LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deep Learning for Physicists

Lecture #8: Graph networks

Kosmas Kepesidis

Outline

- Beyond Cartesian data structures
- Graphs
- Convolutions in the spatial domain
- Convolutions in the spectral domain

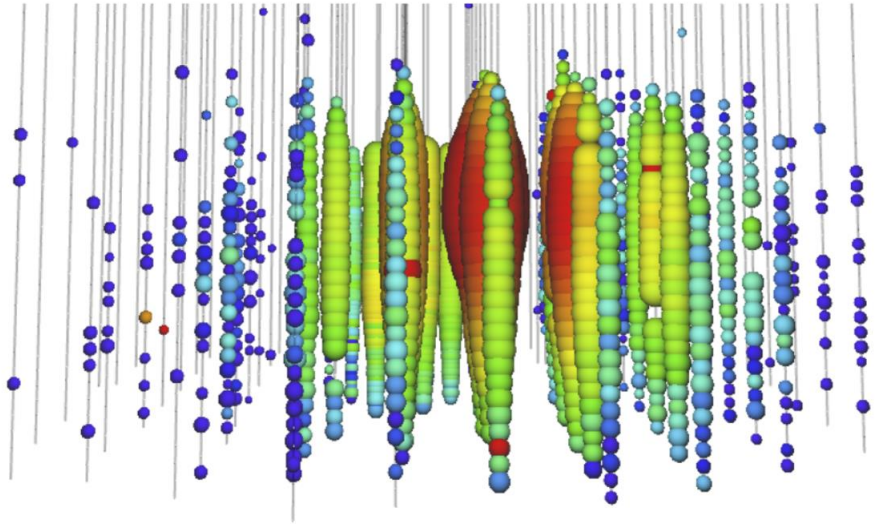
Beyond Cartesian data structures

Beyond Cartesian data structures

- Typical data sets are represented in Cartesian arrangements (tabular data, images, grids, etc.)

- But many physics experiments, provide data that lie on non-Euclidean manifolds (e.g. sphere)

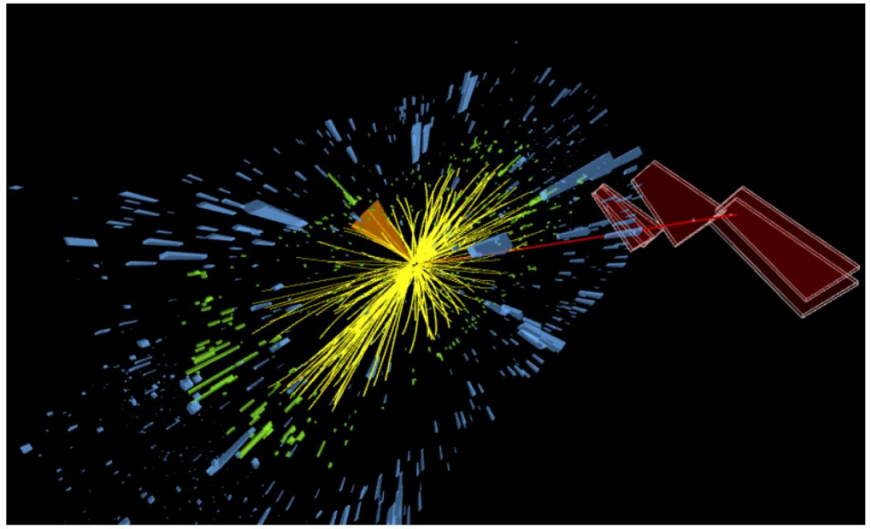
Beyond Cartesian data structures



IceCube experiment:
The experiment features strings of 60 optical modules each, arranged in a hexagonal grid

M. G. Aartsenet al., The IceCube Neutrino Observatory Part II: Atmospheric and Diffuse UHE Neutrino Searches of All Flavors, (2014)

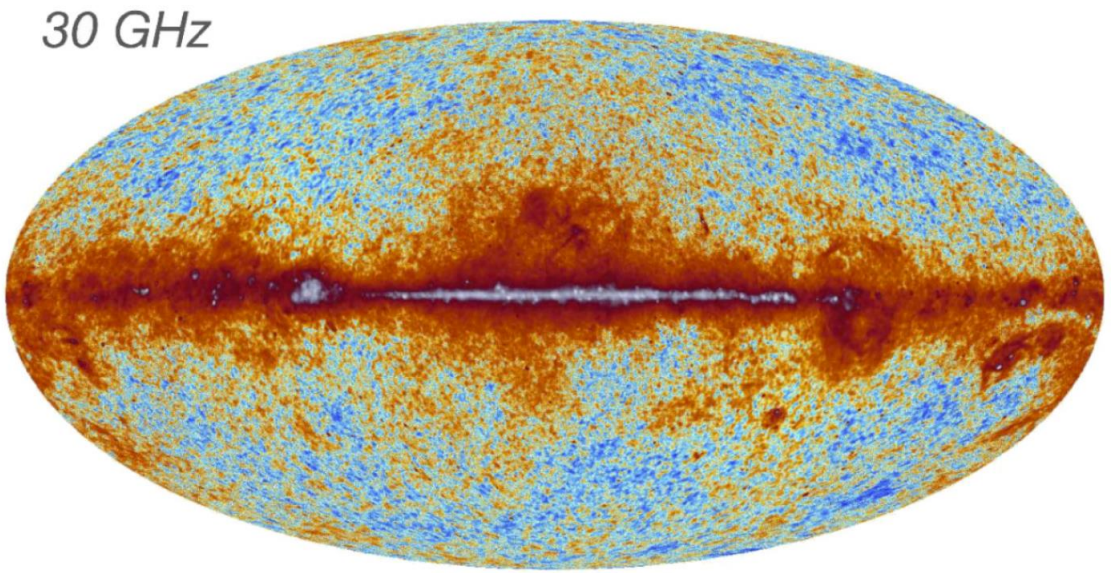
Beyond Cartesian data structures



CMS experiment:
The experiment has various detectors with different sensor arrangements, resulting into non-regular data

The CMS Collaboration, CMS Event displays displaying the use of the tau embedding technique (2020): https://cds.cern.ch/record/2711418

Beyond Cartesian data structures



Planck spacecraft:
Non-Euclidean data measured by Planck spacecraft

Y. Akramiet al., Planck results. I. Overview and the cosmological legacy of Planck, Astronomy and Astrophysics 641, p. 1 (2018)

