Introduction to Graph Neural Networks

Graph Deep Learning 2022

Daniele Grattarola

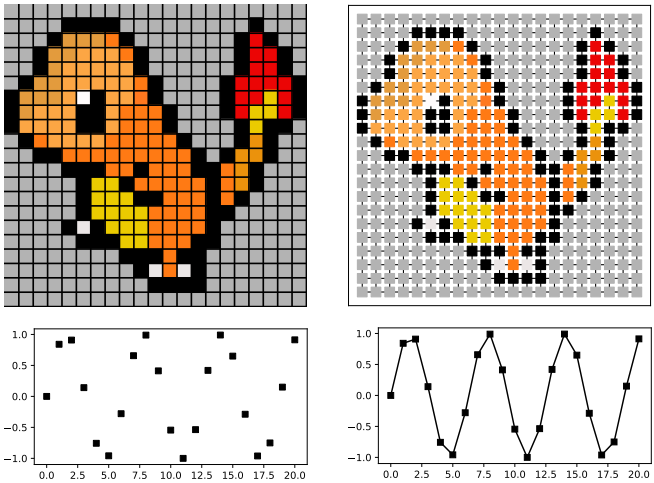
February 28, 2022

Roadmap

Things we are going to cover:

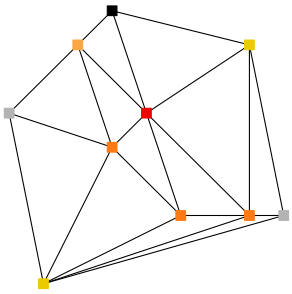
- Practical introduction to GNNs
- Message passing
- Anisotropic GNNs (attention, edge attributes)
- Spectral graph theory and GNNs

From CNNs to GNNs

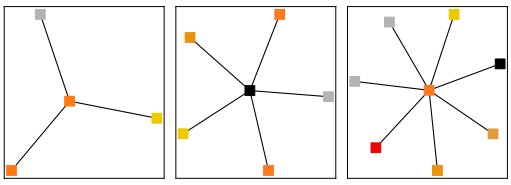


- The receptive field of a CNN reflects the **underlying grid structure**.
- The CNN has an **inductive bias** on how to process the individual pixels/timesteps/nodes.

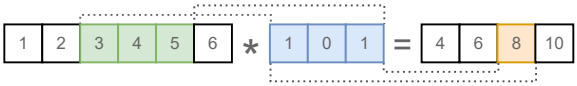
From CNNs to GNNs

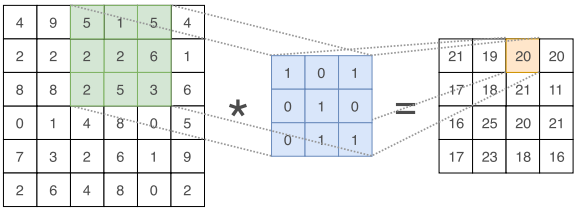


- Drop assumptions about underlying structure: it is now an **input of the problem**.
- The only thing we know: the representation of a node depends on its **neighbors**.



Discrete convolution



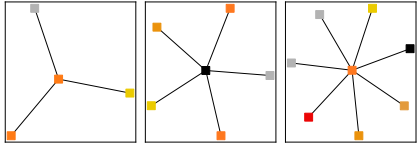


Discrete convolution:

$$(f \star g)[n] = \sum_{m=-M}^{M} f[n-m]g[m]$$

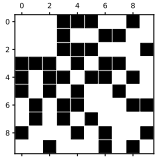
Problems:

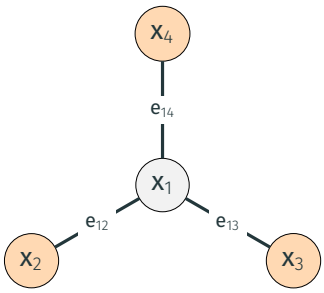
- Variable degree of nodes
- Orientation



Notation recap

- Graph: nodes connected by edges;
- $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$, $\mathbf{x}_i \in \mathbb{R}^F$, node attributes or "graph signal";
- $\mathbf{e}_{ij} \in \mathbb{R}^S$, edge attribute for edge $i \to j$;
- $\mathbf{A}$, $N \times N$ adjacency matrix;
- $\mathbf{D} = \mathrm{diag}([d_1, \ldots, d_N])$, diagonal degree matrix;
- $\mathbf{L} = \mathbf{D} - \mathbf{A}$, Laplacian;
- **Reference operator R**: $\mathbf{r}_{ij} \neq 0$ if $\exists i \to j$
  Note: for us, $\mathbf{R}$ is symmetric.





A quick recipe for a local learnable filter

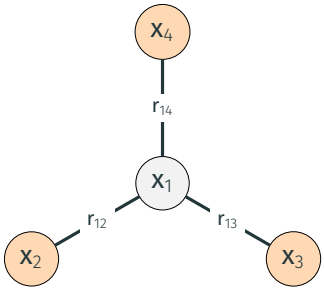
Applying $\mathbf{R}$ to node attributes $\mathbf{X}$ is a **local** action:

$$(\mathbf{RX})_i = \sum_{j=1}^{N} \mathbf{r}_{ij} \cdot \mathbf{x}_j = \sum_{j \in \mathcal{N}(i)} \mathbf{r}_{ij} \cdot \mathbf{x}_j$$

Instead of having a different weight for each neighbor, **we share weights among nodes in the same neighborhood:**

$$\mathbf{X}' = \mathbf{RX}\Theta$$

where $\Theta \in \mathbb{R}^{F \times F'}$.

