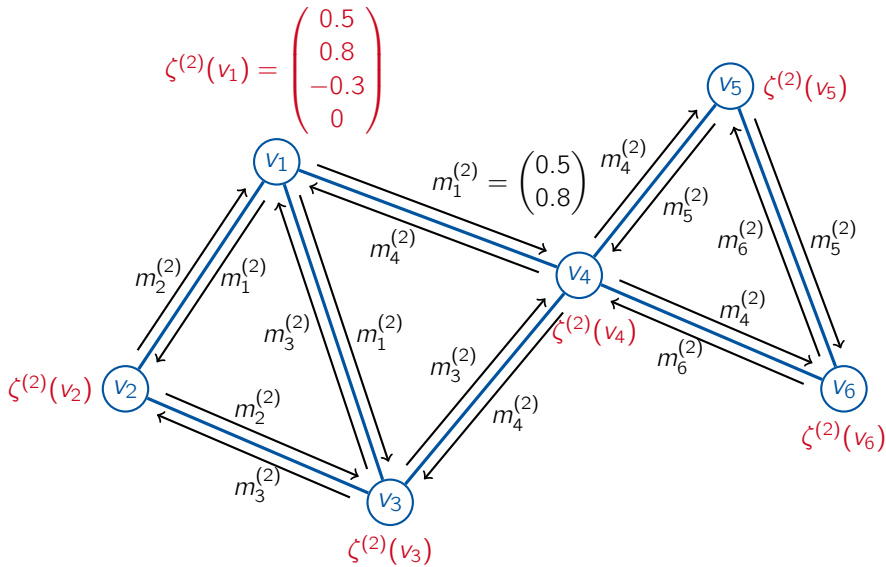
Graph Neural Networks



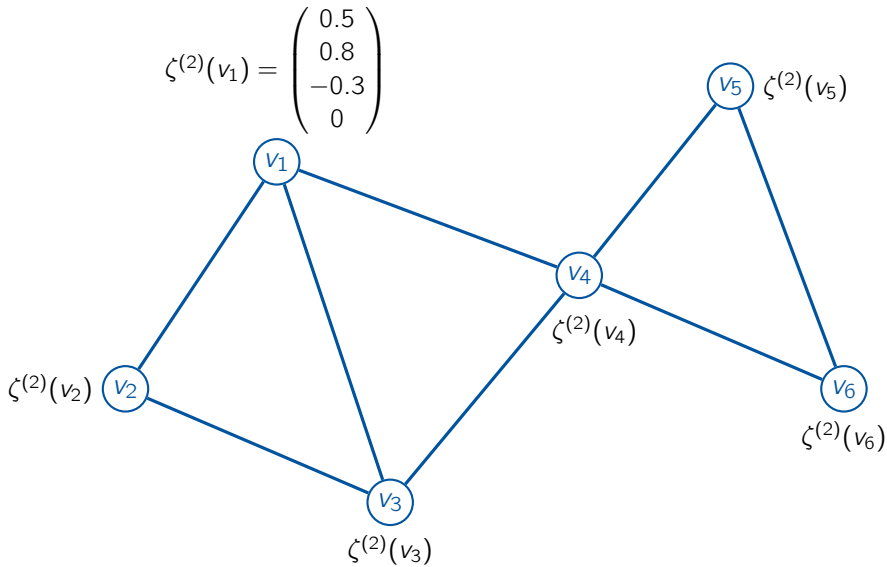
Graph Neural Networks



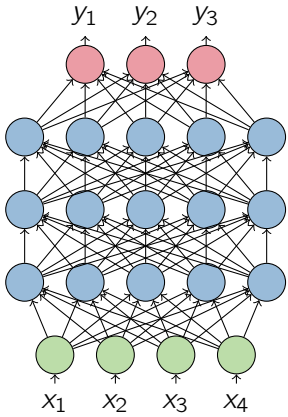
Graph Neural Networks

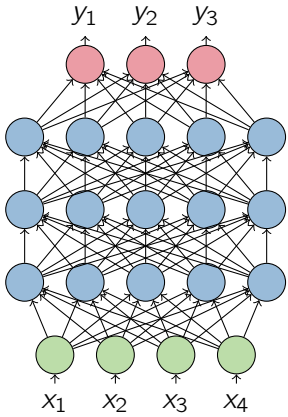


Graph Neural Networks



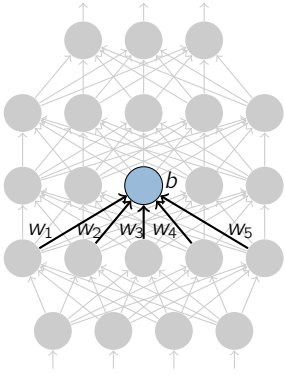
Feedforward Neural Networks





Feedforward Neural Networks

- Nodes and edges are weighted.



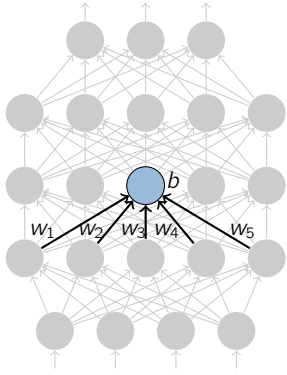
Feedforward Neural Networks

- Nodes and edges are weighted.
- Node with weight $b \in \mathbb{R}$ and incoming edges with weights $w_1, \ldots, w_k \in \mathbb{R}$ computes function

$$x_1, \ldots, x_k \mapsto \sigma\left(b + \sum_{i=1}^{k} w_i x_i\right),$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function.

