


Machine Learning for Graphs and Sequential Data

Graphs – Graph Neural Networks

Lecturer: Prof. Dr. Stephan Günnemann

cs.cit.tum.de/daml

Summer Term 23

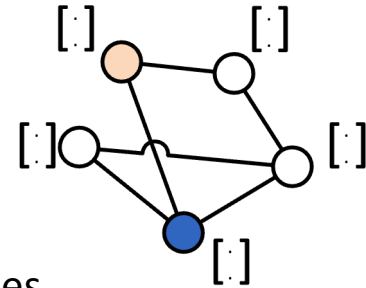
Data Analytics and
Machine Learning 

Roadmap

- **Chapter: Graphs**

1. Graphs & Networks
2. Generative Models
3. Ranking
4. Clustering
5. Classification (Semi-Supervised Learning)
6. Node/Graph Embeddings
- 7. Graph Neural Networks (GNNs)**
 - Spatial Graph Neural Networks
 - Spectral Graph Neural Networks
 - Advanced topics

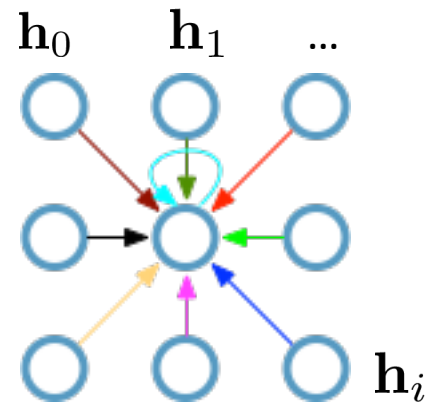
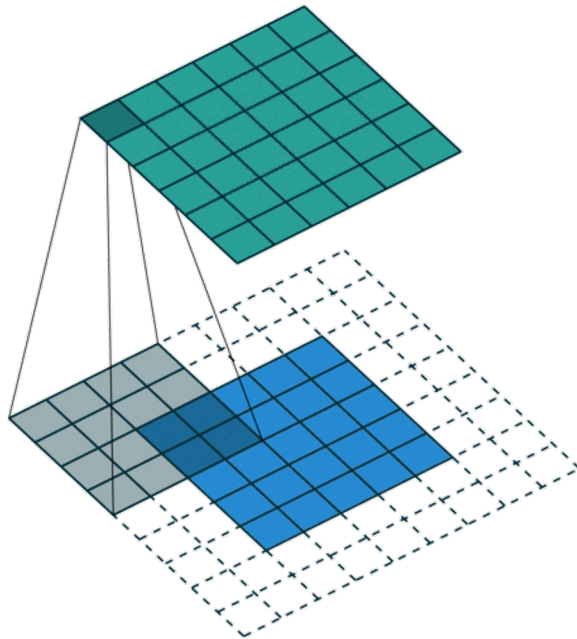
(Semi-supervised) Deep Learning on Graphs



- Neural Networks have achieved outstanding performance for many data types
 - However: usually restricted to simple grid-like (images) or sequential data (text)
- In a lot of real world graphs the nodes have attributes
 - Example: in citation networks nodes are papers, the text gives rise to attributes, and edges are citations; in protein-protein interaction networks the properties of the proteins can be considered as attributes
 - However: Label propagation considers only graph structure
- How can we perform (semi-supervised) deep learning on graphs taking both graph structure and the attributes into account?
 - **Graph Neural Networks:** Neural networks that operate on graph-structured data

From Matrix Convolution ...

- Images are a special kind of graph: every pixel is a node connected to 8 other nodes (up, down, left, right, etc. pixels)
- Recap: Convolutional Neural Networks (CNNs) perform convolutions on images



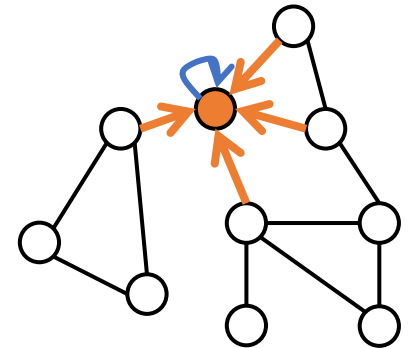
Update for a single pixel:

- Transform each pixel individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

... to Graph Convolution?

Can we generalize the idea of convolutions from images to graphs?

- We cannot just treat the adjacency matrix A as an image and feed it into a CNN: For example, the output should be the same even if we relabel nodes (i.e. even if we permute rows/columns of A)
- Unlike sequences/images, where the convolution operation is clearly defined, there are various versions for graphs
- The two most discussed versions in the literature are:
 - Spatial GNNs (Differentiable message passing)
 - Spectral GNNs (Graph signal processing/spectral filtering)
- Both formulations can be equivalent and often provide alternative views on the same concept



Roadmap

- **Chapter: Graphs**

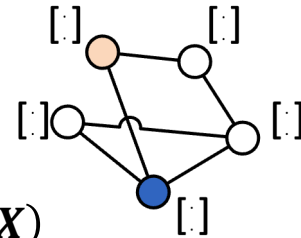
1. Graphs & Networks
2. Generative Models
3. Ranking
4. Clustering
5. Classification (Semi-Supervised Learning)
6. Node/Graph Embeddings
- 7. Graph Neural Networks (GNNs)**
 - **Spatial Graph Neural Networks**
 - Spectral Graph Neural Networks
 - Advanced topics

Differentiable Message Passing Framework (I)

- Spatial GNNs rely on the local aggregation of representations ("message-passing")

- Given:

- A graph G with a set of nodes V , set of edges E , and each node v has features $x_v \in \mathbb{R}^d$ // often written as $G = (\mathbf{A}, \mathbf{X})$
- One might even have edge weights/edge features E_{vu}



- Let $h_v^{(k-1)}$ be the hidden representation of node v at previous $k - 1$ layer

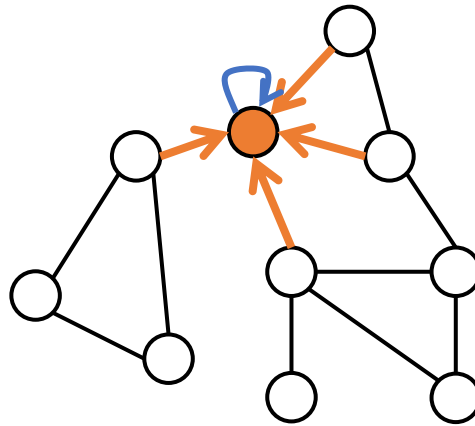
- For each node do:

1. Gather messages from all neighbors $m_v^{(k)} = \sum_{u \in N(v)} M(h_v^{(k-1)}, h_u^{(k-1)}, E_{vu})$
2. Update the hidden representation $h_v^{(k)} = U(h_v^{(k-1)}, m_v^{(k)})$

- M and U are any differentiable functions, e.g. neural networks

Differentiable Message Passing Framework (II)

- For each node do:
 1. Gather messages from all neighbors $m_v^{(k)} = \sum_{u \in N(v)} M(h_v^{(k-1)}, h_u^{(k-1)}, E_{vu})$
 2. Update the hidden representation $h_v^{(k)} = U(h_v^{(k-1)}, m_v^{(k)})$
- Example: Calculating the update for the node in orange



1. Gather message from all neighbors
2. Update hidden representation

Example Instantiation of the Framework

- Let $h_v^{(0)} = x_v$
 - At the first layer the representations are the node features
- Set the message aggregation function to be the average over the neighbors

$$m_v^{(k)} = \sum_{u \in N(v)} \frac{1}{d_v} (W^{(k)} h_u^{(k-1)} + b^{(k)})$$

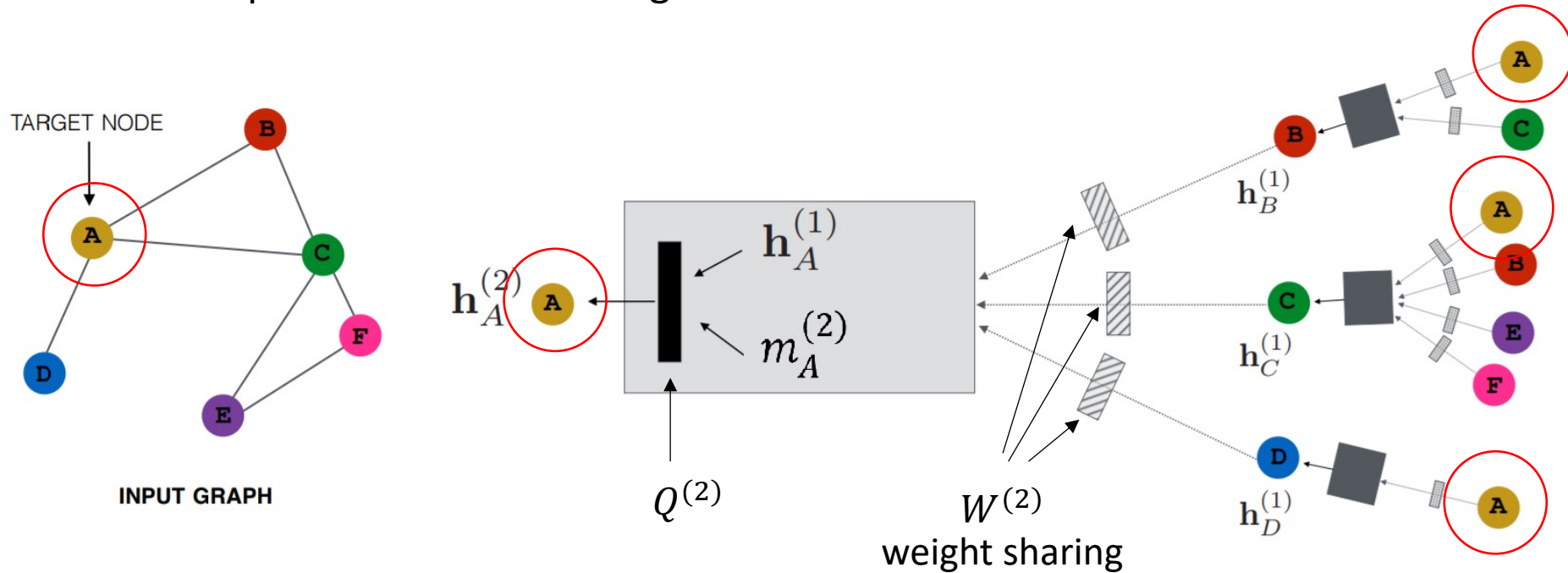
- Calculate hidden representations as simple NNs with a non-linearity

$$h_v^{(k)} = \text{relu}(Q^{(k)} h_v^{(k-1)} + p^{(k)} + m_v^{(k)})$$

- Here $W^{(k)}, b^{(k)}, Q^{(k)}, p^{(k)}$ are the **trainable parameters** of the k -th layer

Recursive View of Differentiable Message Passing

- The hidden representation of each node is recursively defined in terms of the hidden representation of its neighbors

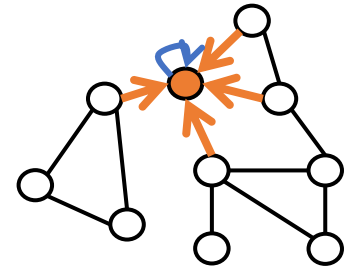


Graph Convolutional Neural Networks

- In the spatial domain, a graph convolution updates each nodes' features by considering the local neighborhood.

- In effect it is some kind of message passing

- Example: $X_i^{(t+1)} = X_i^{(t)} + \sum_{(i,j) \in E} X_j^{(t)}$
or in matrix notation $X^{(t+1)} = X^{(t)} + AX^{(t)} = (A + I_n)X^{(t)}$



- Let $H^{(l)} = [h_{v_1}^{(l)}, \dots, h_{v_2}^{(k)}]$ denote the hidden representations after layer l
- Similar to a (learnable) convolutional layer we can formulate a (learnable) graph convolutional layer

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$$

● Non-linearity

● Message Passing

● Feature Transformation

- Normalizing the propagation matrix avoids exploding gradients
 - Choose $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ where $\tilde{A} = A + I_n$ and $D_{ii} = \sum_j \tilde{A}_{ij}$ is the degree matrix of \tilde{A}

Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. International Conference on Learning Representations (ICLR).

How do Neighbors Influence a Given Node?

