Università della Svizzera italiana

Graph Neural Networks

Outline

Part 1. Neural Message Passing

- Towards graph convolutions
- Message passing

Part 2. Pooling on Graphs

- Select, Reduce, Connect
- Pooling methods
- Global pooling

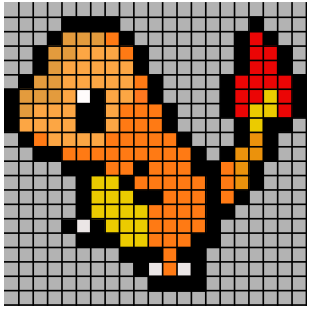
Part 3. Coding GNNs

- Python libraries
- Demo: node classification with PyTorch Geometric

Towards graph convolutions

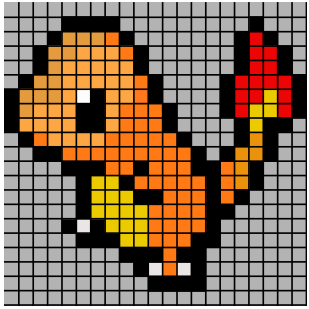
Convolutions on images

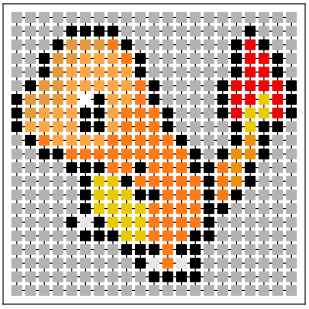
Consider the convolution operation in Convolutional Neural Networks (CNNs).



Convolutions on images

Consider the convolution operation in Convolutional Neural Networks (CNNs).





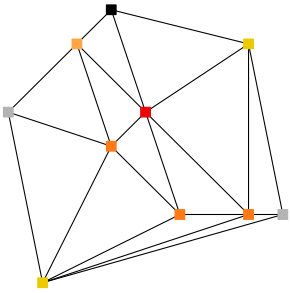
- The receptive field of a CNN reflects the **underlying grid structure**.
- The CNN has an **inductive bias** on how to process the individual pixels/timesteps/nodes.

Going beyond grids

- Unfortunately, not everything can be cast into a grid...

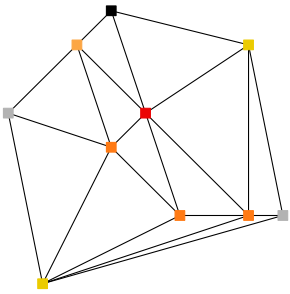
Going beyond grids

- Unfortunately, not everything can be cast into a grid...
- ...but **graphs** are a nice representations for **irregular structures**.



Going beyond grids

- Unfortunately, not everything can be cast into a grid...
- ...but **graphs** are a nice representations for **irregular structures**.
- Can we generalize the concept of convolution on graphs?

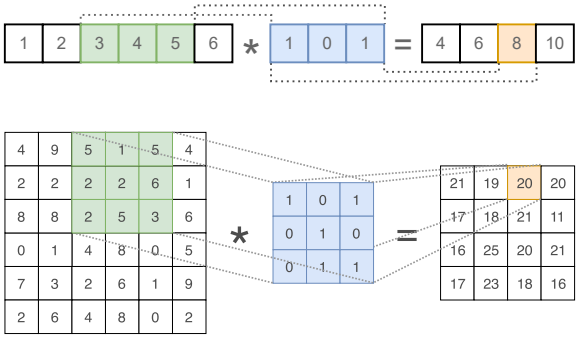


Convolution on Euclidean spaces

The discrete convolution of CNNs:

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

operates on Euclidean spaces.

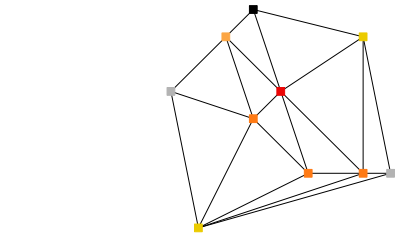


Convolution on non-Euclidean spaces

Moving to non-Euclidean spaces is not that trivial.

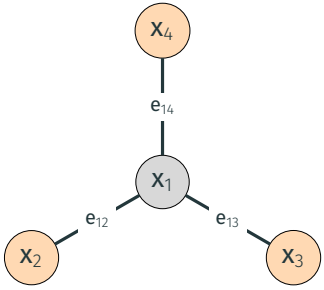
Challenges:

- Variable number of neighbors
- Loss of orientation



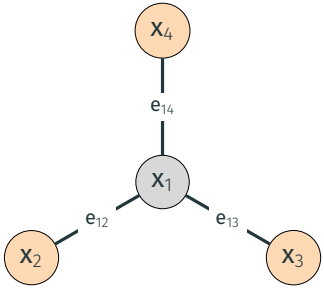
Notation

- Graph $\mathcal{G}\langle\mathcal{V},\mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$



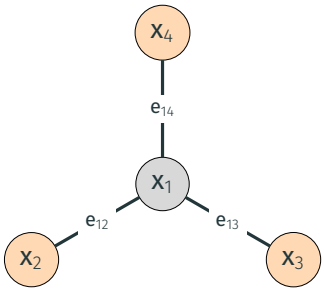
Notation

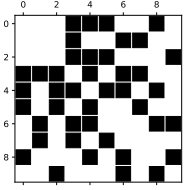
- Graph $\mathcal{G}\langle\mathcal{V}, \mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N \times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector



Notation

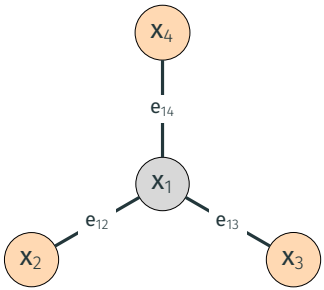
- Graph $\mathcal{G}\langle\mathcal{V}, \mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N \times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N \times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i, j) \in \mathcal{E}$

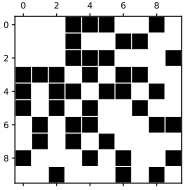




Notation

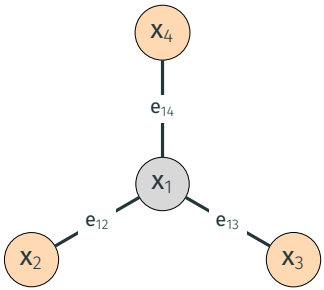
- Graph $\mathcal{G}\langle\mathcal{V},\mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N\times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N\times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i,j) \in \mathcal{E}$
- $\mathbf{D} = diag\,(\mathbf{A}\mathbf{1}_N) \in \mathbb{R}^{N\times N}$, degree matrix

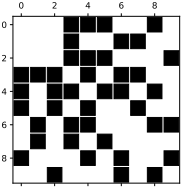




Notation

- Graph $\mathcal{G}\langle\mathcal{V}, \mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N \times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N \times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i, j) \in \mathcal{E}$
- $\mathbf{D} = diag\,(\mathbf{A}\mathbf{1}_N) \in \mathbb{R}^{N \times N}$, degree matrix
- $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$, edge attribute for edge $(i, j) \in \mathcal{E}$

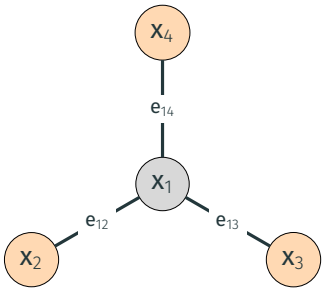


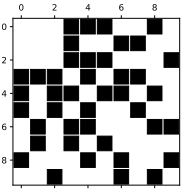


Notation

- Graph $\mathcal{G}\langle\mathcal{V},\mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N\times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N\times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i,j) \in \mathcal{E}$
- $\mathbf{D} = diag\,(\mathbf{A}\mathbf{1}_N) \in \mathbb{R}^{N\times N}$, degree matrix
- $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$, edge attribute for edge $(i,j) \in \mathcal{E}$

In the following, we focus on undirected graphs: $\mathbf{A} = \mathbf{A}^\top$



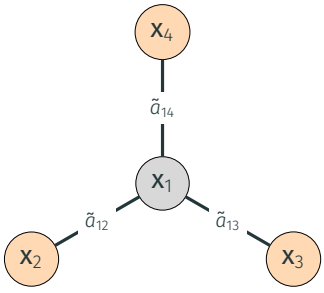


Graph Shift Operator

Graph Shift Operator [1]

A matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is called a **Graph Shift Operator (GSO)** if it satisfies:

$$\tilde{a}_{ij} = 0 \text{ for } (i,j) \notin \mathcal{E} \text{ and } i \neq j.$$



[1] A. Sandryhaila *et al.*, "Discrete signal processing on graphs," 2013.

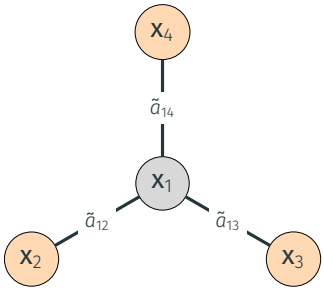
Graph Shift Operator

Graph Shift Operator [1]

A matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is called a **Graph Shift Operator (GSO)** if it satisfies:

$$\tilde{a}_{ij} = 0 \text{ for } (i,j) \notin \mathcal{E} \text{ and } i \neq j.$$

A GSO $\tilde{\mathbf{A}}$ can be viewed as some function of the adjacency matrix $\mathbf{A}$.



