Graph Neural Networks II

Jan Tönshoff, Timo Gervens, Prof. Dr. Martin Grohe

RWTH Aachen

toenshoff@informatik.rwth-aachen.de
gervens@informatik.rwth-aachen.de

November 30, 2022

Message Passing

Input:

- Graph $G = (V, E)$
- Node Features $X_V : V \to \mathbb{R}^d$
- Edge Features $X_E : E \to \mathbb{R}^{d'}$ (optional)

Message Passing

Input:

- Graph $G = (V, E)$
- Node Features $X_V : V \to \mathbb{R}^d$
- Edge Features $X_E : E \to \mathbb{R}^{d'}$ (optional)

Idea of Message Passing:

- Initial embedding $h^0(v) = X_V(v)$.
- Update latent vertex embedding $h^\ell(v) \in \mathbb{R}^{d^\ell}$ in each layer ℓ.

Message Passing

Input:

- Graph $G = (V, E)$
- Node Features $X_V : V \to \mathbb{R}^d$
- Edge Features $X_E : E \to \mathbb{R}^{d'}$ (optional)

Idea of Message Passing:

- Initial embedding $h^0(v) = X_V(v)$.
- Update latent vertex embedding $h^\ell(v) \in \mathbb{R}^{d^\ell}$ in each layer ℓ.
- The update depends on the states of the neighbors.
- Use shared local neural network for the update.

Message Passing

Elements of a GNN Layer [Wu et al., 2020]:

- Trainable functions $\mathbf{M}^{\ell}$, $\mathbf{U}^{\ell}$
- Aggregation function $\bigoplus$ (Sum, Mean, Max,...)

Message Passing

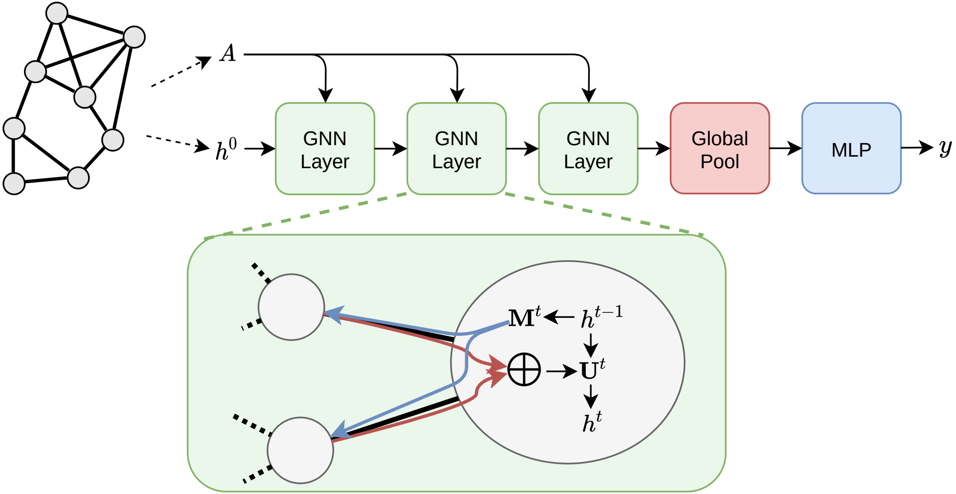
Elements of a GNN Layer [Wu et al., 2020]:

- Trainable functions $\mathbf{M}^\ell$, $\mathbf{U}^\ell$
- Aggregation function $\bigoplus$ (Sum, Mean, Max, ...)

Function computed by the layer:

$$h^\ell(v) = \mathbf{U}^\ell \left(h^{\ell-1}(v), \bigoplus_{u \in \mathcal{N}(v)} \mathbf{M}^\ell \left(h^{\ell-1}(u), X_E(vu) \right) \right)$$

Graph Neural Networks



Implementing Message Passing

How do you actually implement Message Passing in PyTorch?

Implementing Message Passing

How do you actually implement Message Passing in PyTorch?

Simple Solution: Dense Matrix Multiplication

- Easy to implement
- Highly optimized for small graphs

Implementing Message Passing

How do you actually implement Message Passing in PyTorch?

Simple Solution: Dense Matrix Multiplication

- Easy to implement
- Highly optimized for small graphs
- Does not scale to large graphs
- Requires padding

Implementing Message Passing

How do you actually implement Message Passing in PyTorch?