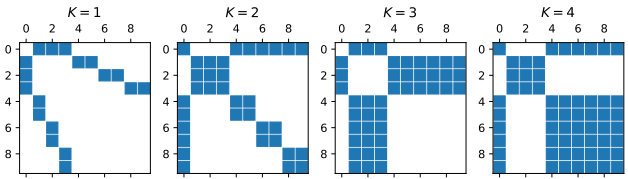


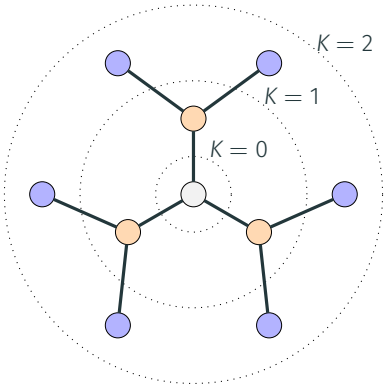
Powers of R

Let's consider the effect of applying $\mathbf{R}^2$ to $\mathbf{X}$:

$$(\mathbf{RRX})_i = \sum_{j \in \mathcal{N}(i)} \mathbf{r}_{ij} (\mathbf{RX})_j = \sum_{j \in \mathcal{N}(i)} \sum_{k \in \mathcal{N}(j)} \mathbf{r}_{ij} \cdot \mathbf{r}_{jk} \cdot \mathbf{x}_k$$

Key idea: by applying $\mathbf{R}^K$ we **aggregate information from the K-th order neighborhood** of a node.

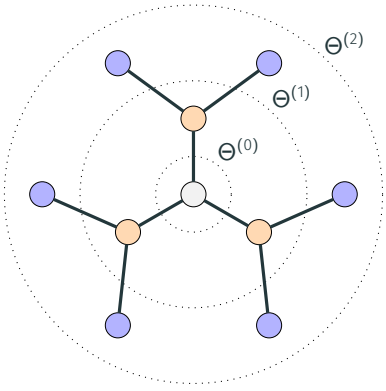




Polynomials of R

To cover all neighbors of order 0 to K, we can just take a polynomial with weights $\Theta^{(k)}$:

$$X' = \sum_{k=0}^{K} R^k X \Theta^{(k)}$$



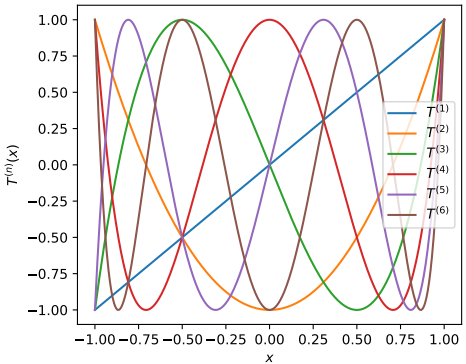
Chebyshev polynomials [1]

A recursive definition using Chebyshev polynomials:

$$
\begin{aligned}
\mathbf{T}^{(0)} &= \mathbf{I} \\
\mathbf{T}^{(1)} &= \tilde{\mathbf{L}} \\
\mathbf{T}^{(k)} &= 2 \cdot \tilde{\mathbf{L}} \cdot \mathbf{T}^{(k-1)} - \mathbf{T}^{(k-2)}
\end{aligned}
$$

Where $\tilde{\mathbf{L}} = \frac{2\mathbf{L}_n}{\lambda_{\max}} - \mathbf{I}$ and $\mathbf{L}_n = \mathbf{I} - \underbrace{\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}}_{\mathbf{A}_n}$

Layer: $\mathbf{X}' = \sum_{k=0}^{K} \mathbf{T}^{(k)}\mathbf{X}\Theta^{(k)}$



[1] M. Defferrard *et al.*, "Convolutional neural networks on graphs with fast localized spectral filtering," 2016.

Graph convolutional networks [2]

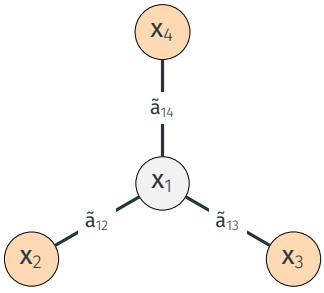
Polynomial of order $K \to K$ layers of order 1;

Three simplifications:

1. $\lambda_{\max} = 2 \to \tilde{\mathbf{L}} = \dfrac{2\mathbf{L}_n}{\lambda_{\max}} - \mathbf{I} = -\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = -\mathbf{A}_n$
2. $K = 1 \to \mathbf{X}' = \mathbf{X}\Theta^{(0)} - \mathbf{A}_n\mathbf{X}\Theta^{(1)}$
3. $\Theta = \Theta^{(0)} = -\Theta^{(1)}$

Layer: $\mathbf{X}' = \underbrace{(\mathbf{I} + \mathbf{A}_n)}_{\tilde{\mathbf{A}}}\mathbf{X}\Theta = \tilde{\mathbf{A}}\mathbf{X}\Theta$

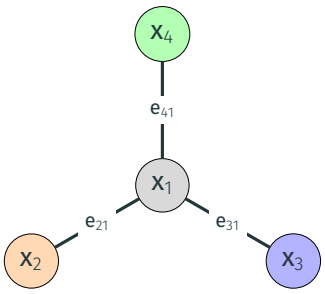
For stability: $(\mathbf{I} + \mathbf{A}_n) \to \mathbf{D}^{-1/2}(\mathbf{I} + \mathbf{A})\mathbf{D}^{-1/2}$



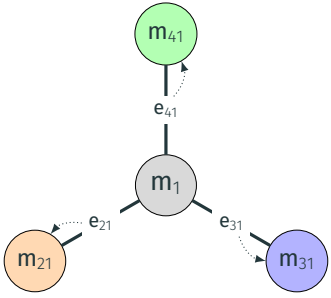
[2] T. N. Kipf *et al.*, "Semi-supervised classification with graph convolutional networks," 2016.

Message passing

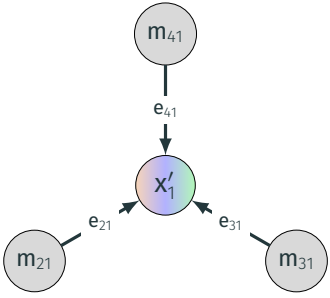
Message passing neural networks [3]



Graph.



Messages.



Propagation.

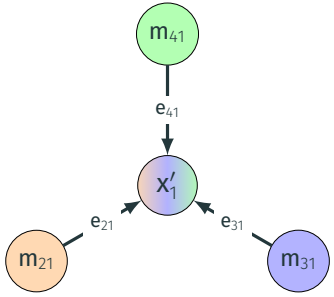
[3] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message passing neural networks [3]

A general scheme for message-passing networks:

$$\mathbf{x}_i' = \gamma\left(\mathbf{x}_i, \square_{j\in\mathcal{N}(i)}\, \phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}\right)\right),$$

- ϕ: **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;
- $\square_{j\in\mathcal{N}(i)}$: **aggregation function** (sum, average, max, or something else...);
- γ: **update function**, final transformation to obtain new attributes after aggregating messages.



