GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

How can we use random walks in deep learning on graphs?

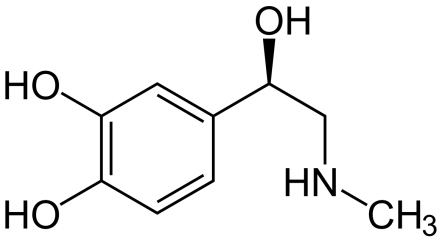
Why do we care?

Martin Ritzert

Work done while at RWTH Aachen together with Jan Tönshoff, Hinrikus Wolf, and Martin Grohe

Introduction

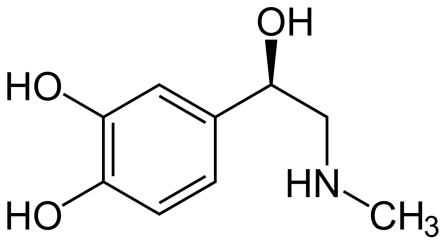
Introduction: Graphs are Everywhere



Molecules

Social Networks

Introduction: Graphs are Everywhere

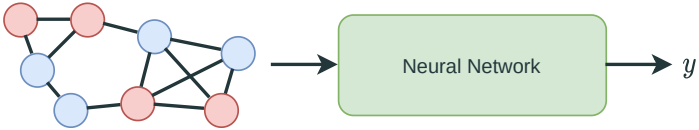


Molecules

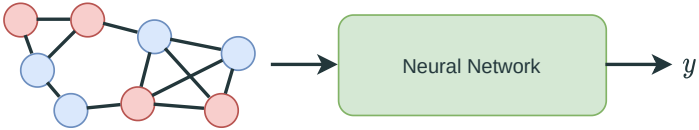
Social Networks

This talk focuses on Graph Classification

Graph Learning

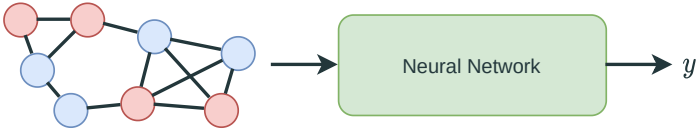


Graph Learning



Fundamental Challenges of Graph Learning:

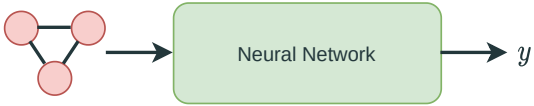
Graph Learning



Fundamental Challenges of Graph Learning:

- Process graphs of any size and structure

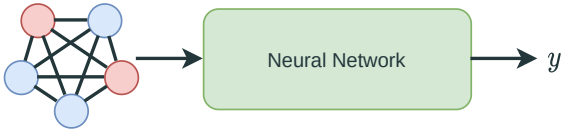
Graph Learning



Fundamental Challenges of Graph Learning:

- Process graphs of any size and structure

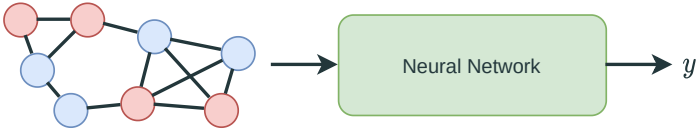
Graph Learning



Fundamental Challenges of Graph Learning:

- Process graphs of any size and structure

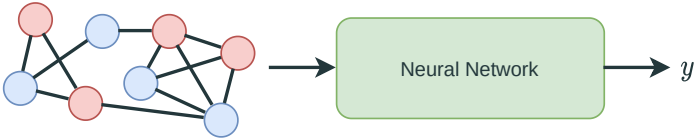
Graph Learning



Fundamental Challenges of Graph Learning:

- Process graphs of any size and structure
- Permutation Invariance

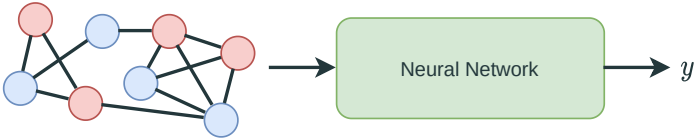
Graph Learning



Fundamental Challenges of Graph Learning:

- Process graphs of any size and structure
- Permutation Invariance

Graph Learning



Fundamental Challenges of Graph Learning:

- Process graphs of any size and structure
- Permutation Invariance

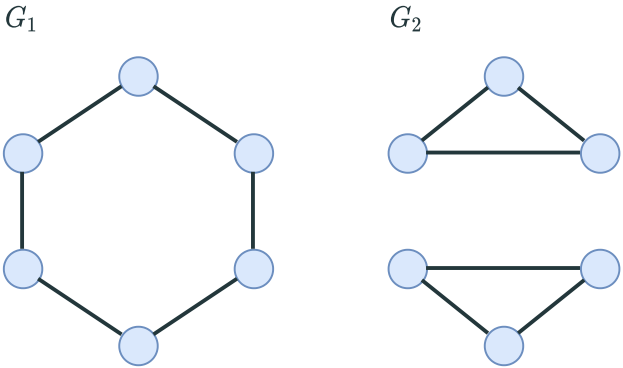
Solution: Message Passing

Message Passing: Extensions

Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

Message Passing: Extensions

Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.