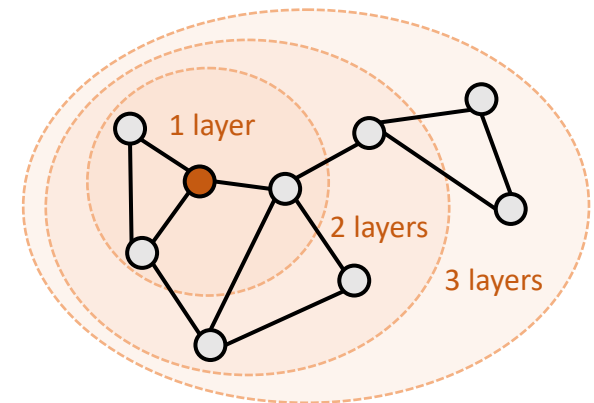
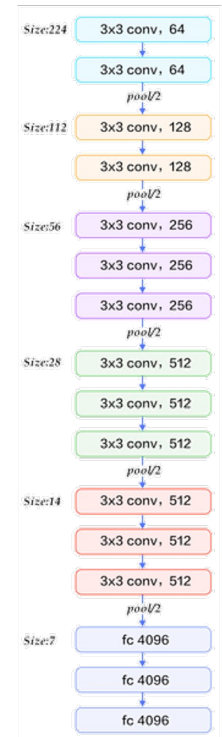
- Observation 1: K hidden layers equal to K steps of message passing
 - At step 1, node v aggregates information from its 1-hop neighbors
 - At step 2, node v (implicitly) aggregates information from the neighbors of its 1-hop neighbors, i.e., from its 2-hop neighbors
 -
 - At step K , node v (implicitly) aggregates information from its K -hop neighbors
- Observation 2: K hidden layers mean that the representation $h_v^{(K)}$ of node v is based on information from all nodes in its K -hop neighborhood

How Deep are Graph Neural Networks?

- In an MLP or CNN, depth means the number of non-linear transformations of the input
- On graphs we can also consider the furthest distance that a model propagates information as depth

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$$

- GCNs apply 1 transformation per message passing step
- *Transformation depth* and *propagation depth* are orthogonal in GNNs
 - We could transform the messages with a multi-layer network in each step or propagate without transforming for multiple steps



Roadmap

- **Chapter: Graphs**

1. Graphs & Networks
2. Generative Models
3. Ranking
4. Clustering
5. Classification (Semi-Supervised Learning)
6. Node/Graph Embeddings
- 7. Graph Neural Networks (GNNs)**
 - Spatial Graph Neural Networks
 - **Spectral Graph Neural Networks**
 - Advanced topics

Background: Graph Signal Processing

- Given:
 - The normalized Laplacian matrix $\hat{L} = I - D^{-1/2}AD^{-1/2}$
 - Its Eigendecomposition $\hat{L} = U\Lambda U^T$ where columns of U are the eigenvectors of \hat{L} and Λ contains the eigenvalues $0 = \lambda_1 \leq \dots \leq \lambda_n \leq 2$ on the diagonal
 - U is an orthogonal matrix: $U^T U = I$

- Analog to classical signal processing:
 - A graph signal is a function on the nodes $x: V \rightarrow \mathbb{R}$, represented as a vector $x \in \mathbb{R}^N$
 - Graph Theory: We can define a “graph” Fourier transform (GFT) of a signal x as $\tilde{x} = U^T x$, and the inverse GFT as $x = U\tilde{x}$
 - Now we can define spectral filtering on graphs as

$$y = Ug(\Lambda)U^T x$$

where g is a so-called *spectral filter*

- The image of g are diagonal matrices

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., & Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3), 83-98.

Spectral Graph Neural Networks

- Main idea of spectral GNNs: Parametrize spectral filters g_θ
 - This yields the following update equation of spectral layers:

$$H^{(l+1)} = \phi \left(U g_\theta(\Lambda) U^T \varphi(H^{(l)}) \right)$$

where ϕ and φ are multi-layer perceptrons.

- Depth: Spectral layers separate the transformation $\varphi(H^{(l)})$ from the “propagation” $U g_\theta(\Lambda) U^T$ (in principle we can still apply several spectral layers)
- Choice of spectral filter g_θ is crucial
 - A common choice is a polynomial filter $g_\theta(\Lambda) = \sum_{k=0}^K \theta_k \Lambda^k$ of degree K since it is localized and limited in the number of parameters
 - For polynomial filters we often also write $H^{(l+1)} = \phi \left(g_\theta(\hat{L}^k) \varphi(H^{(l)}) \right)$ since

$$U g_\theta(\Lambda) U^T = \sum_{k=0}^K \theta_k U \Lambda^k U^T = \sum_{k=0}^K \theta_k \hat{L}^k$$

- Both formulations (spatial and spectral) can be equivalent and often provide different views on the same concept
 - For example, GCNs (Kipf et al., 2017) can be interpreted as a first-order approximation of localized spectral filters on graphs

Discussion: Spectral Graph Neural Networks

- **Computational complexity:** In the original form, spectral GNNs require a spectral decomposition of the graph Laplacian
 - Full eigendecompositions can be avoided, e.g., by using polynomial filters or by introducing update schemes similar to power iteration [1]
- **Receptive field size:** In contrast to message-passing GNNs, spectral GNNs may propagate graph signals through the entire graph
 - Localized filters (e.g. polynomial filters) operate only locally ($\hat{L}^1, \hat{L}^2, \dots, \hat{L}^k$)
- **Transferability:** Can we transfer spectral filters g_θ to unseen graphs?
 - Spectral filters define filter operations in the spectral domain of one specific graph
 - Changing a single edge can yield a large change of the spectrum
 - Transferability is possible for example under the assumption that all graphs are discretizations of the same underlying continuous space [2]

[1] Johannes Klicpera, Aleksandar Bojchevski, Stephan Günnemann. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. ICLR 2019

[2] Ron Levie, Wei Huang, Lorenzo Bucci, Michael M. Bronstein, Gitta Kutyniok: Transferability of Spectral Graph Convolutional Neural Networks. J. Mach. Learn. Res. 22: 272:1-272:59 (2021)

Back to the Big Picture

- Both, spatial and spectral, GNNs give you representations $h_v^{(K)}$ for each node v after the final layer K
 - You can stack them to one matrix $H^{(K)}$
- How to use these representations for machine learning tasks?
- How to learn the parameters of the GNNs?

Semi-Supervised Node Classification

How to handle semi-supervised node classification with GNNs?