[1] A. Sandryhaila *et al.*, "Discrete signal processing on graphs," 2013.

Graph Shift Operator

Graph Shift Operator [1]

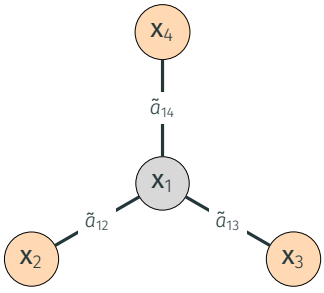
A matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is called a **Graph Shift Operator (GSO)** if it satisfies:

$$\tilde{a}_{ij} = 0 \text{ for } (i,j) \notin \mathcal{E} \text{ and } i \neq j.$$

A GSO $\tilde{\mathbf{A}}$ can be viewed as some function of the adjacency matrix $\mathbf{A}$.

Examples of GSOs are:

- Laplacian: $\tilde{\mathbf{A}} = \mathbf{L} = \mathbf{D} - \mathbf{A}$
- Random-walk matrix: $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$



[1] A. Sandryhaila *et al.*, "Discrete signal processing on graphs," 2013.

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X}$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

Using parameter matrix $\Theta \in \mathbb{R}^{d_x \times d_h}$ we can apply the filter on a different space

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

Using parameter matrix $\Theta \in \mathbb{R}^{d_x \times d_h}$ we can apply the filter on a different space

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \qquad \mathbf{h}_i = (\tilde{\mathbf{A}}\mathbf{X}\Theta)_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta \qquad \mathbf{h}_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

Using parameter matrix $\Theta \in \mathbb{R}^{d_x \times d_h}$ we can apply the filter on a different space

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \qquad \mathbf{h}_i = (\tilde{\mathbf{A}}\mathbf{X}\Theta)_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta \qquad \mathbf{h}_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta$$

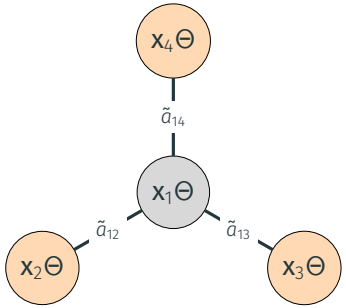
NOTE: We have **local** filters with parameters Θ **shared** among all nodes. Looks familiar?

Graph Convolution

Adding a **nonlinear activation** σ to the operation we have just seen

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \quad \rightarrow \quad \mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$$

we obtain a nonlinear graph convolutional filter.



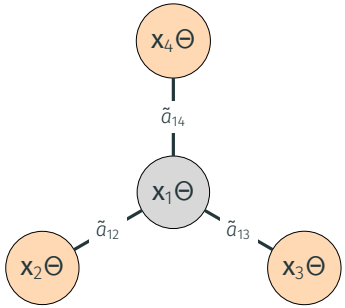
Graph Convolution

Adding a **nonlinear activation** σ to the operation we have just seen

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \quad \rightarrow \quad \mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$$

we obtain a nonlinear graph convolutional filter.

Since this operation is **differentiable**, we can learn Θ with gradient-based optimization methods.



Graph Convolution

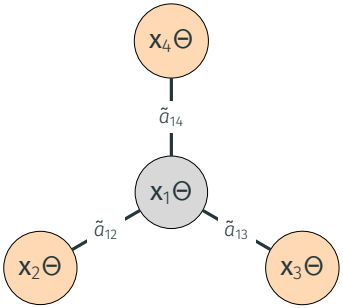
Adding a **nonlinear activation** σ to the operation we have just seen

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \quad \rightarrow \quad \mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$$

we obtain a nonlinear graph convolutional filter.

Since this operation is **differentiable**, we can learn Θ with gradient-based optimization methods.

This enables us to build *neural networks* with *graph*-like inputs, i.e., Graph Neural Networks (GNNs).



Sequence of graph convolutions

What if we apply two graph convolutions in sequence?

$$\mathbf{H}^{(1)} = \tilde{\mathbf{A}}\mathbf{X}\Theta^{(1)}$$
$$\mathbf{H}^{(2)} = \tilde{\mathbf{A}}\mathbf{H}^{(1)}\Theta^{(2)} = \tilde{\mathbf{A}}^2\mathbf{X}\Theta^{(1)}\Theta^{(2)}$$

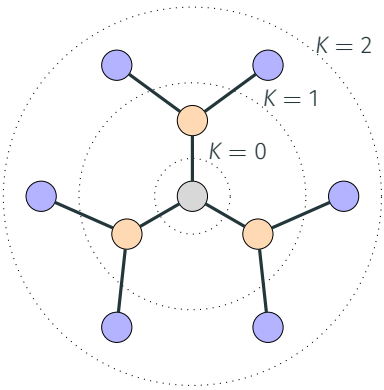
Sequence of graph convolutions

What if we apply two graph convolutions in sequence?

$$\mathbf{H}^{(1)} = \tilde{\mathbf{A}}\mathbf{X}\Theta^{(1)}$$
$$\mathbf{H}^{(2)} = \tilde{\mathbf{A}}\mathbf{H}^{(1)}\Theta^{(2)} = \tilde{\mathbf{A}}^2\mathbf{X}\Theta^{(1)}\Theta^{(2)}$$

Let's focus on the effect of $\tilde{\mathbf{A}}^2\mathbf{X}$:

$$(\tilde{\mathbf{A}}^2\mathbf{X})_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji}(\tilde{\mathbf{A}}\mathbf{X})_j = \sum_{j \in \mathcal{N}(i)} \sum_{k \in \mathcal{N}(j)} \tilde{a}_{ji} \cdot \tilde{a}_{kj} \cdot \mathbf{x}_k$$



Sequence of graph convolutions

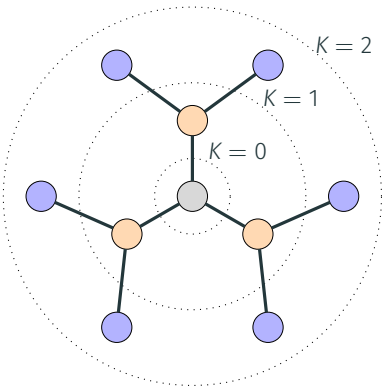
What if we apply two graph convolutions in sequence?

$$\mathbf{H}^{(1)} = \tilde{\mathbf{A}}\mathbf{X}\Theta^{(1)}$$
$$\mathbf{H}^{(2)} = \tilde{\mathbf{A}}\mathbf{H}^{(1)}\Theta^{(2)} = \tilde{\mathbf{A}}^2\mathbf{X}\Theta^{(1)}\Theta^{(2)}$$

Let's focus on the effect of $\tilde{\mathbf{A}}^2\mathbf{X}$:

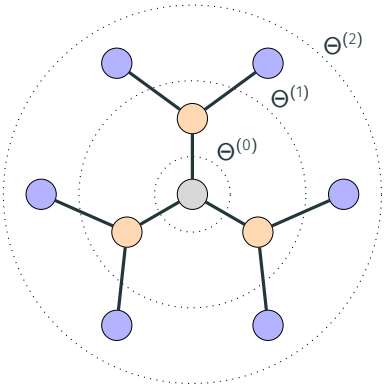
$$(\tilde{\mathbf{A}}^2\mathbf{X})_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji}(\tilde{\mathbf{A}}\mathbf{X})_j = \sum_{j \in \mathcal{N}(i)} \sum_{k \in \mathcal{N}(j)} \tilde{a}_{ji} \cdot \tilde{a}_{kj} \cdot \mathbf{x}_k$$

The second convolution aggregates information from the 2-hop neighbors, i.e., the neighbors' neighbors.



K-th-order filters

To aggregate information **up to the K-th-order neighborhood**, we can either use

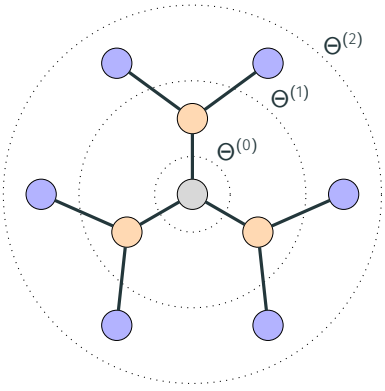


K-th-order filters

To aggregate information **up to the K-th-order neighborhood**, we can either use

- polynomial filters

$$\mathbf{H}^{(K)} = \sum_{k=0}^{K} \tilde{\mathbf{A}}^k \mathbf{X} \Theta^{(k)}$$



K-th-order filters

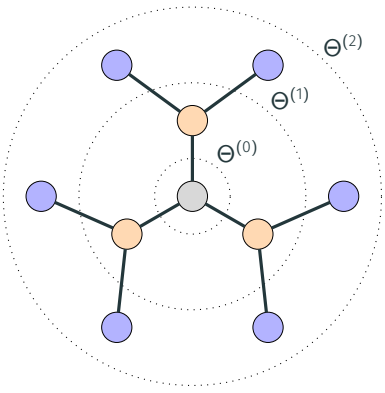
To aggregate information **up to the K-th-order neighborhood**, we can either use

- polynomial filters

$$\mathbf{H}^{(K)} = \sum_{k=0}^{K} \tilde{\mathbf{A}}^k \mathbf{X} \Theta^{(k)}$$

- a sequence of first-order-neighborhood filters

$$\mathbf{H}^{(0)} = \mathbf{X} \qquad \mathbf{H}^{(k)} = \tilde{\mathbf{A}} \mathbf{H}^{(k-1)} \Theta^{(k)}$$



K-th-order filters

To aggregate information **up to the K-th-order neighborhood**, we can either use

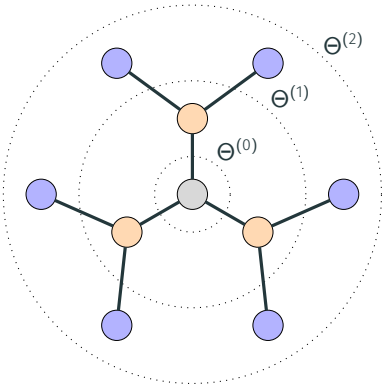
- polynomial filters

$$\mathbf{H}^{(K)} = \sum_{k=0}^{K} \tilde{\mathbf{A}}^k \mathbf{X} \Theta^{(k)}$$

- a sequence of first-order-neighborhood filters

$$\mathbf{H}^{(0)} = \mathbf{X} \qquad \mathbf{H}^{(k)} = \tilde{\mathbf{A}} \mathbf{H}^{(k-1)} \Theta^{(k)}$$

...eventually with nonlinearities.



