Machine learning on graphs

Vlad Gladkikh

IBS CMCM

Nodes on graphs are connected => nodes are not independent and identically distributed.

=> traditional ML techniques cannot be directly applied to graphs.

Traditional machine learning has been designed for regular structured data such as images and sequences.

Graphs are irregular: nodes in a graph are unordered and can have distinct neighborhoods

https://towardsdatascience.com/top-10-learning-resources-for-graph-neural-networks-f24d4eb2cc2b

https://docs.dgl.ai/index.html https://github.com/jermainewang/dgl

https://analyticsindiamag.com/now-you-can-build-graph-neural-networks-with-spektral-based-on-keras/

https://ai.stackexchange.com/questions/12712/are-there-neural-networks-that-accept-graphs-or-trees-as-inputs

https://datascience.stackexchange.com/questions/45459/how-to-use-scikit-learn-label-propagation-on-graph-structured-data

https://neo4j.com/blog/announcing-graph-native-machine-learning-in-neo4j/

https://medium.com/stellargraph/faster-machine-learning-on-larger-graphs-how-numpy-and-pandas-slashed-memory-and-time-in-79b6c63870ef

https://www.udemy.com/course/introduction-to-graph-theory-and-complex-networks-analysis/

https://graph-tool.skewed.de/ https://gephi.org/ https://igraph.org/

https://tkipf.github.io/graph-convolutional-networks/

https://networkx.github.io/

https://www.analyticsvidhya.com/blog/2018/09/introduction-graph-theory-applications-python/

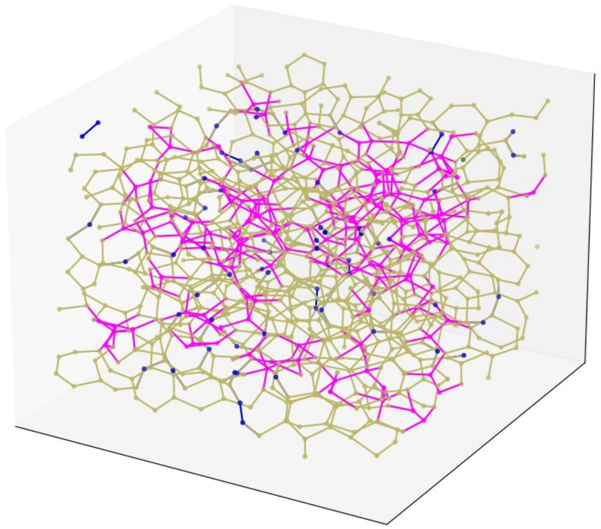
http://www.networksciencebook.com/

Typically, machine learning methods cannot be directly applied to molecular data as the molecule can be of arbitrary size and shape.

Hence, the prediction procedure usually consists of two stages:

1) feature extraction

Extracting molecule fingerprint, a vector representation encoding the molecule.



2) property prediction

Prediction using the extracted fingerprint as input.

Graph neural networks can learn the molecular fingerprints.

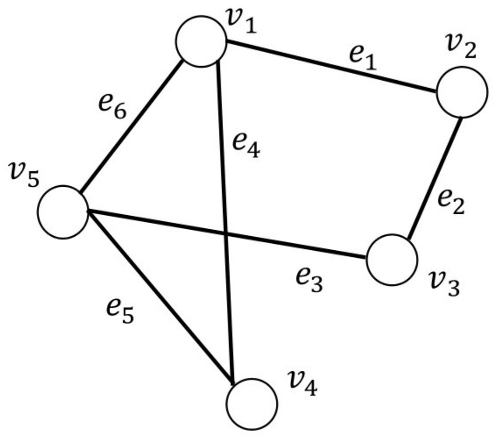
A molecule can be represented as a graph where nodes are the atoms and edges represent the bonds between these atoms.

Thus, the task of molecular property prediction can be regarded as graph classification or graph regression, which requires to learn graph-level representations.

The graph neural network model for this task consists of both graph filtering and graph pooling layers

https://towardsdatascience.com/geometric-ml-becomes-real-in-fundamental-sciences-3b0d109883b5

Graphs describe pairwise relations between entities.



Adjacency matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Degree of a node $d(v_i) = \sum_{j=1}^{N} A_{i,j}$

$$\sum_{v_i \in V} d(v_i) = 2|E| = vol(G)$$

Degree matrix

$$D = diag(d(v_1), \ldots, d(v_N))$$

symmetric $<$ Laplacian matrix: $L = D - A$

Normalized Laplacian matrix

$$L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

$L^k_{i,j} = 0$ if $dis(v_i, v_j) > k$

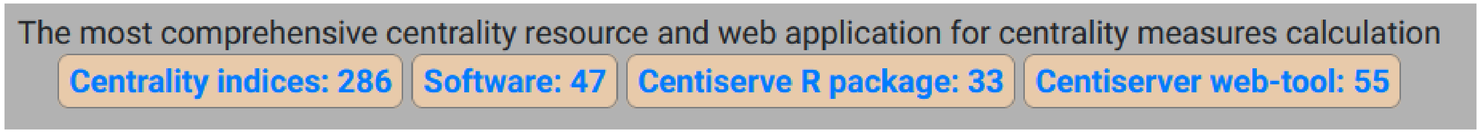
The diameter of a graph is defined as the length of the longest shortest path in the graph.

https://cse.msu.edu/~mayao4/dlg_book/

KDD 2020: Lecture Style Tutorials: Deep Graph Learning Foundations, Advances and Applications
https://www.youtube.com/watch?v=cK1hSxrf01o

Centrality https://www.centiserver.org/

The centrality of a node measures the importance of the node in the graph.