Message Passing: Extensions

Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

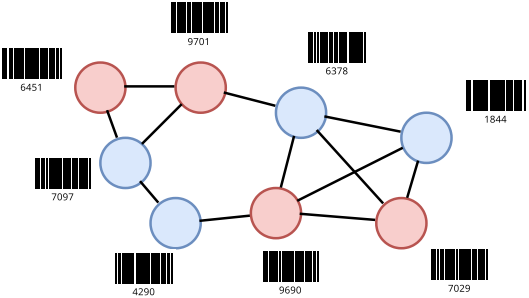
Many extensions have been proposed:

Message Passing: Extensions

Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

Many extensions have been proposed:

- Random Node IDs
  [Abboud et al., 2020]

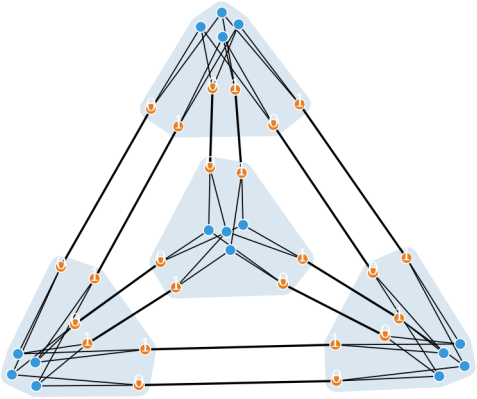


Message Passing: Extensions

Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

Many extensions have been proposed:

- Random Node IDs
  [Abboud et al., 2020]
- Higher Order GNNs
  [Morris et al., 2019, Maron et al., 2019]



Message Passing: Extensions

Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

Many extensions have been proposed:

- Random Node IDs
  [Abboud et al., 2020]
- Higher Order GNNs
  [Morris et al., 2019, Maron et al., 2019]
- Add motif counts as features
  [Bouritsas et al., 2020]

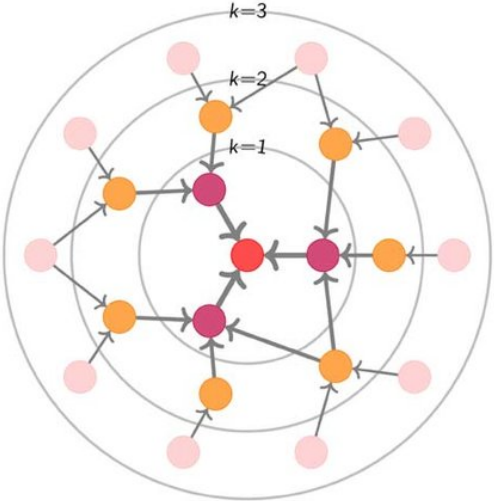
	Motif	Description
3-node		*2-star*
		triangle
4-node		*4-path*
		3-star
		4-tailedtriangle
		4-cycle
		4-chordalcycle
		4-clique
5-node		*5-clique*

Message Passing: Extensions

Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

Many extensions have been proposed:

- Random Node IDs
  [Abboud et al., 2020]
- Higher Order GNNs
  [Morris et al., 2019, Maron et al., 2019]
- Add motif counts as features
  [Bouritsas et al., 2020]
- Aggregate from K-Hop Neighbors
  [Brossard et al., 2020]

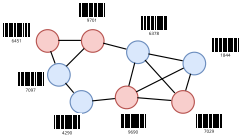


Message Passing: Extensions

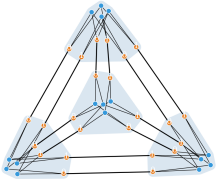
Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

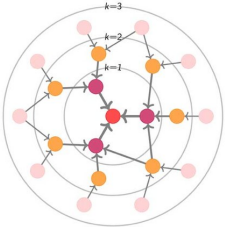
Many extensions have been proposed:

- Random Node IDs
  [Abboud et al., 2020]
- Higher Order GNNs
  [Morris et al., 2019, Maron et al., 2019]
- Add motif counts as features
  [Bouritsas et al., 2020]
- Aggregate from K-Hop Neighbors
  [Brossard et al., 2020]



	Motif	Description
3-node		*2-star*
		triangle
4-node		*4-path*
		3-star
		4-tailedtriangle
		4-cycle
		4-chordalcycle
		4-clique
5-node		*5-clique*



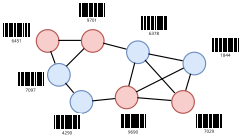


Message Passing: Extensions

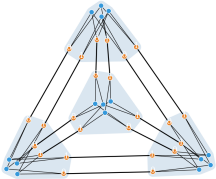
Equivalent to the 1-dimensional Weisfeiler-Leman algorithm.

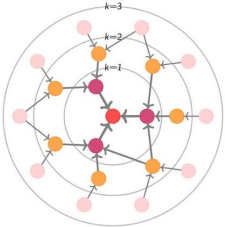
Many extensions have been proposed:

- Random Node IDs
  [Abboud et al., 2020]
- Higher Order GNNs
  [Morris et al., 2019, Maron et al., 2019]
- Add motif counts as features
  [Bouritsas et al., 2020]
- Aggregate from K-Hop Neighbors
  [Brossard et al., 2020]



	Motif	Description
3-node		*2-star*
		triangle
4-node		*4-path*
		3-star
		4-tailedtriangle
		4-cycle
		4-chordalcycle
		4-clique
5-node		*5-clique*





Are there alternatives?