Feedforward Neural Networks

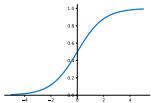


- Nodes and edges are weighted.
- Node with weight $b \in \mathbb{R}$ and incoming edges with weights $w_1, \ldots, w_k \in \mathbb{R}$ computes function

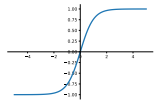
$$x_1, \ldots, x_k \mapsto \sigma\left(b + \sum_{i=1}^{k} w_i x_i\right),$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function.

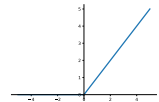
Typical activation functions are:



sigmoid



hyperbolic tangent



rectified linear unit (ReLU)

Universal Approximation Theorem

Theorem (Cybenko 1989, Hornik 1991)

Let

- $\sigma : \mathbb{R} \to \mathbb{R}$ *be be continuous and not polynomial,*
- $f : K \to \mathbb{R}^n$ *continuous, where* $K \subseteq \mathbb{R}^m$ *is a compact set,*
- $\epsilon > 0$.

Universal Approximation Theorem

Theorem (Cybenko 1989, Hornik 1991)

Let

- $\sigma : \mathbb{R} \to \mathbb{R}$ *be be continuous and not polynomial,*
- $f : K \to \mathbb{R}^n$ *continuous, where* $K \subseteq \mathbb{R}^m$ *is a compact set,*
- $\epsilon > 0$.

Then there is a depth-2 feedforward neural network N *with activation functions*

- σ *on layer* 1*,*
- *the identity on layer 2*

computing a function f_N *such that*

$$\sup_{\mathbf{x} \in K} \|f(\mathbf{x}) - f_N(\mathbf{x})\| < \epsilon.$$

Computation of GNNs

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

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

Computation of GNNs

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

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^p$ encodes node labels of v

Computation of GNNs

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

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^p$ encodes node labels of v

Aggregation: $\alpha^{(t)}(v) \coloneqq \mathrm{agg}_t\Big(\Big\{\!\!\Big\{ \mathrm{msg}_t\big(\zeta^{(t-1)}(v), \zeta^{(t-1)}(w)\big) \;\Big|\; w \in N_G(v) \Big\}\!\!\Big\}\Big)$

- $\mathrm{msg}_t : \mathbb{R}^{2p} \to \mathbb{R}^{p'}$ message function computed by a feedforward neural network
- agg_t aggregation function: sum, mean or max

Computation of GNNs

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

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^p$ encodes node labels of v

Aggregation: $\alpha^{(t)}(v) \coloneqq \mathrm{agg}_t\Big(\big\{\!\!\big\{ \mathrm{msg}_t\big(\zeta^{(t-1)}(v), \zeta^{(t-1)}(w)\big) \;\Big|\; w \in N_G(v) \big\}\!\!\big\}\Big)$

- $\mathrm{msg}_t : \mathbb{R}^{2p} \to \mathbb{R}^{p'}$ message function computed by a feedforward neural network
- agg_t aggregation function: sum, mean or max

Global readout: $\gamma^{(t)} \coloneqq \mathrm{agg}'_t\big(\big\{\!\!\big\{ \zeta^{(t-1)}(w) \;\big|\; w \in V(G) \big\}\!\!\big\}\big)$

- agg'_t aggregation function: sum, mean or max

Computation of GNNs

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

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

Initialisation: $\zeta^{(0)}(v) \in \mathbb{R}^p$ encodes node labels of v

Aggregation: $\alpha^{(t)}(v) \coloneqq \mathrm{agg}_t\Big(\Big\{\!\!\Big\{ \mathrm{msg}_t\big(\zeta^{(t-1)}(v), \zeta^{(t-1)}(w)\big) \;\Big|\; w \in N_G(v) \Big\}\!\!\Big\}\Big)$

- $\mathrm{msg}_t : \mathbb{R}^{2p} \to \mathbb{R}^{p'}$ message function computed by a feedforward neural network
- agg_t aggregation function: sum, mean or max

Global readout: $\gamma^{(t)} \coloneqq \mathrm{agg}'_t\big(\{\!\!\{ \zeta^{(t-1)}(w) \mid w \in V(G) \}\!\!\}\big)$

- agg'_t aggregation function: sum, mean or max

Combination: $\zeta^{(t)}(v) \coloneqq \mathrm{comb}_t\big(\zeta^{(t-1)}(v), \alpha^{(t)}(v), \gamma^{(t)}\big)$

- $\mathrm{comb}_t : \mathbb{R}^{2p+p'} \to \mathbb{R}^p$ function computed by a feedforward neural network

Node-Level Functions

To compute a function Φ that maps each graph G to a signal $\Phi(G, \cdot) : V(G) \to \mathbb{R}^q$, we apply a readout function ro such that

$$\Phi(G, v) = \mathrm{ro}\big(\zeta^{(d)}(v)\big).$$

- $\mathrm{ro} : \mathbb{R}^p \to \mathbb{R}^q$ is computed by a feedforward neural network

Functions Computed by GNNs

Node-Level Functions

To compute a function Φ that maps each graph G to a signal $\Phi(G, \cdot) : V(G) \to \mathbb{R}^q$, we apply a readout function ro such that

$$\Phi(G, v) = \mathrm{ro}\big(\zeta^{(d)}(v)\big).$$

- $\mathrm{ro} : \mathbb{R}^p \to \mathbb{R}^q$ is computed by a feedforward neural network

Graph-Level Functions

To compute a function φ that maps graphs to $\mathbb{R}^q$, we apply an aggregate readout function aggro such that

$$\varphi(G) = \mathrm{ro}\Big(\mathrm{agg}\Big(\Big\{\!\!\Big\{ \zeta^{(d)}(v) \;\Big|\; v \in V(G) \Big\}\!\!\Big\}\Big)\Big).$$