The most comprehensive centrality resource and web application for centrality measures calculation

Centrality indices: 286 | Software: 47 | Centiserve R package: 33 | Centiserver web-tool: 55

LightGraphs.jl

Betweenness centrality
Closeness centrality
Degree centrality
Eigenvector centrality
Katz centrality
Page rank
Radiality centrality
Stress centrality

f – a vector where f_i is associated with node v_i

$$h = Lf = Df - Af$$

$$h_i = d(v_i) f_i - \sum_{j=1}^{N} A_{i,j} f_j = \sum_{v_j \in N(v_i)} (f_i - f_j)$$

$h = Lf$ is the summation of the differences of f between node v_i and its neighbors $v_j \in N(v_i)$

$$f^T Lf = \sum_{v_i \in V} f_i \sum_{v_j \in N(v_i)} (f_i - f_j) = \sum_{v_i \in V} \sum_{v_j \in N(v_i)} (f_i f_i - f_i f_j)$$

$$= \sum_{v_i \in V} \sum_{v_j \in N(v_i)} (\frac{1}{2} f_i f_i - f_i f_j + \frac{1}{2} f_j f_j) = \frac{1}{2} \sum_{v_i \in V} \sum_{v_j \in N(v_i)} (f_i - f_j)^2$$

$f^T Lf$ is the sum of the squares of the differences of f between adjacent nodes

$f^T Lf \geq 0 \Rightarrow L$ is positive semi-definite. The eigenvalues of L are non-negative

$$u_1 = \frac{1}{\sqrt{N}}(1, \ldots, 1)$$

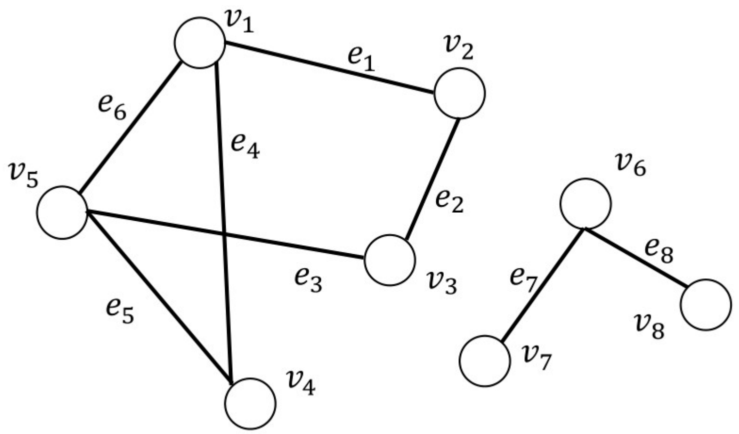
$$Lu_1 = \sum_{v_j \in N(v_i)} (1-1) = 0 = 0 \cdot u_1$$

Eigenvalues of L: $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$

Eigenvectors of L: $v_1, v_2, \ldots v_N$

The number of 0 eigenvalues of a graph's Laplacian matrix equals to the number of connected components in the graph.

A graph with two connected components:



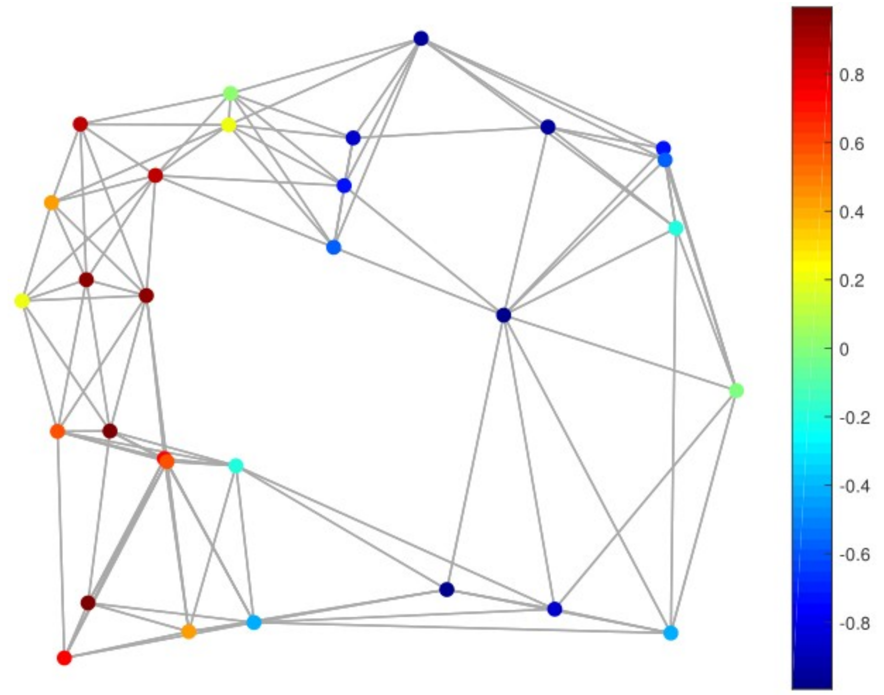
Graph signals: features or attributes associated with nodes

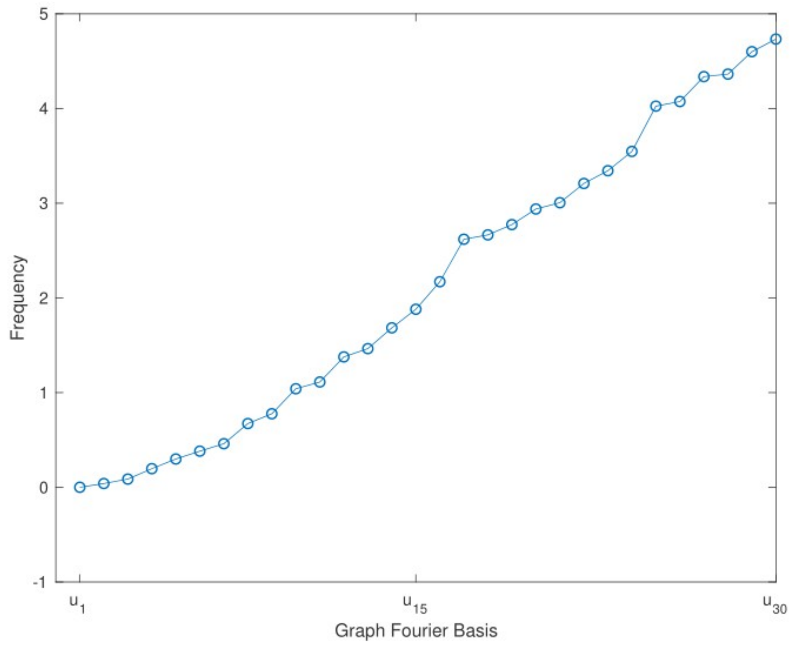
A graph is smooth if the values in connected nodes are similar.

low frequency, as the values change slowly

$f^T L f$ can be used to measure the smoothness (or the frequency) of a graph signal

A one-dimensional graph signal:





The Graph Fourier Transform for a graph signal f

$$\hat{f}_l = \{f, u_l\} = \sum_{i=1}^{N} f_i u_{l,i} \qquad \hat{f} = U^T f$$

$$L u_l = \lambda_l u_l \qquad u_l^T L u_l = \lambda_l u_l^T u_l = \lambda_l$$

λ_l represents the frequency or the smoothness of the eigenvector u_l

The eigenvectors associated with small eigenvalues vary slowly across the graph.