Simple Solution: Dense Matrix Multiplication

- Easy to implement
- Highly optimized for small graphs
- Does not scale to large graphs
- Requires padding
- Only linear aggregations
- No edge features

Implementing Message Passing

How do you actually implement Message Passing in PyTorch?

Simple Solution: Dense Matrix Multiplication

- Easy to implement
- Highly optimized for small graphs
- Does not scale to large graphs
- Requires padding
- Only linear aggregations
- No edge features

Better Solution: **Scatter Operations**

PyTorch Scatter

Open Source Torch Extension[1].

Compatible with CUDA and Backpropagation.

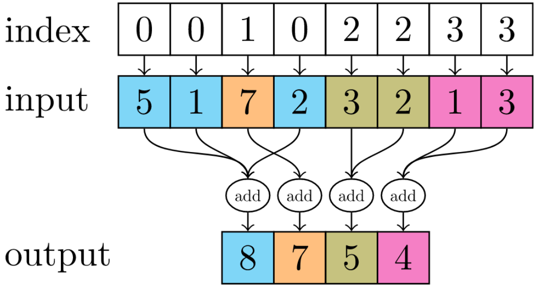
[1] https://github.com/rusty1s/pytorch_scatter

PyTorch Scatter

Open Source Torch Extension[1].

Compatible with CUDA and Backpropagation.

Enables aggregation of input with index list:



[1] https://github.com/rusty1s/pytorch_scatter

PyTorch Scatter Example

$$x = \begin{pmatrix} 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 3 & 5 \\ 0 & 1 \end{pmatrix}$$

$$\mathrm{idx} = [0, 1, 0, 2, 2]$$

PyTorch Scatter Example

$$x = \begin{pmatrix} 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 3 & 5 \\ 0 & 1 \end{pmatrix}$$

$$\mathrm{idx} = [0, 1, 0, 2, 2]$$

$$\texttt{scatter\_sum}(x, idx, \texttt{dim}{=}0) =$$

PyTorch Scatter Example

$$x = \begin{pmatrix} 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 3 & 5 \\ 0 & 1 \end{pmatrix}$$

$$\mathrm{idx} = [0, 1, 0, 2, 2]$$

$$\texttt{scatter\_sum}(x, idx, \texttt{dim}{=}0) = \begin{pmatrix} 1 & 3 \\ 1 & 0 \\ 3 & 6 \end{pmatrix}$$

PyTorch Scatter Example

$$x = \begin{pmatrix} 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 3 & 5 \\ 0 & 1 \end{pmatrix}$$

$$\mathrm{idx} = [0, 1, 0, 2, 2]$$

$$\texttt{scatter\_sum}(x, idx, \texttt{dim}{=}0) = \begin{pmatrix} 1 & 3 \\ 1 & 0 \\ 3 & 6 \end{pmatrix}$$

$$\texttt{scatter\_max}(x, idx, \texttt{dim}{=}0)[0] =$$

PyTorch Scatter Example

$$x = \begin{pmatrix} 0 & 2 \\ 1 & 0 \\ 1 & 1 \\ 3 & 5 \\ 0 & 1 \end{pmatrix}$$

$$\text{idx} = [0, 1, 0, 2, 2]$$

$$\texttt{scatter\_sum}(x, idx, \texttt{dim}{=}0) = \begin{pmatrix} 1 & 3 \\ 1 & 0 \\ 3 & 6 \end{pmatrix}$$

$$\texttt{scatter\_max}(x, idx, \texttt{dim}{=}0)[0] = \begin{pmatrix} 1 & 2 \\ 1 & 0 \\ 3 & 5 \end{pmatrix}$$

Sparse Graph Representation

Sparse representation of $G = (V, E)$:

Sparse Graph Representation

Sparse representation of $G = (V, E)$:

1. *Directed* edge list $\text{idx}_E \in \{0, \dots, |V| - 1\}^{2 \times 2|E|}$

Sparse Graph Representation

Sparse representation of $G = (V, E)$:

1. *Directed* edge list $\text{idx}_E \in \{0, \ldots, |V| - 1\}^{2 \times 2|E|}$
2. Node feature matrix $X_V \in \mathbb{R}^{|V| \times d}$
3. Optional edge feature matrix $X_E \in \mathbb{R}^{2|E| \times d'}$

