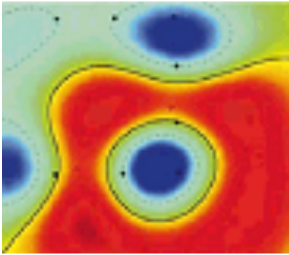
SoSe 2023

Deep Learning 2

Technische Universität Berlin



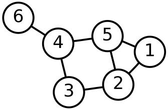
Lecture 5

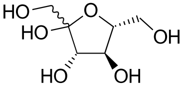
Graph Neural Networks

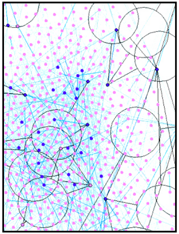
Graphs

Graphs can be used to represent

- information
  knowledge graphs, IMDB
- interactions
  social networks, phone calls, citations
- chemical compounds
- topology







Definition

A graph is a tuple $G = (V, E)$ with nodes V and edges E with

$$V = \{v_1, \ldots, v_N\}$$
$$E = \{(v, w) | (v, w) \in V^2\}.$$

Challenges of graph learning

- A graph is invariant to the order of nodes and edges.

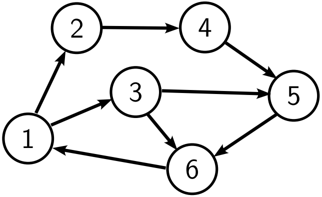
$$\{v_1, v_2, v_3\} = \{v_2, v_1, v_3\}$$

- An undirected graph can be represented using symmetric edges.

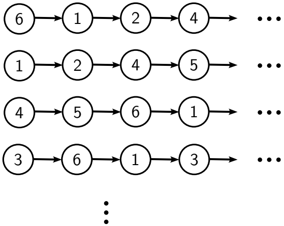
$$(v, w) \in E \implies (w, v) \in E$$

- Input graphs may have a varying number of nodes and edges.
- Labels may be attached to nodes and edges, e.g.
  - chemical elements and bond types in a molecule
  - profile information and type of connection in social networks

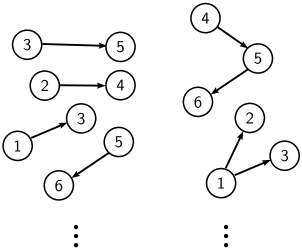
Classical graph features



Random Walks



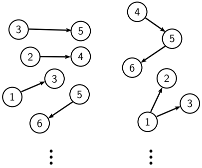
Subgraphs



Properties of classical graph features

Subgraphs

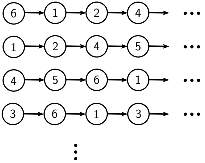
- number grows exponentially with size
- give a limited size, only local information is captured



Embedding into high-dimensional feature space for MLP

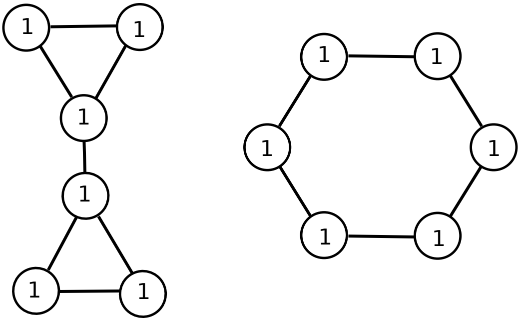
Random Walks

- sampling noise $\Longrightarrow$ many walks are required
- information about (non)locality of walks may be hard to recover