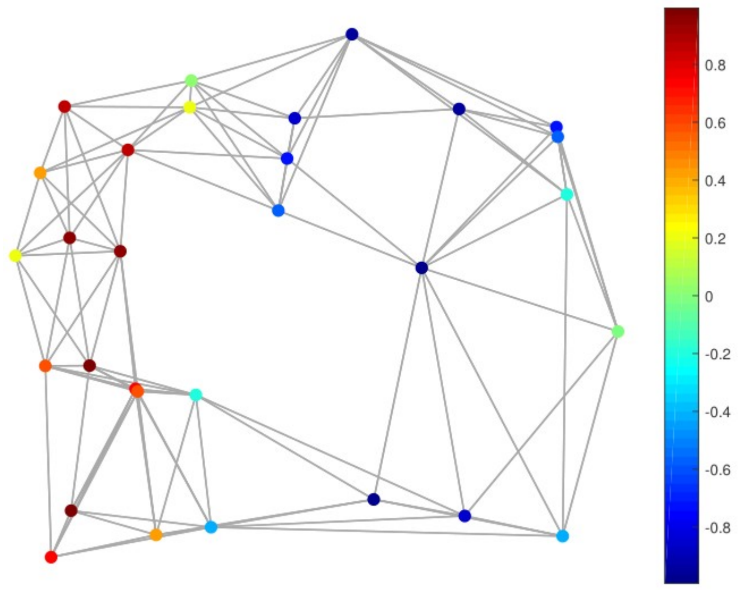
The values of such eigenvector at connected nodes are similar.

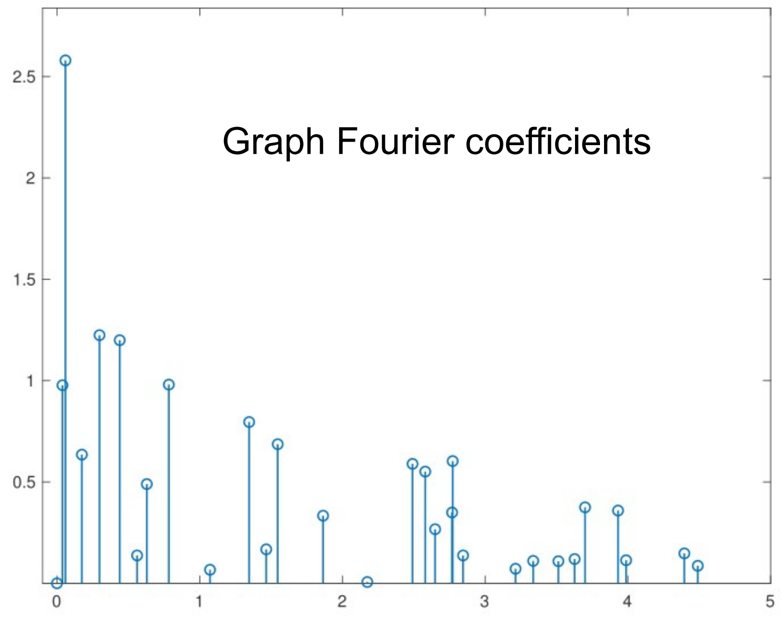
An extreme example is the first eigenvector associated with the eigenvalue 0 – it is constant over all the nodes.

The eigenvectors corresponding to large eigenvalues may have very different values on two nodes, even if connected.

The inverse graph Fourier transform: $f_i = \langle \hat{f}, u_i \rangle = \sum_{l=1}^{N} \hat{f}_l u_{i,l}$ $\qquad f = U \hat{f}$

A graph signal can be denoted in two domains, i.e., the spatial domain and the spectral domain.





Computational Tasks on Graphs

Node-focused tasks

- Node classification
- Node ranking
- Link prediction
- Community detection

Graph-focused tasks

- Graph classification
- Graph matching
- Graph generation
- Graph structure learning

Graph embedding

– mapping each node in a given graph into a vector representation (aka node embedding) that preserves some key information of the node in the original graph.

Key questions

1) what information to preserve?

    node's neighborhood information

    node's structural role

    node status

    community information

2) how to preserve this information?

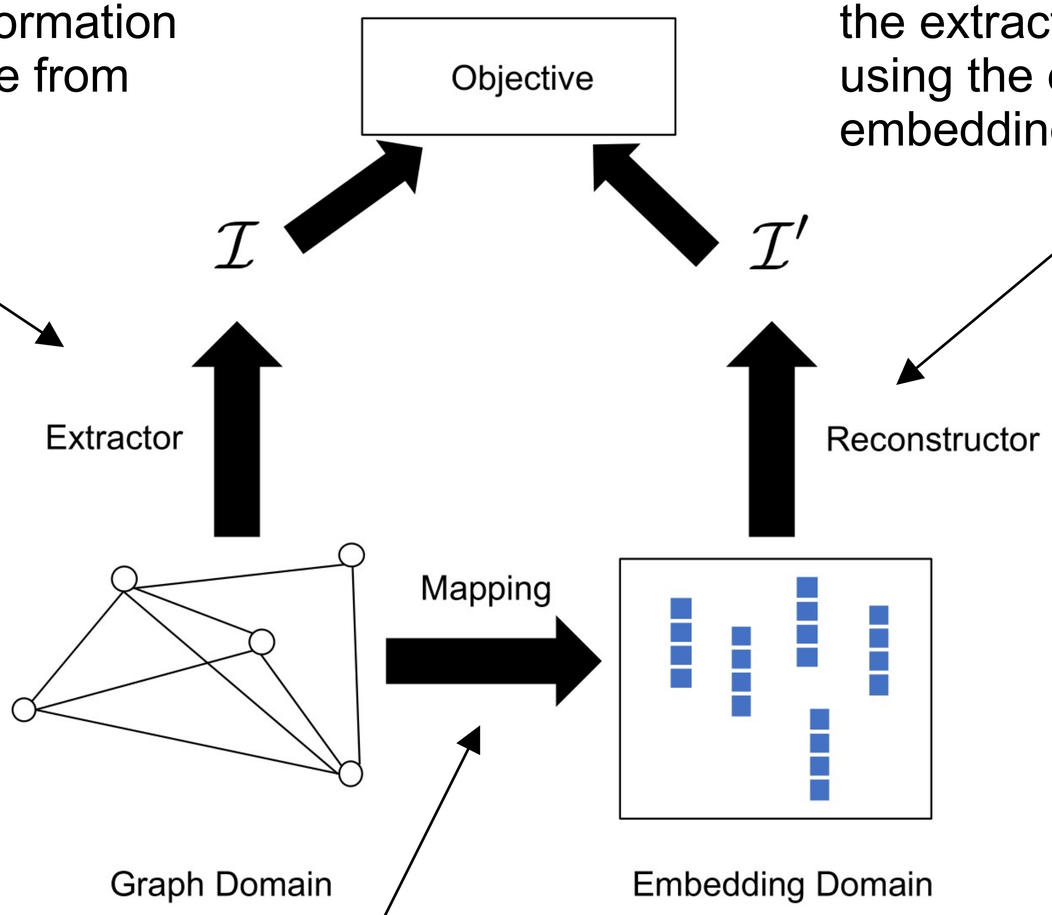
Node representations should be able to reconstruct the information we desire to preserve.