Graph convolutional layers

Examples of graph convolutional layers from the literature:

- GCN [2]:

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}(\mathbf{I}_N + \mathbf{A})\mathbf{D}^{-1/2}$$

- Diffusion Convolution [3]:

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$$

- GIN [4]:

$$\tilde{\mathbf{A}} = \mathbf{A} + (1 + \epsilon) \cdot \mathbf{I}_N$$

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

[3] Y. Li *et al.*, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," 2017.

[4] K. Xu *et al.*, "How powerful are graph neural networks?" 2019.

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

What if we want to:

- take into account **edge attributes**?

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

What if we want to:

- take into account **edge attributes**?
- make the filter dependent also on the **nodes' features**, not only on the topology?

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

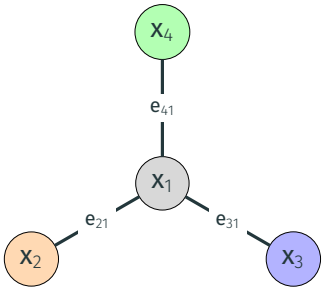
- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

What if we want to:

- take into account **edge attributes**?
- make the filter dependent also on the **nodes' features**, not only on the topology?
- e.g., **weigh** the contribution of a neighbor based on the **root node features**?

Message passing

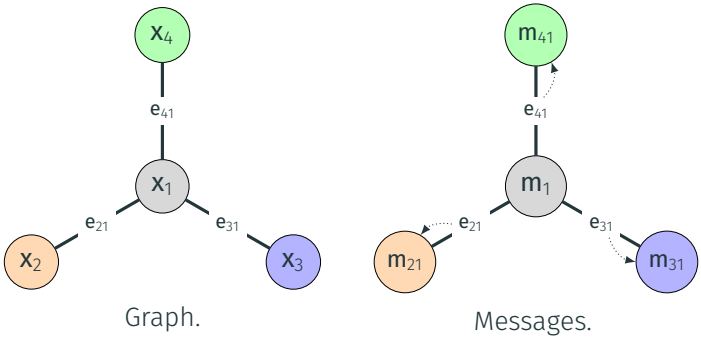
Message-passing neural networks [5]



Graph.

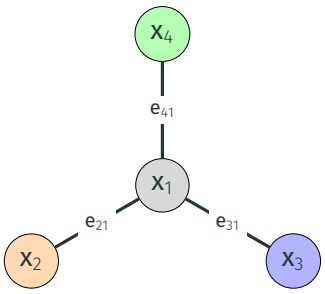
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks [5]

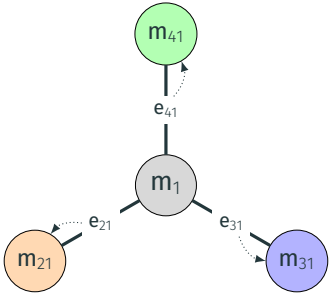


[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

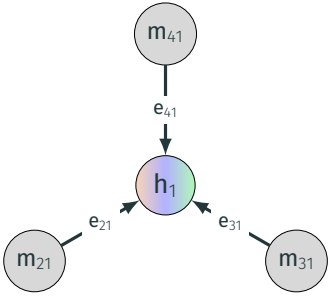
Message-passing neural networks [5]



Graph.



Messages.



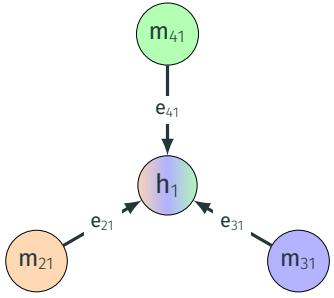
Propagation.

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

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

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



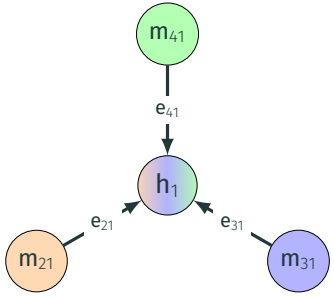
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

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

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;



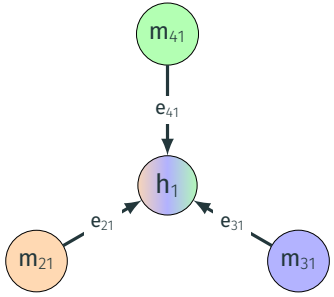
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

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

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;
- $\boldsymbol{Aggr}$: permutation-invariant **aggregation function** (e.g., sum, mean, max);



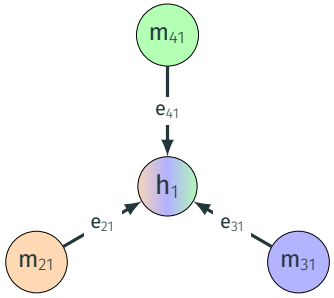
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

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

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;
- Aggr: permutation-invariant **aggregation function** (e.g., sum, mean, max);
- γ **update function**, to obtain new attributes from aggregated messages and previous attributes.



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

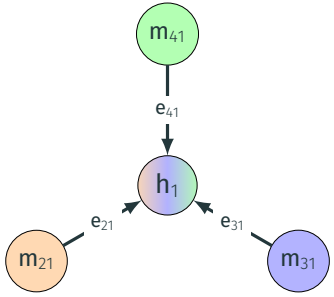
Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

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

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;
- Aggr: permutation-invariant **aggregation function** (e.g., sum, mean, max);
- γ **update function**, to obtain new attributes from aggregated messages and previous attributes.

Note: ϕ and γ are usually **parametric** (e.g., MLPs).



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

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

E.g., $\mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$ can be rewritten as:

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} a_{ji}\mathbf{x}_j\Theta\right)$$

where:

- $\phi\left(\mathbf{x}_j\right) = a_{ji}\mathbf{x}_j\Theta$
- $\boldsymbol{Aggr}$ is the sum
- $\gamma(\,\cdot\,) = \sigma(\,\cdot\,)$

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

E.g., $\mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$ can be rewritten as:

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} a_{ji}\mathbf{x}_j\Theta\right)$$

where:

- $\phi\left(\mathbf{x}_j\right) = a_{ji}\mathbf{x}_j\Theta$
- $\boldsymbol{Aggr}$ is the sum
- $\gamma(\,\cdot\,) = \sigma(\,\cdot\,)$

MP operations whose message function depends **only on the sender node's features** are called isotropic.

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

E.g., $\mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$ can be rewritten as:

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} a_{ji}\mathbf{x}_j\Theta\right)$$

where:

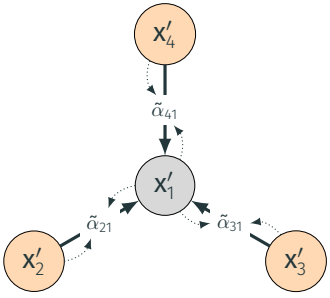
- $\phi\left(\mathbf{x}_j\right) = a_{ji}\mathbf{x}_j\Theta$
- $\boldsymbol{Aggr}$ is the sum
- $\gamma(\,\cdot\,) = \sigma(\,\cdot\,)$

MP operations whose message function depends **only on the sender node's features** are called isotropic.

We call them anisotropic when also **edge's or receiver node's features** are exploited.

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.



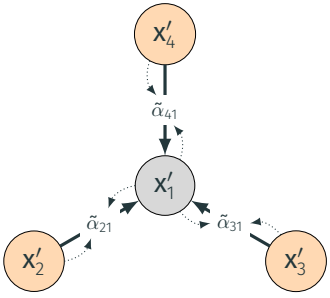
$\|$ indicates concatenation

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

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.



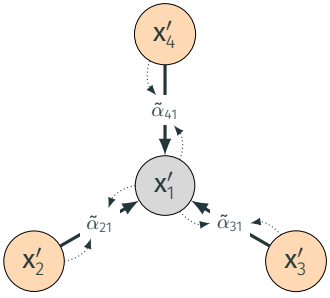
$\|$ indicates concatenation

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

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.
2. Compute **attention scores** between neighbors:
   2.1 Score: $\alpha_{ji} = \sigma\left([\mathbf{x}'_i \parallel \mathbf{x}'_j]\theta_2\right)$, with $\theta_2 \in \mathbb{R}^{2d_h \times 1}$.