Sparse Graph Representation

Sparse representation of $G = (V, E)$:

1. *Directed* edge list $\text{idx}_E \in \{0, \dots, |V| - 1\}^{2 \times 2|E|}$
2. Node feature matrix $X_V \in \mathbb{R}^{|V| \times d}$
3. Optional edge feature matrix $X_E \in \mathbb{R}^{2|E| \times d'}$
4. Graph label $y \in \mathbb{R}^c$

my guess
j'th column gives you source
and target of j'th edge
(u)
(v)

Message Passing with Scatter

Given: $H^{(\ell-1)} \in \mathbb{R}^{|V| \times d_h}, \quad X_E \in \mathbb{R}^{2|E| \times d'}, \quad \mathrm{idx}_E \in \{0, \ldots, |V|-1\}^{2 \times 2|E|}$

Compute:

Message Passing with Scatter

Given: $H^{(\ell-1)} \in \mathbb{R}^{|V| \times d_h}$, $\quad X_E \in \mathbb{R}^{2|E| \times d'}$, $\quad \mathrm{idx}_E \in \{0, \dots, |V|-1\}^{2 \times 2|E|}$

Compute:

$$Y^{(\ell)} = \mathbf{M}\left(H^{(\ell-1)}\left[\mathrm{idx}_E[0]\right], X_E\right)$$

Message Passing with Scatter

Given: $H^{(\ell-1)} \in \mathbb{R}^{|V| \times d_h}$, $\quad X_E \in \mathbb{R}^{2|E| \times d'}$, $\quad \mathrm{idx}_E \in \{0, \ldots, |V|-1\}^{2 \times 2|E|}$

Compute:

$$Y^{(\ell)} = \mathbf{M}\left(H^{(\ell-1)}\left[\mathrm{idx}_E[0]\right], X_E\right)$$

$$Z^{(\ell)} = \texttt{scatter\_sum}(Y^{(\ell)}, \mathrm{idx}_E[1], \mathrm{dim}{=}0)$$

Message Passing with Scatter

Given: $H^{(\ell-1)} \in \mathbb{R}^{|V| \times d_h}$, $X_E \in \mathbb{R}^{2|E| \times d'}$, $\text{idx}_E \in \{0, \ldots, |V|-1\}^{2 \times 2|E|}$

Compute:

$$Y^{(\ell)} = \mathbf{M}\left(H^{(\ell-1)}\left[\text{idx}_E[0]\right], X_E\right)$$

H[idxs[0]] - (2|E|, d_h)
H_Xe_ - (2|E|, d_h + d')

$$Z^{(\ell)} = \texttt{scatter\_sum}(Y^{(\ell)}, \text{idx}_E[1], \text{dim}{=}0)$$

$$H^{(\ell)} = \mathbf{U}\left(H^{(\ell-1)}, Z^{(\ell)}\right)$$

Batching Sparse Graphs

Given: Batch of training graphs $(\mathrm{idx}_{E_1}, X_{V_1}, X_{E_1}, y_1), \ldots, (\mathrm{idx}_{E_b}, X_{V_b}, X_{E_b}, y_b)$

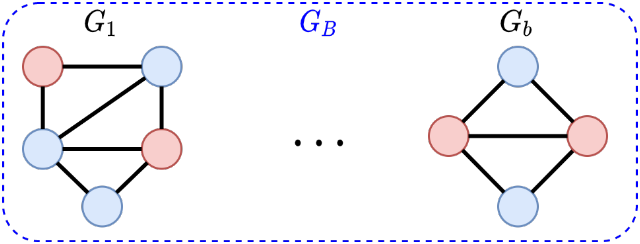
Batching Sparse Graphs

Given: Batch of training graphs $(\mathrm{idx}_{E_1}, X_{V_1}, X_{E_1}, y_1), \ldots, (\mathrm{idx}_{E_b}, X_{V_b}, X_{E_b}, y_b)$

How do we combine these into one input for parallel processing?

Batching Sparse Graphs

Given: Batch of training graphs $(\mathrm{idx}_{E_1}, X_{V_1}, X_{E_1}, y_1), \ldots, (\mathrm{idx}_{E_b}, X_{V_b}, X_{E_b}, y_b)$



How do we combine these into one input for parallel processing?