Beyond Cartesian data structures

- Applying convolutional operations on non-regular or non-Euclidean data is not a straightforward task

  - Convolutional operations take advantage of:
    1. Translational invariance in data
    2. Scale invariance of features in data
    3. Parameters do not directly depend on the input
    4. Locality – filters are localized in space

  - Very challenging to define filters with a fixed number of adaptive parameters

Beyond Cartesian data structures

- Applying convolutional operations on non-regular or non-Euclidean data is not a straightforward task

- To define convolutions in irregular data, one needs to understand the underlying structure as a **graph**

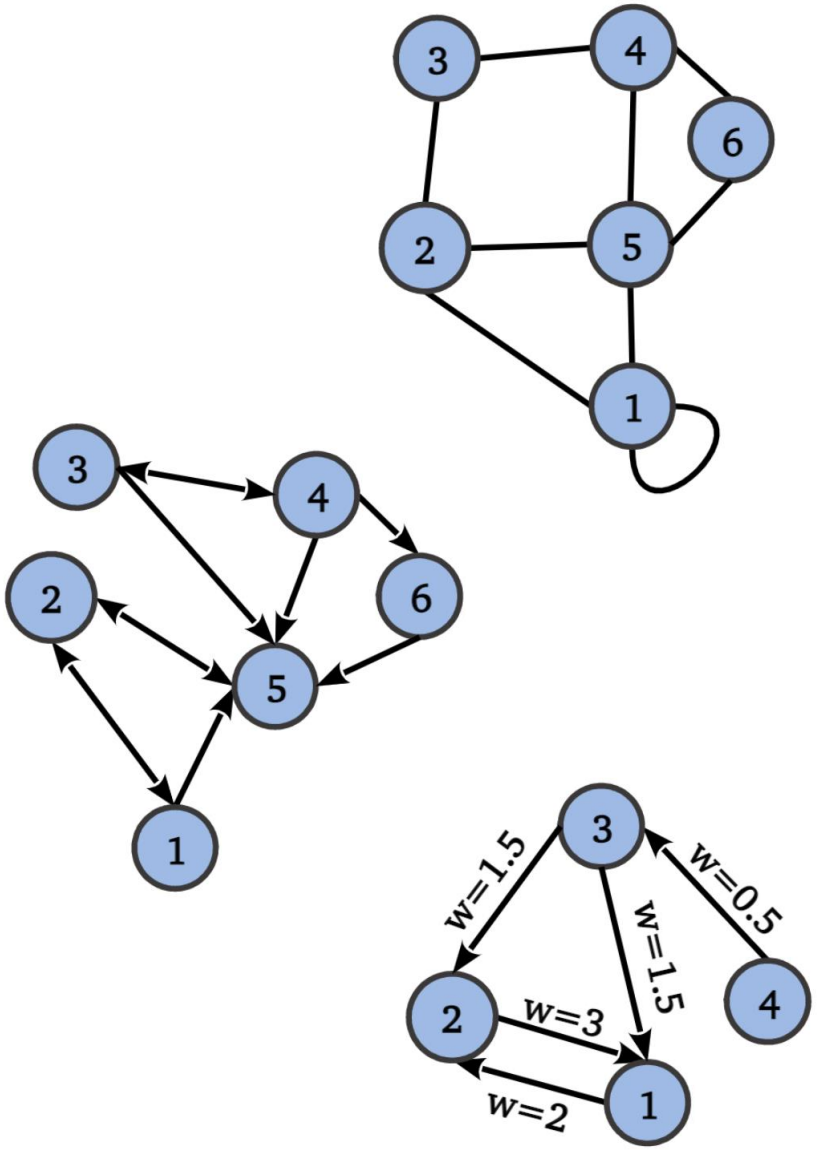
- **Graph neural networks** (GNNs) can extend the concept of convolutions to irregular domains, using basic **graph theory**

Graphs

Graphs

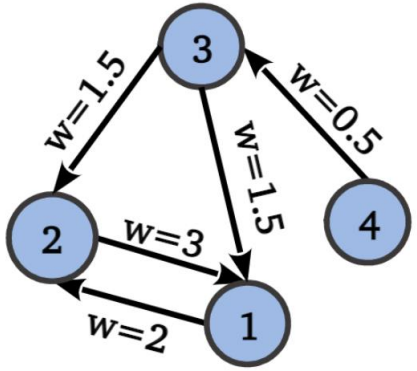
- A graph consists of **nodes** and **edges**

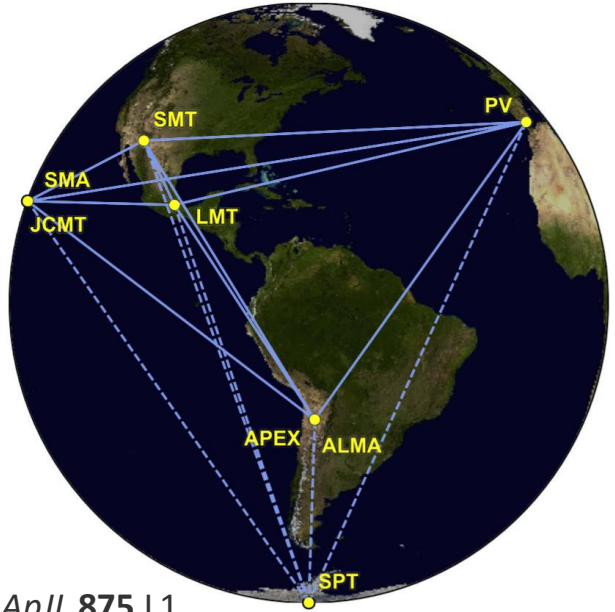
- Three types considered here:
  - Undirected graphs
  - Directed graphs
  - Weighted (embedded) graphs



Graphs

- Weighted graphs (or embedded in a manifold)
- Manifold: fixed neighbourhood described by a single property
- Examples:
  - Non-regular arrangement of sensors (telescopes) that together form a spherical detector, where the edges can describe the distance between sensors