Random Walks for Graph Learning

Random Walk Graph Kernel

- Compare graphs through common random walks
- based on the number of length $p \in [P]$ random walks

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}^p$$

Random Walk Graph Kernel

- Compare graphs through common random walks
- based on the number of length $p \in [P]$ random walks
- Various usecases:
  - Graph kernel, e.g. with SVM
  - Structural embedding, e.g. in Graph Transformers
  - As GNN, e.g. RWGNN [Nikolentzos and Vazirgiannis, 2020]

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}^p$$

Random Walk Graph Kernel

- Compare graphs through common random walks
- based on the number of length $p \in [P]$ random walks
- Various usecases:
  - Graph kernel, e.g. with SVM
  - Structural embedding, e.g. in Graph Transformers
  - As GNN, e.g. RWGNN [Nikolentzos and Vazirgiannis, 2020]

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}^p$$

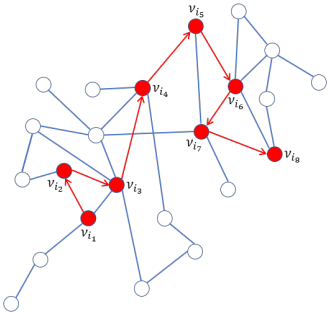
Intuition based on random walks, deterministic computation

Random Walk Based Node Embeddings

DeepWalk and Node2Vec

1. Sample random walks
2. Run Skip-Gram to get the embedding:
   - Positive: co-occuring nodes
   - Negative: other nodes

$\Longrightarrow$ Embedding is a random variable

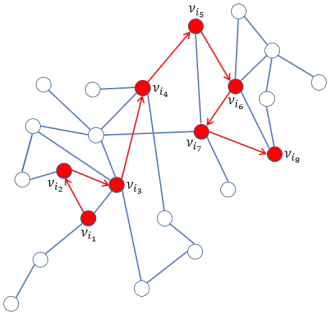


Random Walk Based Node Embeddings

DeepWalk and Node2Vec

1. Sample random walks
2. Run Skip-Gram to get the embedding:
   - Positive: co-occuring nodes
   - Negative: other nodes

$\implies$ Embedding is a random variable



Node2Vec just encodes proximity, can we use more structure?

CRAWL

CRAWL

CNNs for **Ra**ndom **W**a**l**ks

CRAWL

CNNs for **Ra**ndom **W**al**l**ks

- Sample random walks

CRAWL

CNNs for **Ra**ndom **Wal**ks

- Sample random walks
- Construct a feature matrix for each walk

CRAWL

CNNs for **Ra**ndom **Wal**ks

- Sample random walks
- Construct a feature matrix for each walk
- Process features with 1D-CNNs

CRAWL

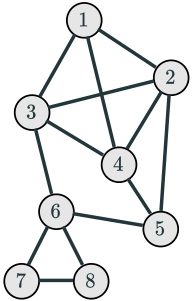
CNNs for **Ra**ndom **Wa**l**k**s

- Sample random walks
- Construct a feature matrix for each walk
- Process features with 1D-CNNs
- Use result to update latent node embeddings

CRAWL Layer

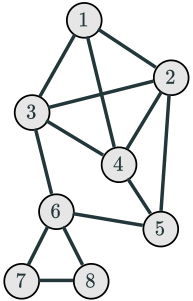
Input: Graph $G = (V, E)$ with node and edge features h, g

Hyperparameters: number m and length ℓ of walks, window size s of the CNN



CRAWL Layer

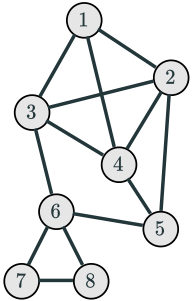
Step 1: Sample m random walks of length ℓ



CRAWL Layer

Step 1: Sample m random walks of length ℓ

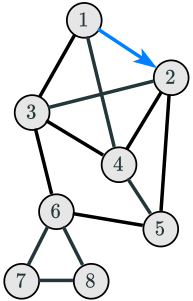
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

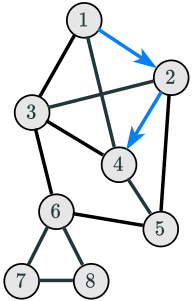
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

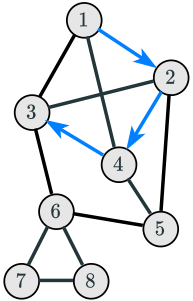
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

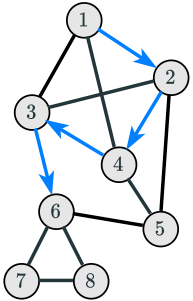
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

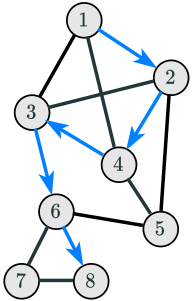
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

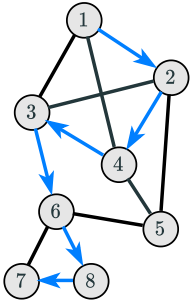
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

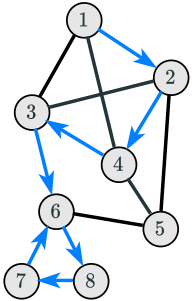
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

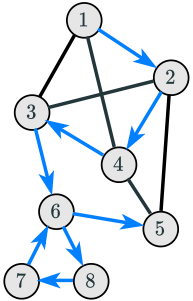
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

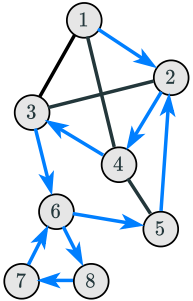
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

Step 1: Sample m random walks of length ℓ

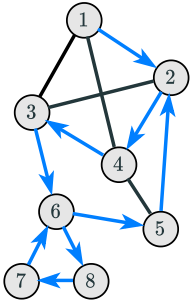
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