[3] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

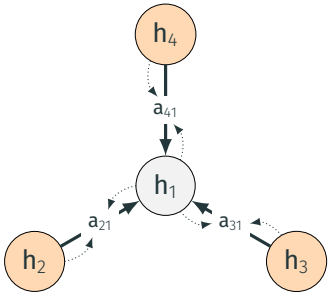
Anisotropic GNNs

Graph attention networks [4]

1. Messages: $\mathbf{h}_i = \Theta_f \mathbf{x}_i$ with $\Theta_f \in \mathbb{R}^{F' \times F}$.
2. Attention between neighbors:
   2.1 Score: $\mathbf{e}_{ij} = \sigma\left(\theta_a^\top [\mathbf{h}_i \parallel \mathbf{h}_j]\right)$, with $\theta_a \in \mathbb{R}^{2F'}$.
   2.2 Normalize with Softmax: $\mathbf{a}_{ij} = \dfrac{\exp(\mathbf{e}_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})}$
3. Aggregate using the attention coefficients:

$$\mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \mathbf{a}_{ij} \mathbf{h}_j$$

$\parallel$ indicates concatenation



[4] P. Velickovic *et al.*, "Graph attention networks," 2017.

Edge-conditioned convolution [5]

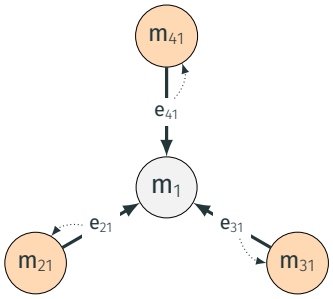
Key idea: incorporate edge attributes into the messages.

Use a MLP $\phi : \mathbb{R}^S \to \mathbb{R}^{FF'}$ to **generate weights**:

$$\Theta^{(ji)} = \text{reshape}(\phi(\mathbf{e}_{ji}))$$

Use the edge-dependent weights to compute messages:

$$\mathbf{x}'_i = \Theta^{(i)}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \Theta^{(ji)}\mathbf{x}_j + \mathbf{b}$$



[5] M. Simonovsky *et al.*, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," 2017.

So which GNN do I use?

GCNConv *Kipf & Welling*	**ChebConv** *Defferrard et al.*	**GraphSageConv** *Hamilton et al.*	**ARMAConv** *Bianchi et al.*
ECCConv *Simonovsky & Komodakis*	**GATConv** *Velickovic et al.*	**GCSConv** *Bianchi et al.*	**APPNPConv** *Klicpera et al.*
GINConv *Xu et al.*	**DiffusionConv** *Li et al.*	**GatedGraphConv** *Li et al.*	**AGNNConv** *Thekumparampil et al.*
TAGConv *Du et al.*	**CrystalConv** *Xie & Grossman*	**EdgeConv** *Wang et al.*	**MessagePassing** *Gilmer et al.*

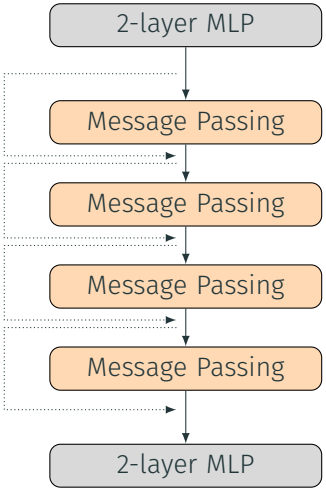
A good recipe [6]

Message passing scheme:

- Message: $\mathbf{m}_{ji} = \text{PReLU}\left(\text{BatchNorm}\left(\Theta \mathbf{x}_j + \mathbf{b}\right)\right)$
- Aggregate: $\mathbf{m}^{\text{agg}} = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ji}$
- Update: $\mathbf{x}' = \mathbf{x} \,||\, \mathbf{m}^{\text{agg}}$;

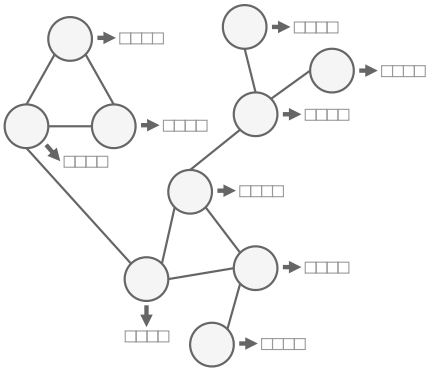
Architecture:

- Pre- and post-process node features using 2-layer MLPs;
- 4-6 message passing steps;

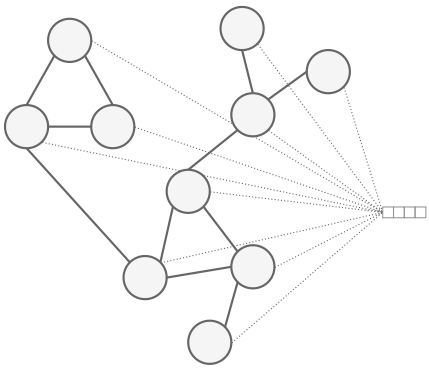


[6] J. You *et al.*, "Design space for graph neural networks," 2020.

How do we use this?