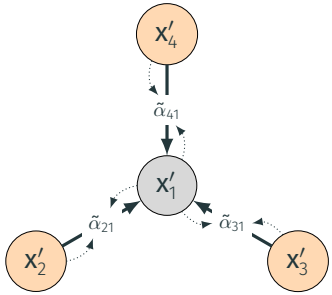
$\parallel$ indicates concatenation

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

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.
2. Compute **attention scores** between neighbors:
   - 2.1 Score: $\alpha_{ji} = \sigma\left([\mathbf{x}'_i \parallel \mathbf{x}'_j]\theta_2\right)$, with $\theta_2 \in \mathbb{R}^{2d_h \times 1}$.
   - 2.2 Normalize with Softmax: $\tilde{\alpha}_{ji} = \dfrac{\exp\left(\alpha_{ji}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\alpha_{ki}\right)}$



$\parallel$ indicates concatenation

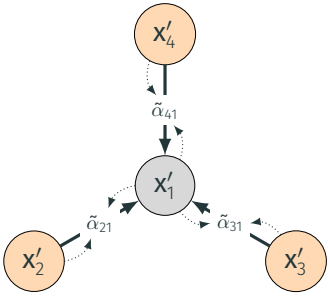
[6] P. Velickovic *et al.*, "Graph attention networks," 2017.

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.
2. Compute **attention scores** between neighbors:
   2.1 Score: $\alpha_{ji} = \sigma\left([\mathbf{x}'_i \parallel \mathbf{x}'_j]\theta_2\right)$, with $\theta_2 \in \mathbb{R}^{2d_h \times 1}$.
   2.2 Normalize with Softmax: $\tilde{\alpha}_{ji} = \dfrac{\exp\left(\alpha_{ji}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\alpha_{ki}\right)}$
3. Aggregate using attention coefficients as weights:

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

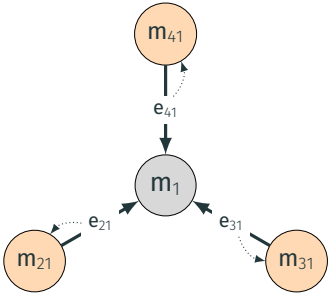


$\parallel$ indicates concatenation

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

Edge-conditioned convolution [7]

Key idea: incorporate edge attributes into the messages.



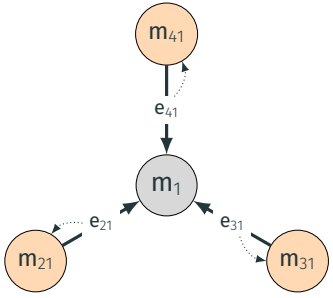
[7] M. Simonovsky *et al.*, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," 2017.

Edge-conditioned convolution [7]

Key idea: incorporate edge attributes into the messages.

Use a MLP $\rho : \mathbb{R}^{d_e} \to \mathbb{R}^{d_x \times d_h}$ to **generate weights**:

$$\Theta_{ji} = \rho(\mathbf{e}_{ji})$$



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

Edge-conditioned convolution [7]

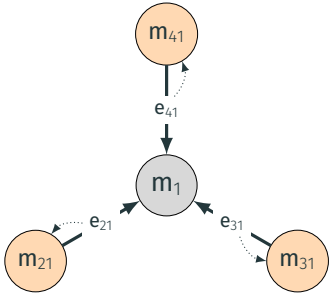
Key idea: incorporate edge attributes into the messages.

Use a MLP $\rho : \mathbb{R}^{d_e} \to \mathbb{R}^{d_x \times d_h}$ to **generate weights**:

$$\Theta_{ji} = \rho(\mathbf{e}_{ji})$$

Use the edge-dependent weights to compute messages:

$$\mathbf{h}_i = \mathbf{x}_i \Theta_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \Theta_{ji}$$



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

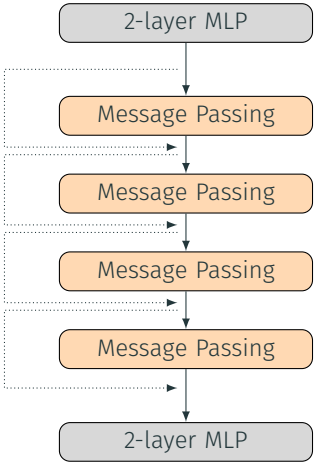
The zoo of GNNs

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 [8]

Architecture:

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



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

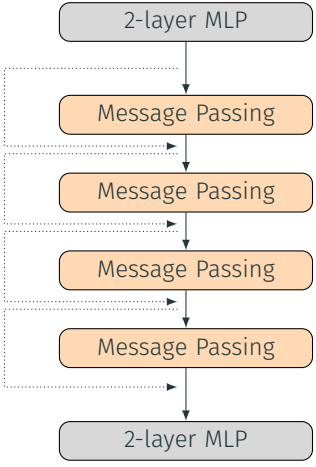
A good recipe [8]

Architecture:

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

Message passing at l-th layer:

- Message: $\mathbf{m}_{ji}^{l} = \text{PReLU}\left(\text{BatchNorm}\left(\mathbf{h}_{j}^{l}\Theta^{l} + \mathbf{b}^{l}\right)\right)$
- Aggregation: sum, i.e., $\mathbf{m}_{i}^{l} = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ji}^{l}$
- Update: $\mathbf{h}_{i}^{l+1} = \mathbf{h}_{i}^{l} \,||\, \mathbf{m}_{i}^{l}$;



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

A good recipe [8]

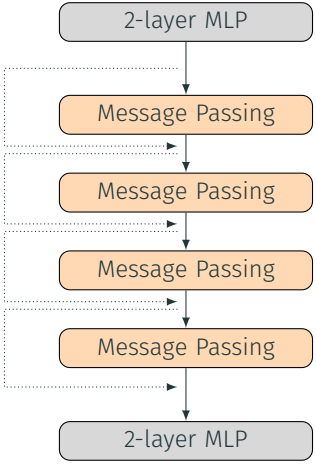
Architecture:

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

Message passing at l-th layer:

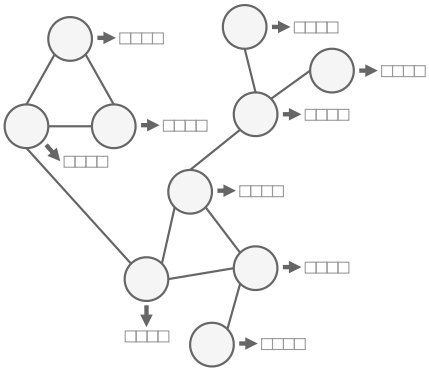
- Message: $\mathbf{m}_{ji}^{l} = \text{PReLU}\left(\text{BatchNorm}\left(\mathbf{h}_{j}^{l}\Theta^{l} + \mathbf{b}^{l}\right)\right)$
- Aggregation: sum, i.e., $\mathbf{m}_{i}^{l} = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ji}^{l}$
- Update: $\mathbf{h}_{i}^{l+1} = \mathbf{h}_{i}^{l} \,||\, \mathbf{m}_{i}^{l}$;

Quick test: is this MP operation *anisotropic*?

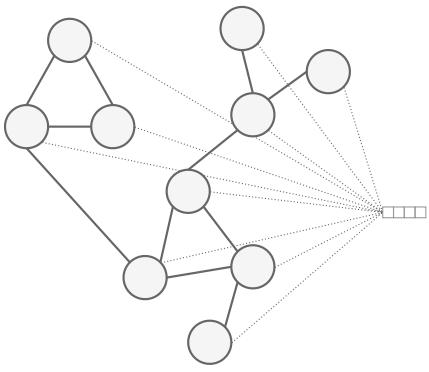


[8] J. You *et al.*, "Design space for graph neural networks," 2020.

How do we use this?



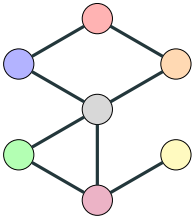
Node-level learning.
(e.g., social networks)



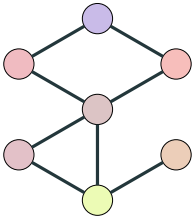
Graph-level learning.
(e.g., molecules)

A little warning

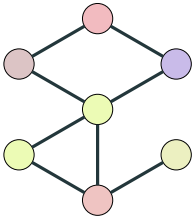
Graph convolutions act as **low-pass** filters, reducing the dissimilarity of neighbors' features at every application.



$l = 0$

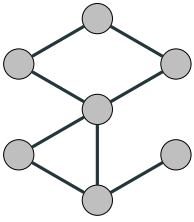


$l = 1$



$l = 2$

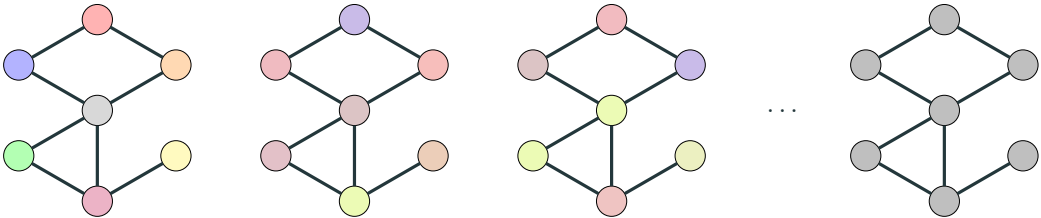
$\ldots$



$l = K$

A little warning

Graph convolutions act as **low-pass** filters, reducing the dissimilarity of neighbors' features at every application.