CRAWL Layer

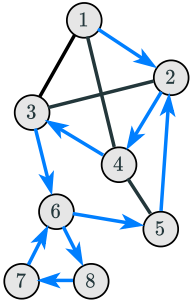
Step 2: Compute feature matrix for each walk with window size s

$$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$$



Step 2: Compute feature matrix for each walk with window size s

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$

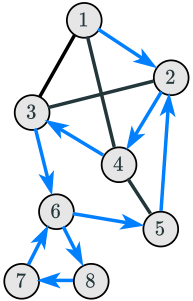


$$X(w_1, h, g, 4)$$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

Column 1: Node features

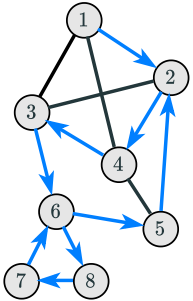
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$$h_w \quad \begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

Column 2: Edge features

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



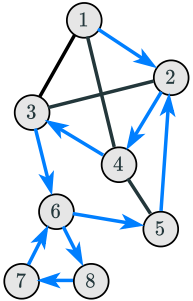
$$g_w$$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

Column 3: Binary identity features
(1 if current node and j-th predecessor are the same)

Within the window size

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$$I_w^s = \begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

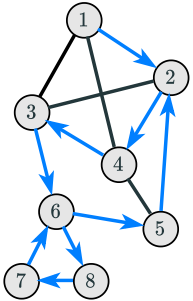
Column 4: Binary edge features

(1 of current node and $(j+1)$-th predecessor are connected)

Again within the window

We skip the first node (always 1)

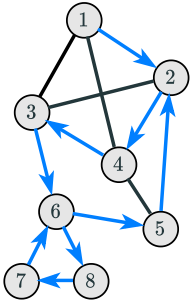
$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$$
\begin{array}{c}
A_w^s \\
\begin{pmatrix}
h(1) & 00000 & 0000 & 000 \\
h(2) & g(1,2) & 0000 & 000 \\
h(4) & g(2,4) & 0000 & 100 \\
h(3) & g(4,3) & 0000 & 110 \\
h(6) & g(3,6) & 0000 & 000 \\
h(8) & g(6,8) & 0000 & 000 \\
h(7) & g(8,7) & 0000 & 100 \\
h(6) & g(7,6) & 0010 & 100 \\
h(5) & g(6,5) & 0000 & 000 \\
h(2) & g(5,2) & 0000 & 000
\end{pmatrix}
\end{array}
$$

Step 2: The feature matrix fully describes induced subgraphs of size s

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$

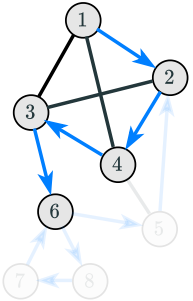


$$X(w_1, h, g, 4)$$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

Step 2: The feature matrix fully describes induced subgraphs of size s

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$

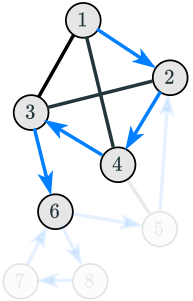


$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

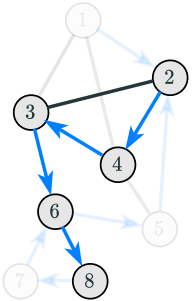
1D CNN

C

$$\begin{pmatrix} c_1 \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

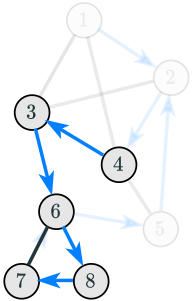
1D CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ \\ \\ \\ \\ \\ \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

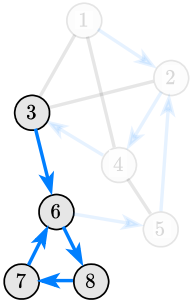
1D CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \\ \\ \\ \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

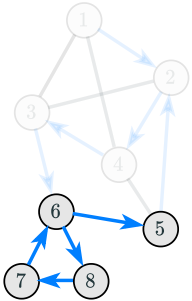
1D CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \\ \\ \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

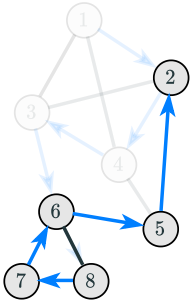
1D CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

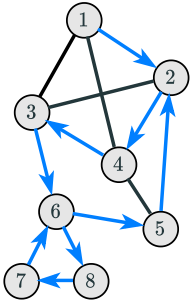
1D CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

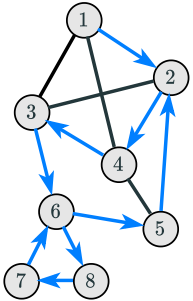
1D-CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}$$

Step 3: Process induced subgraphs with a 1D CNN

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

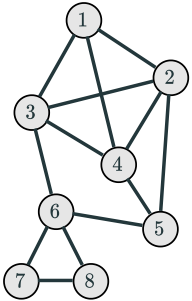
$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

1D-CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}$$

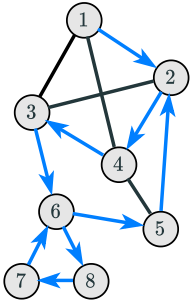
Pool & Update



Step 4: Pool the output

(Every node averages all c_i where it was the center of the walk)

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

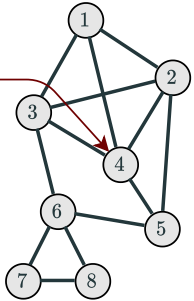
$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

1D-CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}$$

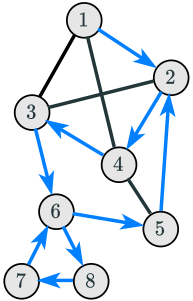
Pool & Update



Step 4: Pool the output

(Every node averages all c_i where it was the center of the walk)

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

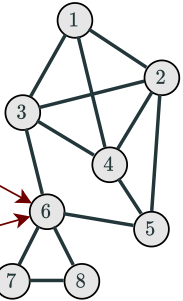
$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