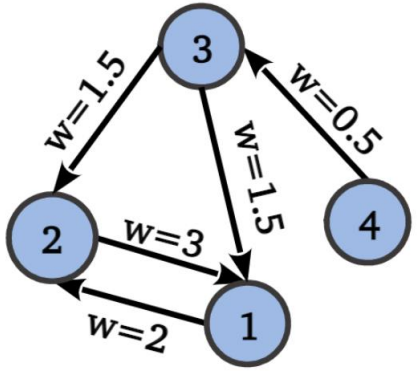




The Event Horizon Telescope Collaboration *et al* 2019 *ApJL* **875** L1

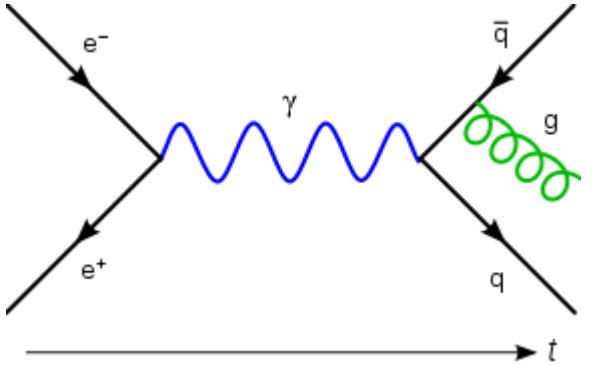
Graphs

- Weighted graphs are said to be imbedded in a manifold – fixed neighbourhood described by a single property



- Examples:

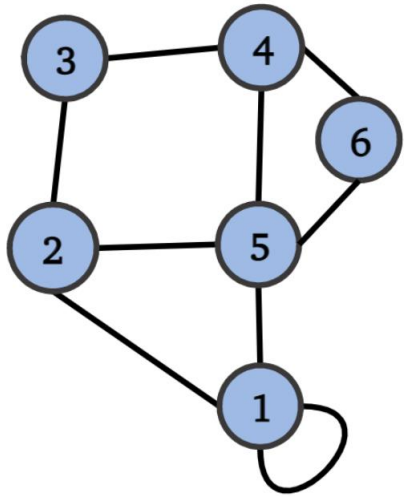
  - Feynman diagrams:
    - Edges indicate the type of particle
    - Nodes which describe the coupling strength



Graphs

- relationship of nodes in a graph is described by the so-called **adjacency matrix** – square matrix similar to stochastic matrices
  - Describes the structure of a graph
  - Undirected graphs have a symmetric $\mathbf{A}$ matrix
  - Each entry corresponds to the number of connections of each node

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

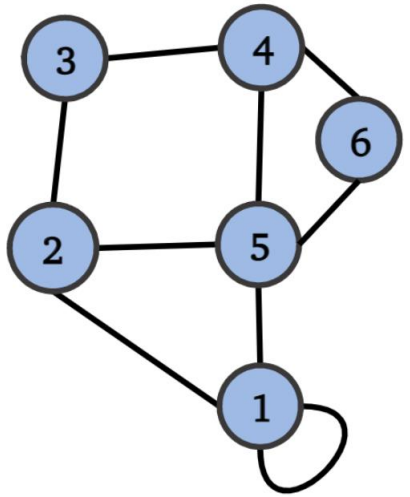


Graphs

- Another important matrix is the **degree matrix**

  - Gives the number of connections (degree of connectivity) of each node
  - Matrix $\mathbf{D}$ is always diagonal

$$\mathbf{D} = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$



Graphs

- Using the adjacency matrix $\mathbf{A}$ and the degree matrix $\mathbf{D}$, one can construct the discrete version of the Laplace operator: $\mathbf{L}' = \mathbf{D} - \mathbf{A}$
- This discrete operator converges to the known Laplace operator, $\mathbf{L}' \rightarrow \Delta$, as the graph becomes increasingly dense
- Operator $\mathbf{L}'$ can be interpreted as a measure of smoothness of a scalar function on a particular manifold
- Operator $\mathbf{L}'$ describes how “smoothly” signals propagate across the graph

M. M. Bronstein et al., Geometric deep learning: Going beyond Euclidean data, IEEE Signal Processing Magazine 34, 4, pp. 18–42 (2017)

Graphs

- A normalized version of the Laplacian can be constructed as follows: $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{L}'\ \mathbf{D}^{-1/2} = \mathbf{1} - \mathbf{D}^{-1/2}\mathbf{A}\ \mathbf{D}^{-1/2}$

$$\mathbf{L}_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and node } v_i \text{ is adjacent to node } v_j \\ 0 & \text{otherwise} \end{cases}$$

➢Where v_i are edges of nodes and $\deg(v_i)$ is the degree of node i

M. M. Bronstein et al., Geometric deep learning: Going beyond Euclidean data, IEEE Signal Processing Magazine 34, 4, pp. 18–42 (2017)

Graphs

- A normalized version of the Laplacian can be constructed as follows: $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{L}'\ \mathbf{D}^{-1/2} = \mathbf{1} - \mathbf{D}^{-1/2}\mathbf{A}\ \mathbf{D}^{-1/2}$

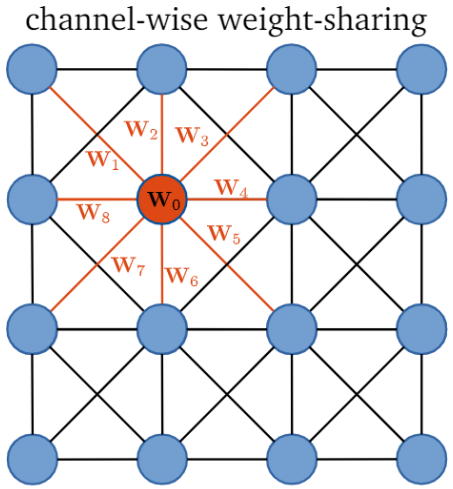
  - Interesting property:
    - Its eigenfunctions for a Fourier basis of the respective graph, i.e. the Fourier transform diagonalizes the Laplacian: $F(\Delta f) = -\omega^2 F(f)$
    - This means that the graph Laplacian can be interpreted in terms of frequencies and its eigenfunctions as eigenmodes of the graph

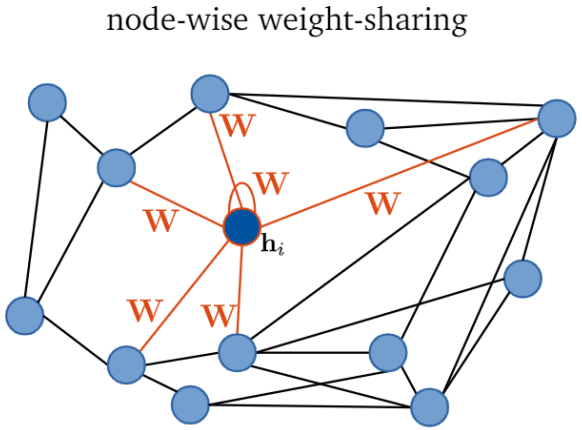
M. M. Bronstein et al., Geometric deep learning: Going beyond Euclidean data, IEEE Signal Processing Magazine 34, 4, pp. 18–42 (2017)

Convolutions in the spatial domain

Convolutions in the spatial domain

- Most straightforward way to design GNNs would be a direct application of convolutional operations to graphs
  - In grid-like data the weights are shared over the image, while this is not in general possible for non-regular graphs, since the number and distance of neighbors can be different from node to node
  - Modifications are required!