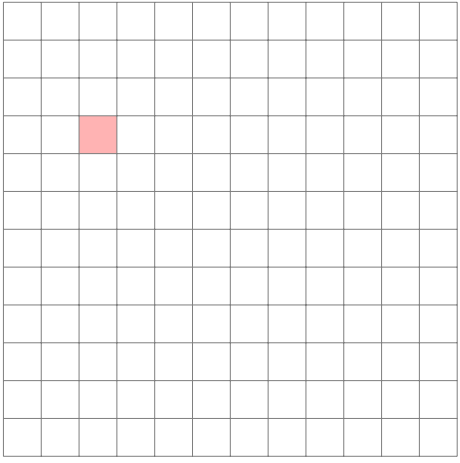


$l = 0$ $l = 1$ $l = 2$ $l = K$

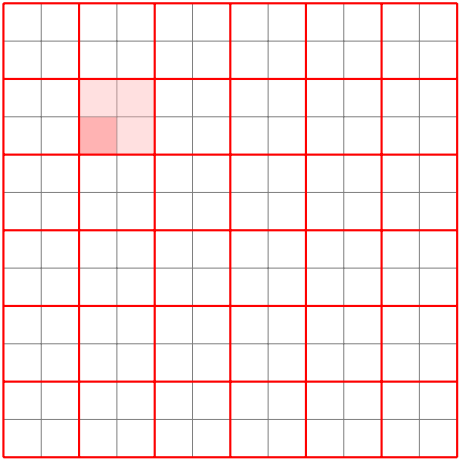
This phenomenon is reffered to as over-smoothing. Can you guess why it can be harmful?

Pooling on Graphs

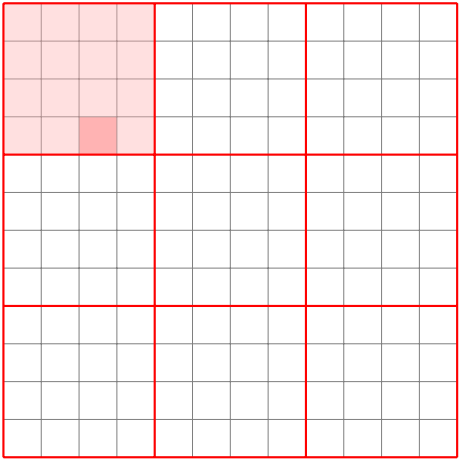
Pooling in CNNs



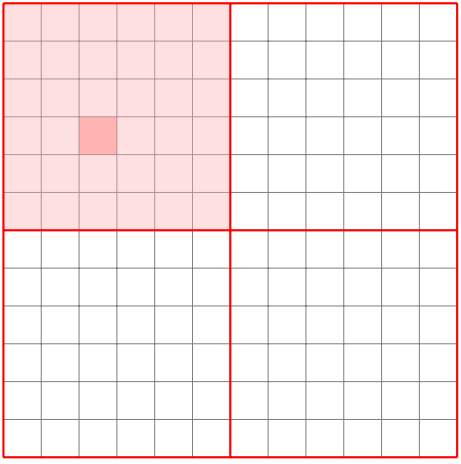
Pooling in CNNs



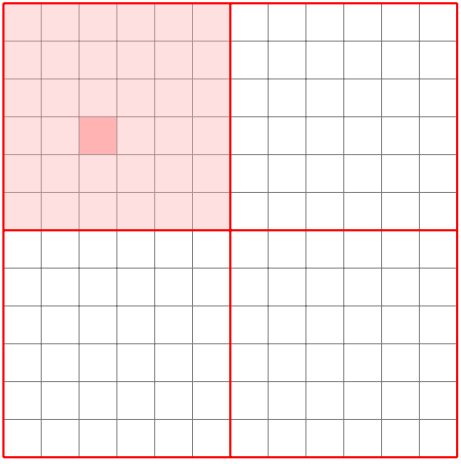
Pooling in CNNs

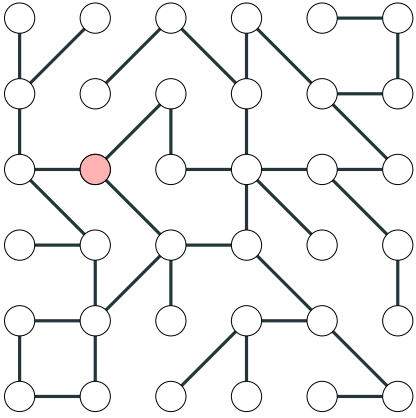


Pooling in CNNs

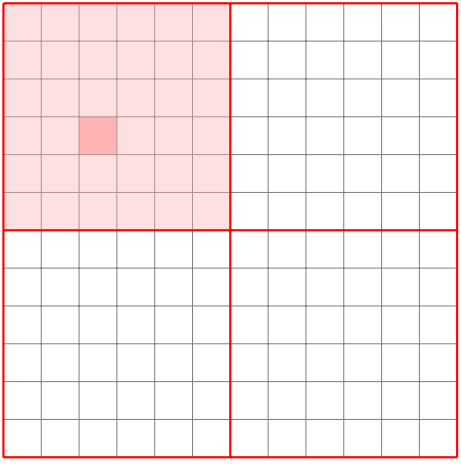


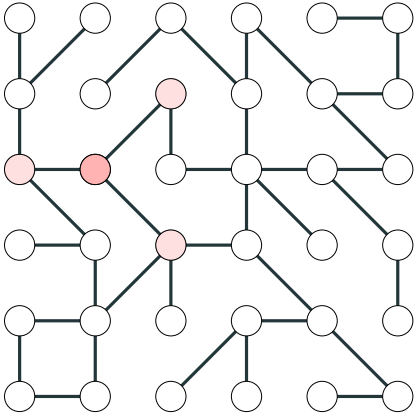
Pooling in CNNs



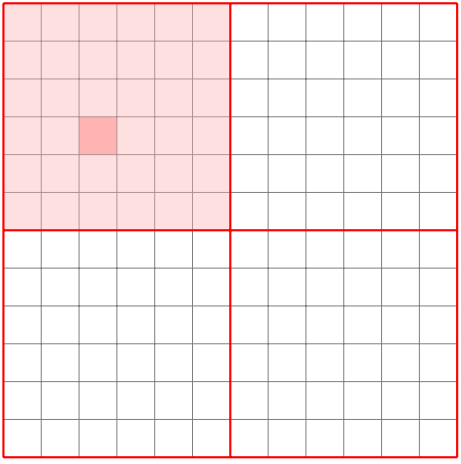


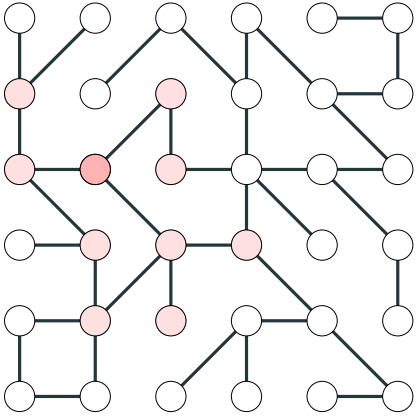
Pooling in CNNs



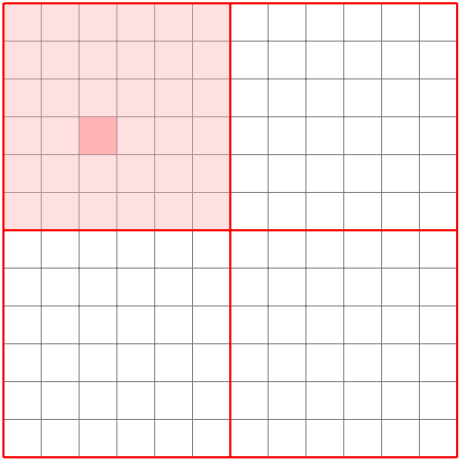


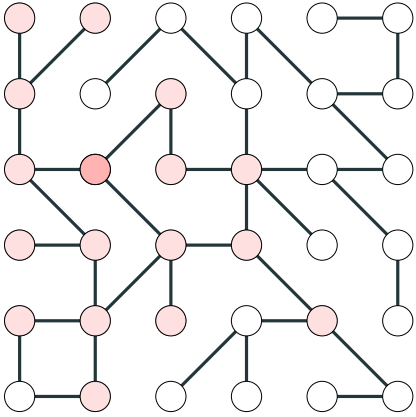
Pooling in CNNs





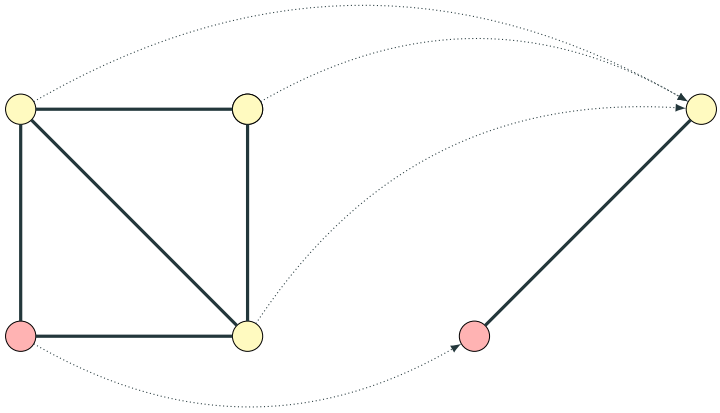
Pooling in CNNs





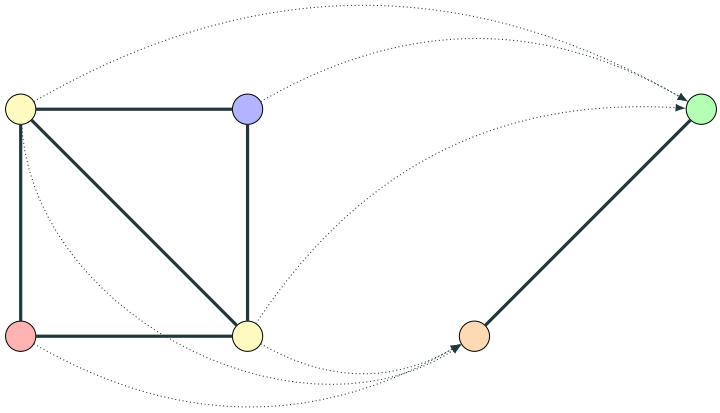
Graph pooling by example

Strategy 1: aggregate same attributes (Candy Crush pooling).