Compute disjoint union $G_B = (V_B, E_B)$: $V_B = \bigcup_{i=1}^{b} V_i$, $E_B = \bigcup_{i=1}^{b} E_i$

Batching Sparse Graphs

Representation of the batched graph G_B:

1. $\text{idx}_E \in \{0, \ldots, |V_B| - 1\}^{2\times 2|E_B|}$
2. $X_{V_B} \in \mathbb{R}^{|V_B|\times d}$
3. $X_{E_B} \in \mathbb{R}^{|E_B|\times d'}$
4. $y_B \in \mathbb{R}^{B\times c}$
5. $\text{batch\_idx} \in \{0, \ldots, b - 1\}^{|V_B|}$ (remember original graph for each node)

Batching Sparse Graphs

Representation of the batched graph G_B:

1. $\text{idx}_E \in \{0, \dots, |V_B| - 1\}^{2\times 2|E_B|}$
2. $X_{V_B} \in \mathbb{R}^{|V_B|\times d}$
3. $X_{E_B} \in \mathbb{R}^{|E_B|\times d'}$
4. $y_B \in \mathbb{R}^{B\times c}$
5. $\text{batch\_idx} \in \{0, \dots, b - 1\}^{|V_B|}$ (remember original graph for each node)

The indices in idx_{E_B} are re-enumerated.

Batching Sparse Graphs

Representation of the batched graph G_B:

1. $\mathrm{idx}_E \in \{0, \ldots, |V_B| - 1\}^{2 \times 2|E_B|}$
2. $X_{V_B} \in \mathbb{R}^{|V_B| \times d}$
3. $X_{E_B} \in \mathbb{R}^{|E_B| \times d'}$
4. $y_B \in \mathbb{R}^{B \times c}$
5. batch_idx $\in \{0, \ldots, b - 1\}^{|V_B|}$ (remember original graph for each node)

The indices in idx_{E_B} are re-enumerated.

Pooling becomes a scatter operation with batch_idx.

Batching Sparse Graphs

Representation of the batched graph G_B:

1. $\text{idx}_E \in \{0, \ldots, |V_B| - 1\}^{2 \times 2|E_B|}$
2. $X_{V_B} \in \mathbb{R}^{|V_B| \times d}$
3. $X_{E_B} \in \mathbb{R}^{|E_B| \times d'}$
4. $y_B \in \mathbb{R}^{B \times c}$
5. $\text{batch\_idx} \in \{0, \ldots, b - 1\}^{|V_B|}$ (remember original graph for each node)

The indices in idx_{E_B} are re-enumerated.

Pooling becomes a scatter operation with batch_idx.

Batching Sparse Graphs in PyTorch

Write custom collation function in Python:

- Input: $[(\mathrm{idx}_{E_1}, X_{V_1}, X_{E_1}, y_1), \ldots, (\mathrm{idx}_{E_b}, X_{V_b}, X_{E_b}, y_b)]$
- Output: $(\mathrm{idx}_{E_B}, X_{V_B}, X_{E_B}, y_B, \mathrm{batch\_idx})$
- Pass function to `DataLoader` as `collate_fn` parameter.

GNNs in Practice

Typical Hyperparameters:

- 3-10 layers
- Latent dimension $50 \leq d \leq 500$

GNNs in Practice

Typical Hyperparameters:

- 3-10 layers
- Latent dimension $50 \leq d \leq 500$

Most GNNs are relatively shallow and small (compared to CNNs, Transformers, etc.)

GNN Components

Update function $\mathbf{U}$:

- Usually MLP with one or two layers
- If $\mathbf{U}$ has two layer, the second layer is usually linear

GNN Components

Update function $\mathbf{U}$:

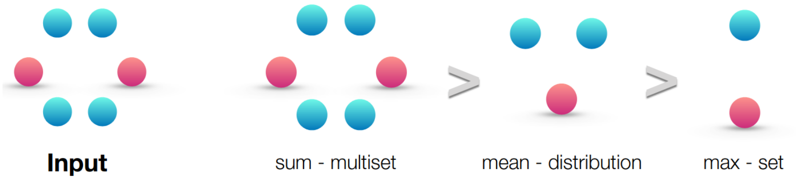
- Usually MLP with one or two layers
- If $\mathbf{U}$ has two layer, the second layer is usually linear

Message function $\mathbf{M}$:

- If no edge features are given, simply use identity (like the GCN)
- If edge features are given, $\mathbf{M}$ should be non-linear.