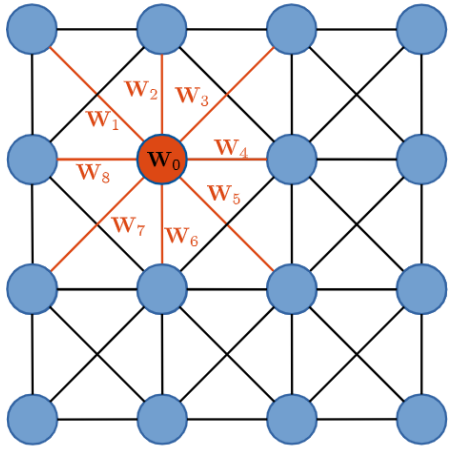


T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, (2017)

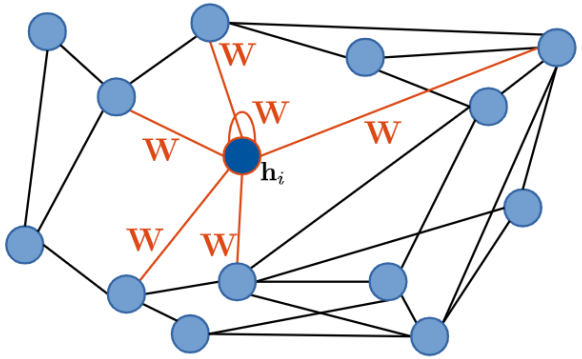
Convolutions in the spatial domain

- Most straightforward way to design GNNs would be a direct application of convolutional operations to graphs
  - Each node has a feature vector $\mathbf{h}_i$ and the same matrix with weights $\mathbf{W}$ is applied on each node

channel-wise weight-sharing



node-wise weight-sharing



T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, (2017)

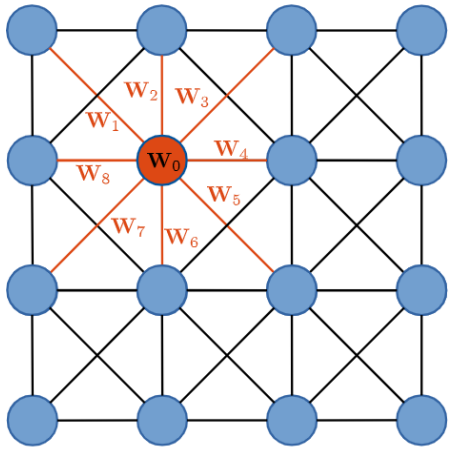
Convolutions in the spatial domain

- Most straightforward way to design GNNs would be a direct application of convolutional operations to graphs

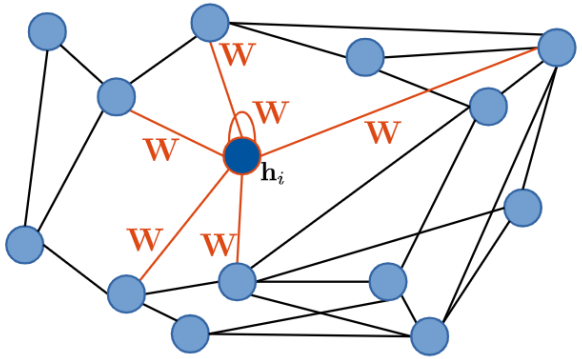
$$\mathbf{h}_i'^{\top} = \sigma\left(\mathbf{h}_i^{\top}\mathbf{W} + \sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} \mathbf{h}_j^{\top}\mathbf{W}\right)$$

  - $\mathbf{h}_i$: vector with node features
  - c_{ij}: normalization constants

channel-wise weight-sharing



node-wise weight-sharing



T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, (2017)