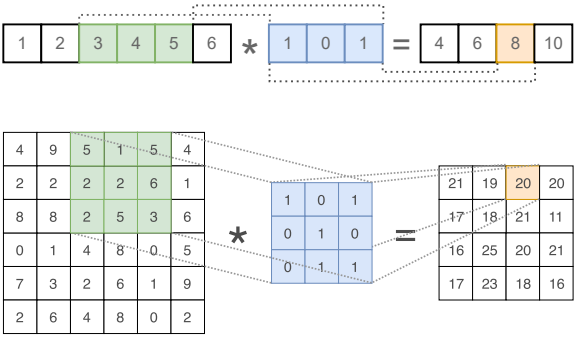
Node-level learning.
(e.g., social networks)



Graph-level learning.
(e.g., molecules)

Graph convolution

Discrete convolution



Recall: CNNs compute a discrete convolution

$$(f \star g)[n] = \sum_{m=-M}^{M} f[n-m]g[m] \quad (1)$$

Convolution theorem

Given two functions f and g, their convolution $f \star g$ can be expressed as:

$$f \star g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\} \tag{2}$$

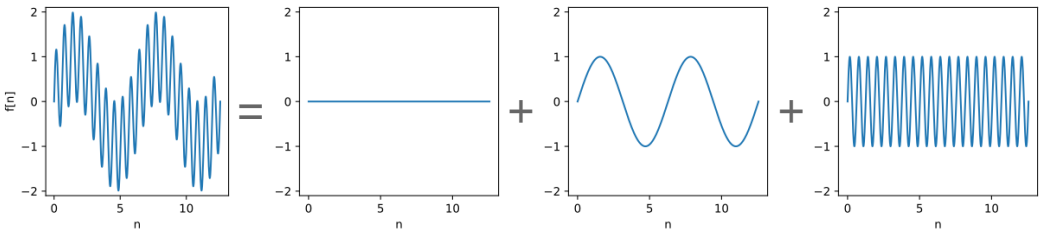
Where $\mathcal{F}$ is the **Fourier transform** and $\mathcal{F}^{-1}$ its inverse.

Can we use this major property?

What is the Fourier transform?

Key intuition – we are representing a function in a different basis.

$$\mathcal{F}\{f\}[k] = \hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{-i\frac{2\pi}{N}kn} \qquad \mathcal{F}^{-1}\{\hat{f}\}[n] = f[n] = \frac{1}{N}\sum_{k=0}^{N-1} \hat{f}[k] e^{i\frac{2\pi}{N}kn}$$

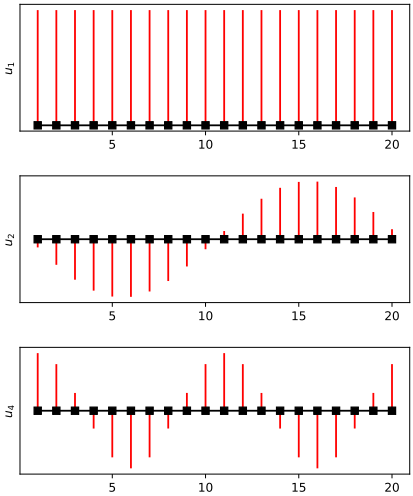


From FT to GFT

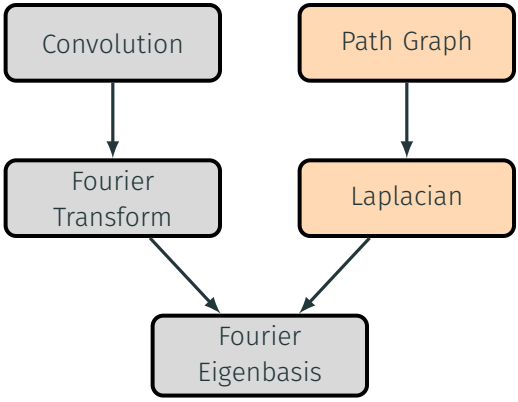
The eigenvectors of the Laplacian for a path graph can be obtained analytically:

$$\mathbf{u}_k[n] = \begin{cases} 1, & \text{for } k = 0 \\ e^{i\pi(k+1)n/N}, & \text{for odd } k, k < N-1 \\ e^{-i\pi kn/N}, & \text{for even } k, k > 0 \\ cos(\pi n), & \text{for odd } k, k = N-1 \end{cases}$$

Looks familiar?



From FT to GFT



- Drop the “grid” assumption
- Replace $e^{-i\frac{2\pi}{N}kn}$ with generic $\mathbf{u}_k[n]$:

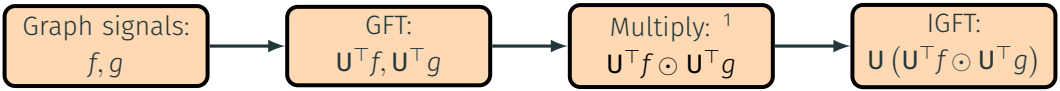
$$\mathcal{F}_G\{f\}[k] = \sum_{n=0}^{N-1} f[n]\mathbf{u}_k[n]$$

- GFT: $\mathcal{F}_G\{f\} = \hat{f} = \mathbf{U}^\top f$;
- IGFT: $\mathcal{F}_G^{-1}\{\hat{f}\} = f = \mathbf{U}\hat{f}$

Graph convolution

Recall:

- Convolution theorem: $f \star g = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{f\right\} \cdot \mathcal{F}\left\{g\right\}\right\}$
- Spectral theorem: $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \sum_{i=0}^{N-1} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$



Graph filter: $\mathbf{U}\left(\mathbf{U}^\top f \odot \mathbf{U}^\top g\right) = \mathbf{U} \cdot \underbrace{\operatorname{diag}(\mathbf{U}^\top g)}_{g(\mathbf{\Lambda})} \cdot \mathbf{U}^\top f = \underbrace{\mathbf{U} \cdot g(\mathbf{\Lambda}) \cdot \mathbf{U}^\top}_{g(\mathbf{L})} f = g(\mathbf{L}) f$

[1] $\odot$ indicates element-wise multiplication

Spectral GCNs

Spectral GCNs

A first idea [7]: transformation of **each individual eigenvalue** is learned with a free parameter θ_i.

Problems:

- $O(N)$ parameters;
- not **localized** in node space (the only thing that we want);
- $\mathbf{U} \cdot g(\Lambda) \cdot \mathbf{U}^\top$ costs $O(N^2)$;

$$g_\theta(\Lambda) = \begin{bmatrix} \theta_0 & & & & \\ & \theta_1 & & & \\ & & \ddots & & \\ & & & \theta_{N-2} & \\ & & & & \theta_{N-1} \end{bmatrix}$$

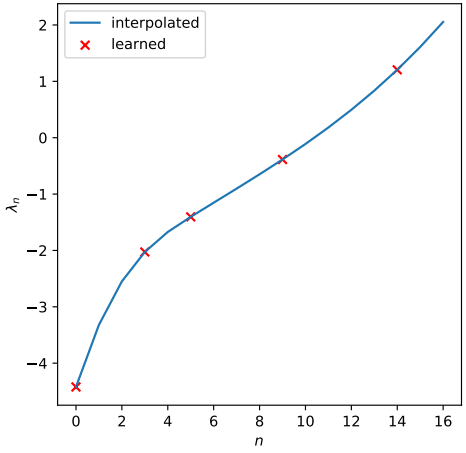
[7] J. Bruna *et al.*, "Spectral networks and locally connected networks on graphs," 2013.