- agg aggregation function: sum, mean or max
- $\mathrm{ro} : \mathbb{R}^p \to \mathbb{R}^q$ is computed by a feedforward neural network

Invariance of Graph-Level Functions

Let φ be a graph-level function computed by a GNN. Then for all isomorphic graphs G, H,

$$\varphi(G) = \varphi(H).$$

Invariance of Graph-Level Functions

Let φ be a graph-level function computed by a GNN. Then for all isomorphic graphs G, H,

$$\varphi(G) = \varphi(H).$$

Equivariance of Node-Level Functions

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

$$\Phi(G, v) = \Phi(H, h(v)).$$

Invariance of Graph-Level Functions

Let φ be a graph-level function computed by a GNN. Then for all isomorphic graphs G, H,

$$\varphi(G) = \varphi(H).$$

Equivariance of Node-Level Functions

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

$$\Phi(G, v) = \Phi(H, h(v)).$$

We are mainly interested in Boolean queries and unary queries, that is, invariant graph level functions $G \mapsto \{0, 1\}$ and equivariant node-level functions $G \mapsto \{0, 1\}^{V(G)}$.

Recurrent GNNs

As described so far, GNNs have a fixed number d of layers, and each layer t has its own aggregation function agg_t and combination function comb_t.

As described so far, GNNs have a fixed number d of layers, and each layer t has its own aggregation function agg_t and combination function comb_t.

Recurrent GNNs

Instead, we can take a single layer and apply it repeatedly. That is, we have a single aggregation function agg and combination function comb and let:

$$\zeta^{(t)}(v) = \text{comb}\Big(\zeta^{(t-1)}(v), \text{agg}\big(\{\!\{\zeta^{(t-1)}(w) \mid w \in N_G(v)\}\!\}\big)\Big)$$

for all $t \geq 1$.

As described so far, GNNs have a fixed number d of layers, and each layer t has its own aggregation function agg_t and combination function comb_t.

Recurrent GNNs

Instead, we can take a single layer and apply it repeatedly. That is, we have a single aggregation function agg and combination function comb and let:

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

for all $t \geq 1$.

We determine the number d of iterations at runtime (maybe depending on the size of the input graph, or even depending on the evolution of the sequence $\big(\zeta^{(t)}\big)_{t\geq 0}$).

As described so far, GNNs have a fixed number d of layers, and each layer t has its own aggregation function agg_t and combination function comb_t.

Recurrent GNNs

Instead, we can take a single layer and apply it repeatedly. That is, we have a single aggregation function agg and combination function comb and let:

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

for all $t \geq 1$.

We determine the number d of iterations at runtime (maybe depending on the size of the input graph, or even depending on the evolution of the sequence $\big(\zeta^{(t)}\big)_{t\geq 0}$). *We do not require any kind of convergence.*

As described so far, GNNs have a fixed number d of layers, and each layer t has its own aggregation function agg_t and combination function comb_t.

Recurrent GNNs

Instead, we can take a single layer and apply it repeatedly. That is, we have a single aggregation function agg and combination function comb and let:

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

for all $t \geq 1$.

We determine the number d of iterations at runtime (maybe depending on the size of the input graph, or even depending on the evolution of the sequence $\big(\zeta^{(t)}\big)_{t\geq 0}$). *We do not require any kind of convergence.*

As before, we apply the readout function to the dth layer.

Digression: The Weisfeiler-Leman Algorithm

A Simple Combinatorial Algorithm

The Colour Refinement algorithm iteratively computes a colouring of the vertices of graph G.

A Simple Combinatorial Algorithm

The Colour Refinement algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

A Simple Combinatorial Algorithm

The Colour Refinement algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

Refinement Step 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.

A Simple Combinatorial Algorithm

The Colour Refinement algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

Refinement Step 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.

Refinement is repeated until colouring stays stable.

Run Colour Refinement (Demo by Holger Dell)

A Simple Combinatorial Algorithm

The Colour Refinement algorithm iteratively computes a colouring of the vertices of graph G.

Initialisation All vertices get the same colour.

Refinement Step 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.

Refinement is repeated until colouring stays stable.

Run Colour Refinement (Demo by Holger Dell)

Remark

Colour Refinement is essentially the same as the 1-dimensional Weisfeiler-Leman algorithm. There is a subtle difference that is actually relevant here (but we ignore it in the talk).

Colour Refinement as an Isomorphism Test

Colour Refinement distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

Colour Refinement as an Isomorphism Test

Colour Refinement distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

Thus Colour Refinement can be used as an incomplete isomorphism test.

Colour Refinement as an Isomorphism Test

Colour Refinement distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

Thus Colour Refinement can be used as an incomplete isomorphism test.

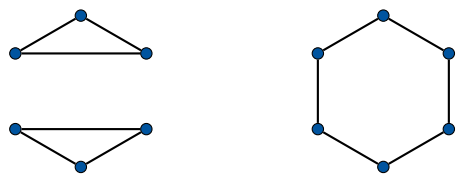
- works on almost all graphs (Babai, Erdös, Selkow 1980)

Colour Refinement as an Isomorphism Test

Colour Refinement distinguishes two graphs G, H if their colour histograms differ, that is, some colour appears a different number of times in G and H.

Thus Colour Refinement can be used as an incomplete isomorphism test.