Convolutions in the spatial domain

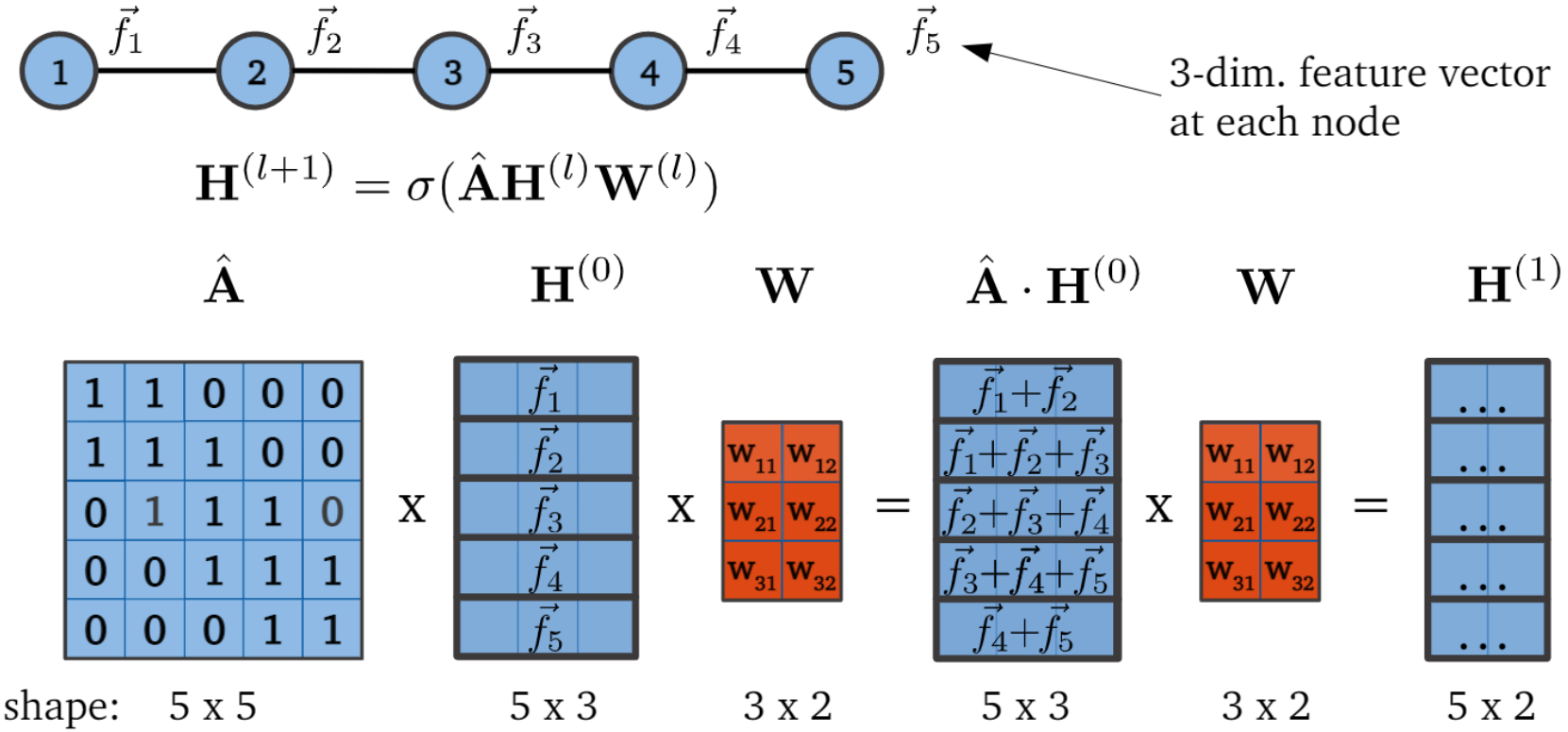
- Assume a graph with n nodes and an $n \times n$ adjacency matrix $\mathbf{A}$
- Input data given by an $n \times d$ matrix $\mathbf{X}$
- Weight matrix $\mathbf{W}$ with $d \times f$ dimensions, with f features
- Propagations would read: $\mathbf{H} = \mathbf{X}\,\mathbf{W}$
- Including self-coupling, the adjacency operations is: $\widehat{\mathbf{A}} = \mathbf{1} + \mathbf{A}$
- By normalizing $\widehat{\mathbf{A}} \rightarrow \widehat{\mathbf{D}}^{-1/2}\,\widehat{\mathbf{A}}\;\widehat{\mathbf{D}}^{-1/2}$, the adjacency is inversely scaled by the number of edges at each node

Convolutions in the spatial domain

- Finally, we have the layer: $f(\mathbf{H}, \mathbf{A}) = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}\mathbf{W}\right)$
- Stacking several such layers with the same graph structure (and same adjacency matrix) forms a **graph convolutional network** (GCN)

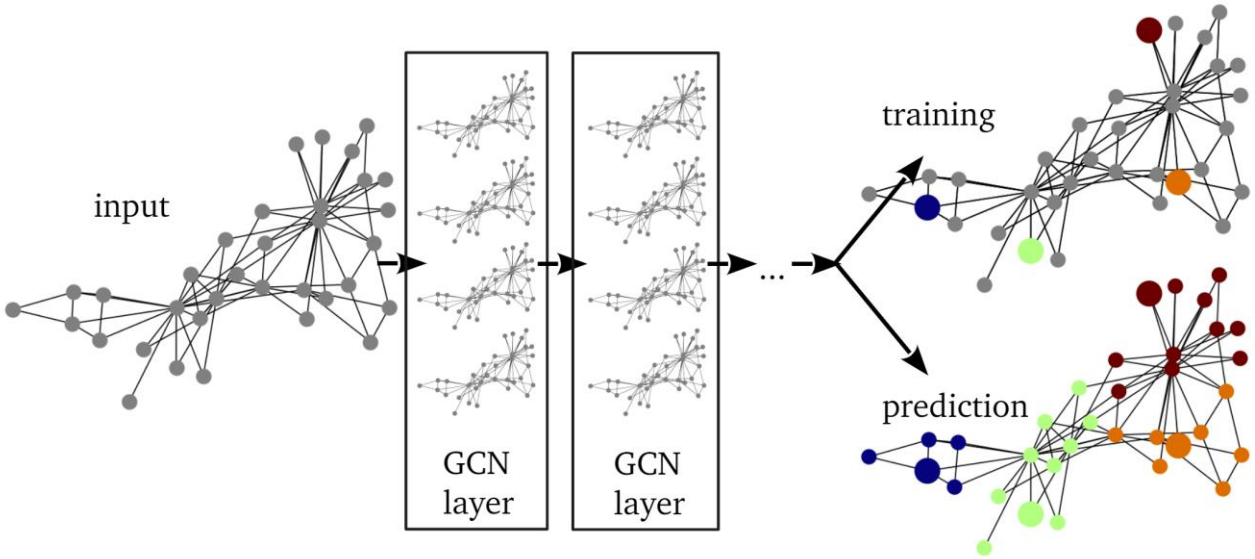
Convolutions in the spatial domain

- 1D example: time series of five numbers



Convolutions in the spatial domain

- Example: Semi-supervised classification on a social network
  - Each of 34 people belongs in one of 4 groups
  - Only one person from each group is known
  - Task is to identify in which group each person belongs based on social connections (incomplete supervised learning)
  - a GCN can be used to analyze the social network, using an undirected, unweighted graph



Convolutions in the spatial domain

- **Edge convolutions** is another method for graph convolutions in the spatial domain

  - Edge convolutions do not require a graph but only a point cloud in a d-dimensional space (advantage)

  - A graph is formed using information of the neighborhood of each point in the space

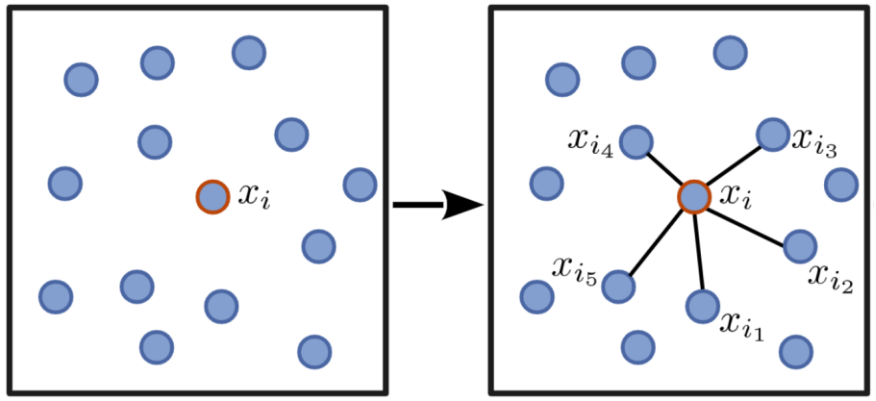
  - Edge convolutions are defined by three steps:
    1. The definition a graph using k-nearest neighbors clustering (kNN)
    2. Features generation by convolution with a kernel function
    3. Aggregation over neighborhood