- That is, besides the graph, a subset of labeled nodes $S \subseteq V$ is given, where y_v denotes the label of node $v \in S$
- Consider the representations $h_v^{(K)}$ at the final layer K as logits
- Use the softmax function to obtain the probability of node v to belong to class c

$$p_v = \text{softmax}\left(h_v^{(K)}\right)$$
- Train the network using the standard cross entropy loss between the predicted probabilities and the observed labels
 - Recall: y_{vc} is the one-hot vector encoding the label for node v
 - Denote with p_{vc} the probability that node v to belong to class c
 - Denote with \mathcal{C} the set of all classes, and S the set of labeled nodes

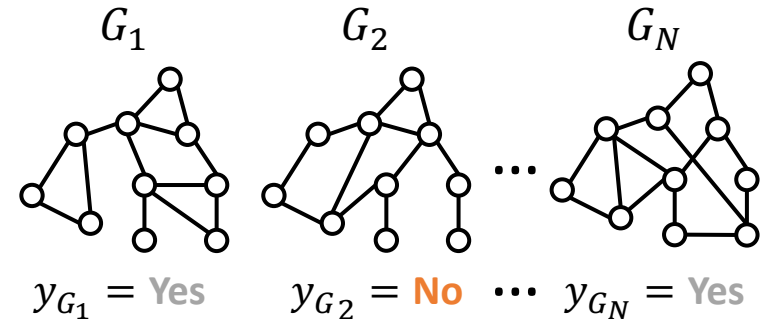
$$\min_{\{W^{(k)}, Q^{(k)}, p^{(k)}, b^{(k)}\}_{k=1..K}} - \sum_{v \in S} \sum_{c \in \mathcal{C}} y_{vc} \log p_{vc}$$

Graph Classification

How to handle graph classification?

Given:

- A set of training graphs $\{G_i = (V_i, E_i)\}_{i=1..N}$
- along with labels y_{G_i} denoting the graph level target of graph G_i



- Let $H_{G_i} = [h_1^{(K)}, h_2^{(K)}, \dots, h_{|V_i|}^{(K)}]$ be a matrix containing the final representations for all nodes of graph G_i
- Define an aggregation function $R(H_{G_i})$ over the node representations of a given graph that produces a representation for the entire graph $h_{G_i} = R(H_{G_i})$
 - Example: $R(H_{G_i}) = \frac{1}{|V_i|} \sum_{v \in V_i} h_v^{(K)}$ is the average of the node embeddings
- For classification: consider h_{G_i} as the logits and train a model using standard cross-entropy loss, i.e. $\mathcal{L}(y_{G_i}, \text{softmax}(h_{G_i}))$
 - For regression, e.g. predicting functional properties of molecules, you can use squared loss

Bigger Picture: Equivariant Machine Learning

- What if we change the (arbitrary) ordering of nodes?
 - (1) Graph-level representations h_{G_i} should be independent of node-ordering
 - (2) The order of node-level representations $h_v^{(K)}$ should change accordingly

- Two attributed graphs $G_1 = (A_1, X_1), G_2 = (A_2, X_2)$ are called isomorphic if there exists a permutation matrix P such that $PA_1P^T = A_2$ and $PX_1 = X_2$

- Ideally, Graph Neural Networks f behave the same under graph isomorphism:
 - For (1) we require **invariance**, i.e. for any permutation matrix P :

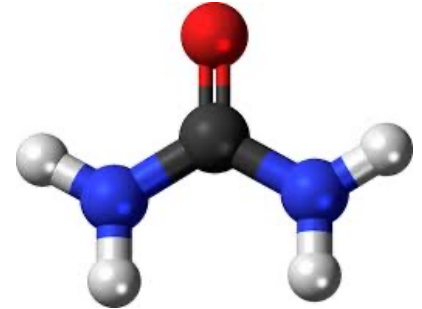
$$f(PX, PAP^T) = f(X, A)$$
 - For (2) we require **equivariance**, i.e. for any permutation matrix P :

$$f(PX, PAP^T) = Pf(X, A)$$

- In the message-passing framework this can be achieved by choosing permutation-invariant aggregation functions (e.g. sum, max, or avg)

Equivariant Machine Learning for Spatial Graphs

- A **spatial graph** is a graph in which nodes (or edges) are spatial elements corresponding to geometric objects in a vector space equipped with a metric (i.e. a Hilbert Space)
- Example: molecules can be interpreted as graphs where nodes correspond to atoms with positions in 3D-space and edges correspond to bond types
- For spatial graphs, GNNs should be invariant to further transformations beyond node permutations
- $E(n)$ -equivariant GNNs are equivariant to rotation and translation, i.e. to group operations under the Euclidean group $E(N) = \{RX + T : R \in O(N)\}$



Satorras, Garcia, Hoogeboom, Welling. “E(n) equivariant graph neural networks”. ICML 2021.
Gasteiger, Groß, Günnemann. “Directional Message Passing for Molecular Graphs”. ICLR 2020.

Roadmap

▪ Chapter: Graphs

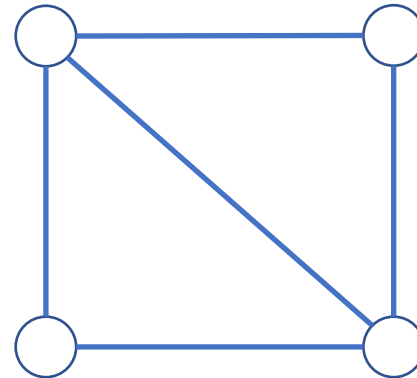
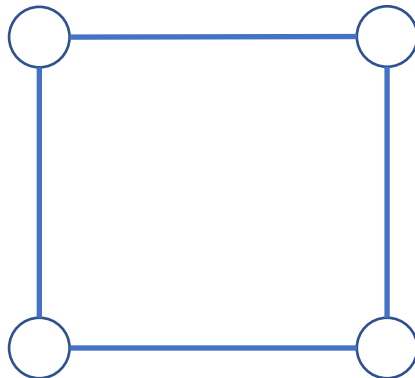
1. Graphs & Networks
2. Generative Models
3. Ranking
4. Clustering
5. Classification (Semi-Supervised Learning)
6. Node/Graph Embeddings
- 7. Graph Neural Networks (GNNs)**
 - Spatial Graph Neural Networks
 - Spectral Graph Neural Networks
 - **Advanced topics**

Limitations of Graph Neural Networks

- GNNs have become extremely popular and are getting used in many domains
 - Though, still a very young research field – many open questions
 - In particular, various limitations regarding early models
- Expressiveness
 - Worse than the simple WL test on the graph isomorphism problem
 - GCN-like models tend to overly smooth predictions with increasing depth
- Reliability / (Non-)Robustness
 - GNNs are susceptible to adversarial graph manipulations, see next block of the lecture
- Scalability to large graphs
 - Message passing requires communication and synchronization at every step
 - At the same time you have to process the whole dataset at once because the nodes are not i.i.d.; batching/stochastic training not trivial
 - Adding a new node or edge affects large parts of the graph because of the small world phenomenon and requires a complete recomputation of the predictions/embeddings