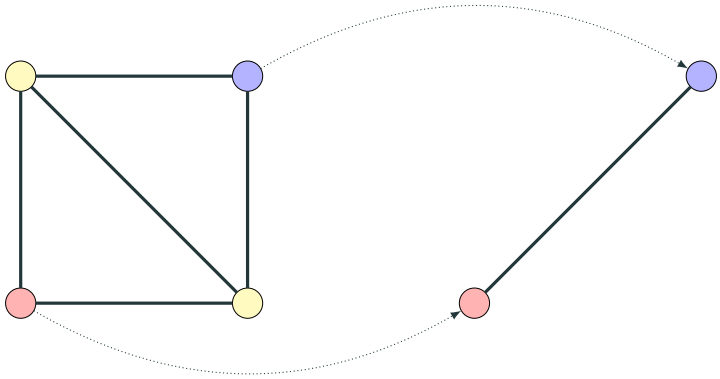
Graph pooling by example

Strategy 2: aggregate cliques.



Graph pooling by example

Strategy 3: keep only some types/colors.



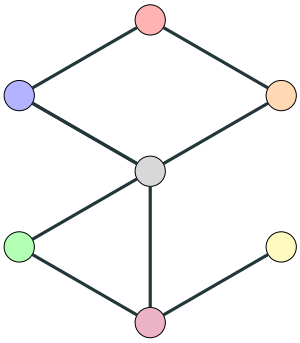
Three main questions [9]

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?

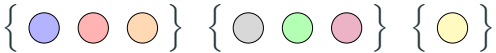
[9] D. Grattarola *et al.*, "Understanding pooling in graph neural networks," 2022.

Step 1: Select

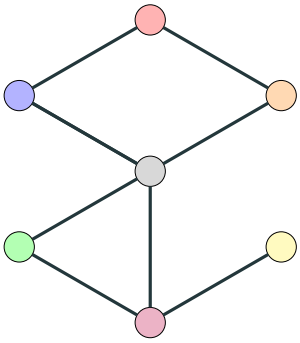
Selecting nodes



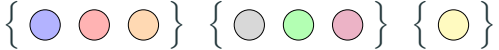
Example 1: partition.



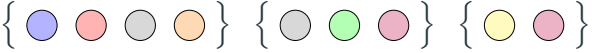
Selecting nodes



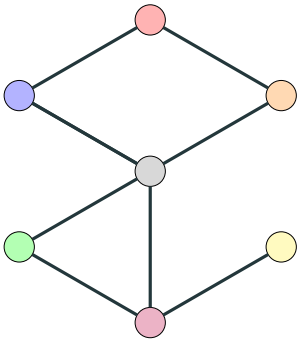
Example 1: partition.



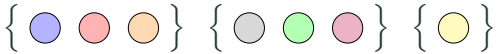
Example 2: cover (possible overlaps).



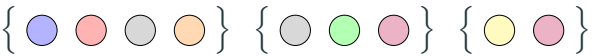
Selecting nodes



Example 1: partition.



Example 2: cover (possible overlaps).



Example 3: sparse.

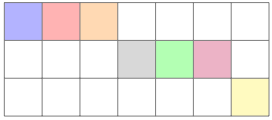
Selecting nodes

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

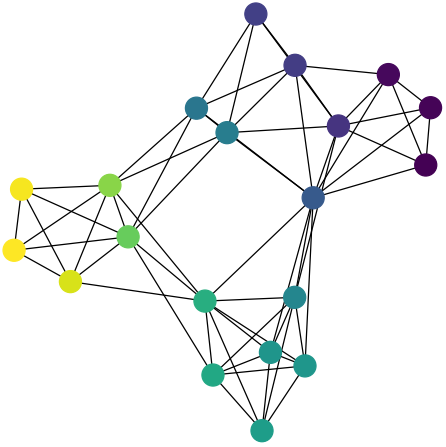
$$\textsc{Sel} : \mathcal{G} \mapsto \mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_K\}.$$

Each supernode is a set of nodes (with relative features) associated with a score:

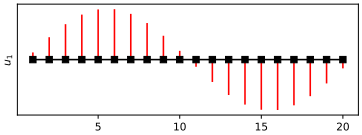
$$\mathcal{S}_k = \{(\mathbf{x}_i, s_{ki}) \mid s_{ki} > 0\}$$

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

Spectral clustering [11]



The low-frequency eigenvectors of the Laplacian naturally cluster the nodes.

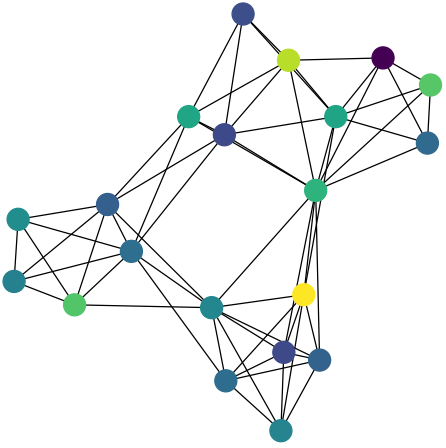


Idea: run k-means clustering (or similar) using the first few eigenvectors.

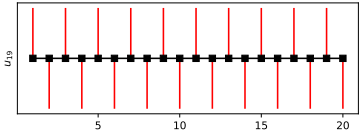
[10] J. Shi *et al.*, "Normalized cuts and image segmentation," 2000.

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

Node decimation [13]



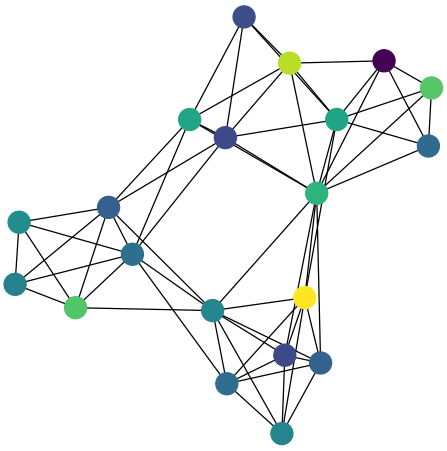
Alternative: use the highest-frequency eigenvector to do something similar to a regular subsampling.



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

[13] 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**.

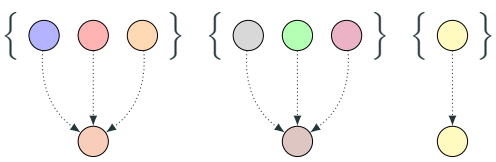
But we get the general idea...

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):

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

Step 3: Connect

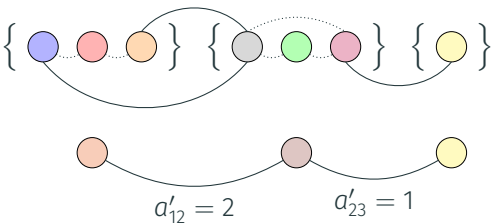
Connecting supernodes

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

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

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

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



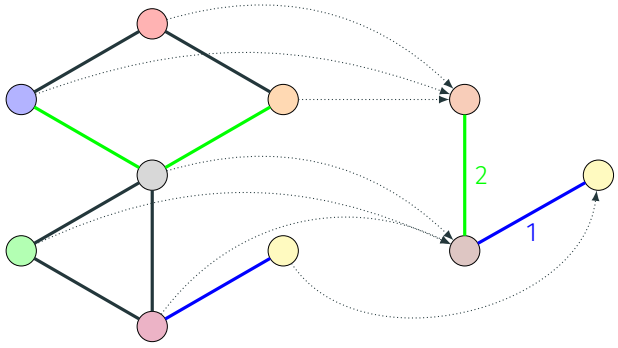
Select, Reduce, Connect [9]

Putting everything together:

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

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

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



[9] D. Grattarola *et al.*, "Understanding pooling in graph neural networks," 2022.