1D-CNN

C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}$$

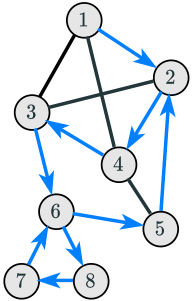
Pool & Update



Step 4: Pool the output

(Every node averages all c_i where it was the center of the walk)

$w_1 = (1, 2, 4, 3, 6, 8, 7, 6, 5, 2)$



$X(w_1, h, g, 4)$

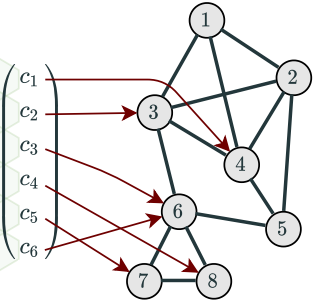
$$\begin{pmatrix} h(1) & 00000 & 0000 & 000 \\ h(2) & g(1,2) & 0000 & 000 \\ h(4) & g(2,4) & 0000 & 100 \\ h(3) & g(4,3) & 0000 & 110 \\ h(6) & g(3,6) & 0000 & 000 \\ h(8) & g(6,8) & 0000 & 000 \\ h(7) & g(8,7) & 0000 & 100 \\ h(6) & g(7,6) & 0010 & 100 \\ h(5) & g(6,5) & 0000 & 000 \\ h(2) & g(5,2) & 0000 & 000 \end{pmatrix}$$

1D-CNN

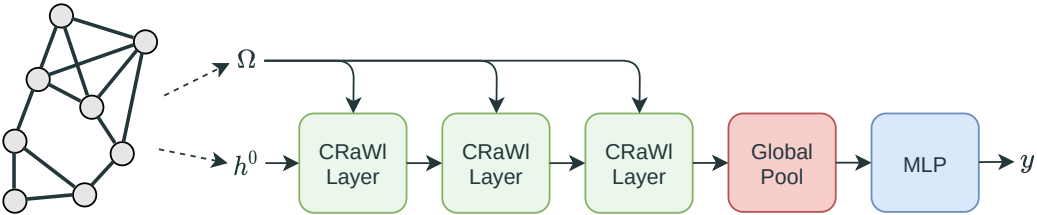
C

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix}$$

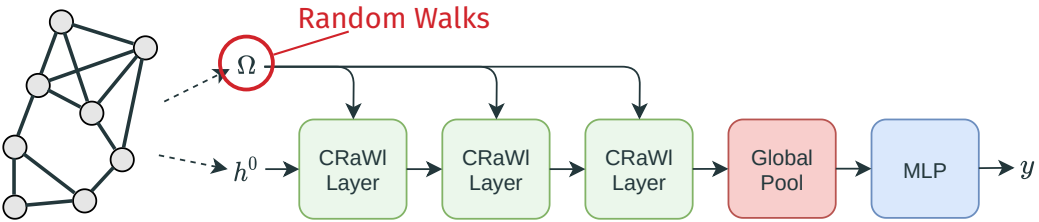
Pool & Update



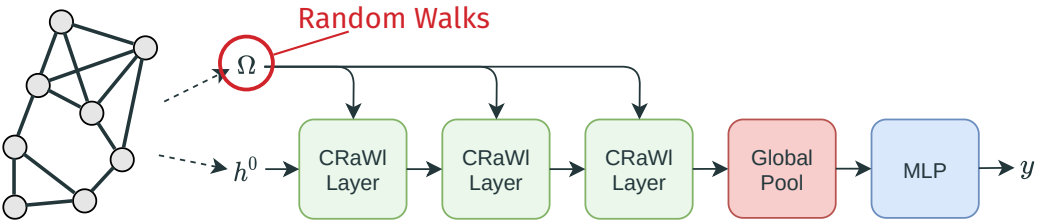
CRAWL Network



CRaWl Network



CRaWl Network



Every layer updates a node embedding

$\Longrightarrow$ Fully compatible with message passing and graph transformer layers

Experiments & Expressiveness

Experiments

Method	ZINC Test MAE ↓	MOLPCBA Test AP ↑	CSL Test Acc ↑ (%)
GIN [Xu et al., 2019]	0.526 ± 0.051	0.2703 ± 0.0023	10.0 ± 0.0
GCN [Kipf and Welling, 2017]	0.367 ± 0.011	0.2483 ± 0.0037	10.0 ± 0.0
3WLGNN [Maron et al., 2019]	0.303 ± 0.068	-	*95.7 ± 14.9*
PNA [Corso et al., 2020]	0.142 ± 0.010	0.2838 ± 0.0035	10.0 ± 0.0
PHC-GNN [Le et al., 2021]	0.164 ± 0.003	$\mathbf{0.2947 \pm 0.0026}$	-
GSN [Bouritsas et al., 2020]	0.108 ± 0.018	-	-
GINE+ [Brossard et al., 2020]	-	$\mathbf{0.2979 \pm 0.0030}$	-
GPS [Rampášek et al., 2022]	$\mathbf{0.070 \pm 0.004}$	0.2907 ± 0.0028	-
CRaWl	*0.085 ± 0.004*	$\mathbf{0.2986 \pm 0.0025}$	$\mathbf{100.0 \pm 0.0}$

Expressiveness

Considerations:

Expressiveness

Considerations:

- Output of CRAWL is a random variable.

Expressiveness

Considerations:

- Output of CRAWL is a random variable.
- CRAWL is permutation invariant. (Because random walks are, too.)