The mapping can be learned by minimizing the reconstruction error.

The objective is optimized to learn all parameters involved in the mapping and/or reconstructor.

The information extractor extracts the key information we want to preserve from the graph domain.

The reconstructor constructs the extracted graph information using the embeddings from the embedding domain.



The mapping function maps the node from the graph domain to the embedding domain.

Mapping function $f(v_i) = u_i = e_i^T W$

$e_i \in \{0,1\}^N$ – one-hot encoding of the node v_i

$W^{N \times d}$ – the embedding parameters to be learned

d – the dimension of the embedding

The *i*-th row of W denotes the embedding of node v_i

Preserving node co-occurrence

e.g. via performing random walks

Nodes are considered similar to each other if they tend to co-occur in a random walk.

The mapping function is optimized so that the learned node representations can reconstruct the similarity extracted from random walks.

$$p\left(v^{(t+1)} \middle| v^{(t)}\right) = \begin{cases} \frac{1}{d\left(v^{(t)}\right)}, & if\ v^{(t+1)} \in N\left(v^{(t)}\right) \\ 0, & otherwise \end{cases}$$

To generate random walks that can capture the information of the entire graph, each node is considered as a starting node to generate γ random walks

$N \cdot \gamma$ random walks, each of the length T

These random walks can be treated as sentences in an artificial language where the set of nodes is its vocabulary.

For a given center node, all its context nodes are treated either equally, regardless of the distance between them or differently according to their distance to the center node.

https://www.analyticsvidhya.com/blog/2019/11/graph-feature-extraction-deepwalk/

https://medium.com/analytics-vidhya/an-intuitive-explanation-of-deepwalk-84177f7f2b72

Perozzi, Bryan, Al-Rfou, Rami, and Skiena, Steven. 2014. Deepwalk: Online learning of social representations. Pages 701–710 of: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.

Grover, Aditya, and Leskovec, Jure. 2016. node2vec: Scalable feature learning for networks. Pages 855–864 of: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.

Tang, Jian, Qu, Meng, Wang, Mingzhe, Zhang, Ming, Yan, Jun, and Mei, Qiaozhu. 2015. Line: Large-scale information network embedding. Pages 1067–1077 of: Proceedings of the 24th international conference on world wide web. International World Wide Web Conferences Steering Committee.

Fouss, Francois, Pirotte, Alain, Renders, Jean-Michel, and Saerens, Marco. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Transactions on knowledge and data engineering, 19(3), 355–369.

Andersen, Reid, Chung, Fan, and Lang, Kevin. 2006. Local graph partitioning using pagerank vectors. Pages 475–486 of: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06).

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. 2013. Distributed representations of words and phrases and their compositionality. Pages 3111–3119 of: Advances in neural information processing systems.

Cao, Shaosheng, Lu, Wei, and Xu, Qiongkai. 2015. Grarep: Learning graph representations with global structural information. Pages 891–900 of: Proceedings of the 24th ACM international on conference on information and knowledge management.

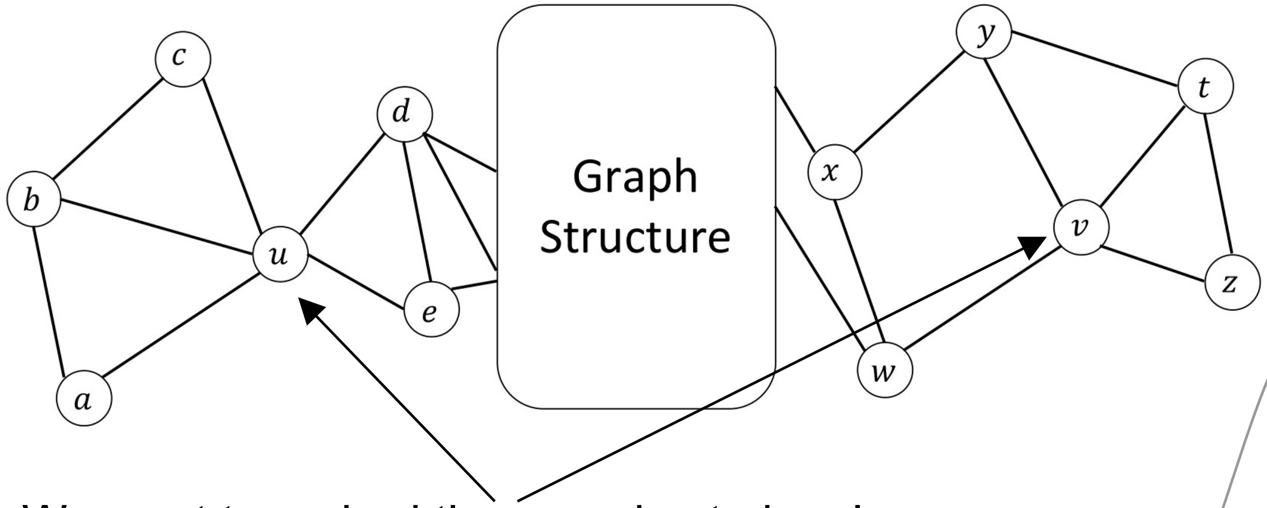
Qiu, Jiezhong, Dong, Yuxiao, Ma, Hao, Li, Jian, Wang, Kuansan, and Tang, Jie. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. Pages 459–467 of: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining.

Preserving structural role

Two nodes close to each other in the graph domain tend to co-occur in many random walks.

Therefore, the co-occurrence preserving methods are likely to learn similar representations for these nodes in the embedding domain.

However, we might want to embed nodes that are far from each other to be close in the embedding domain since they share a similar structural role.



We want to embed these nodes to be close in the embedding domain since they share a similar structural role.

A degree-based method: measure the pairwise structural role similarity, then build a new graph.

The edge in the new graph denotes structural role similarity.