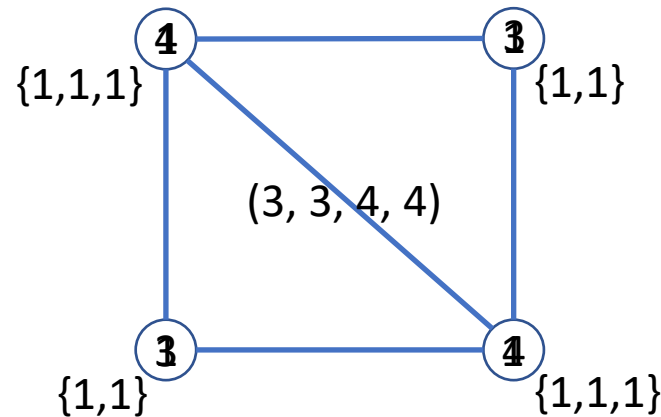
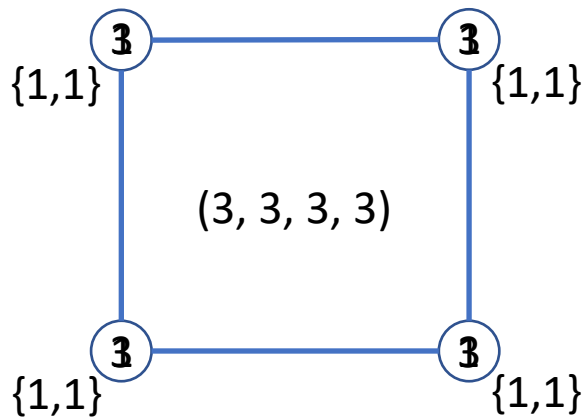
Graph Isomorphism Problem

- Determine whether two graphs are structurally identical
 - Can we map all nodes from one graph onto the nodes from the second graph such that the edge structure is preserved?
- Best known algorithm has exponential worst-case runtime
- Are these two graphs isomorphic?



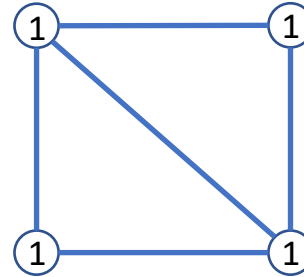
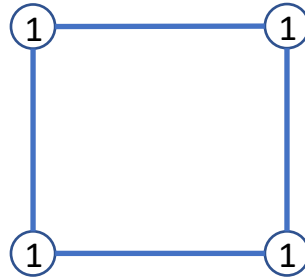
Weisfeiler-Lehman Test

1. Label all nodes as 1
2. Collect all neighboring labels in ordered multisets
3. Relabel each node by hashing the current label concatenated with the neighbor labels with some hash function
4. Compare the sorted node labels
 - If they are different, the two graphs are definitely not isomorphic
 - Otherwise, the graphs *might* be isomorphic and we continue with step 2 until the sorted labels have converged or our computation budget is used up



Weisfeiler-Lehman vs. Graph Neural Networks

- The WL test can distinguish many non-isomorphic graphs but not all of them
- However, that looked a lot like message passing with hash aggregation
- How do GNNs compare to WL?
 - They often can distinguish even fewer graphs (consider mean or max aggregation in our running example)



- Increasing expressiveness of GNNs:
 - carefully designed aggregation functions lead to the same power as WL [Xu2019]
 - use of higher order structure [Morris2019] or directional/edge information [Klicpera2020]

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. How Powerful are Graph Neural Networks. ICLR 2019

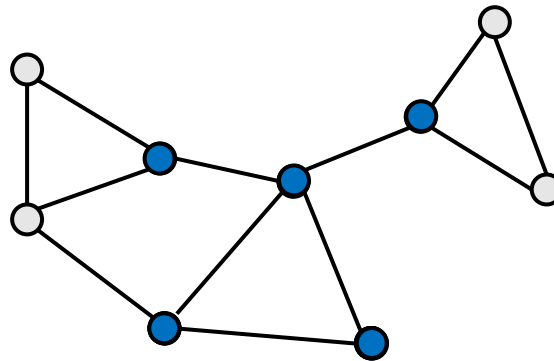
C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, M. Grohe: Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. AAAI 2019

J. Klicpera, J. Groß, S. Günnemann: Directional Message Passing for Molecular Graphs. ICLR 2020

Blogpost on WL and GNNs: <https://towardsdatascience.com/expressive-power-of-graph-neural-networks-and-the-weisfeiler-lehman-test-b883db3c7c49>

Oversmoothing (I)

- In a 3-layer GCN information propagates from the blue to the red node along all possible walks of length 3 or less with about equal weight

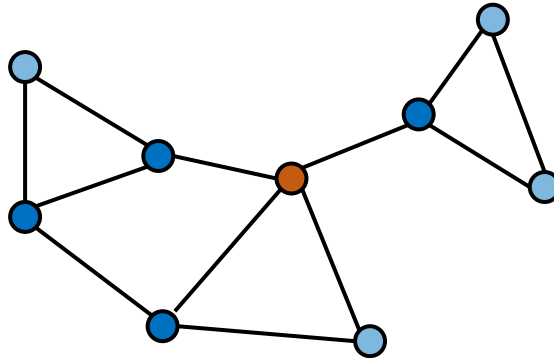


All paths along which information travels from blue to red

- Influence between the nodes is proportional to the probability of visiting blue from red via a random walk of length k

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., & Jegelka, S. (2018). Representation Learning on Graphs with Jumping Knowledge Networks. In ICML 2018: Thirty-fifth International Conference on Machine Learning (pp. 5449–5458).

Oversmoothing (II)

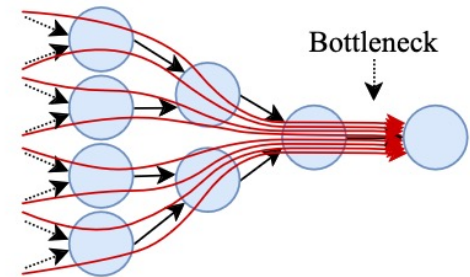


- In the limit of infinite layers, the influence becomes proportional to the stationary distribution of the graph as a markov chain (see PageRank)
- The stationary distribution is a global property of the graph
 - The blue node influences every other node in the same way regardless of their distance
- Deep GCN-like networks are unable to represent local neighborhoods

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., & Jegelka, S. (2018). Representation Learning on Graphs with Jumping Knowledge Networks. In ICML 2018: Thirty-fifth International Conference on Machine Learning (pp. 5449–5458).