Expressiveness

Considerations:

- Output of CRAWL is a random variable.
- CRAWL is permutation invariant. (Because random walks are, too.)
- CRAWL can (theoretically) detect any substructure with up to s nodes.
  - Caveat: The feature matrix is not permutation invariant $\implies$ the CNN has to cope

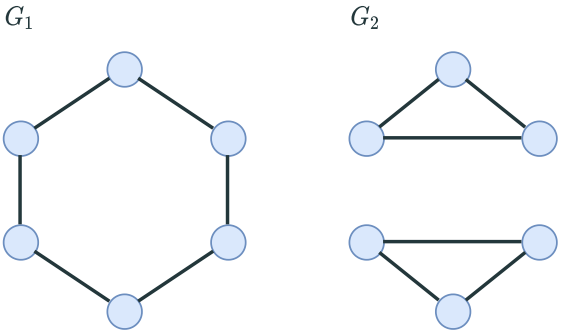
Expressiveness

Considerations:

- Output of CRAWL is a random variable.
- CRAWL is permutation invariant. (Because random walks are, too.)
- CRAWL can (theoretically) detect any substructure with up to s nodes.
  - Caveat: The feature matrix is not permutation invariant $\implies$ the CNN has to cope
- Substructures need to be traversed to be detected.

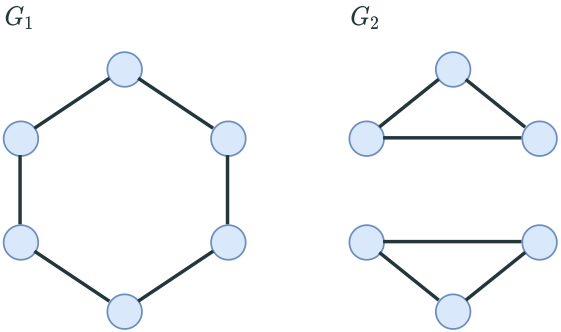
CRAWL vs MPGNNs

MPGNNs cannot distinguish the following graphs:



CRAWL vs MPGNNs

MPGNNs cannot distinguish the following graphs:



Distinguished by CRAWL with $s = 3$ and non-backtracking walks.

Expressiveness

Theorem (informal)

CRAWL sees things that even higher-order MPGNNs cannot.

Expressiveness

Theorem (formal)

For every $k \geq 1$ there are graphs of maximum degree 3 that are distinguishable by CRaWl with window size and walk length $O(k^2)$, but not by k-WL (and hence not by k-dimensional GNNs).

Expressiveness

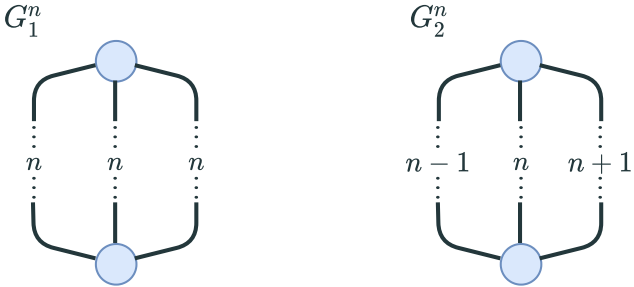
Theorem (formal)

For every $k \geq 1$ there are graphs of maximum degree 3 that are distinguishable by CRaWl with window size and walk length $O(k^2)$, but not by k-WL (and hence not by k-dimensional GNNs).

Those graphs are standard CFI graphs.

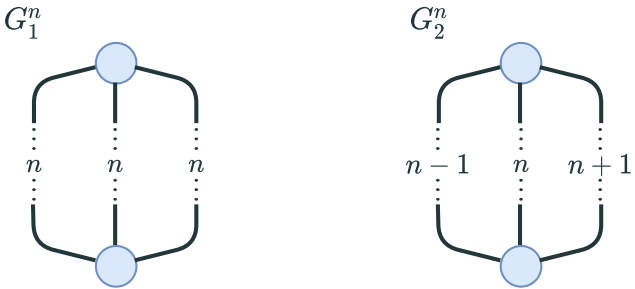
CRAWL vs MPGNNs

For the other direction:



CRAWL vs MPGNNs

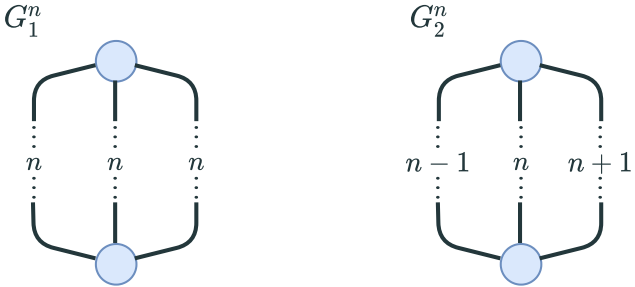
For the other direction:



Indistinguishable to CRAWL with $s = o(n)$.

CRAWL vs MPGNNs

For the other direction:



Indistinguishable to CRAWL with $s = o(n)$.

Distinguishable with 1-WL in $\mathcal{O}(n)$ steps.

(Small) Conclusion

Conclusion

CRAWL:

Conclusion

CRAWL:

- Distinguishes local structures (it can detect triangles!)