Pooling methods

Pooling methods

A few ideas:

1. **Graclus** [14], approximately halves nodes:
   1.1 select a (not merged) node i randomly;
   1.2 merge i with (not merged) neighbor j such that $\underset{j}{\text{argmax}}\, a_{ji} \left(\frac{1}{deg_i} + \frac{1}{deg_j} \right)$

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

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

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

Pooling methods

A few ideas:

1. **Graclus** [14], approximately halve nodes:
   1.1 select a (not merged) node i randomly;
   1.2 merge i with (not merged) neighbor j such that $\operatorname*{argmax}_j a_{ji} \left(\frac{1}{deg_i} + \frac{1}{deg_j} \right)$
2. **Clique Pooling** [15]: merge together cliques, i.e., fully-connected subgraphs.

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

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

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

Pooling methods

A few ideas:

1. **Graclus** [14], approximately halve nodes:
   1.1 select a (not merged) node i randomly;
   1.2 merge i with (not merged) neighbor j such that $\operatorname*{argmax}_j a_{ji} \left(\frac{1}{deg_i} + \frac{1}{deg_j} \right)$
2. **Clique Pooling** [15]: merge together cliques, i.e., fully-connected subgraphs.
3. **LaPool** [16]: select "leaders" that have highest local variation $\|\mathbf{L}\mathbf{X}\|$ w.r.t. all their neighbors. Create clusters by assigning nodes to nearest leader.

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

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

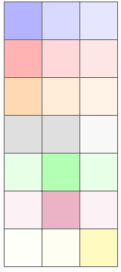
[16] E. Noutahi *et al.*, "Towards Interpretable Sparse Graph Representation Learning with Laplacian Pooling," 2019.

Learning to pool

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

- **DiffPool** [17]: GNN for $\mathbf{S}^\top$, regularize with "link prediction" loss;

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



[17] R. Ying *et al.*, "Hierarchical Graph Representation Learning withDifferentiable Pooling," 2018.

[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

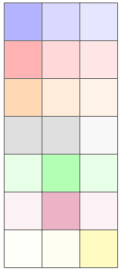
[19] C. Bodnar *et al.*, "Deep Graph Mapper: Seeing Graphs through the Neural Lens," 2020.

Learning to pool

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

- **DiffPool** [17]: GNN for $\mathbf{S}^\top$, regularize with "link prediction" loss;
- **MinCutPool** [18]: MLP for $\mathbf{S}^\top$, regularize with "minimum cut" loss (same objective as spectral clustering);

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



[17] R. Ying *et al.*, "Hierarchical Graph Representation Learning withDifferentiable Pooling," 2018.

[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

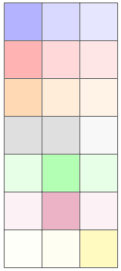
[19] C. Bodnar *et al.*, "Deep Graph Mapper: Seeing Graphs through the Neural Lens," 2020.

Learning to pool

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

- **DiffPool** [17]: GNN for $\mathbf{S}^\top$, regularize with "link prediction" loss;
- **MinCutPool** [18]: MLP for $\mathbf{S}^\top$, regularize with "minimum cut" loss (same objective as spectral clustering);
- **Deep Graph Mapper** [19]: combine Mapper [20] and GCN [2] to compute clusters.

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



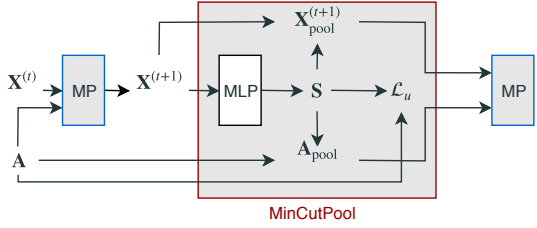
[17] R. Ying *et al.*, "Hierarchical Graph Representation Learning withDifferentiable Pooling," 2018.

[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

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

MinCut Pooling [18]

- 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:
  $\mathcal{L}_o = \left\| \frac{\mathbf{SS}^\top}{\|\mathbf{SS}^\top\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}} \right\|_F$

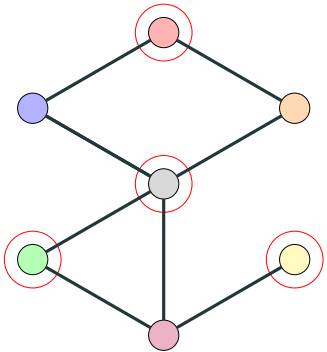


[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

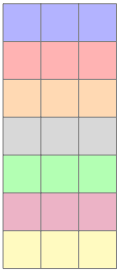
Top-K methods

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

Can we learn a sparse selection?

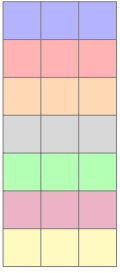


Top-K methods

$X =$ 

Features

Top-K methods

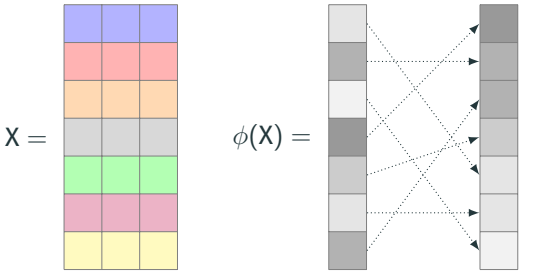
$\mathbf{X} =$ 

Features

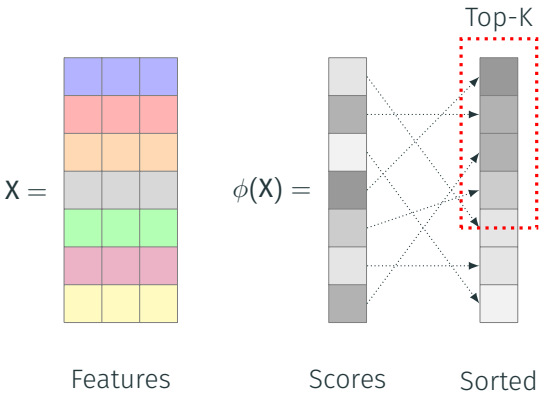
$\phi(\mathbf{X}) =$ 

Scores

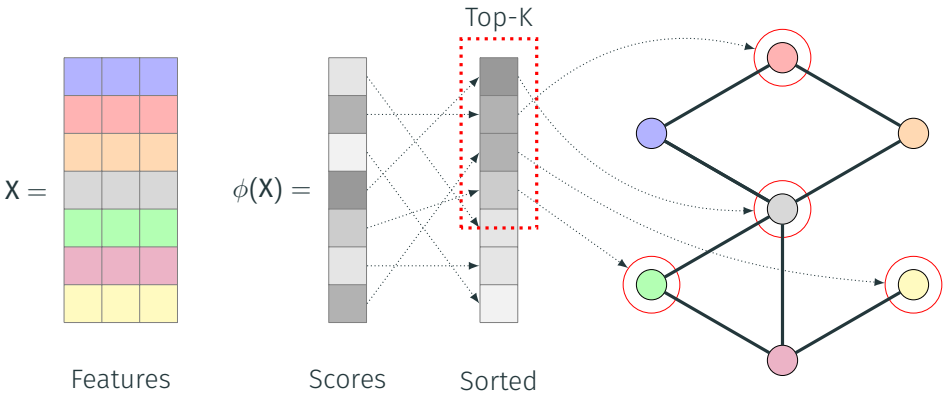
Top-K methods



Top-K methods



Top-K methods



Top-K methods

Different ways of computing the selection indices :

- Select with a simple linear projection $\theta \in \mathbb{R}^{d_x}$ [21];
- Select with a GNN [22];
- Train the selection with a supervised objective (needs ground truth for which nodes to keep) [23].

[21] H. Gao *et al.*, "Graph U-Nets," 2019.

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

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

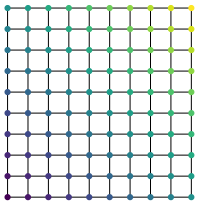
Top-K methods

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

Problems:

- Top-k selection is **non-differentiable**.
  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;

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;

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 convolutions, we usually **flatten** out the matrix representation to give a vector as input to an MLP:

1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

Global Pooling

In CNNs, after convolutions, we usually **flatten** out the matrix representation to give a vector as input to an MLP:

We may want to do the same operation on graphs, e.g., for *graph classification* tasks.

This operation is called global pooling.

Global Pooling

In CNNs, after convolutions, we usually **flatten** out the matrix representation to give a vector as input to an MLP:

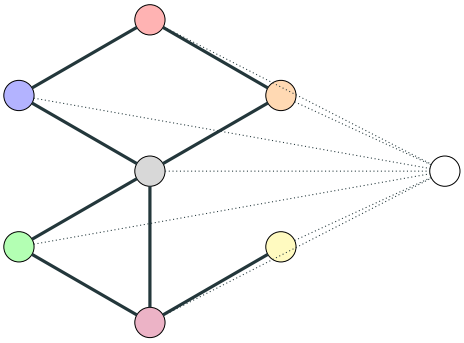
1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

We may want to do the same operation on graphs, e.g., for *graph classification* tasks.

This operation is called global pooling.

Global pooling must be **invariant to permutations** of the nodes:



Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;

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

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

Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;
- Weighted sum with attention [24];

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

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

Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;
- Weighted sum with attention [24];
- Sum and then apply a neural network [25];

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

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

Coding GNNs

GNNs libraries

Spektral is a Python library based on Keras providing a simple but flexible framework for creating graph neural networks (GNNs).

GitHub: danielegrattarola/spektral
Website: graphneural.network

PyG (PyTorch Geometric) is a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs).

GitHub: pyg-team/pytorch_geometric
Website: pyg.org

Demo

In this demo, we will use **PyG** to address the **node classification** task with GNNs.