The random walk based algorithm is utilized to extract co-occurrence relations from the new graph.

Ribeiroet al 2017. struc2vec: Learning node representations from structural identity. Pages 385–394 of: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Preserving community structure

$$P = \underbrace{A}_{\text{pairwise connectivity}} + \eta \underbrace{S}_{\text{similarity between the neighborhoods of nodes}}$$

$\eta > 0$ controls the importance of the neighborhood similarity

$$S: \quad s_{i,j} = \frac{A_i A_j^T}{\|A_i\| \|A_j\|}$$

The **modularity** for a graph with m communities: $Q = tr(H^T BH)$

$$B_{i,j} = A_{i,j} - \frac{d(v_i) d(v_j)}{vol(G)}$$

$H \in \{0,1\}^{N \times m}$ Each column of H represents a community.

The *i*-th row of H is a one-hot vector indicating the community of node v_i

$\frac{d(v_i) d(v_j)}{vol(G)}$ is the expected number of edges between nodes v_i and v_j in a randomly generated graph where the same number of edges as G are randomly placed.

The modularity measures the difference between the number of edges within communities of a given graph and a randomly generated graph with the same number of edges.

H can be learned by maximizing the modularity: $\max_H Q = tr(H^T B H) \quad s.t. \quad tr(H^T H) = N$

P, H are reconstructed from the embedding domain

Newman, Mark EJ. 2006. Modularity and community structure in networks. Proceedings of the national academy of sciences, 103(23), 8577–8582.

Wang, Xiao, Cui, Peng, Wang, Jing, Pei, Jian, Zhu, Wenwu, and Yang, Shiqiang. 2017. Community preserving network embedding. In: Thirty-first AAAI conference on artificial intelligence.

Li, Ye, Sha, Chaofeng, Huang, Xin, and Zhang, Yanchun. 2018d. Community detection in attributed graphs: An embedding approach. In: Thirty-Second AAAI Conference on Artificial Intelligence.

Graph Neural Networks

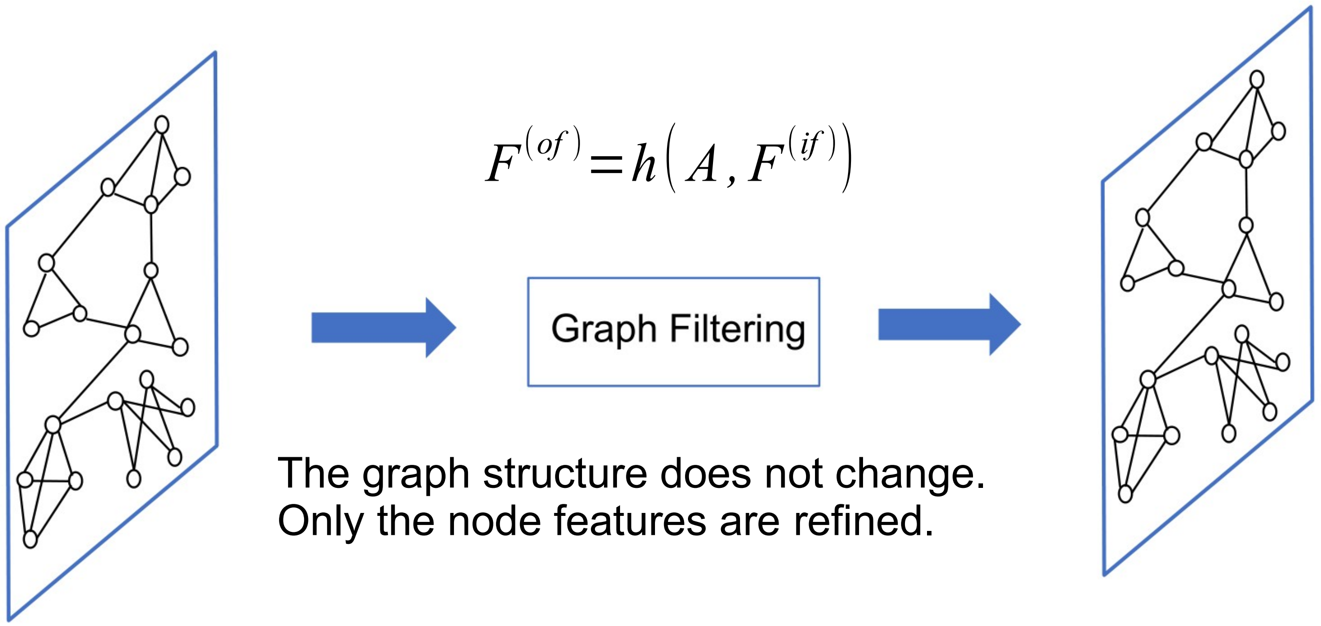
GNN can be viewed as a process of representation learning on graphs.

Node-focused tasks target on learning good features for each node.

For graph-focused tasks, they aim to learn representative features for the entire graph where learning node features is typically an intermediate step.

Learning node features aka **graph filtering**:

$$F^{(of)} = h\left(A, F^{(if)}\right)$$



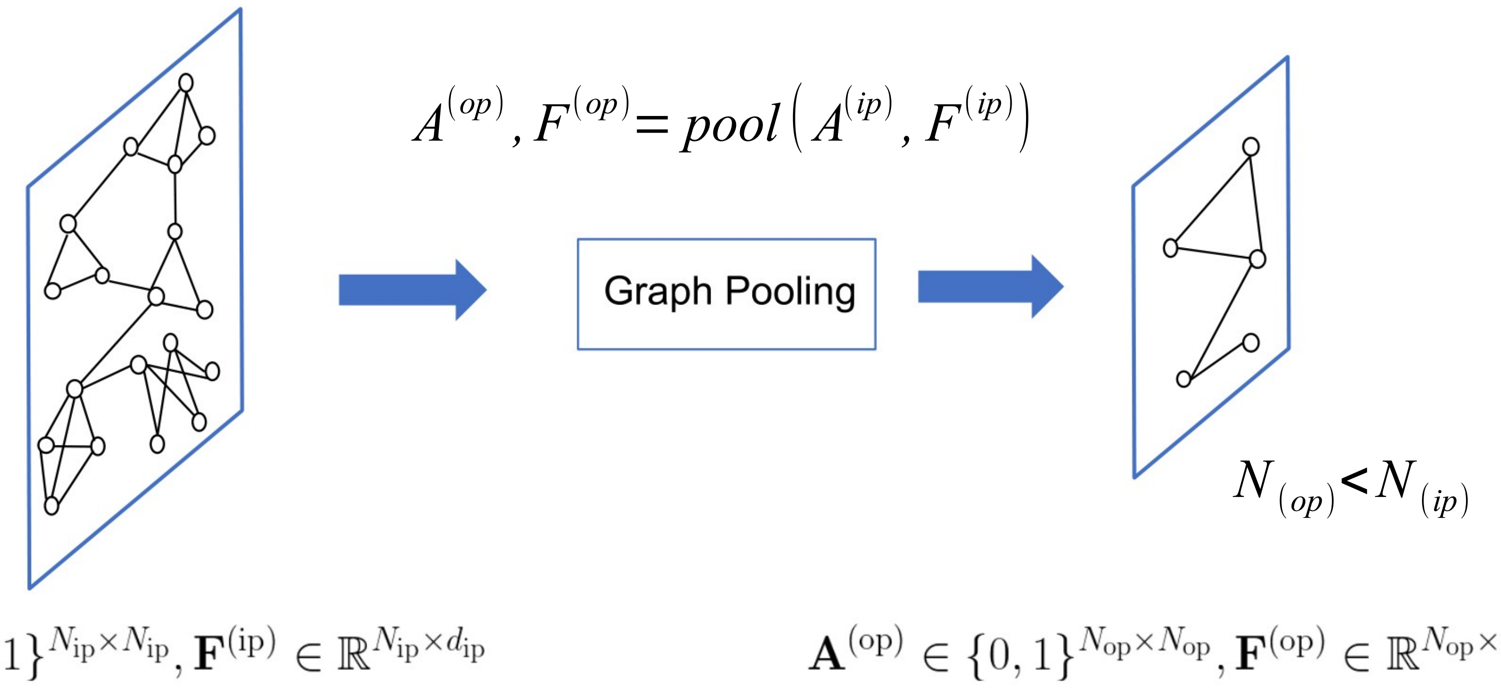
The graph structure does not change.
Only the node features are refined.

$\mathbf{A} \in \{0,1\}^{N\times N}, \mathbf{F}^{(\mathrm{if})} \in \mathbb{R}^{N\times d_{\mathrm{if}}}$

$\mathbf{A} \in \{0,1\}^{N\times N}, \mathbf{F}^{(\mathrm{of})} \in \mathbb{R}^{N\times d_{\mathrm{of}}}$

For node-focused tasks, the graph filtering operation is sufficient

Pooling operations summarize node features to generate graph-level features.



$$A^{(op)}, F^{(op)} = pool\left(A^{(ip)}, F^{(ip)}\right)$$

$$N_{(op)} < N_{(ip)}$$

$$\mathbf{A}^{(\text{ip})} \in \{0,1\}^{N_{\text{ip}} \times N_{\text{ip}}}, \mathbf{F}^{(\text{ip})} \in \mathbb{R}^{N_{\text{ip}} \times d_{\text{ip}}}$$

$$\mathbf{A}^{(\text{op})} \in \{0,1\}^{N_{\text{op}} \times N_{\text{op}}}, \mathbf{F}^{(\text{op})} \in \mathbb{R}^{N_{\text{op}} \times d_{\text{op}}}$$

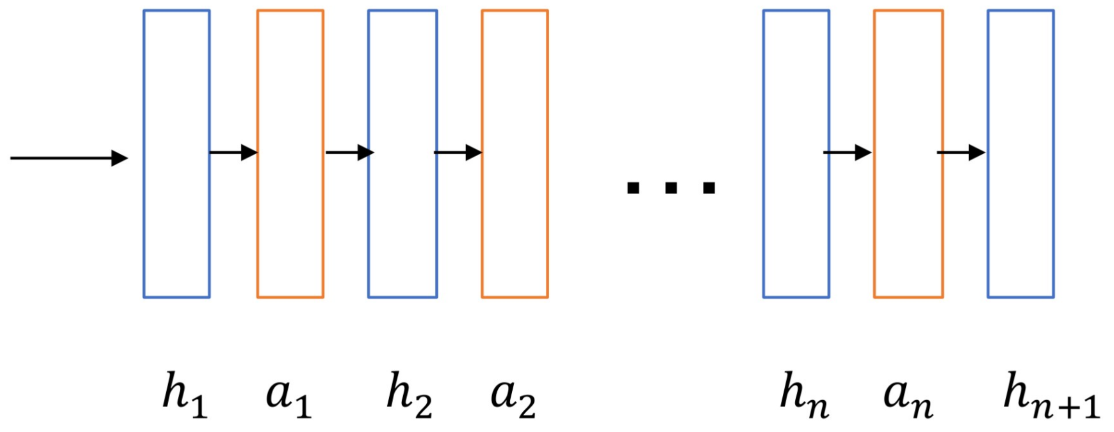
For graph-focused tasks, GNNs require both the graph filtering and the graph pooling.

Scarselli et al 2005. Graph neural networks for ranking web pages. Pages 666–672 of: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence.

Scarselli et al 2008. The graph neural network model. IEEE Transactions on Neural Networks, 20(1), 61–80.