Conclusion

CRaWl:

- Distinguishes local structures (it can detect triangles!)
- Strong empirical performance (especially MOLPCBA)

Conclusion

CRaWl:

- Distinguishes local structures (it can detect triangles!)
- Strong empirical performance (especially MOLPCBA)
- Expressiveness perpendicular to WL-hierarchy

Conclusion

CRaWl:

- Distinguishes local structures (it can detect triangles!)
- Strong empirical performance (especially MOLPCBA)
- Expressiveness perpendicular to WL-hierarchy

Future Work:

- Node- & edge-level tasks

Conclusion

CRaWl:

- Distinguishes local structures (it can detect triangles!)
- Strong empirical performance (especially MOLPCBA)
- Expressiveness perpendicular to WL-hierarchy

Future Work:

- Node- & edge-level tasks
- Random walk strategies

Conclusion

CRaWl:

- Distinguishes local structures (it can detect triangles!)
- Strong empirical performance (especially MOLPCBA)
- Expressiveness perpendicular to WL-hierarchy

Future Work:

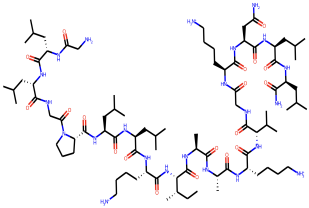
- Node- & edge-level tasks
- Random walk strategies
- Use transformers for processing the feature matrix

Excursion: Baseline Training

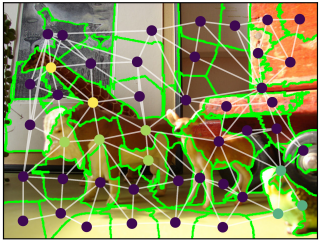
Excursion: Baseline Training

On the LRGB Dataset

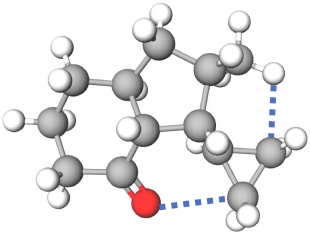
Long-Range Graph Benchmark



Peptides-Func
Peptides-Struct

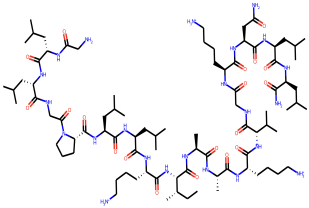


Superpixel
Pascal and COCO

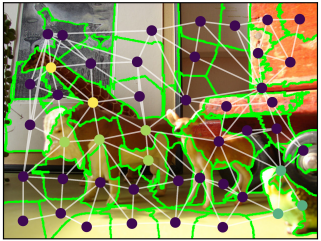


PCQM-Contact
(link prediction)

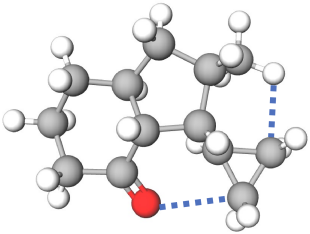
Long-Range Graph Benchmark



Peptides-Func
Peptides-Struct



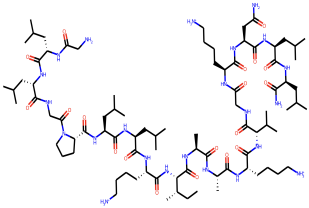
Superpixel
Pascal and COCO



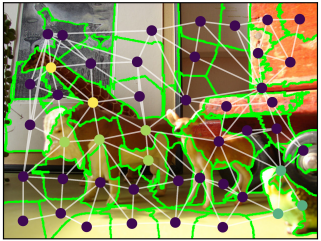
PCQM-Contact
(link prediction)

- Focus on long-range interaction
- Young dataset (from summer 2022), rapidly adopted

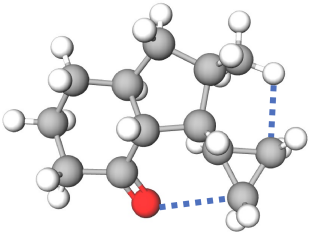
Long-Range Graph Benchmark



Peptides-Func
Peptides-Struct



Superpixel
Pascal and COCO

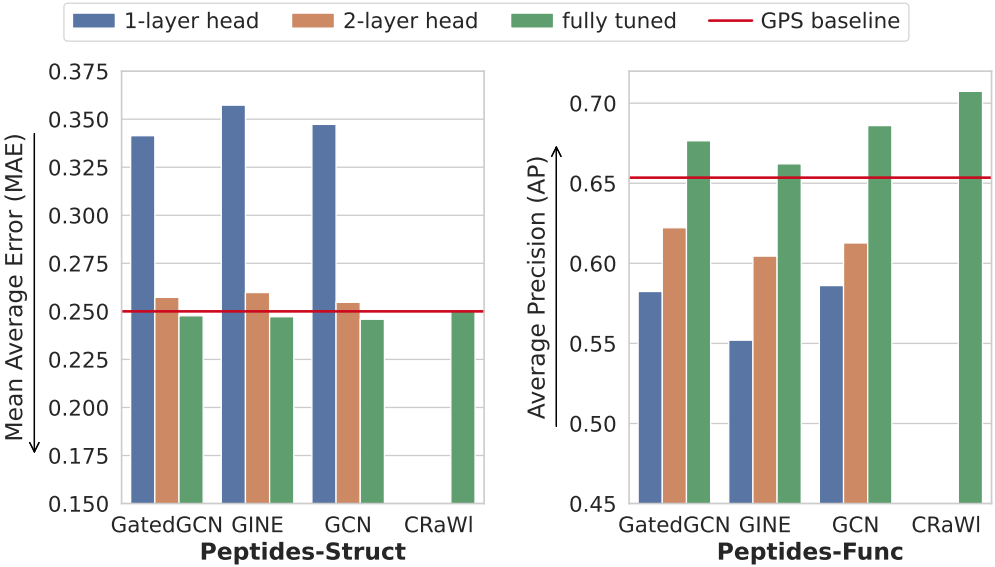


PCQM-Contact
(link prediction)

- Focus on long-range interaction
- Young dataset (from summer 2022), rapidly adopted

!! **But:**
Baselines have weak hyperparameters → MPGNNs can do much better !!