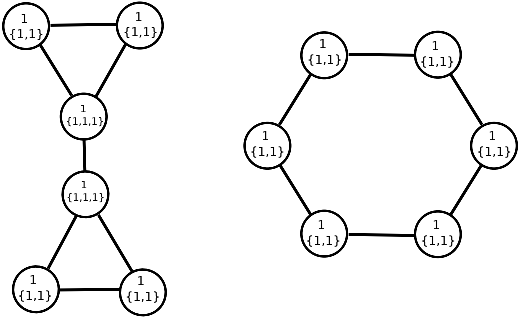
RNNs as graph neural networks

Weisfeiler-Lehman isomorphism test



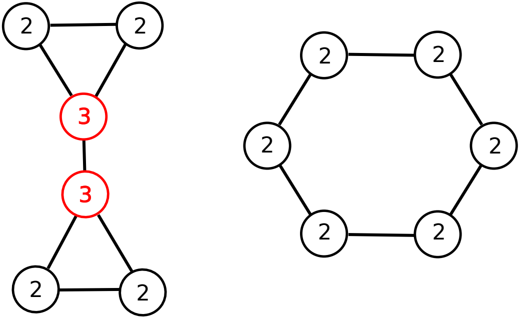
1. **If nodes are not labeled, assign the same label to each node.**

Weisfeiler-Lehman isomorphism test



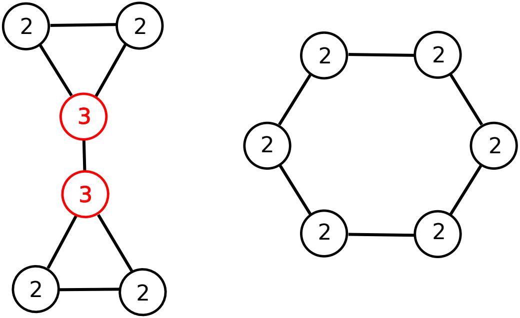
2. Relabel nodes with tuple of own label and sorted multiset of neighbor labels.

Weisfeiler-Lehman isomorphism test



3. **Compress labels using hash function (here: sequential id).**

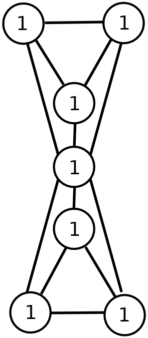
Weisfeiler-Lehman isomorphism test

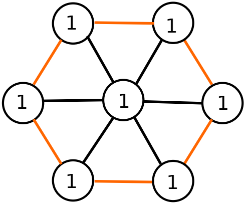


4. Check stop conditions

- If sorted labels of both graphs are not equal: Not isomorphic!
- Did label partition change (i.e. same nodes have identical labels)?
  - Yes → continue with step 2
  - No → isomorphic according to Weisfeiler-Lehman

Limitations of Weisfeiler-Lehman





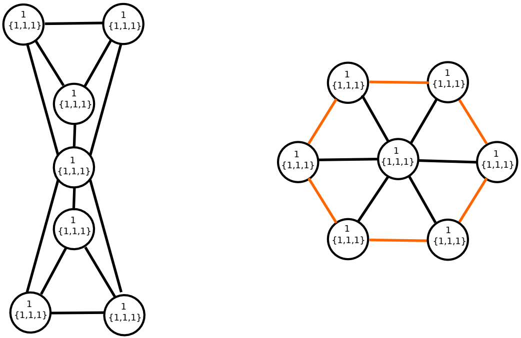
The subgraph marked in orange is not contained in the left graph

$\Longrightarrow$ **not isomorphic!**

Weisfeiler-Lehman wrongly shows isomorphism.

Weisfeiler-Lehman is necessary, but not sufficient condition!

Limitations of Weisfeiler-Lehman



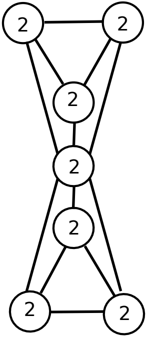
The subgraph marked in orange is not contained in the left graph

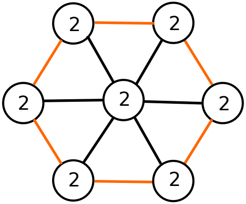
$\implies$ **not isomorphic!**

Weisfeiler-Lehman wrongly shows isomorphism.

Weisfeiler-Lehman is necessary, but not sufficient condition!