A general GNN architecture for node-focused tasks

$$F^{(i)} = h_i\left(A, \alpha_{i-1}\left(F^{(i-1)}\right)\right)$$



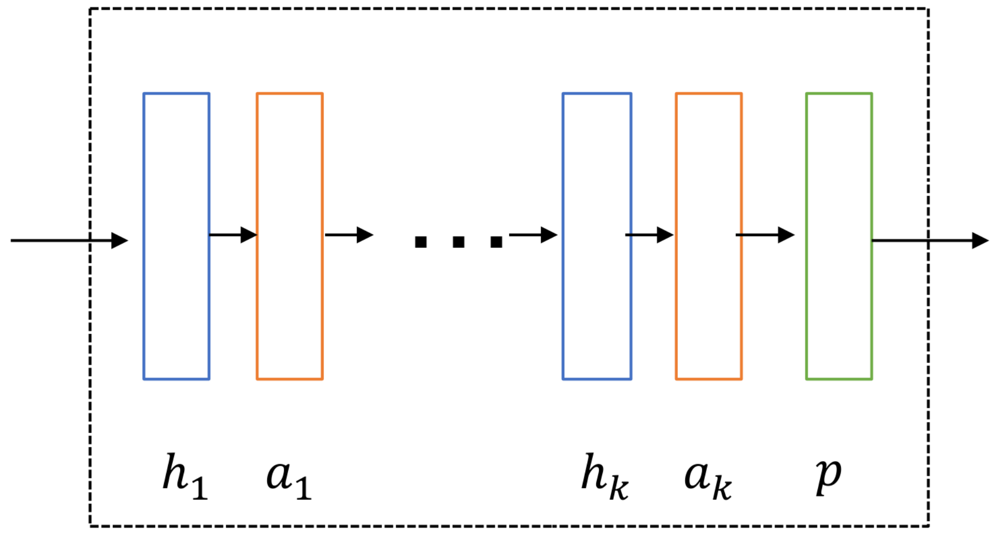
🟦 Filtering 🟧 Activation 🟩 Pooling

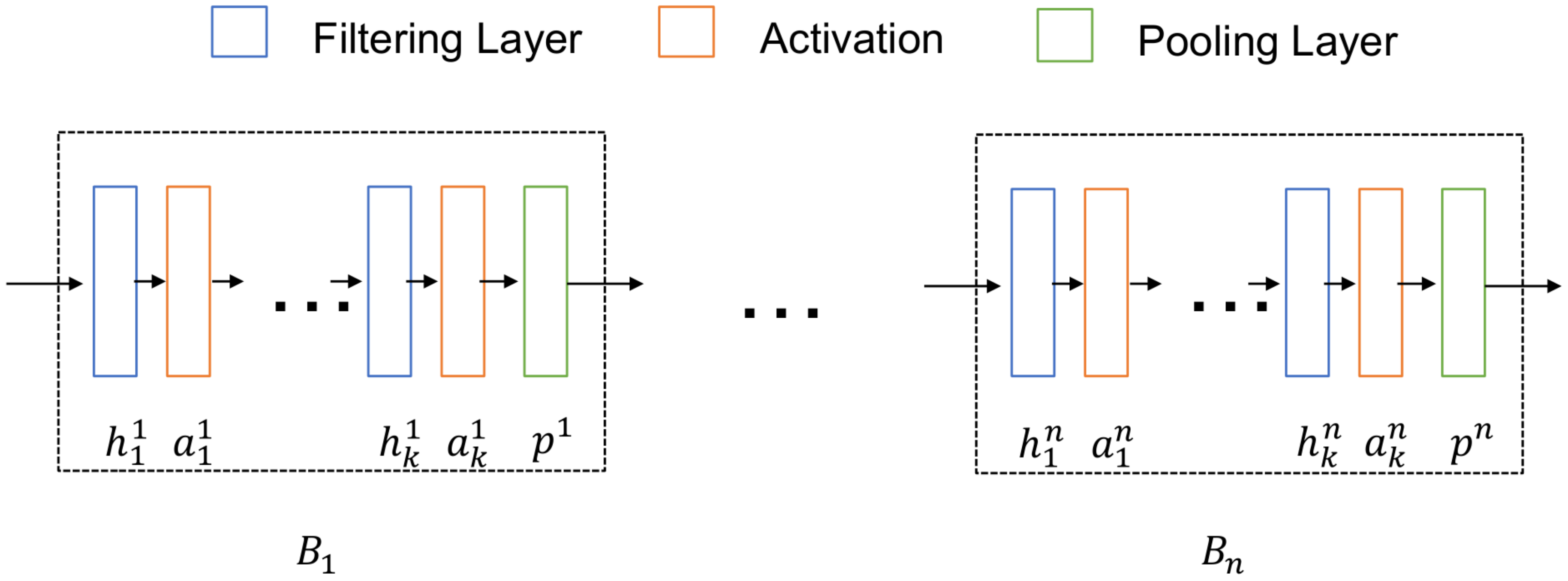
A block in GNNs for graph-focused tasks

$$F^{(i)} = h_i\left(A^{(ib)}, \alpha_{i-1}\left(F^{(i-1)}\right)\right), \quad i = 1, \ldots, k$$

$$A^{(ob)}, F^{(ob)} = pool\left(A^{(ib)}, F^{(k)}\right)$$

$$A^{(ob)}, F^{(ob)} = B\left(A^{(ib)}, F^{(ib)}\right)$$





Filtering Layer
Activation
Pooling Layer
h_1^1 a_1^1 h_k^1 a_k^1 p^1
h_1^n a_1^n h_k^n a_k^n p^n
B_1
B_n

$$A^{(j)}, F^{(j)} = B^{(j)}\left(A^{(j-1)}, F^{(j-1)}\right), \quad j = 1, \ldots, L$$

Spectral-based graph filters

$$\begin{tikzcd}
\boxed{f} \arrow[r, "GFT", "\text{Decompose}"'] & \boxed{U^T f} \arrow[r, "\hat{g}(\Lambda)", "\text{Filter}"'] & \boxed{\hat{g}(\Lambda)U^T f} \arrow[r, "IGFT", "\text{Reconstruct}"'] & \boxed{U\hat{g}(\Lambda)U^T f} \\
 & \text{Coefficients} & \text{Filtered coefficients} &
\end{tikzcd}$$

The idea is to modulate the frequencies of a graph signal.

The graph filters can be learned in a data-driven way.

$$\hat{g}(\Lambda) = \begin{pmatrix} \theta_1 & & 0 \\ & \ddots & \\ 0 & & \theta_N \end{pmatrix}$$

E.g., we can model $\hat{g}(\Lambda)$ with certain functions and then learn the parameters with the supervision from data.

Poly-Filter

$$\hat{g}(\lambda_l) = \sum_{k=0}^{K} \theta_k \lambda_l^k$$

Matrix form: $\hat{g}(\Lambda) = \sum_{k=0}^{K} \theta_k \Lambda^k$

In this case, $U\hat{g}(\Lambda)U^T$ is a polynomial of the Laplacian: $U\hat{g}(\Lambda)U^T f = \sum_{k=0}^{K} \theta_k L^k f$

It means that:

1) no eigendecomposition is needed

2) the polynomial parametrized filtering operator is spatially localized, i.e., the calculation of each element of the output only involves a small number of nodes in the graph.

Cheby-Filter

A limitation of the Poly-Filter: the basis of the polynomial $(1, x, x^2, \ldots)$ is not orthogonal.

Solution: Chebyshev polynomials

As the domain for the Chebyshev polynomials is $[-1, 1]$, to approximate the filter, we rescale and shift the eigenvalues of the Laplacian:

$$\tilde{\lambda}_l = \frac{2 \cdot \lambda_l}{\lambda_{max}} - 1 \qquad \lambda_{max} = \lambda_N$$

Cheby-Filter: $\hat{g}(\Lambda) = \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$

$$U \hat{g}(\Lambda) U^T f = \sum_{k=0}^{K} \theta_k U T_k(\tilde{\Lambda}) U^T f = \sum_{k=0}^{K} \theta_k T_k(\tilde{L}) f \quad \text{with} \quad \tilde{L} = \frac{2L}{\lambda_{max}} - I$$

Defferrard et al 2016. Convolutional neural networks on graphs with fast localized spectral filtering.
Pages 3844–3852 of: Advances in neural information processing systems.