Oversquashing

- GNNs typically fail to propagate messages over long-range distances due to bottlenecks in the graph
- Main reason is that an exponential amount of information is “squashed” into node representations of fixed size
- As a result, GNNs may have low predictive performance on tasks where long-range information is crucial



[Alon2021]

Oversmoothing vs. Oversquashing

- Oversmoothing and oversquashing are related but distinct problems
- Both occur when the problem radius and the number of GNN layers is large
 - Fully connected graphs: oversmoothing but no oversquashing (no bottlenecks)
 - Directed tree-structured graphs: oversquashing (bottlenecks) but no oversmoothing (subtrees are independent of each other)

Uri Alon, Eran Yahav: On the Bottleneck of Graph Neural Networks and its Practical Implications. ICLR 2021

PageRank in the Message Passing Framework

$$r^{(t+1)} = AD^{-1}r^{(t)}$$

PageRank iteration

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

GCN iteration

- PageRank and a GCN layer exhibit some structural similarity
- If we “push” the nonlinearity one layer up, the similarity becomes even more apparent

PageRank in the Message Passing Framework

$$r^{(t+1)} = AD^{-1}r^{(t)}$$

PageRank iteration

$$H^{(l+1)} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} \sigma(H^{(l)}) W^{(l)}$$

GCN iteration

- PageRank and a GCN layer exhibit some structural similarity
- If we “push” the nonlinearity one layer up, the similarity becomes even more apparent

● Features

● Message Passing

- We know how to adapt PageRank to focus on a node’s local neighborhood
 - Personalized PageRank with teleport vector $\pi = e_i$, the i -th unit vector for the i -th node
 - Extend GCN in the same way!

Personalized Propagation of Neural Predictions (PPNP)

$$r^{(t+1)} = AD^{-1}r^{(t)}$$

PageRank iteration

$$H^{(l+1)} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \sigma(H^{(l)}) W^{(l)}$$

GCN iteration

1. Separate transformation and propagation

α teleport probability

π teleport vector

- Instead of $H^{(0)} = X$, let $H_{i,:}^{(0)} = f_{\theta}(X_{i,:})$ for some f_{θ} , e.g. a neural network

Personalized Propagation of Neural Predictions (PPNP)

$$r^{(t+1)} = (1 - \alpha)AD^{-1}r^{(t)} + \alpha\pi$$

Personalized PageRank iteration

$$H^{(l+1)} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}$$

PNP iteration

α teleport probability

π teleport vector

1. Separate transformation and propagation

- Instead of $H^{(0)} = X$, let $H_{i,:}^{(0)} = f_{\theta}(X_{i,:})$ for some f_{θ} , e.g. a neural network

2. Introduce personalized teleportation

- As with PageRank, we can take the limit of infinite propagation steps
- We arrive at the PPNP model for node classification

$$Z = \text{softmax}\left(\alpha(I_n - (1 - \alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})^{-1}H^{(0)}\right)$$

Personalized Propagation of Neural Predictions (PPNP)

$$r^{(t+1)} = (1 - \alpha)AD^{-1}r^{(t)} + \alpha\pi$$

Personalized PageRank iteration

$$H^{(l+1)} = (1 - \alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)} + \alpha H^{(0)}$$

Personalized PNP iteration

α teleport probability

π teleport vector

1. Separate transformation and propagation

- Instead of $H^{(0)} = X$, let $H_{i,:}^{(0)} = f_{\theta}(X_{i,:})$ for some f_{θ} , e.g. a neural network

2. Introduce personalized teleportation

- As with PageRank, we can take the limit of infinite propagation steps
- We arrive at the PPNP model for node classification

$$Z = \text{softmax}\left(\alpha(I_n - (1 - \alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})^{-1}H^{(0)}\right)$$

PPNP Alleviates Some of the Limitations

- Personalized PageRank prevents oversmoothing
 - Immediate neighborhood gets higher weight
- Clear separation between depth of transformation and message passing
 - Transformation depth is the depth of the deep model f_{θ}
 - PPNP uses the limit distribution of personalized PageRank which corresponds to infinite message passing depth
- Scalability
 - The predictions $f_{\theta}(v_i)$ can be computed efficiently for arbitrarily large graphs since they are made individually per node (no interaction)
 - Propagation only operates on the lower dimensional predictions instead of high dimensional node attributes or intermediate representations
 - However: As with PageRank the exact solution is too costly
 - Computation involves a matrix inversion with runtime at least $O(n^{2.373})$
 - Inverse is dense and requires $O(n^2)$ memory

Approximate PPNP (APPNP)

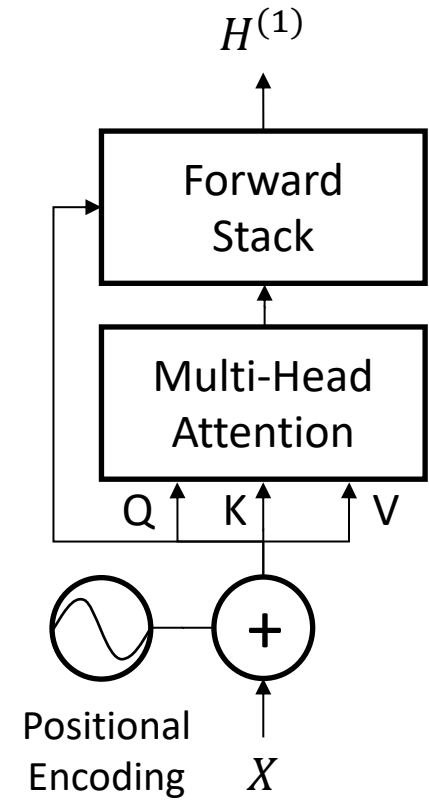
- Observation: The exact solution is the limit of an iterative update equation
- Approximate the exact solution as $Z \approx H^{(K)}$, i.e. with K iterations of

$$H^{(l+1)} = (1 - \alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)} + \alpha H^{(0)}$$

- Never realizes a dense $n \times n$ matrix and exploits the sparsity of \tilde{A}
 - Only needs $O(Knd)$ runtime and $O(nd)$ memory
- Setting $K = 10$ already approximates the exact solution effectively