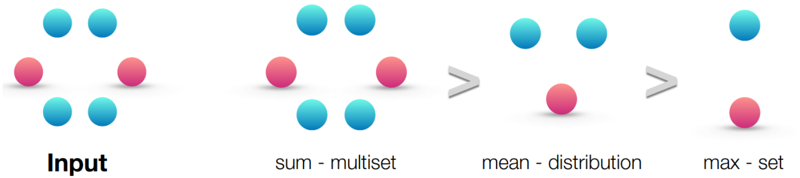
Aggregation

Expressiveness of aggregation functions [Xu et al., 2019]:



Aggregation

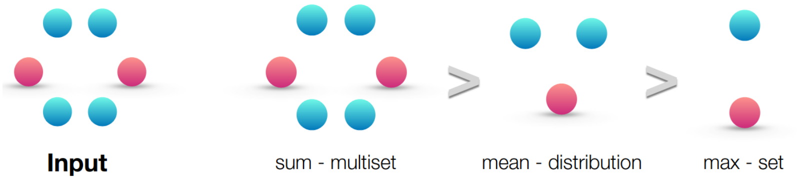
Expressiveness of aggregation functions [Xu et al., 2019]:



- SUM carries the most information but is not very robust.

Aggregation

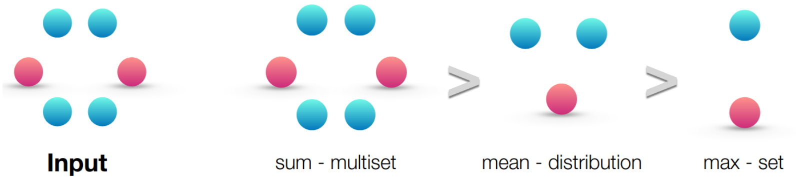
Expressiveness of aggregation functions [Xu et al., 2019]:



- SUM carries the most information but is not very robust.
- MEAN is more robust but you may need feature engineering.

Aggregation

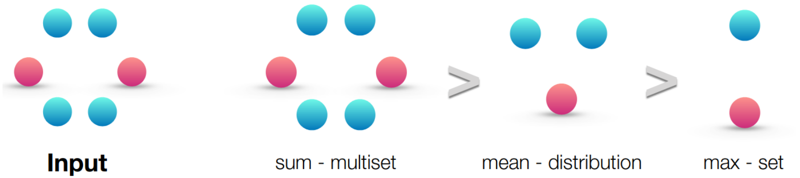
Expressiveness of aggregation functions [Xu et al., 2019]:



- SUM carries the most information but is not very robust.
- MEAN is more robust but you may need feature engineering.
- MAX carries the least information but can handle extreme ranges of degrees.

Aggregation

Expressiveness of aggregation functions [Xu et al., 2019]:



- SUM carries the most information but is not very robust.
- MEAN is more robust but you may need feature engineering.
- MAX carries the least information but can handle extreme ranges of degrees.

More advanced: Predict edge weights with attention. [Veličković et al., 2018]

Special GNNs

Recurrent GNNs:

- Choose $\mathbf{U}$ as GRU-Cell and reuse it in each message pass.
- Number of message passes not fixed.
- Application: Learning Heuristics for NP-hard problems.

Special GNNs

Recurrent GNNs:

- Choose **U** as GRU-Cell and reuse it in each message pass.
- Number of message passes not fixed.
- Application: Learning Heuristics for NP-hard problems.

Heterogeneous GNNs:

- Heterogeneous graphs contain nodes of different types.
- Train different functions **U**, **M** for each type of node.