Spectral GCNs

Better idea [7]:

- **Localized** in node domain $\leftrightarrow$ **smooth** in spectral domain;
- Learn only a few parameters θ_i;
- **Interpolate** the other eigenvalues using a smooth **cubic spline**;

Localized and $O(1)$ parameters, but **multiplying by U twice is still expensive**.

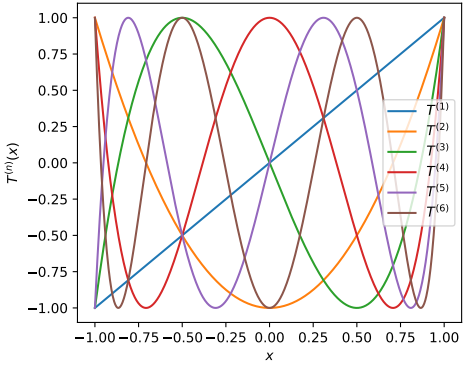


[7] J. Bruna *et al.*, "Spectral networks and locally connected networks on graphs," 2013.

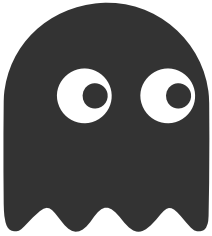
Chebyshev polynomials [1]

The same recursion is used to filter eigenvalues:

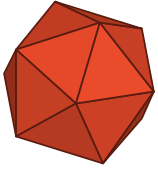
$$\mathbf{T}^{(0)} = \mathbf{I}$$
$$\mathbf{T}^{(1)} = \tilde{\mathbf{\Lambda}}$$
$$\mathbf{T}^{(k)} = 2 \cdot \tilde{\mathbf{\Lambda}} \cdot \mathbf{T}^{(k-1)} - \mathbf{T}^{(k-2)}$$



[1] M. Defferrard *et al.*, "Convolutional neural networks on graphs with fast localized spectral filtering," 2016.



Spektral



PyTorch geometric

References i

[1] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[3] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[4] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[5] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

References ii

[6] J. You, R. Ying, and J. Leskovec, “Design space for graph neural networks,” *arXiv preprint arXiv:2011.08843*, 2020.

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

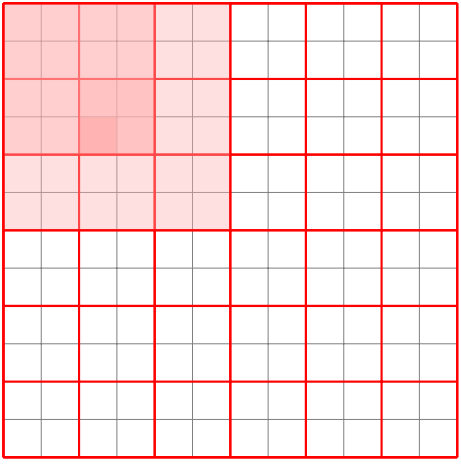
Pooling in Graph Neural Networks

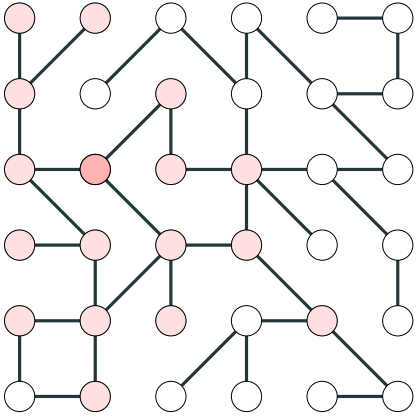
Graph Deep Learning 2022

Daniele Grattarola

February 28, 2022

Pooling in CNNs





Roadmap

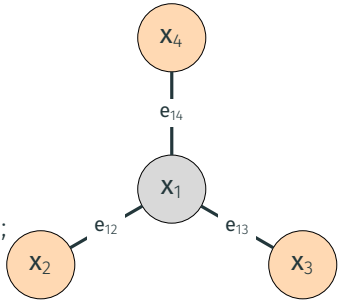
Things we are going to cover:

- A “message passing” for pooling
- Methods
- Global pooling
- Open questions

Source: “Understanding pooling in graph neural networks”, Grattarola et al., 2021
https://arxiv.org/abs/2110.05292

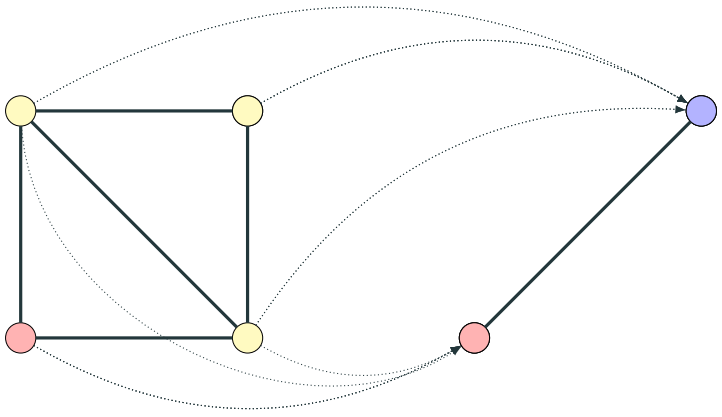
Notation

- Graph: nodes connected by edges;
- $\mathbf{A}$, adjacency matrix of shape $N \times N$;
- $\mathbf{D} = \mathrm{diag}([d_1, \ldots, d_N])$, diagonal degree matrix;
- $\mathbf{L} = \mathbf{D} - \mathbf{A}$, Laplacian matrix;
- $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$, $\mathbf{x}_i \in \mathbb{R}^F$, node attributes or "graph signal";
- $\mathbf{e}_{ij} \in \mathbb{R}^S$, edge attribute for edge $i \to j$;



Graph pooling by example

Strategy 1: aggregate same attributes (Candy Crush pooling). Strategy 2: aggregate cliques. Strategy 3: keep only some types/colors.