- works on almost all graphs (Babai, Erdös, Selkow 1980)
- fails on some very simple graphs:



Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

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

1. *colour refinement does not distinguish G and H;*

Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

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

1. *colour refinement does not distinguish G and H;*
2. *G and H satisfy the same sentences of the logic C^2, the 2-variable fragment of first-order logic with counting quantifiers $\exists^{\geq n} x$.*

Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

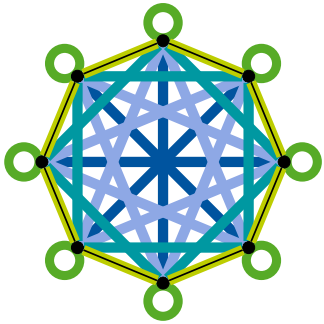
For all graphs G, H, the following are equivalent:

1. *colour refinement does not distinguish G and H;*
2. *G and H satisfy the same sentences of the logic C^2, the 2-variable fragment of first-order logic with counting quantifiers $\exists^{\geq n} x$.*
3. *G and H are fractionally isomorphic, that is, there is a doubly stochastic matrix X such that $A_G X = X A_H$.*

Theorem (Immerman and Lander 1990, Tinhofer 1991, Dvorak 2010)

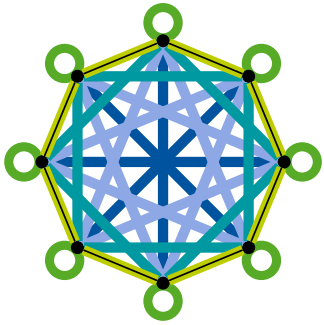
For all graphs G, H, the following are equivalent:

1. *colour refinement does not distinguish G and H;*
2. *G and H satisfy the same sentences of the logic C^2, the 2-variable fragment of first-order logic with counting quantifiers $\exists^{\geq n} x$.*
3. *G and H are fractionally isomorphic, that is, there is a doubly stochastic matrix X such that $A_G X = X A_H$.*
4. *For all trees T, the number of homomorphisms from T to G equals the number of homomorphisms from T to H.*



Higher-Dimensional Weisfeiler-Leman

The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

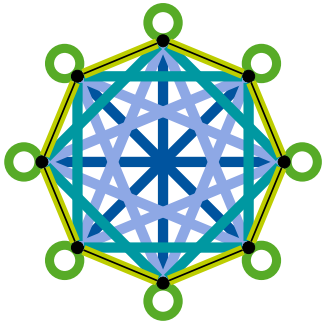


Higher-Dimensional Weisfeiler-Leman

The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

Running time: $O(n^{k+1} \log n)$

Higher-Dimensional Weisfeiler-Leman

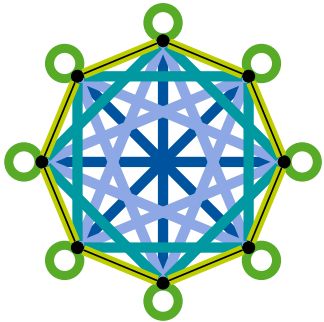


The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

Running time: $O(n^{k+1} \log n)$

- 1-WL is essentially the same as Colour Refinement.

Higher-Dimensional Weisfeiler-Leman

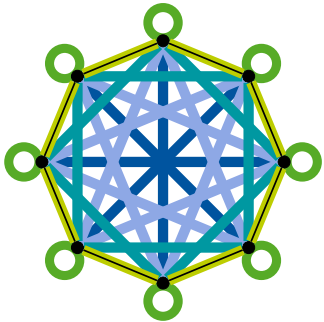


The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

Running time: $O(n^{k+1} \log n)$

- 1-WL is essentially the same as Colour Refinement.
- k-WL is much more powerful, but still not a complete isomorphism test: *for every k there are non-isomorphic graphs G_k, H_k of size $O(k)$ not distinguished by k-WL* (Cai, Fürer, Immerman 1991).

Higher-Dimensional Weisfeiler-Leman



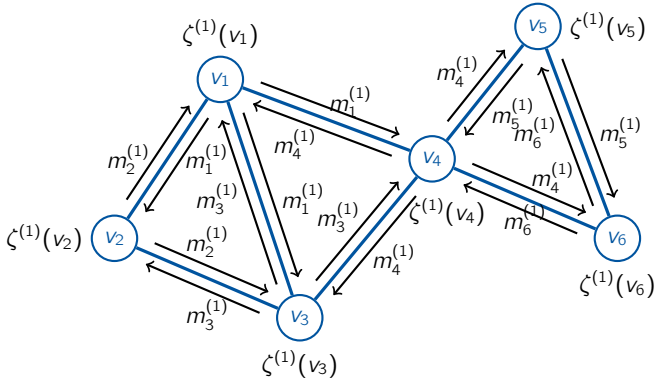
The k-dimensional Weisfeiler-Leman algorithm (k-WL) iteratively colours k-tuples of nodes (Weisfeiler and Leman 1968, Babai $\sim$1980)

Running time: $O(n^{k+1} \log n)$

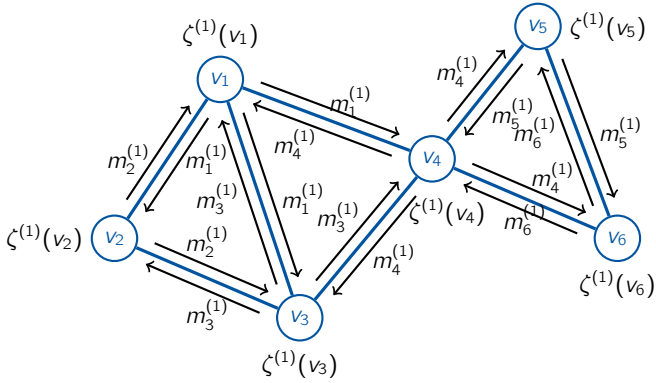
- 1-WL is essentially the same as Colour Refinement.
- k-WL is much more powerful, but still not a complete isomorphism test: *for every k there are non-isomorphic graphs G_k, H_k of size $O(k)$ not distinguished by k-WL* (Cai, Fürer, Immerman 1991).
- The characterisations of Colour Refinement in terms of logic, linear (in)equalities, and homomorphism counts can be generalised to k-WL.