D. Duvenaudet al., Convolutional networks on graphs for learning molecular fingerprints, (2015)
M. Simonovsky and N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, (2017)

Convolutions in the spatial domain

- **Edge convolutions** is another method for graph convolutions in the spatial domain

1. The definition a graph using **k nearest neighbors** clustering (kNN)
   - Produced graph is directed (non-symmetric $\mathbf{A}$ matrix)



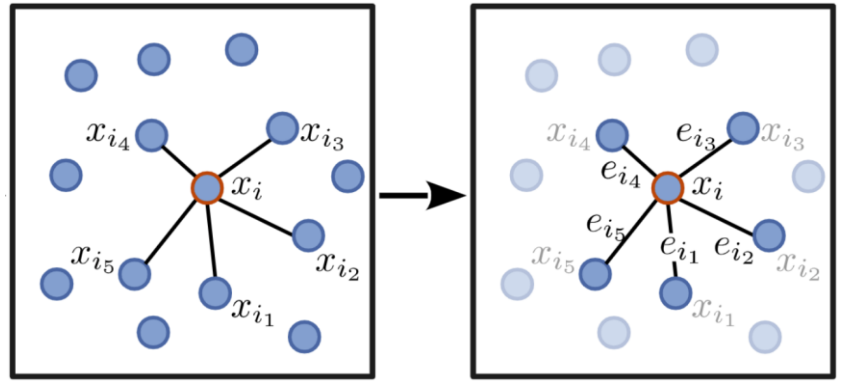
D. Duvenaudet al., Convolutional networks on graphs for learning molecular fingerprints, (2015)
M. Simonovsky and N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, (2017)

Convolutions in the spatial domain

- **Edge convolutions** is another method for graph convolutions in the spatial domain

2. Estimation of edge features, using a kernel function h_θ

  - This function corresponds to a model (typically a fully-connected neural network) with a set of adaptive parameters θ
  - Kernel function is similar to the filter in CNNs
  - The same kernel function is used for all nodes, i.e. **weight sharing**
  - In contrast to the CNNs, the kernel function is continuous



$$e_{i_j} = h_\theta(x_i, x_{i_j})$$

D. Duvenaudet al., Convolutional networks on graphs for learning molecular fingerprints, (2015)

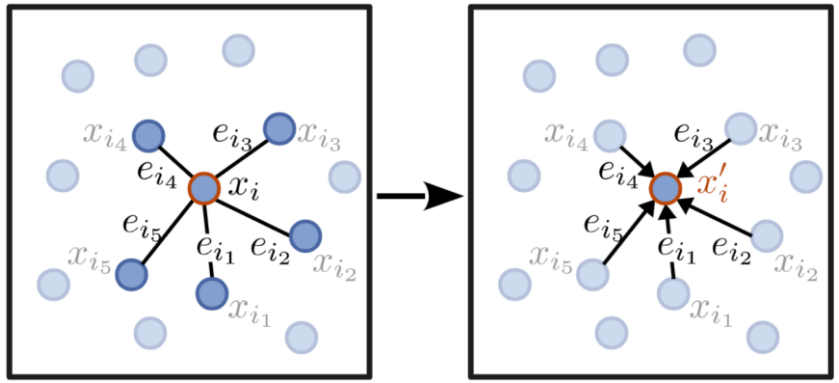
M. Simonovsky and N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, (2017)

Convolutions in the spatial domain

- **Edge convolutions** is another method for graph convolutions in the spatial domain

3. Aggregation over the neighbourhood is performed for calculating final response of the filter

- Aggregation could be summation, mean/maximum value, or other..
- User has full control over the choice of aggregation operation as well as the type of the kernel function:
  - Ability to design convolutions with very different properties
- Output dimension could be more than one, resulting in a feature vector that could be interpreted as the result of different filters



D. Duvenaudet al., Convolutional networks on graphs for learning molecular fingerprints, (2015)

M. Simonovsky and N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, (2017)

Convolutions in the spatial domain

- **Edge convolutions** is another method for graph convolutions in the spatial domain

  - Different symmetries can be exploited using different kernel functions

$$\text{if } h_\theta(x_i - x_{i_j}) \text{ — translational invariance,}$$
$$\text{if } h_\theta(x_i, x_i - x_{i_j}) \text{ — translational invariance + local information,}$$
$$\text{if } h_\theta\left(|x_i - x_{i_j}|\right) \text{ — rotational invariance,}$$

D. Duvenaudet al., Convolutional networks on graphs for learning molecular fingerprints, (2015)
M. Simonovsky and N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, (2017)

Convolutions in the spatial domain

- **Dynamic graph convolutional neural networks** (DGCNNs)

  - When different layers are stacked to form a network, the same graph can be used in all layers
  - Sometimes in might be useful to update the graph from layer to layer
  - In the first layer kNN clustering is used to define the graph by considering neighbors in the spatial domain
  - But in subsequent layers, kNN can be used to define a new graph by considering neighbors in terms of features (outputs of previous layer)
  - Distances now do not have the meaning of spatial proximity but proximity in the feature space

  ➢ DGCNNs find uses in fundamental physics.
    - An example is an architecture called **ParticleNet** used for tagging of particle jets
    - **ParticleNet** achieved the highest accuracy compared to other methods

M. Simonovsky and N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, (2017)
N. G. Kasieczka et al., The machine learning landscape of top taggers, Sci-Post Phys.7, p. 14 (2019)

Convolutions in the spatial domain

- **Deep Sets**
  - More simplified approach: all nodes are connected to one central node, but not to each other
  - Simpler but still robust in many applications
  - Simple to implement and computationally cheaper compared to the full graph networks

M. Zaheer, et. al, Deep sets, NeurIPS (2017)

Convolutions in the spatial domain

- **Message passing neural networks** (MPNNs)