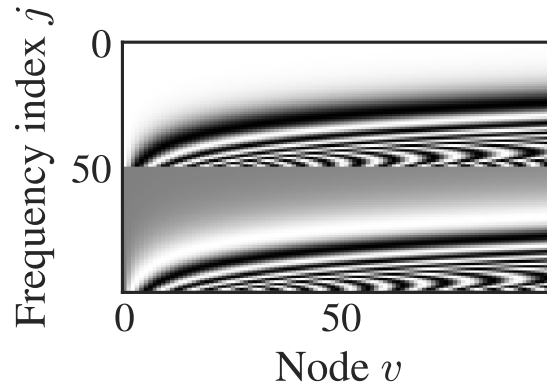
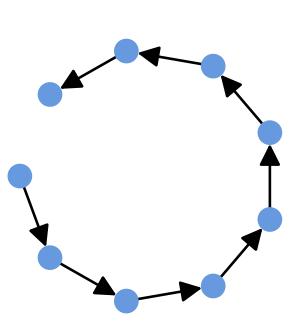
Graph Transformers

- As we have seen (most) spatial and spectral GNNs effectively pass messages along the edges of a graph
- However, is this constraint of information propagation optimal for all applications (e.g. see oversquashing)?
- Recall that a similar reasoning justifies transformers for sequences
- Can we use transformers for graphs?
- **Core challenge: How to define useful positional encodings for nodes?**



Given: attributed graph $G = (A, X)$

Sequences: Sinusoidal Encodings



d - # of features/
embedding dim.
 $2\pi 10,000$ - largest wave length
 v - node index/position
 j - frequency index

- Sinusoidal Positional Encodings $\text{PE}^{(\sin)} \in \mathbb{R}^{n \times d}$

$$\text{PE}_{v,j+d/2}^{(\sin)} = \sin\left(v \cdot 1/10,000^{2j/d}\right)$$

$$\text{PE}_{v,j}^{(\sin)} = \cos\left(v \cdot 1/10,000^{2j/d}\right)$$

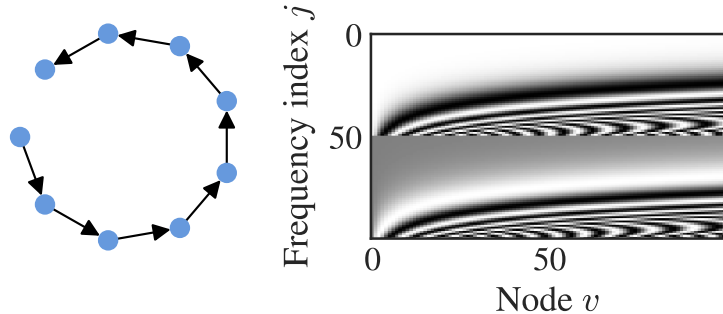
- Positional encodings inspired by Discrete Fourier Transformation (DFT):

$$\hat{x}_j = \sum_{v=0}^{n-1} x_v \left[\cos\left(v \cdot \frac{2\pi}{n} j\right) - i \cdot \sin\left(v \cdot \frac{2\pi}{n} j\right) \right]$$

x - signal (spatial domain)
 \hat{x} - signal (spectral domain)
 n - sequence length

Graph Transformers: Positional Encoding

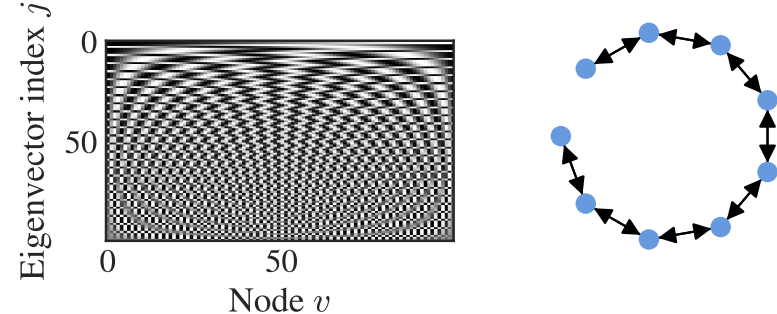
■ Sinusoidal Encodings



$$\text{PE}_{v,j+d/2}^{(\sin)} = \sin(v \cdot 1/10,000^{2j/d})$$

$$\text{PE}_{v,j}^{(\sin)} = \cos(v \cdot 1/10,000^{2j/d})$$

↔ Eigenvectors of Laplacian



Closed form for $L = D - A = U\Lambda U^T$ of a sequence:

$$U = \text{PE}_{v,j}^{(\text{lap})} = \pm \cos\left(v \cdot \frac{j\pi}{n} + \frac{j\pi}{2n}\right)$$

- Recall that Graph Fourier Transformation (GFT) for (undirected) graphs can be defined via the Eigenvectors of Laplacian
- In the case of sequences, the resulting eigenvectors consist of cosine waves (aka Cosine Transformation Type II)

S. Geisler, Y. Li, D. Mankowitz, AT. Cemgil, S. Günnemann, C. Paduraru: Transformers Meet Directed Graphs. ICML 2023

Graph Transformers: Directed Graphs

- Eigendecomposition has desirable properties for, e.g., real symmetric matrices (eigenvectors are orthogonal, both forward and inverse GFT exist, ...)
- $L = D - A$ and $\hat{L} = I - D^{-1/2}AD^{-1/2}$ are both real symmetric if A is symmetric and, thus, if the *graph is undirected*
- The *direction-aware* Magnetic Laplacian $L^{(q)}$ encodes the edge directions using complex numbers