Weisfeiler and Leman go Neural

GNNs Revisited

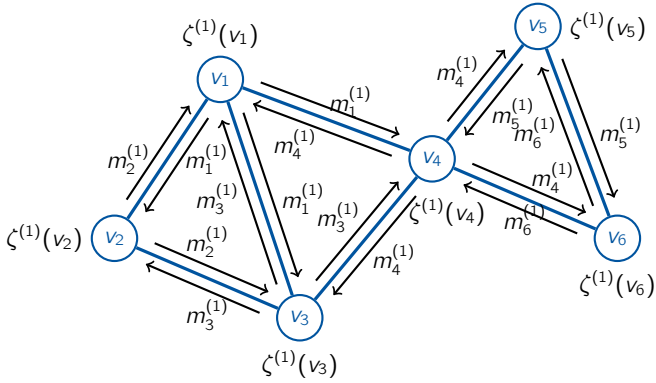


GNNs Revisited



Initialisation: $\zeta^{(0)}(v)$ encodes node label of v

GNNs Revisited

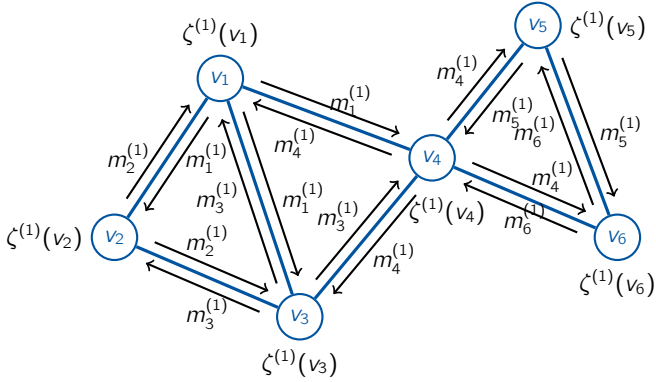


Initialisation: $\zeta^{(0)}(v)$ encodes node label of v

Aggregation and Combination:

$$\zeta^{(t)}(v) = \mathrm{comb}_t\Big(\zeta^{(t-1)}(v), \mathrm{agg}_t\big(\{\!\{\mathrm{msg}_t(\zeta^{(t-1)}(v), \zeta^{(t-1)}(w)) \mid w \in N_G(v)\}\!\}\big), \mathrm{agg}'_t\big(\{\!\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

GNNs Revisited



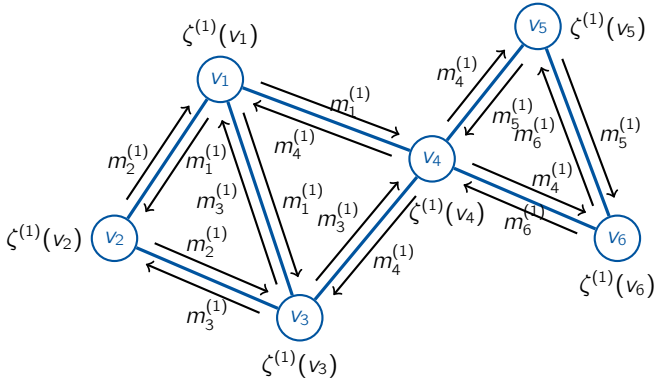
Initialisation: $\zeta^{(0)}(v)$ encodes node label of v

Aggregation and Combination:

$$\zeta^{(t)}(v) = \mathrm{comb}_t\Big(\zeta^{(t-1)}(v), \mathrm{agg}_t\big(\{\!\{\mathrm{msg}_t(\zeta^{(t-1)}(v), \zeta^{(t-1)}(w)) \mid w \in N_G(v)\}\!\}\big), \mathrm{agg}'_t\big(\{\!\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

Read-out: $\Phi_N(G, v) \coloneqq \mathrm{ro}(\zeta^{(d)}(v))$ (node level)

GNNs Revisited



Initialisation: $\zeta^{(0)}(v)$ encodes node label of v

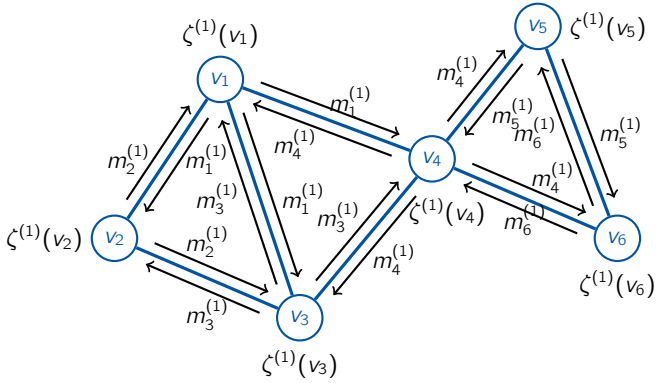
Aggregation and Combination:

$$\zeta^{(t)}(v) := \mathrm{comb}'_t\Big(\zeta^{(t-1)}(v), \mathrm{agg}'_t\big(\{\!\{(\zeta^{(t-1)}(v), \zeta^{(t-1)}(w)) \mid w \in N_G(v)\}\!\}\big), \mathrm{agg}'_t\big(\{\!\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

Notation: Index N refers to GNN N

Read-out: $\Phi_N(G, v) := \mathrm{ro}(\zeta^{(d)}(v))$ (node level)