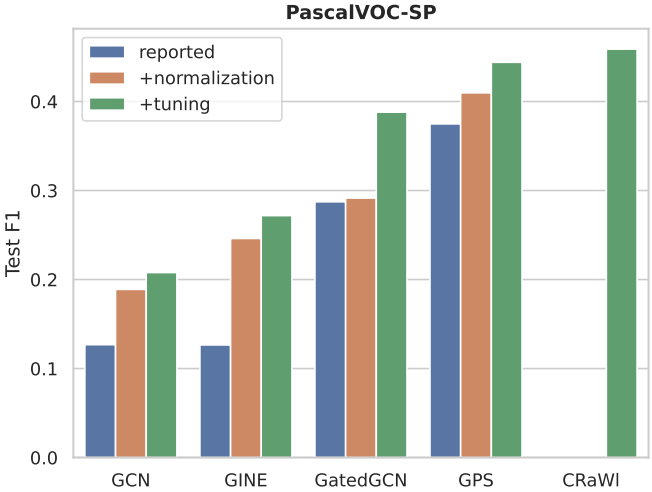
Baseline Tuning



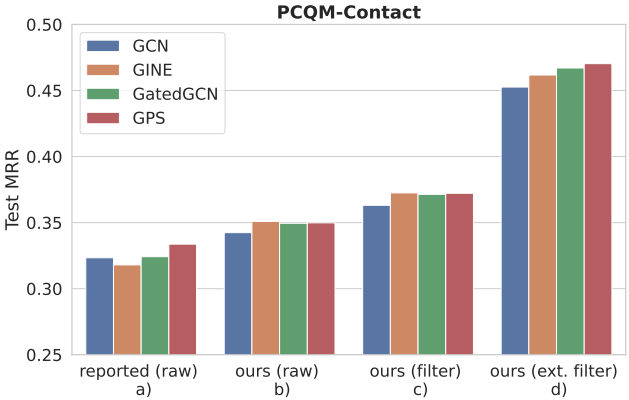
Feature Normalization

- Input features may have very different ranges
- Not all GNNs use internal normalization

⇒ Feature normalization



Link Prediction Metric: Filtering False Negatives



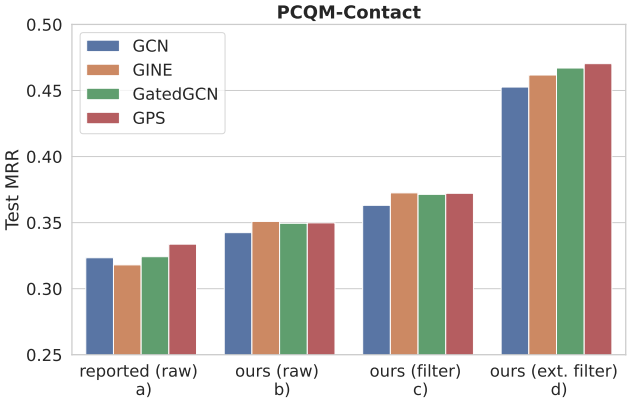
a) Implementation erroneously uses “raw MRR”

b) After hyperparameter tuning

c) Switching to “filtered MRR”

d) Filtering self-loops
   - Can never be true edges
   - Emphasized by dot-product

Link Prediction Metric: Filtering False Negatives



a) Implementation erroneously uses “raw MRR”

b) After hyperparameter tuning

c) Switching to “filtered MRR”

d) Filtering self-loops
   - Can never be true edges
   - Emphasized by dot-product

We argue that d) is the best metric for this task

Second Conclusion

CRaWl

- There are alternatives to standard MPGNNs (and Graph Transformers)
- Random walks are good at capturing local structure
  - And long-range interactions for extended window sizes ($\rightarrow$ LRGB)
- CRaWl layers are fully compatible with other layers (MPGNN, GT)

Second Conclusion

CRaWl

- There are alternatives to standard MPGNNs (and Graph Transformers)
- Random walks are good at capturing local structure
  - And long-range interactions for extended window sizes ($\rightarrow$ LRGB)
- CRaWl layers are fully compatible with other layers (MPGNN, GT)

LRGB

- Baseline tuning is vitally important (always!)
- Message passing is much better than reported
  - What does that tell us about long-range interactions or the LRGB datasets?
- Currently, the link prediction metric is broken

References

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. arXiv preprint arXiv:2010.01179, 2020.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI), pages 4602–4609, 2019.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. pages 2153–2164, 2019.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. arXiv preprint arXiv:2006.09252, 2020.

Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure. arXiv preprint arXiv:2011.15069, 2020.

Giannis Nikolentzos and Michalis Vazirgiannis. Random walk graph neural networks. Advances in Neural Information Processing Systems, 33:16211–16222, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In Proceedings of the Seventh International Conference on Learning Representations (ICLR), 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR), 2017.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. arXiv preprint arXiv:2004.05718, 2020.

Tuan Le, Marco Bertolini, Frank Noé, and Djork-Arné Clevert. Parameterized hypercomplex graph neural networks for graph classification. arXiv preprint arXiv:2103.16584, 2021.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. Advances in Neural Information Processing Systems, 35:14501–14515, 2022.

Random Walks

How many walks? Which length?

- Default: $m = |V|$. Start one walk at each vertex.
- Training: $\ell = 50$, Testing: $\ell = 100$

Walk Strategies:

- uniform
- non-backtracking
- Many more: pq-Walks,...

Which window size s?

- Default: $s = 8$

Hyperparameter Tuning