Standard Tricks

Most of the standard tricks from Deep Learning also work for GNNs:

- Batch/Layer Normalization
- Dropout

Standard Tricks

Most of the standard tricks from Deep Learning also work for GNNs:

- Batch/Layer Normalization
- Dropout
- Use Residual Connections:

$$h^{\ell}(v) = h^{\ell-1}(v) + \mathbf{U}^{\ell}\left(h^{\ell-1}(v), \bigoplus_{u \in \mathcal{N}(v)} \mathbf{M}^{\ell}\left(h^{\ell-1}(u), X_E(vu)\right)\right)$$

Virtual Nodes

Perform global pooling after each GNN layer and pass result back to nodes:

$$h^\ell(G) = \mathbf{V}^\ell\Big(\sum_{v \in V} h^\ell(v)\Big)$$
$$\tilde{h}^\ell(v) = h^\ell(v) + h^\ell(G)$$

Here, $\mathbf{V}^\ell$ is a trainable MLP.

Virtual Nodes enable global information exchange after each layer.

Local Tasks on Massive Graphs

Dataset: One huge graph with billions of vertices.

Task: Node Classification with a GNN.

Local Tasks on Massive Graphs

Dataset: One huge graph with billions of vertices.

Task: Node Classification with a GNN.

Solution: Sample subgraph rooted in v. Only process this subgraph to predict $y(v)$.

Local Tasks on Massive Graphs

Dataset: One huge graph with billions of vertices.

Task: Node Classification with a GNN.

Solution: Sample subgraph rooted in v. Only process this subgraph to predict $y(v)$.

Sampling Algorithms:

- GraphSAGE [Hamilton et al., 2017]
- HGSampler [Hu et al., 2020]

Overcoming WL

Standard GNNs are at most as powerful as the (1-dimensional) Weisfeiler-Leman-Test.

Overcoming WL

Standard GNNs are at most as powerful as the (1-dimensional) Weisfeiler-Leman-Test.

Extensions:

- Higher order GNNs (equivalent to higher order WL-Test) [Morris et al., 2019]

Overcoming WL

Standard GNNs are at most as powerful as the (1-dimensional) Weisfeiler-Leman-Test.

Extensions:

- Higher order GNNs (equivalent to higher order WL-Test) [Morris et al., 2019]
- Random node features [Sato et al., 2020]

Overcoming WL

Standard GNNs are at most as powerful as the (1-dimensional) Weisfeiler-Leman-Test.

Extensions:

- Higher order GNNs (equivalent to higher order WL-Test) [Morris et al., 2019]
- Random node features [Sato et al., 2020]
- Pre-Compute substructures of interest [Bouritsas et al., 2020]

Overcoming WL

Standard GNNs are at most as powerful as the (1-dimensional) Weisfeiler-Leman-Test.

Extensions:

- Higher order GNNs (equivalent to higher order WL-Test) [Morris et al., 2019]
- Random node features [Sato et al., 2020]
- Pre-Compute substructures of interest [Bouritsas et al., 2020]

Is this even the correct measure of "power"?

Overcoming WL

Standard GNNs are at most as powerful as the (1-dimensional) Weisfeiler-Leman-Test.

Extensions:

- Higher order GNNs (equivalent to higher order WL-Test) [Morris et al., 2019]
- Random node features [Sato et al., 2020]
- Pre-Compute substructures of interest [Bouritsas et al., 2020]

Is this even the correct measure of "power"?

This is an active field of research.

Exercise 3

Implement a more general GNN layer in PyTorch:

- Implement custom collation function.

Exercise 3

Implement a more general GNN layer in PyTorch:

- Implement custom collation function.
- Implement GNN layer with scatter operations (SUM, MEAN and MAX).

Exercise 3

Implement a more general GNN layer in PyTorch:

- Implement custom collation function.
- Implement GNN layer with scatter operations (SUM, MEAN and MAX).
- Implement global pooling with scatter operations.

Exercise 3

Implement a more general GNN layer in PyTorch:

- Implement custom collation function.
- Implement GNN layer with scatter operations (SUM, MEAN and MAX).
- Implement global pooling with scatter operations.
- Implement a Virtual Node layer.

Exercise 3

Implement a more general GNN layer in PyTorch:

- Implement custom collation function.
- Implement GNN layer with scatter operations (SUM, MEAN and MAX).
- Implement global pooling with scatter operations.
- Implement a Virtual Node layer.
- Evaluate your implementation on the ZINC dataset.

References I

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 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.

Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

References II

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.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *CoRR*, abs/2002.03155, 2020.

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.