GNNs Revisited



Initialisation: $\zeta^{(0)}(v)$ encodes node label of v

Aggregation and Combination:

$$\zeta^{(t)}(v) = \mathrm{comb}_t\Big(\zeta^{(t-1)}(v), \mathrm{agg}_t\big(\{\!\{\mathrm{msg}_t(\zeta^{(t-1)}(v), \zeta^{(t-1)}(w)) \mid w \in N_G(v)\}\!\}\big), \mathrm{agg}'_t\big(\{\!\{\zeta^{(t-1)}(w) \mid w \in V(G)\}\!\}\big)\Big)$$

Read-out: $\Phi_N(G, v) \coloneqq \mathrm{ro}(\zeta^{(d)}(v))$ (node level)

$\varphi_N(G) \coloneqq \mathrm{ro}(\mathrm{agg}(\{\!\{\zeta^{(d)}(v)) \mid v \in V(G)\}\!\}))$ (graph level)

Distinguishing Graphs

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

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

1. *G and H are distinguishable by a GNN, that is, there is a GNN N such that $\varphi_N(G) \neq \varphi_N(H)$;*
2. *1-WL distinguishes G and H.*

Expressing Graph Properties

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

Let $\mathcal{Q}$ be a query expressible in the logic C^2. Then there is a GNN computing $\mathcal{Q}$.

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

Let $\mathcal{Q}$ be a query expressible in the logic C^2. Then there is a GNN computing $\mathcal{Q}$.

Remarks

- Barceló et al. also prove a converse of the theorem for queries expressible in first-order logic.

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

Let $\mathcal{Q}$ be a query expressible in the logic C^2. Then there is a GNN computing $\mathcal{Q}$.

Remarks

- Barceló et al. also prove a converse of the theorem for queries expressible in first-order logic.
- This is a uniform expressibility result: the property can be expressed by a single GNN across input graphs of all sizes.

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

Let $\mathbb{Q}$ be a query expressible in the logic C^2. Then there is a GNN computing $\mathbb{Q}$.

Remarks

- Barceló et al. also prove a converse of the theorem for queries expressible in first-order logic.
- This is a uniform expressibility result: the property can be expressed by a single GNN across input graphs of all sizes.
- The proof assumes activation functions like ReLU or linearised sigmoid. It is open whether the theorem also holds for GNNs with logistic (sigmoid) or tanh activations.

Higher-Order Graph Neural Networks

We also proposed a model of higher-order GNNs passing messages between tuples of vertices.

We also proposed a model of higher-order GNNs passing messages between tuples of vertices.

Theorem (Morris, Ritzert, Fey, Hamilton, Lenssen, Rattan, G. 2019)

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

1. *G and H are distinguishable by a GNN, that is, there is a order $(k+1)$-GNN N such that $\varphi_N(G) \neq \varphi_N(H)$;*
2. *k-WL distinguishes G and H.*

Random Node Initialisation

Suppose we initialise the states of the nodes of a GNN randomly:

$$\zeta^{(0)}(v) \sim N(0, 1).$$

Random Node Initialisation

Suppose we initialise the states of the nodes of a GNN randomly:

$$\zeta^{(0)}(v) \sim N(0, 1).$$

- In GNNs with RI, we also assume that the GNNs use a global readout on each layer.

Random Node Initialisation

Suppose we initialise the states of the nodes of a GNN randomly:

$$\zeta^{(0)}(v) \sim N(0, 1).$$

- In GNNs with RI, we also assume that the GNNs use a global readout on each layer.
- A GNN with random initialisation (RI) computes a random variable and no longer a deterministic function.

Random Node Initialisation

Suppose we initialise the states of the nodes of a GNN randomly:

$$\zeta^{(0)}(v) \sim N(0, 1).$$

- In GNNs with RI, we also assume that the GNNs use a global readout on each layer.
- A GNN with random initialisation (RI) computes a random variable and no longer a deterministic function.
- This random variable is invariant/equivariant.

Random Node Initialisation

Suppose we initialise the states of the nodes of a GNN randomly:

$$\zeta^{(0)}(v) \sim N(0,1).$$

- In GNNs with RI, we also assume that the GNNs use a global readout on each layer.
- A GNN with random initialisation (RI) computes a random variable and no longer a deterministic function.
- This random variable is invariant/equivariant.
- GNNs with RI are substantially more expressive than GNNs with constant initialisation. This has been experimentally demonstrated by (Sato, Yamada, Kashima 2020) and (Abboud et al 2020)

Random Node Initialisation

Suppose we initialise the states of the nodes of a GNN randomly:

$$\zeta^{(0)}(v) \sim N(0,1).$$

- In GNNs with RI, we also assume that the GNNs use a global readout on each layer.
- A GNN with random initialisation (RI) computes a random variable and no longer a deterministic function.
- This random variable is invariant/equivariant.
- GNNs with RI are substantially more expressive than GNNs with constant initialisation. This has been experimentally demonstrated by (Sato, Yamada, Kashima 2020) and (Abboud et al 2020)
- (Sato et al. 2020) proved that many interesting combinatorial problems can be expressed by GNNs with RI.

Universality of GNNs with RI

Theorem (Abboud, Ceylan, G., Lukasiewicz 2021)

Let $\epsilon, \delta > 0$ *and* $n \in \mathbb{N}$ *and* $\psi : \mathscr{G}_n \to [0, 1]$.

Universality of GNNs with RI

Graphs of order n

Theorem (Abboud, Ceylan, G., Lukasiewicz 2021)

Let $\epsilon, \delta > 0$ *and* $n \in \mathbb{N}$ *and* $\psi : \mathcal{G}_n \to [0, 1]$.