  - This method can create a big variety of graph networks

  - It consists of three steps:
    1. Message phase: each node collects “messages” from its neighbors, using a message function (model) f that depends on node locations x_i, x_j and edge features $e_{i,j}$:

$$\vec{m}_{ij} = f(\vec{x}_i, \vec{x}_j, \vec{e}_{ij})$$

    2. Aggregation phase: combining all messages associated with a node into a single one

$$\vec{m}'_i = \square_{j \in \mathcal{N}_i} \vec{m}_{ij}$$

    3. Update phase:

$$\vec{x}'_i = g(\vec{x}_i, \vec{m}'_i)$$

J. Gilmer, et. al, Neural message passing for quantum chemistry (2017)

Convolutions in the spatial domain

- **Message passing neural networks** (MPNNs)
  - This method can create a big variety of graph networks
  - The three steps, more concretely:
    1. Message phase: each node receives messages accord-ing to the normalized adjacency matrix

$$\vec{m}_{ij} = \frac{1}{\sqrt{\deg(i)\deg(j)}} A_{ij} \vec{x_j}$$

    2. Aggregation phase: received messages are summed up

$$\vec{m}_i = \sum_{j \in \mathcal{N}_i} \vec{m}_{ij}$$

    3. Update phase: applying affine mapping with weights $\mathbf{W}$ and activation function σ

$$\vec{x}'_i = \sigma(\mathbf{W}\vec{m}_i)$$

J. Gilmer, et. al, Neural message passing for quantum chemistry (2017)

Convolutions in the spectral domain

Convolutions in the spectral domain

- Convolutions on graphs can be realized in the Fourier (spectral) spectral
  - Mathematical definition of convolution in the spatial domain, with filter g acting on function f (signal):

$$(f * g)(x) \coloneqq \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$

  - *Convolution theorem*: the Fourier transform $\hat{f} = \mathcal{F}\{f\}$ of a convolution of two functions (or signals) is the pointwise product of their Fourier transforms

$$\mathcal{F}\{f * g\} = \hat{f} \cdot \hat{g}$$

  - It is then useful to write down the convolutional operation using the Fourier transform

$$(f * g) = \mathcal{F}^{-1}\{\hat{g}\, \hat{f}\}$$

F. Monti et al., Geometric deep learning on graphs and manifolds using mixture model CNNs (2016)
J. Bruna et al., Spectral networks and locally connected networks on graphs, (2014)

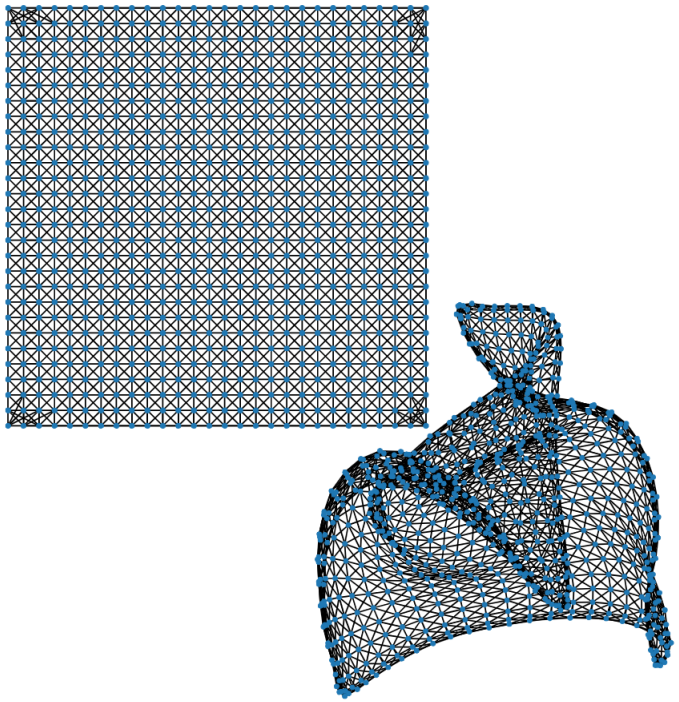
Convolutions in the spectral domain

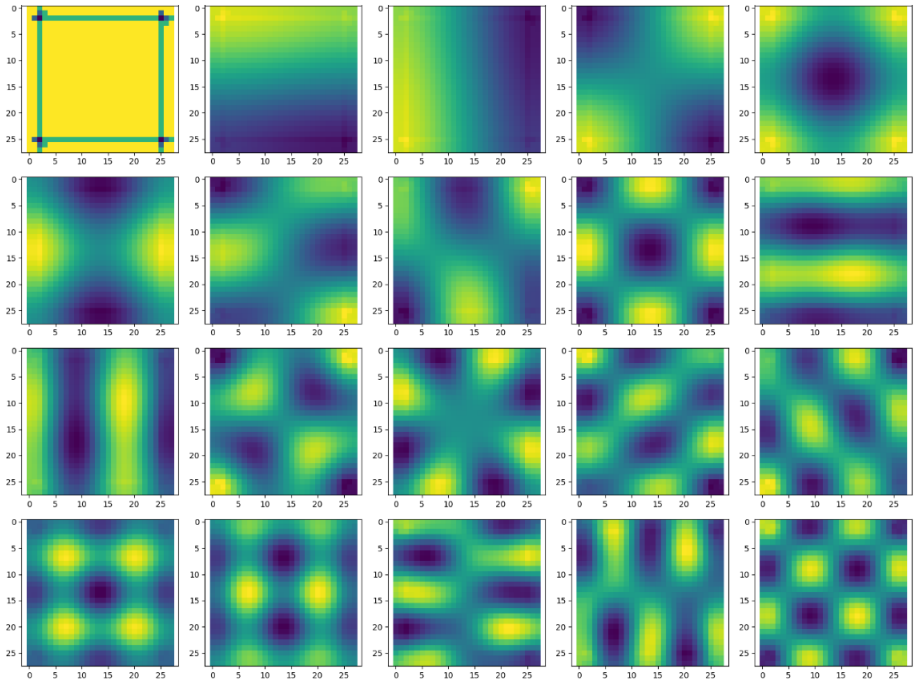
- For graphs, the discrete Fourier transform $\mathbf{\Phi}^{\mathrm{T}}$ is done using the orthonormal eigenvectors of the graph Laplacian $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{\mathrm{T}}$
- Eigenvectors form a Fourier basis: $\mathbf{\Phi} = (\phi_0, \phi_1, \dots, \phi_n)$
- Eigenvalues: $\mathbf{\Lambda} = diag(\lambda_0, \lambda_1, \dots, \lambda_n)$