GCN-Filter: simplified Cheby-Filter involving 1-hop neighbors

Set the order of Chebyshev polynomials to $K=1$ and approximate $\lambda_{max} \approx 2$

$$\hat{g}(\Lambda) = \theta_0 T_0(\widetilde{\Lambda}) + \theta_1 T_1(\widetilde{\Lambda}) = \theta_0 I + \theta_1 \widetilde{\Lambda} = \theta_0 I + \theta_1 (\Lambda - I)$$

$$U \hat{g}(\Lambda) U^T f = \theta_0 f + \theta_1 (L - I) f = \theta_0 f - \theta_1 (D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) f$$

where $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

Further simplification $\theta = \theta_0 = -\theta_1$ $\quad U \hat{g}(\Lambda) U^T f = \theta (I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) f$

$I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ has eigenvalues in the range $[0,2]$ which may lead to numerical instabilities when this operator is repeatedly applied to a specific signal

Renormalization: $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ ➔ $\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}}$ $\quad \widetilde{A} = A + I$ $\quad \widetilde{D}_{ii} = \sum_j \widetilde{A}_{i,j}$

The *i,j*-th element of $\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}}$ is non-zero only when nodes v_i and v_j are connected.

Thus, the GCN-Filter can also be viewed as a spatial-based filter, which only involves directly connected neighbors when updating node features.

Kipf, Thomas N, and Welling, Max. 2016a. Semi-supervised classification with graph convolutional networks. arXiv:1609.02907.

The ultimate intro to Graph Neural Networks. Maybe. https://www.youtube.com/watch?v=me3UsMm9QEs

Graph Convolutional Networks (GCNs) made simple https://www.youtube.com/watch?v=2KRAOZIULzw

Deep learning on graphs: successes, challenges, and next steps https://www.youtube.com/watch?v=PLGcx65MhCc

Graph neural networks https://www.youtube.com/watch?v=cWIeTMklzNg

Learning the Structure of Graph Neural Networks https://www.youtube.com/watch?v=9XoCQn34tXo

Graph Convolutional Networks https://www.youtube.com/watch?v=uMtDrG107Ws

Graph Convolutional Neural Networks https://www.youtube.com/watch?v=Mf-mIRF3ao8

Convolutional Neural Networks on Graphs https://www.youtube.com/watch?v=v3jZRkvIOIM

An Introduction to Graph Neural Networks https://www.youtube.com/watch?v=zCEYiCxrL_0

Unsupervised Learning with Graph Neural Networks https://www.youtube.com/watch?v=9jSFBcptZ9A

Large-scale Graph Representation Learning https://www.youtube.com/watch?v=oQL4E1gK3VU

Graph Node Embedding Algorithms https://www.youtube.com/watch?v=7JELX6DiUxQ

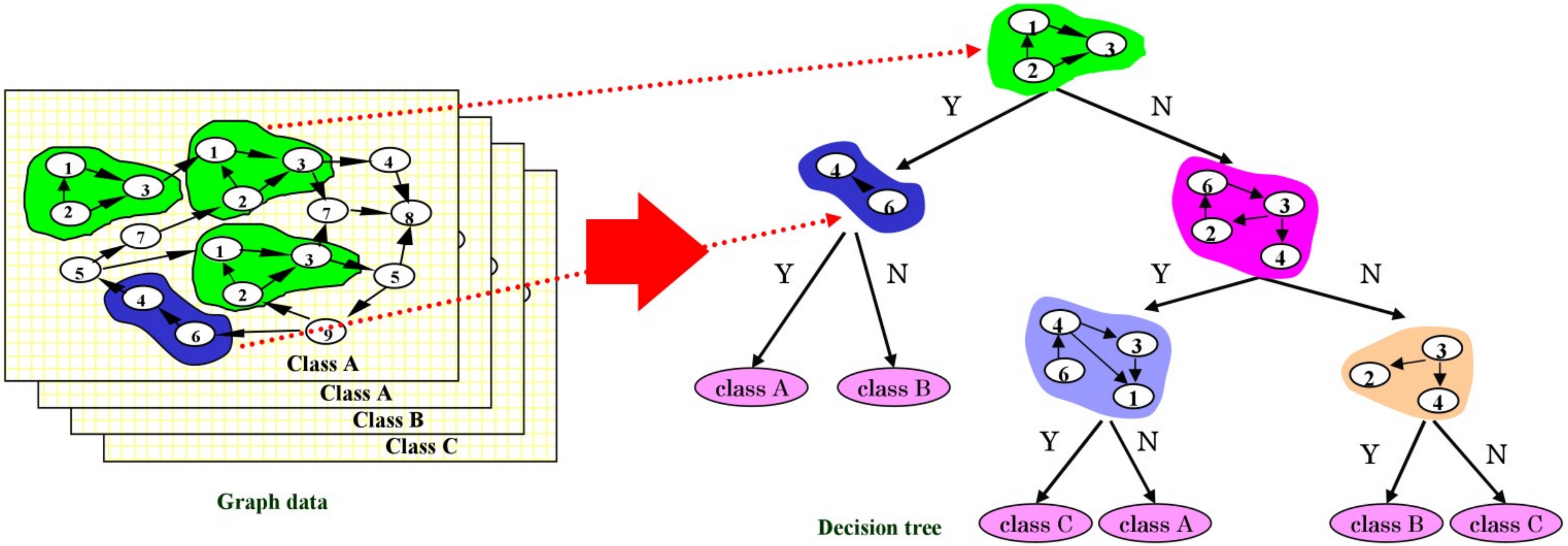
Graph Representation Learning https://www.youtube.com/watch?v=YrhBZUtgG4E

Graph learning python libraries

- PyTorch Geometric
- Graph Nets
- Deep Graph

Decision trees for graph structured data

Nguyen et al. Constructing Decision Trees for Graph-Structured Data by Chunkingless Graph-Based Induction. PAKDD 2006, LNAI 3918, pp. 390–399, 2006 https://link.springer.com/chapter/10.1007/11731139_45



Ivanov S., Prokhorenkova L. Boost then Convolve: Gradient Boosting Meets Graph Neural Networks. ICLR 2021 https://arxiv.org/abs/2101.08543