Theorem (Abboud, Ceylan, G., Lukasiewicz 2021)

Let $\epsilon, \delta > 0$ *and* $n \in \mathbb{N}$ *and* $\psi : \mathscr{G}_n \to [0, 1]$.
Then there is a GNN with RI N *such that for all* $G \in \mathscr{G}_n$,

$$\Pr\left(\left|\psi(G) - \varphi_N(G)\right| \leq \epsilon\right) \geq 1 - \delta.$$

Theorem (Abboud, Ceylan, G., Lukasiewicz 2021)

Let $\epsilon, \delta > 0$ and $n \in \mathbb{N}$ and $\psi : \mathcal{G}_n \to [0, 1]$.
Then there is a GNN with RI N such that for all $G \in \mathcal{G}_n$,

$$\mathsf{Pr}\left(\left|\psi(G) - \varphi_N(G)\right| \leq \epsilon\right) \geq 1 - \delta.$$

Remarks

- There is also a node-level version of this result.

Theorem (Abboud, Ceylan, G., Lukasiewicz 2021)

Let $\epsilon, \delta > 0$ and $n \in \mathbb{N}$ and $\psi : \mathcal{G}_n \to [0, 1]$.
Then there is a GNN with RI N such that for all $G \in \mathcal{G}_n$,

$$\Pr\left(\left|\psi(G) - \varphi_N(G)\right| \leq \epsilon\right) \geq 1 - \delta.$$

Remarks

- There is also a node-level version of this result.
- These approximation results are non-uniform, that is, for each size of the input graph we need a separate GNN.

Theorem (Abboud, Ceylan, G., Lukasiewicz 2021)

Let $\epsilon, \delta > 0$ and $n \in \mathbb{N}$ and $\psi : \mathcal{G}_n \to [0, 1]$.
Then there is a GNN with RI N such that for all $G \in \mathcal{G}_n$,

$$\Pr\left(\left|\psi(G) - \varphi_N(G)\right| \leq \epsilon\right) \geq 1 - \delta.$$

Remarks

- There is also a node-level version of this result.
- These approximation results are non-uniform, that is, for each size of the input graph we need a separate GNN.
- The size of the GNN N can be exponential in n.

The Descriptive Complexity of GNNs

Goal

Understand the power of GNNs to compute queries and compare it to classical models of complexity theory and logic.

Circuits with Threshold Gates

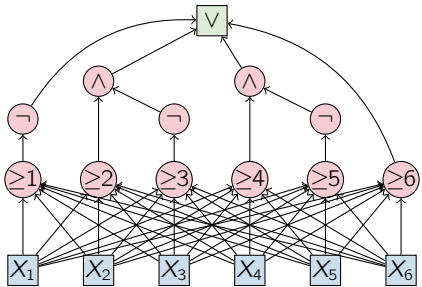
We consider Boolean circuits with threshold gates. For all $t \in \mathbb{N}$, a t-threshold gate evaluates to 1 if at least t of its inputs are 1.

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.

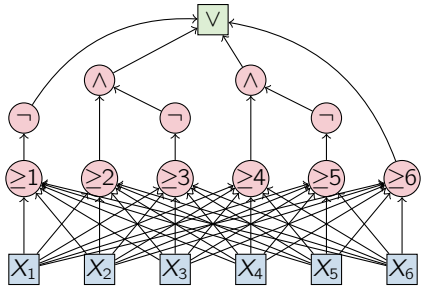


Circuits with Threshold Gates

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.

Theorem (Maass 1997)

Let $\mathscr{f} = (f_n)_{n \geq 1}$ be a family of Boolean functions $f_n : \{0,1\}^n \to \{0,1\}$. Then the following are equivalent.

1. *$\mathscr{f}$ is in TC^0.*
2. *$\mathscr{f}$ is computable by a bounded-depth polynomial-size family of feedforward neural networks with piecewise polynomial activation functions.*

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 $\exists^{\geq k} x \ldots$, 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 $\exists^{\geq k} x \ldots$, whereas FO+C has numerical variables.

Theorem (Barrington, Immerman, Straubing 1990)

Uniform Version: *A language $L \subseteq \{0,1\}^*$ is in dlogtime-uniform TC^0 if and only if it is definable in* FO+C.

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 $\exists^{\geq k} x \ldots$, whereas FO+C has numerical variables.

Theorem (Barrington, Immerman, Straubing 1990)

Uniform Version: *A language $L \subseteq \{0,1\}^*$ is in dlogtime-uniform TC^0 if and only if it is definable in* FO+C.

Nonuniform Version: *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.*

Upper Bound for the Logical Expressivity of GNNs

FO^2+C is the 2-variable fragment of FO+C

Upper Bound for the Logical Expressivity of GNNs

FO^2+C is the 2-variable fragment of FO+C

Theorem (G. 2023)

Let $\mathcal{Q}$ be a unary query computable by a GNN with rational weights and piecewise linear activations. Then $\mathcal{Q}$ is expressible in in FO^2+C.

Upper Bound for the Logical Expressivity of GNNs

FO^2+C is the 2-variable fragment of FO+C

Theorem (G. 2023)

Let $\mathcal{Q}$ be a unary query computable by a GNN with rational weights and piecewise linear activations. Then $\mathcal{Q}$ is expressible in in FO^2+C.

Corollary

Combined with Barcelo et al. (2019), we get

$$\mathrm{C}^2 \quad \subseteq \quad \mathrm{GNN} \quad \subseteq \quad \mathrm{FO}^2\text{+C}.$$

Both inclusions are strict.

Rational piecewise linear (rpl) approximable functions include all common activation functions.

Rational piecewise linear (rpl) approximable functions include all common activation functions.

Theorem (G. 2023)

For all queries $\mathcal{Q}$, the following are equivalent.