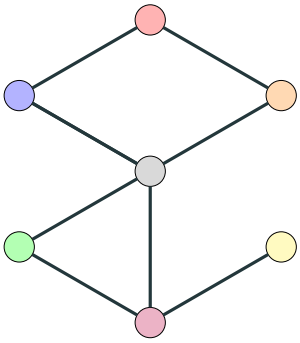
Three main questions [1]

1. How to identify **groups of related nodes**?
2. How to get **new node attributes** from the groups?
3. How to **connect** the new nodes?

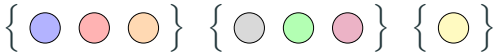
[1] D. Grattarola *et al.*, "Understanding Pooling in Graph Neural Networks," 2021 (In preparation).

Step 1: Select

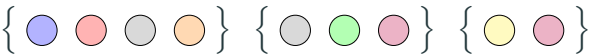
Selecting nodes



Example 1: partition.



Example 2: cover (possible overlaps).

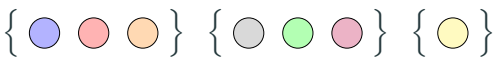


Example 3: sparse.

Selecting nodes

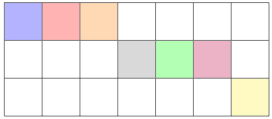
The **selection** stage computes K **supernodes**:

$$\textsc{Sel} : \mathcal{G} \mapsto \mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_K\}.$$

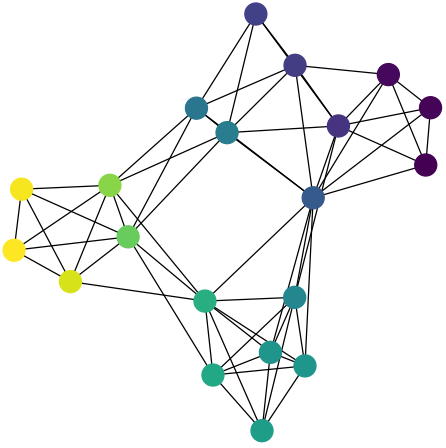


Each supernode is a set of nodes associated with a score:

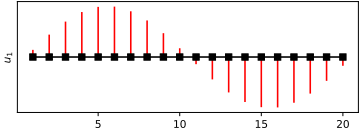
$$\mathcal{S}_k = \{(\mathbf{x}_i, \mathbf{s}_i) \mid \mathbf{s}_i \in \mathbb{R}_{>0}\},$$

$\mathbf{S} =$  $\in \mathbb{R}^{K \times N}$

Spectral clustering [3]



The low-frequency eigenvectors naturally cluster the nodes.

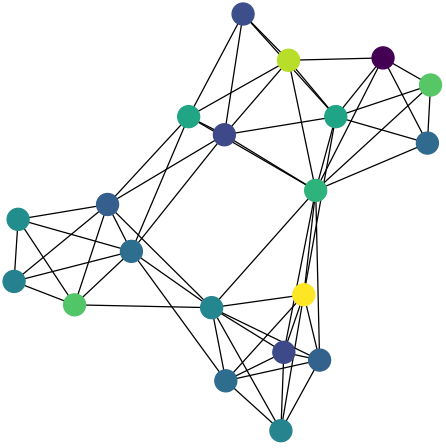


Idea: run k-means clustering (or similar) using the first few eigenvectors.

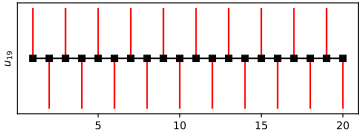
[2] J. Shi *et al.*, "Normalized cuts and image segmentation," 2000.

[3] U. Von Luxburg, "A tutorial on spectral clustering," 2007.

Node decimation [5]



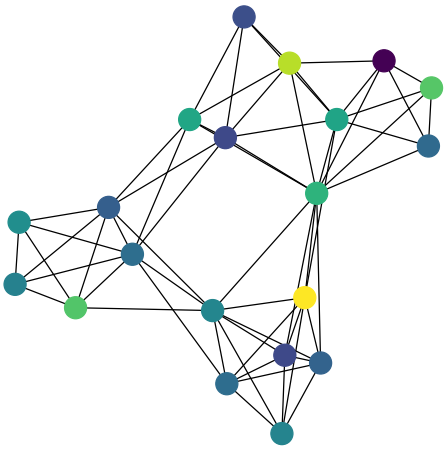
Alternative: use the highest-frequency eigenvector to do something similar to a regular subsampling.



[4] L. Palagi *et al.*, "Computational approaches to max-cut," 2012.

[5] F. M. Bianchi *et al.*, *Hierarchical Representation Learning in Graph Neural Networks with Node Decimation Pooling*, 2019.

Some problems



Problems with spectral methods:

- Computing eigenvectors is **expensive** ($O(N^3)$);
- They do not consider **attributes**.

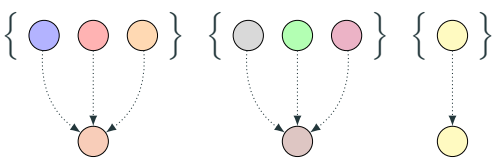
But we get the general idea...