Limitations of Weisfeiler-Lehman

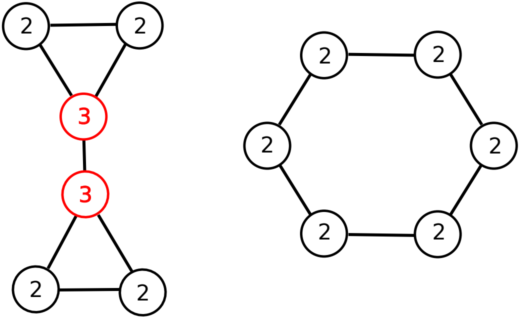




The subgraph marked in orange is not contained in the left graph
$\Longrightarrow$ **not isomorphic!**

Weisfeiler-Lehman wrongly shows isomorphism.
Weisfeiler-Lehman is necessary, but not sufficient condition!

Weisfeiler-Lehman generates graph features



Labels generated by WL can be used as signatures for a node environment

- efficient subgraph features
- Weisfeiler-Lehman graph kernels

Graph neural networks use similar approaches with learnable encodings

The graph neural network model Scarselli et al. [2008]

Similar to Weisfeiler-Lehman, nodes are updated using a message function

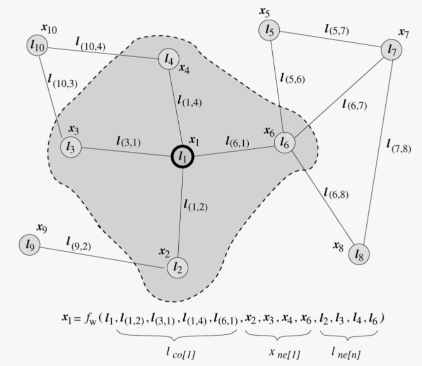
$$\mathbf{x}_n = f_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{\mathrm{co}[n]}, \mathbf{x}_{\mathrm{ne}[n]}, \mathbf{l}_{\mathrm{ne}[n]})$$
$$= \sum_{u \in \mathrm{ne}[n]} h_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{x}_u, \mathbf{l}_u)$$

A readout function produces an output for each node

$$\mathbf{o}_n = g_{\mathbf{w}}(\mathbf{x}_n, \mathbf{l}_n)$$

Compact notation of combined functions:

$$\mathbf{x} = F_{\mathbf{w}}(\mathbf{x}, \mathbf{l})$$
$$\mathbf{o} = G_{\mathbf{w}}(\mathbf{x}, \mathbf{l}_{\mathbf{N}})$$



→ **generalized, parametrizable Weisfeiler-Lehman**

Fixed-point iteration

The states of the nodes are computed recurrently until convergence to a fixed-point:

$$\mathbf{x}(t+1) = F_{\mathbf{w}}(\mathbf{x}(t), \mathbf{l}) \tag{1}$$

Banach fixed-point theorem

If $F_{\mathbf{w}}$ is a contraction map w.r.t. to $\mathbf{x}$, i.e.

$$\|F_{\mathbf{w}}(\mathbf{x}, \mathbf{l}) - F_{\mathbf{w}}(\mathbf{y}, \mathbf{l})\| \leq \mu \|\mathbf{x} - \mathbf{y}\|, \quad \exists 0 \leq \mu \leq 1,$$

then (1) converges to a unique fixed-point.

For general message functions, this can be achieved by Lipschitz regularization, i.e. adding a gradient penalty to the loss function:

$$\left\| \frac{\partial F_{\mathbf{w}}}{\partial \mathbf{x}} \right\|$$

Special case: Linear GNN

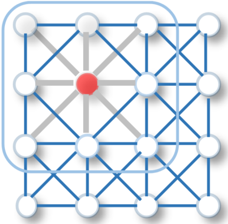
$$h_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{x}_u, \mathbf{l}_u) = \mathbf{A}_{n,u}\mathbf{x}_n + \rho_w(\mathbf{l}_n)$$
$$\mathbf{A}_{n,u} = \frac{\mu}{s|ne[u]|}\Psi_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{(n,u)}, \mathbf{l}_u) \in \mathbb{R}^{s\times s}$$

This transition function is a contraction map if the entries of Ψ are bounded to $[-1, 1]$, e.g. using a hyperbolic tangent activation function.