1. *$\mathcal{Q}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with arbitrary real weights and rpl approximable activations.*
2. *$\mathcal{Q}$ is computable by a polynomial-size bounded-depth family of GNNs with random initialisation and with rational weights and ReLU activations, using only sum aggregation.*
3. *$\mathcal{Q}$ is expressible in $\mathsf{FO}^2{+}\mathsf{C}$ with built-in relations.*
4. *$\mathcal{Q}$ is in TC^0.*

Lemma

Let $\mathcal{Q}$ be a query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with arbitrary real weights and rpl approximable activation functions.
Then $\mathcal{Q}$ is expressible in $\mathrm{FO}^2{+}\mathrm{C}$ *with built-in relations.*

Lemma

Let $\mathcal{Q}$ be a query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with arbitrary real weights and rpl approximable activation functions.
Then $\mathcal{Q}$ is expressible in $\mathrm{FO}^2{+}\mathrm{C}$ *with built-in relations.*

Proof Ideas

- Simulate GNNs with rational weights and piecewise linear activations in $\mathrm{FO}^2{+}\mathrm{C}$ (previous theorem).

Lemma

Let $\mathcal{Q}$ be a query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with arbitrary real weights and rpl approximable activation functions.
Then $\mathcal{Q}$ is expressible in $\mathrm{FO}^2{+}\mathrm{C}$ *with built-in relations.*

Proof Ideas

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

Lemma

Let $\mathcal{Q}$ be a query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with arbitrary real weights and rpl approximable activation functions.
Then $\mathcal{Q}$ is expressible in $\mathrm{FO}^2{+}\mathrm{C}$ *with built-in relations.*

Proof Ideas

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

Lemma

Let $\mathcal{Q}$ be a query that is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with arbitrary real weights and rpl approximable activation functions.
Then $\mathcal{Q}$ is expressible in $\mathrm{FO}^2{+}\mathrm{C}$ *with built-in relations.*

Proof Ideas

- Simulate GNNs with rational weights and piecewise linear activations in $\mathrm{FO}^2{+}\mathrm{C}$ (previous theorem).
- 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

Let $\mathcal{Q}$ be a query that is expressible in $\mathsf{FO}^2{+}\mathsf{C}$ *with built-in relations. Then $\mathcal{Q}$ is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with rational weights, piecewise linear activation functions, and sum aggregation.*

Lemma

Let $\mathcal{Q}$ be a query that is expressible in FO^2+C *with built-in relations.*
Then $\mathcal{Q}$ is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with rational weights, piecewise linear activation functions, and sum aggregation.

Proof Ideas

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

Lemma

Let $\mathbb{Q}$ be a query that is expressible in FO^2+C *with built-in relations. Then $\mathbb{Q}$ is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and 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

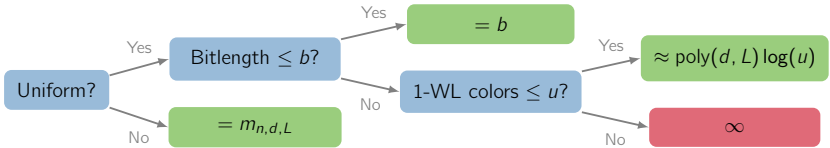
Let $\mathbb{Q}$ be a query that is expressible in FO^2+C *with built-in relations. Then $\mathbb{Q}$ is computable by a polynomial-size bounded-depth family of graph neural networks with random initialisation and with rational weights, piecewise linear activation functions, and sum aggregation.*

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.

Learning and Generalisation

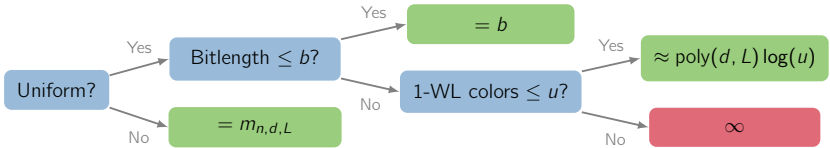
The VC Dimension of GNNs



Theorem (Morris, Geerts, G. 2023)

1. *In the non-uniform regime, the VC-dimension of GNNs is essentially the number of of Colour-Refinement equivalence classes.*

The VC Dimension of GNNs



Theorem (Morris, Geerts, G. 2023)

1. *In the non-uniform regime, the VC-dimension of GNNs is essentially the number of of Colour-Refinement equivalence classes.*
2. *In the uniform regime, the VC-dimension of GNNs is linear in the bitlength of the GNN's weights.*

Concluding Remarks

Concluding Remarks

- GNNs are a very flexible learning architecture, which allows us to adapt them to logical formalisms such as CSPs

Concluding Remarks

- GNNs are a very flexible learning architecture, which allows us to adapt them to logical formalisms such as CSPs
- We have a good understanding of their expressiveness. Yet many interesting questions remain open, in particular regarding uniformity (expressiveness results across input sizes).

Concluding Remarks

- GNNs are a very flexible learning architecture, which allows us to adapt them to logical formalisms such as CSPs
- We have a good understanding of their expressiveness. Yet many interesting questions remain open, in particular regarding uniformity (expressiveness results across input sizes).

  For example:

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

Concluding Remarks

- GNNs are a very flexible learning architecture, which allows us to adapt them to logical formalisms such as CSPs
- We have a good understanding of their expressiveness. Yet many interesting questions remain open, in particular regarding uniformity (expressiveness results across input sizes).

  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. However, most of the results presented here have good experimental support.

A Few References

Grohe, M. (2021). The Logic of Graph Neural Networks.
In: Proc. LICS 2021.
arXiv:2104.14624

Grohe, M. (2023). The Descriptive Complexity of Graph Neural Networks.
In: Proc. LICS 2023.
arXiv:2303.04613

Morris, C., Geerts, F., and Grohe, M. (2023). WL meet VC.
In: Proc. ICML 2023.
arXiv:2301.11039