Step 2: Reduce

Reducing supernodes

The **reduction** stage aggregates the supernodes in a **permutation-invariant** way:

$$\textsc{Red} : \mathcal{G}, \mathcal{S}_k \mapsto \mathbf{x}'_k$$



Typical approach is to take a **weighted sum** (weights given by the scores in the supernodes):

$$\mathbf{X}' = \mathbf{S}\mathbf{X} \quad (\in \mathbb{R}^{K \times F})$$

Step 3: Connect

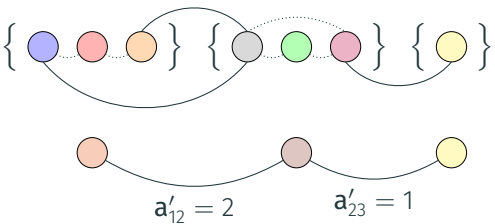
Connecting supernodes

The **connection** function decides whether two supernodes are connected (and, in case, computes the associated attributes):

$$\textsc{Con} : \mathcal{G}, \mathcal{S}_k, \mathcal{S}_l \mapsto \mathbf{e}'_{kl}$$

Typical approach is again to take a **weighted sum** of edges between two supernodes:

$$\mathbf{A}' = \mathbf{S}\mathbf{A}\mathbf{S}^\top \quad (\in \mathbb{R}^{K \times K})$$



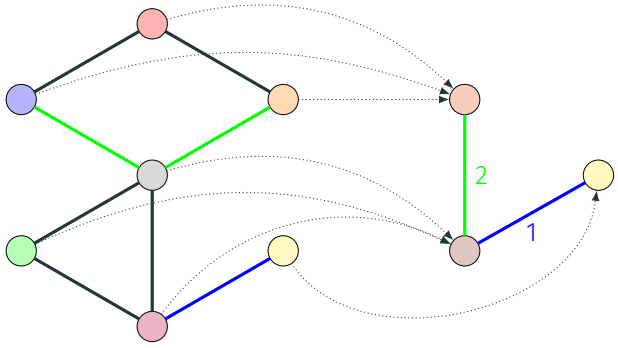
Select, Reduce, Connect [1]

Putting everything together:

$$\underbrace{\mathcal{S} = \{\mathcal{S}_k\}_{k=1:K} = \textsc{Sel}(\mathcal{G})}_{\text{Selection}};$$

$$\underbrace{\mathcal{X}' = \{\textsc{Red}(\mathcal{G}, \mathcal{S}_k)\}_{k=1:K}}_{\text{Reduction}};$$

$$\underbrace{\mathcal{E}' = \{\textsc{Con}(\mathcal{G}, \mathcal{S}_k, \mathcal{S}_l)\}_{k,l=1:K}}_{\text{Connection}};$$



[1] D. Grattarola *et al.*, "Understanding Pooling in Graph Neural Networks," 2021 (In preparation).

Methods

Pooling methods

A few ideas:

1. **Graclus** [6]: visit nodes randomly, merge pairs that maximize $\frac{\mathbf{a}_{ij}}{w_i} + \frac{\mathbf{a}_{ij}}{w_j}$; [1]

   In [7], they reduce supernodes with element-wise max.
2. **Clique Pooling** [8]: merge together cliques.
3. **LaPool** [9]: select "leaders" that have higher local variation $\|\mathbf{LX}\|$ w.r.t. all their neighbors. Create clusters by assigning nodes to nearest leader.

[6] I. S. Dhillon *et al.*, "Weighted graph cuts without eigenvectors a multilevel approach," 2007.

[7] M. Defferrard *et al.*, "Convolutional neural networks on graphs with fast localized spectral filtering," 2016.

[8] E. Luzhnica *et al.*, "Clique pooling for graph classification," 2019.

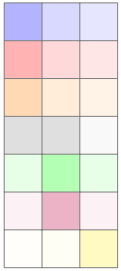
[9] E. Noutahi *et al.*, "Towards Interpretable Sparse Graph Representation Learning with Laplacian Pooling," 2019.

Learning to pool

Key idea: learn to output $\mathbf{S}^\top$ by giving node features $\mathbf{X}$ as input to a neural network.

- **DiffPool** [10]: GNN for $\mathbf{S}^\top$, regularize with "link prediction" loss;
- **MinCutPool** [11]: MLP for $\mathbf{S}^\top$, regularize with "minimum cut" loss (same objective as spectral clustering);
- **Deep Graph Mapper** [12]: combine Mapper [13] and GCN [14] to compute clusters.

$$\phi(\mathbf{X}) = \mathbf{S}^\top = \quad \in \mathbb{R}^{N \times K}$$



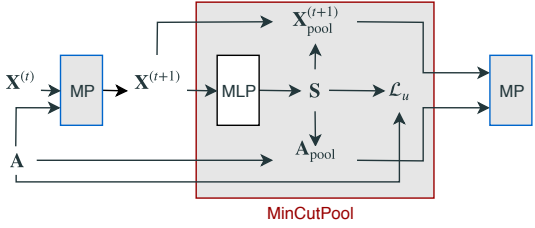
[10] R. Ying *et al.*, "Hierarchical Graph Representation Learning with Differentiable Pooling," 2018.

[11] F. M. Bianchi *et al.*, "Mincut pooling in Graph Neural Networks," 2019.

[12] C. Bodnar *et al.*, "Deep Graph Mapper: Seeing Graphs through the Neural Lens," 2020.

MinCut Pooling [11]

- Select: $\mathbf{S}^\top = \text{MLP}(\mathbf{X})$
- Reduce: $\mathbf{X}' = \mathbf{SX}$
- Connect: $\mathbf{A}' = \mathbf{SAS}^\top$
- MinCut loss: $\mathcal{L}_c = -\frac{Tr(\mathbf{SAS}^\top)}{Tr(\mathbf{SDS}^\top)}$
- Orthogonality loss:
$$\mathcal{L}_o = \left\| \frac{\mathbf{SS}^\top}{\|\mathbf{SS}^\top\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}} \right\|_F$$

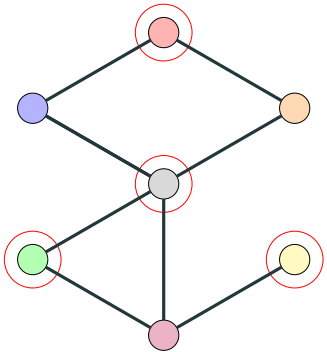


[11] F. M. Bianchi *et al.*, "Mincut pooling in Graph Neural Networks," 2019.