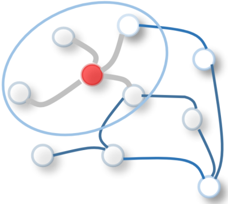
Proof: Given the block matrix A with blocks $\mathbf{A}_{n,u}$ for all neighbors n, u, the Lipschitz constant is bounded by

$$\left\|\frac{\partial F_{\mathbf{w}}}{\partial \mathbf{x}}\right\|_1 = \|\mathbf{A}\|_1 \leq \max_{u\in V}\left(\sum_{u\in \mathrm{ne}[u]} \|\mathbf{A}_{n,u}\|\right)$$
$$\leq \max_{u\in V}\left(\frac{\mu}{s|ne[u]|}\sum_{u\in \mathrm{ne}[u]} \|\mathbf{A}_{n,u}\|\right) \leq \mu$$

Graph Convolutional Networks



Convolution on grid



Graph convolution

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

- Instead of iterating until convergence, stack a couple of convolutional layers $\implies$ constraining of parameters is not required
- Convolution cannot directly be applied to graphs as there is no corresponding translation operator

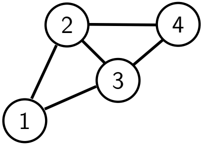
In the following slides, we will derive a convolution operator for graphs.

Image source: Wu et al. 2019, A Comprehensive Survey on Graph Neural Networks

Graph Fourier Transform

Given an adjacency matrix **A** and the degree matrix **D**, the normalized graph Laplacian is defined as $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

Example:



$$\mathbf{L} = \mathbf{I} - \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}^{-\frac{1}{2}} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}^{-\frac{1}{2}}$$

$$= \begin{pmatrix} 1 & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \\ -\frac{1}{\sqrt{6}} & 1 & -\frac{1}{3} & -\frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{6}} & -\frac{1}{3} & 1 & -\frac{1}{\sqrt{6}} \\ 0 & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 1 \end{pmatrix}$$

Spectral Graph Convolution Bruna et al. [2013], Kipf and Welling [2016]

Using the eigendecomposition of the graph Laplacian $\mathbf{L} = \mathbf{U}^T \Lambda \mathbf{U}$ with eigenvectors $\mathbf{U} = (\mathbf{u}_1, \dots, \mathbf{u}_N)$, we can define a graph Fourier transform for a function $f : V \to \mathbb{R}$ assigning values to each node:

$$\mathcal{GF}(f)(l) = \sum_{i=0}^{N-1} f(i) u_l(i)$$

A generalized convolution can be defined in the spectral domain using the convolution theorem:

$$(f * g)(i) = \mathcal{IGF}\left[\mathcal{GF}[f] \cdot \mathcal{GF}[g]\right](i)$$

With the convolution filter $g_\theta(\Lambda) = \mathrm{diag}(\theta)$ already defined in the Fourier domain, we arrive at

$$(g_\theta * \mathbf{x}) = U\, g_\theta(\Lambda)\, U^T \mathbf{x}$$

Fast approximations (1) Hammond et al. [2011], Defferrard et al. [2016]

One can approximate the non-parametric filter by a truncated expansion of Chebyshev polynomials

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k \left(\frac{2}{\Lambda_{\max}} \Lambda - \mathbb{1}_N \right)$$

$$T_k(x) = 2x T_{k-1}(x) - T_{k-2}(x), \quad T_0 = 1, T_1 = x$$

Since $(U \Lambda U^T)^k = (U \Lambda^k U^T)$, the full eigendecomposition is not required and the convolution can be written as:

$$g_\theta(\Lambda) * x = \sum_{k=0}^{K-1} \theta_k T_k \left(\frac{2}{\lambda_{\max}} \mathbf{L} - \mathbf{I}_N \right)$$

The filter is K-localized, i.e. nodes up to a distance of K edges influence a central node x_i.