F. Monti et al., Geometric deep learning on graphs and manifolds using mixture model CNNs (2016)
J. Bruna et al., Spectral networks and locally connected networks on graphs, (2014)

Convolutions in the spectral domain

- Example of eigenvector decomposition of a graph





F. Monti et al., Geometric deep learning on graphs and manifolds using mixture model CNNs (2016)
J. Bruna et al., Spectral networks and locally connected networks on graphs, (2014)

Convolutions in the spectral domain

- **Spectral convolutional networks** realize convolutions in the spectral domain using the convolutional theorem
- Assume we need to analyze a signal function f using a filter a filter function w, then the convolution operation in the spatial domain is given by $f * w = \mathbf{W}_\theta f$
- $\mathbf{W}_\theta$ is a matrix with all adaptive weights for the spatial domain
- For learning filters in the spectral domain, the Fourier-transformed signal $\hat{f} = \mathbf{\Phi}^{\mathrm{T}} f$ needs to be used instead:

$$f * w = \mathbf{W}_\theta(\mathbf{L})f = \mathbf{W}_\theta(\Phi\mathbf{\Lambda}\Phi^T)f = \mathbf{\Phi}\hat{\mathbf{W}}_\theta(\mathbf{\Lambda})\mathbf{\Phi}^T f = \mathbf{\Phi}\hat{\mathbf{W}}_\theta\mathbf{\Phi}^T f$$

- $\widehat{\mathbf{W}}_\theta = diag(\theta_0, \theta_1, \dots, \theta_n)$ is the kernel matrix with adaptive weights in the spectral domain

F. Monti et al., Geometric deep learning on graphs and manifolds using mixture model CNNs (2016)
J. Bruna et al., Spectral networks and locally connected networks on graphs, (2014)

Convolutions in the spectral domain

- The approach of using convolutions in the spectral domain appears to be very powerful
- However, for the same reason, it is prone to limitations. The issue is that the learned kernels in the spectral domain typically do not act locally in the spatial domain but globally
- Another issue is that since the produced filters depend on the eigenvalues of $\mathbf{L}$, they also depend on the particular graph structure that was used for training. This means that the that the trained network model will not generalizable
- Another limitation is that the number of adaptive parameters scales with the number of nodes

F. Monti et al., Geometric deep learning on graphs and manifolds using mixture model CNNs (2016)
J. Bruna et al., Spectral networks and locally connected networks on graphs, (2014)

Convolutions in the spectral domain

- To overcome these limitations, $\widehat{\mathbf{W}}_\theta$ is approximated using a smooth transfer function $\tau_\theta(\lambda)$, which depends on the eigenvalues λ

- To design filters that depend less on the eigenvalues of $\mathbf{L}$, the transfer function is expanded in powers of λ, with a cutoff value k (hyperparameter) controlling the complexity of the filter

$$\tau_\theta(\lambda) \approx \sum_{i=0}^{k-1} \theta_i \lambda^i$$

- Applying this approximation, yields a more stable and localized convolution

$$\mathbf{\Phi}\hat{\mathbf{W}}_\theta\mathbf{\Phi}^T f \approx \mathbf{\Phi} \begin{pmatrix} \tau_\theta(\lambda_0) & & \\ & \ddots & \\ & & \tau_\theta(\lambda_{n-1}) \end{pmatrix} \mathbf{\Phi}^T f$$

M. Henaff, J. Bruna and Y. LeCun, Deep convolutional networks on graph-structured data (2015)

Convolutions in the spectral domain

- This modified convolution results into a set of parameters that do not depend on the number of nodes – can produce models that are more stable (generalizable)

- Also, due to the fact that the $\tau_\theta(\lambda)$ is a smooth function in the spectral domain, it corresponds to a localized filter in the spatial domain

M. Henaff, J. Bruna and Y. LeCun, Deep convolutional networks on graph-structured data (2015)

Convolutions in the spectral domain

- Even though this formulation allows for stable and localized filters, it is computationally expensive due to the eigenvector decomposition of $\mathbf{L}$
- For different graph structures, the eigenvector decomposition would have to be carried out individually for each graph
- But a method called **fast filtering** exists for speeding up calculations

M. Defferrard et al., Convolutional neural networks on graphs with fast localized spectral filtering (2017)

Convolutions in the spectral domain

- **Fast filtering** works by rewriting the convolution in the original form

$$\mathbf{\Phi}\hat{\mathbf{W}}_\theta\mathbf{\Phi}^T f = \mathbf{\Phi}\hat{\mathbf{W}}_\theta(\mathbf{\Lambda})\mathbf{\Phi}^T f = \mathbf{W}_\theta(\mathbf{L})f$$

- The idea is to approximate $\mathbf{W}_\theta(\mathbf{L})f$ directly

$$\mathbf{W}_\theta(\tilde{\mathbf{L}})f \approx \sum_{i=0}^{k-1} \theta_i T_i(\tilde{\mathbf{L}})f$$

- T_i correspond to the Chebyshev polynomials, which form an orthogonal basis in $[-1,1]$

- This leads to a rescaled Laplacian $\tilde{\mathbf{L}} = 2\frac{\mathbf{L}}{\lambda_{max}} - \mathbf{1}$ with eigenvalues between [−1,1]

M. Henaff, J. Bruna and Y. LeCun, Deep convolutional networks on graph-structured data (2015)

Convolutions in the spectral domain

- The trick comes from recursive definition of the Chebyshev polynomials:

$$T_{n+1}(\tilde{\mathbf{L}}) = 2\tilde{\mathbf{L}} \cdot T_n(\tilde{\mathbf{L}}) - T_{n-1}(\tilde{\mathbf{L}}), \quad \text{and } T_0(\tilde{\mathbf{L}}) = \mathbb{1}, \quad T_1(\tilde{\mathbf{L}}) = \tilde{\mathbf{L}}$$

- The expansion can be determined very efficiently, since the eigenvectors of the Laplacian do not need to be determined

- Neural networks that are based on this convolution approach are called **ChebNets**

- In this case, k does not correspond to the complexity of the filter, but also to the distance in the spatial domain that is affected by the filter

  - For $k = 0$, only the central node is considered
  - For $k = 1$, all adjacent neighbors are considered
  - For $k = 2$, the neighbors of the neighbors… and so on

M. Henaff, J. Bruna and Y. LeCun, Deep convolutional networks on graph-structured data (2015)