Introduction to Graph Neural Networks

References i

[1] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE transactions on signal processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

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

[3] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[4] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" In *International Conference on Learning Representations (ICLR)*, 2019.

[5] 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.

References ii

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

[7] 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.

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

[9] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[10] 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.

References iii

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

[12] 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.

[13] 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]*.

[14] 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.

References iv

[15] 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.

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

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

[18] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proceedings of the 37th international conference on Machine learning*, ACM, 2020.

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

[20] 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.

[21] H. Gao and S. Ji, “Graph u-nets,” *CoRR*, vol. abs/1905.05178, 2019. arXiv: *1905.05178*. [Online]. Available: *http://arxiv.org/abs/1905.05178*.

[22] J. Lee, I. Lee, and J. Kang, “Self-attention graph pooling,” *CoRR*, vol. abs/1904.08082, 2019. arXiv: *1904.08082*. [Online]. Available: *http://arxiv.org/abs/1904.08082*.

[23] 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*.

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

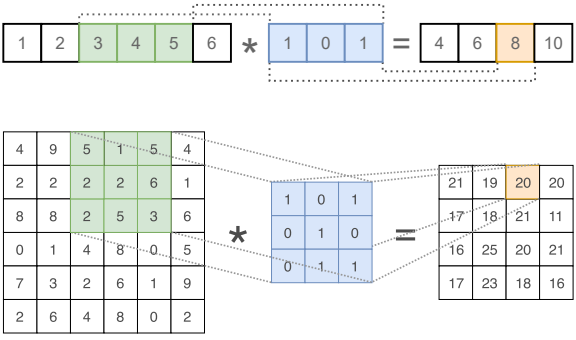
[25] N. Navarin, D. Van Tran, and A. Sperduti, "Universal readout for graph convolutional neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–7.

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

[27] 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.

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:

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

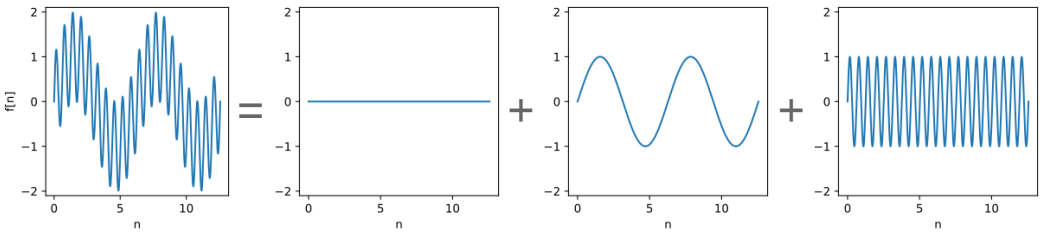
Where $\mathcal{F}$ is the **Fourier transform** and $\mathcal{F}^{-1}$ its inverse.

Can we use this major property?

What is the Fourier transform?

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

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

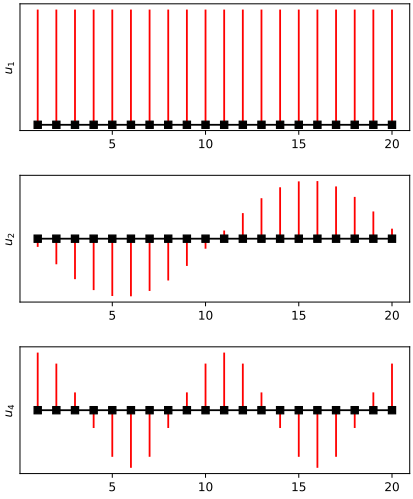


From FT to GFT

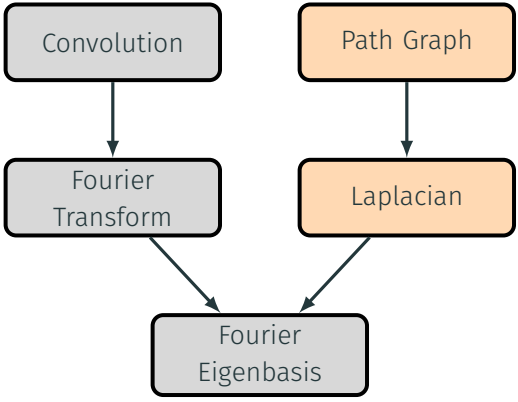
The eigenvectors of the Laplacian for a path graph can be obtained analytically:

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

Looks familiar?



From FT to GFT



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

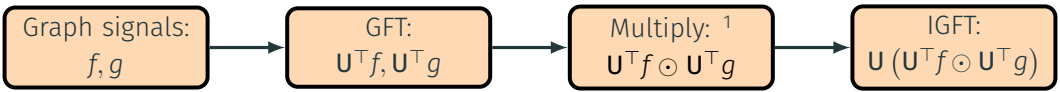
$$\mathcal{F}_G\{f\}[k] = \sum_{n=0}^{N-1} f[n]\mathbf{u}_k[n]$$

- 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: $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \sum_{i=0}^{N-1} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$

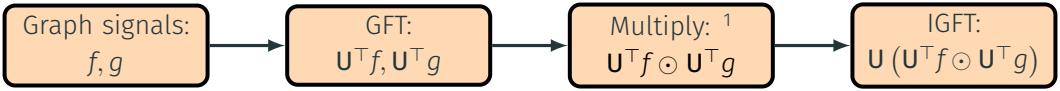


[1] $\odot$ indicates element-wise multiplication

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: $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \sum_{i=0}^{N-1} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$



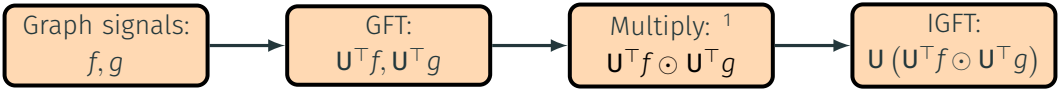
Graph filter: $\mathbf{U}\left(\mathbf{U}^\top f \odot \mathbf{U}^\top g\right) =$

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

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: $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \sum_{i=0}^{N-1} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$



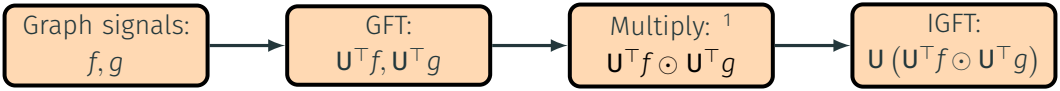
Graph filter: $\mathbf{U}\left(\mathbf{U}^\top f \odot \mathbf{U}^\top g\right) = \mathbf{U} \cdot \underbrace{\operatorname{diag}(\mathbf{U}^\top g)}_{g(\mathbf{\Lambda})} \cdot \mathbf{U}^\top f =$

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

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: $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \sum_{i=0}^{N-1} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$



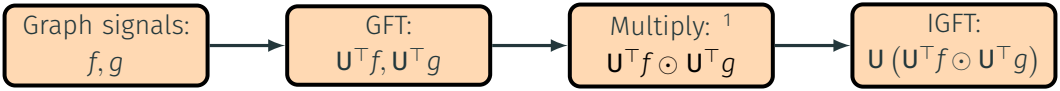
Graph filter: $\mathbf{U}\left(\mathbf{U}^\top f \odot \mathbf{U}^\top g\right) = \mathbf{U} \cdot \underbrace{\operatorname{diag}(\mathbf{U}^\top g)}_{g(\mathbf{\Lambda})} \cdot \mathbf{U}^\top f = \underbrace{\mathbf{U} \cdot g(\mathbf{\Lambda}) \cdot \mathbf{U}^\top}_{g(\mathbf{L})} f =$

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

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 [26]: 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 $O(N^2)$;

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

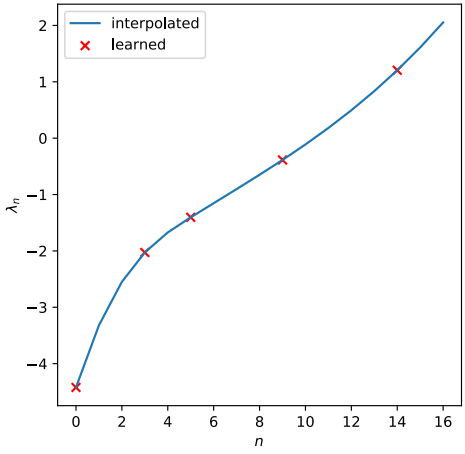
[26] J. Bruna *et al.*, "Spectral networks and locally connected networks on graphs," 2013.

Spectral GCNs

Better idea [26]:

- **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**.

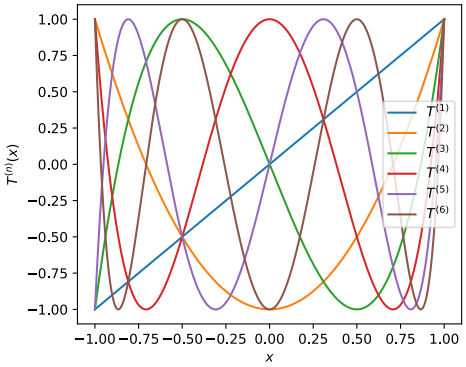


[26] J. Bruna *et al.*, "Spectral networks and locally connected networks on graphs," 2013.

Chebyshev polynomials [27]

The same recursion is used to filter eigenvalues:

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



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