Fast approximations (2) Kipf and Welling [2016]

Starting from the previous construction, set K=1, i.e. linear functions of the Laplacian, set $\theta = \theta_0 = -\theta_1$ and approximate $\lambda_{\text{max}} \approx 2$:

$$
\begin{aligned}
g_\theta * x &= \theta_0 x + \theta_1 (\mathbf{L} - \mathbf{I}_N) x \\
&= \theta \left(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) x
\end{aligned}
$$

Renormalization of the eigenvalues to range [-1, 1] to avoid vanishing/exploding gradients:

$$
\mathbf{I} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \to \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}
$$

$$
\text{with } \tilde{\mathbf{A}} = \mathbf{A} + I_N, \quad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}
$$

Finally, we obtain a **convolutional layer for a graph** signal $X \in \mathbb{R}^{N \times C}$ with C channel

$$
g_\theta * x = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} x \Theta,
$$

with filter matrix $\Theta \in \mathbb{R}^{C \times F}$ and output signal $Z \in \mathbb{R}^{N \times F}$.

Message-passing neural network Gilmer et al. [2017]

MPNNs have been proposed as a shared framework of previously described graph neural networks. A message function M describes the interaction between a pair of nodes, taking the node states $\mathbf{x}_i^t$ and edge states $\mathbf{e_{ij}}$ from teh previous layer t. The message $\mathbf{m}_i^{t+1}$ received by a node is aggregated from these:

$$\mathbf{m}_i^{t+1} = \sum_{j \in \mathcal{N}(i)} \mathbf{M}_t(\mathbf{x}_i^t, \mathbf{x}_j^t, \mathbf{e_{ij}})$$

Then, the states of nodes are updated using an update function:

$$\mathbf{x}_i^{t+1} = \mathbf{U}_t(\mathbf{x}_i^t, \mathbf{m}_i^{t+1})$$

This procedure is similar to the GNN by Scarselli et al. [2008], only that the updates are not iterated until convergence, but for a fixed number of layers. The specific choice of message and update function allows to fit many different GNNs in this

Examples for GNNs in the MPNN framework

GCN, Kipf and Welling [2016]

$$\mathbf{M}_t(\mathbf{x}_i^t, \mathbf{x}_j^t, \mathbf{e_{ij}}) = (\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}})_{ij} \mathbf{x}_j$$
$$\mathbf{U}_t(\mathbf{x}_i^t, \mathbf{m}_i^{t+1}) = \mathrm{ReLU}(\mathbf{m}_i \Theta)$$

Gated Graph Neural Networks, Li et al. [2015]

$$\mathbf{M}_t(\mathbf{x}_i^t, \mathbf{x}_j^t, \mathbf{e_{ij}}) = \mathbf{A}_{e_{ij}} \mathbf{x}_j^t$$
$$\mathbf{U}_t(\mathbf{x}_i^t, \mathbf{m}_i^{t+1}) = \mathrm{GRU}(\mathbf{x}_i^t, \mathbf{m}_i^t)$$

Deep Tensor Neural Networks, Schütt et al. [2017]

$$\mathbf{M}_t(\mathbf{x}_i^t, \mathbf{x}_j^t, \mathbf{e_{ij}}) = \tanh\left(\mathbf{W}^{fc}\left(\mathbf{W}^{cf}\mathbf{x}_j^t + b_1\right) \circ \left(\mathbf{W}^{df}\mathbf{e}_{ij} + b_2\right)\right)$$
$$\mathbf{U}_t(\mathbf{x}_i^t, \mathbf{m}_i^{t+1}) = \mathbf{x}_i^t + \mathbf{m}_i^t$$

Readout functions

Finally, the node representation has to be transformed to the prediction. Depending on the task, this could be node-wise or graph-wise. In the two former case, one can simply apply an output network $y_i = f(\mathbf{x}_i)$.
For graph-wise prediction, a permutationally invariant pooling is required:

$$y = f(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\})$$

Common options are sum and average pooling:

$$y = \sum_{i=1}^{N} f(\mathbf{x}_i)$$

$$y = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i)$$

Summary

- graphs are a natural way to structure data
- RNN and CNNs have corresponding variants for graphs
- MPNNs build a family of approaches are reminiscent of the Weisfeiler-Lehman isomorphism test
- Further topics
  - graph attention and transformers
  - graph embeddings and unsupervised learning

Bibliography

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations*, 2016.

Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8:13890, 2017.