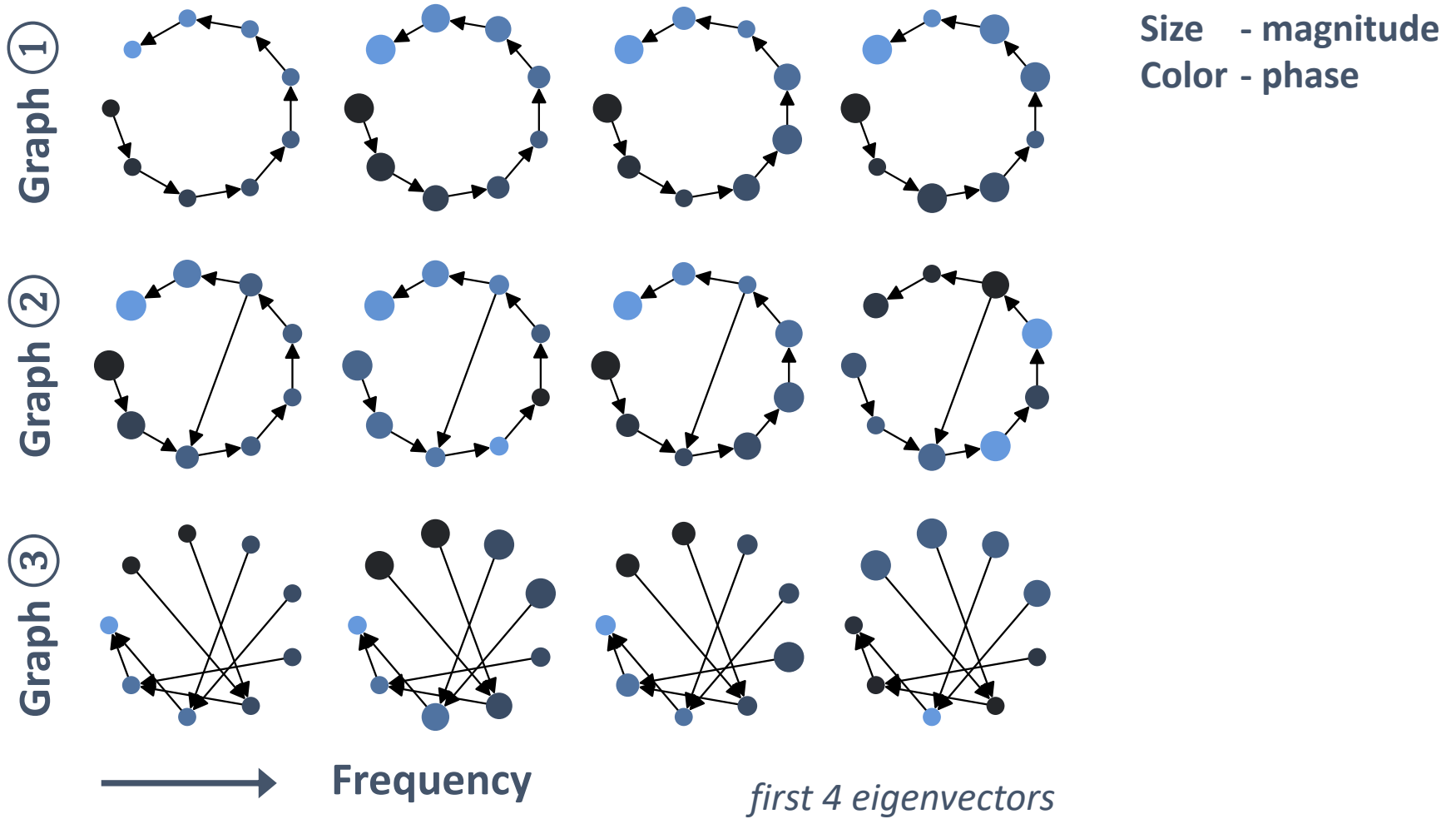
$$L^{(q)} := D_S - A_S \odot \exp(i\Theta^{(q)}) \quad \hat{L}^{(q)} := I - D_S^{-1/2}A_S D_S^{-1/2} \odot \exp(i\Theta^{(q)})$$

$$\Theta_{u,v}^{(q)} := 2\pi q(A_{u,v} - A_{v,u})$$

with symmetrized A_S/D_S , potential q , element-wise \exp/\odot , and $i = \sqrt{-1}$

- The Magnetic Laplacian is Hermitian, i.e., $L^{(q)} = (\overline{L^{(q)}})^T$ and a generalization of real symmetric matrices → eigenvectors are in \mathbb{C} but orthogonal etc.

Graph Transformers: Directed Graphs



S. Geisler, Y. Li, D. Mankowitz, AT. Cemgil, S. Günnemann, C. Paduraru: Transformers Meet Directed Graphs. ICML 2023

Summary

Graph Neural Networks

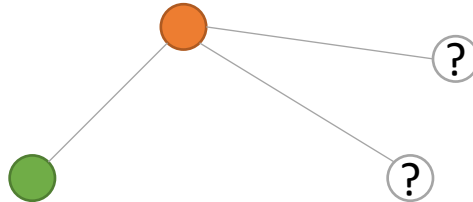
- Differentiable message passing
 - Flexible framework to apply the power of deep learning to graphs
 - The nodes aggregate information from their k-hop neighbors
 - Message passing is a generalization of convolution from grids to general graphs
- Spatial GNNs operate directly in the graph domain
- Spectral GNNs define filtering operations in the spectral domain
- Equivariant layers ensure that GNNs are independent, e.g. of the node-ordering

Advanced Topics on GNNs

- GNNs show superior predictive performance on many tasks but still suffer from many limitations regarding e.g. expressiveness, reliability and scalability
- Problems including oversmoothing and oversquashing further limit GNN performance
- Advanced architectures address existing shortcomings, examples include higher-order GNNs, PPNP and APPNP, and Graph Transformers

Questions (I)

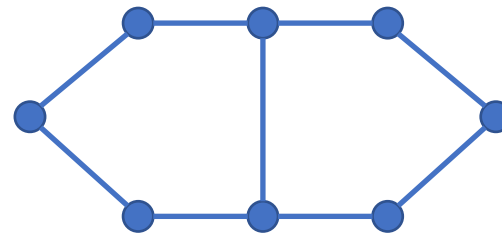
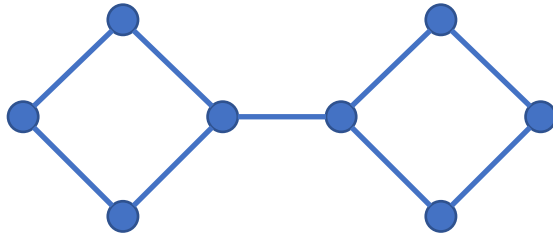
- Consider the graph below. What is the influence of the green node on the unlabeled nodes for GNNs with $K = 2$? Why?
What is the difference compared to Label Propagation?



- What could be alternative aggregation functions beside summation in the Gather step of GNNs? Which ones are permutation-invariant?
- Can you apply GNNs to vector data without a specified graph structure?

Questions (II)

- The Weisfeiler-Lehman test works similar to message passing with hashing as an aggregation function. Why don't we just use hash aggregation in GNNs to lift them to the same level of expressiveness?



- Above you see two non-isomorphic graphs that the WL test cannot distinguish
 - Can you make this counter example smaller?
- Why is oversmoothing a problem for node-level prediction models?