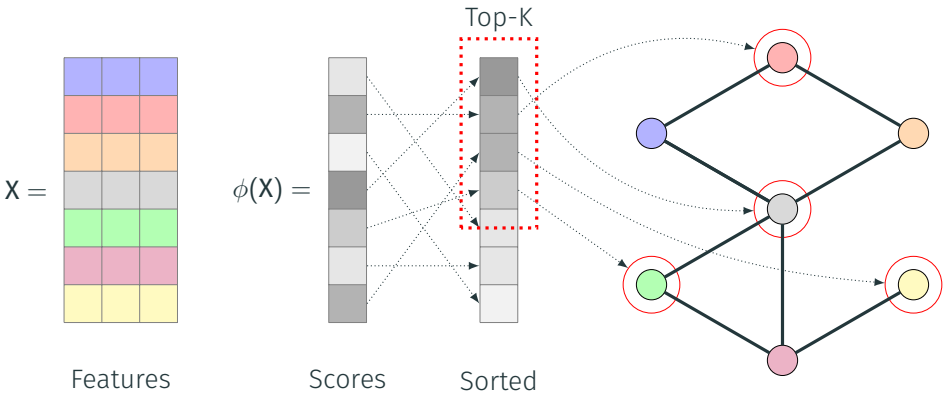
Top-K methods

Problem: computing **S** with neural network is likely to yield a very **dense** matrix.

Can we learn a sparse selection?



Top-K methods



Top-K methods

Different ways of computing the selection indices $\mathbf{i}$:

- Select with a simple linear projection $\theta \in \mathbb{R}^F$ [15];
- Select with a GNN [16];
- Train the selection with a supervised objective (needs ground truth for which nodes to keep) [17].

[15] S. J. Hongyang Gao, "Graph U-Net," 2019.

[16] J. Lee *et al.*, "Self-Attention Graph Pooling," 2019.

[17] B. Knyazev *et al.*, "Understanding attention in graph neural networks," 2019.

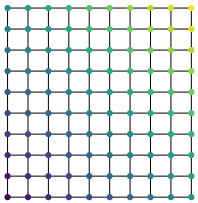
Top-K methods

Reduce: $\mathbf{X}' = \mathbf{X}_i$ - Connect: $\mathbf{A}' = \mathbf{A}_{i,i}$

Problems:

- Top-k selection is **non-differentiable** (no way of training ϕ).
  Solved by **gating** (multiplying) the node attributes with the scores.
- Graph is likely to be **disconnected** or simply **cut off** (like in the image on the right).
  Not really solvable...

Original



Top-K

Main properties of pooling operators

- **Dense vs. Sparse**: how many nodes are selected for the supernodes;
- **Fixed vs. Adaptive**: how many supernodes does the selection compute;
- **Trainable vs. Non-trainable**: learn to pool from data or not;

Global pooling

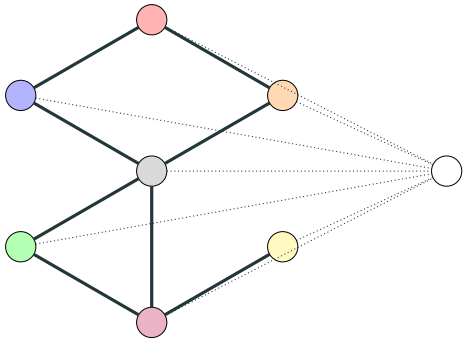
Global Pooling

In CNNs, after convolution, we usually **flatten** out the images to give a vector as input to a MLP:

1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

The graph equivalent must be **invariant to permutations** of the nodes:



Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;
- Weighted sum with attention [18];
- Sum and then apply a neural network [19];

[18] Y. Li *et al.*, "Gated graph sequence neural networks," 2015.

[19] N. Navarin *et al.*, "Universal readout for graph convolutional neural networks," 2019.

Open questions

Open questions

- Does pooling really work?
  - Dense selection + message passing + small graphs is a bad idea [20]
  - Which tasks benefit from pooling *a priori*?
  - Problems with inherent hierarchy?
- Can we make a pooling layer that is dense, trainable, and adaptive?

[20] D. Mesquita *et al.*, "Rethinking pooling in graph neural networks," 2020.

References i

[1] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks,", 2021 (In preparation).

[2] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[3] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[4] L. Palagi, V. Piccialli, F. Rendl, G. Rinaldi, and A. Wiegele, "Computational approaches to max-cut," in *Handbook on semidefinite, conic and polynomial optimization*, Springer, 2012, pp. 821–847.

[5] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, *Hierarchical representation learning in graph neural networks with node decimation pooling*, 2019. arXiv: *1910.11436 [cs.LG]*.

References ii

[6] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[8] E. Luzhnica, B. Day, and P. Lio, "Clique pooling for graph classification," *International Conference of Learning Representations (ICLR) – Representation Learning on Graphs and Manifolds workshop*, 2019.

[9] E. Noutahi, D. Beani, J. Horwood, and P. Tossou, "Towards interpretable sparse graph representation learning with laplacian pooling," *arXiv preprint arXiv:1905.11577*, 2019.

References iii

[10] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *arXiv preprint arXiv:1806.08804*, 2018.

[11] F. M. Bianchi, D. Grattarola, and C. Alippi, "Mincut pooling in graph neural networks," *CoRR*, vol. abs/1907.00481, 2019. arXiv: *1907.00481*. [Online]. Available: *http://arxiv.org/abs/1907.00481*.

[12] C. Bodnar, C. Cangea, and P. Liò, "Deep graph mapper: Seeing graphs through the neural lens," *arXiv preprint arXiv:2002.03864*, 2020.

[13] G. Singh, F. Mémoli, and G. E. Carlsson, "Topological methods for the analysis of high dimensional data sets and 3d object recognition.," in *SPBG*, 2007, pp. 91–100.

[14] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2016.

References iv

[15] S. J. Hongyang Gao, "Graph u-net," *Submitted to ICLR*, 2019.

[16] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," *arXiv preprint arXiv:1904.08082*, 2019.

[17] B. Knyazev, G. W. Taylor, and M. R. Amer, "Understanding attention in graph neural networks," *CoRR*, vol. abs/1905.02850, 2019. arXiv: *1905.02850*. [Online]. Available: *http://arxiv.org/abs/1905.02850*.

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

[19] N. Navarin, D. Van Tran, and A. Sperduti, "Universal readout for graph convolutional neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–7.

[20] D. Mesquita, A. Souza, and S. Kaski, “Rethinking pooling in graph neural networks,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.