Graph Neural Networks (GNN)

Makoto Yamada

myamada@i.kyoto-u.ac.jp

Kyoto University

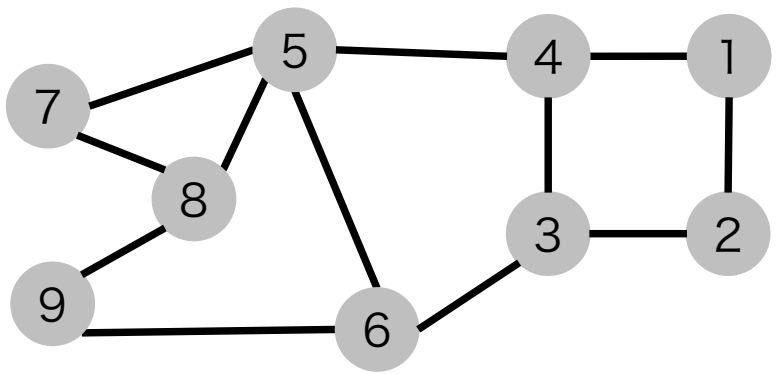
2020/7/20

Graph

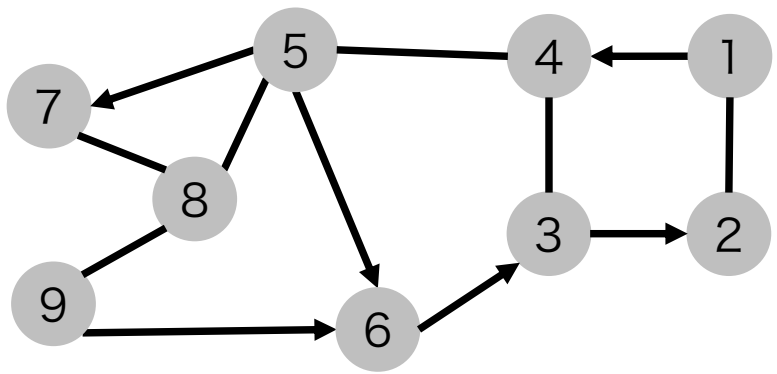
Graph

A graph consists of Nodes V and edges E. $G = (V, E)$

Undirected graph



Directed graph



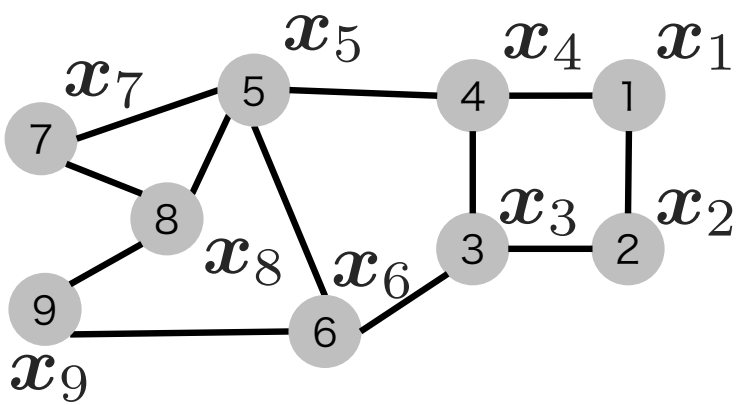
Attributed graph

Graph + features

In each node, a feature vector is given. $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$

We mainly use **Attributed graphs** for GNN.

Attributed graph



Adjacency matrix & Degree matrix

Adjacency matrix A

$n \times n$ matrix with **edge** information

$$a_{ij} = \begin{cases} 1 & \text{(i-th node is connected to j-th node)} \\ 0 & \text{(Otherwise)} \end{cases}$$

Degree matrix D

$n \times n$ matrix with **degree** information.

$$d_{ii} = \sum_{k=1}^{n} a_{ik}$$

$$\boldsymbol{D} = \text{diag}(d_{11}, d_{22}, \ldots, d_{nn}) \in \mathbb{R}^{n \times n}$$

Graph Laplacian

Graph Laplacian

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A},$$

- $\boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(u_i - u_j)^2 \geq 0$ (PSD)
- $\boldsymbol{L}\mathbf{1}_n = \mathbf{0}_n$ (minimum eigenvalue is zero)

Normalized graph Laplacian

$$\bar{\boldsymbol{L}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I}_n - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$$

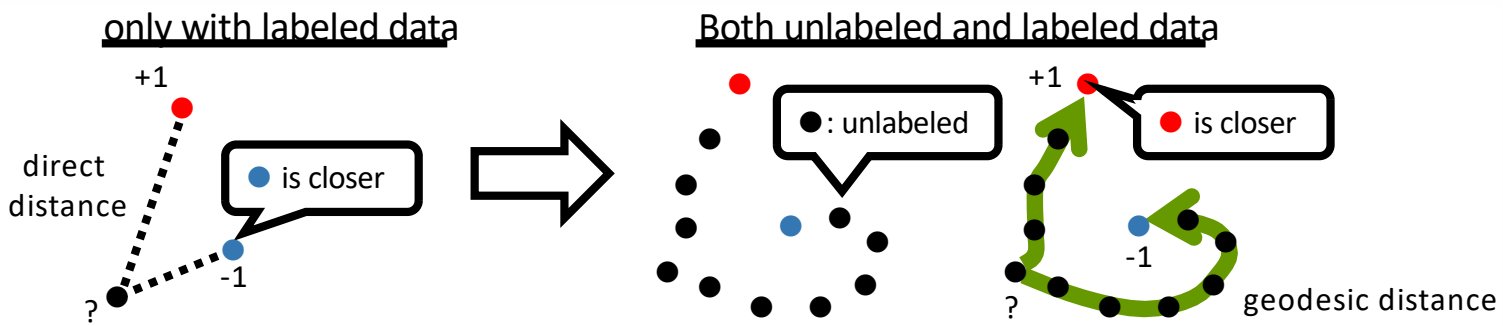
- $\boldsymbol{u}^\top \bar{\boldsymbol{L}}\boldsymbol{u} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(\frac{u_i}{\sqrt{d_{ii}}} - \frac{u_j}{\sqrt{d_{jj}}})^2 \geq 0$ (PSD)
  ($\widetilde{\boldsymbol{u}} = \boldsymbol{D}^{-1/2}\boldsymbol{u}$ and $\boldsymbol{x}^\top \bar{\boldsymbol{L}}\boldsymbol{u} = \widetilde{\boldsymbol{u}}^\top \boldsymbol{L}\widetilde{\boldsymbol{u}}$)
- $\bar{\boldsymbol{L}}\boldsymbol{D}^{1/2}\mathbf{1}_n = \bar{\boldsymbol{L}}\boldsymbol{d} = \mathbf{0}_n$ (minimum and maximum eigenvalues are 0 and 2, respectively)

Graph based Machine Learning

We construct graphs and use it for prediction.

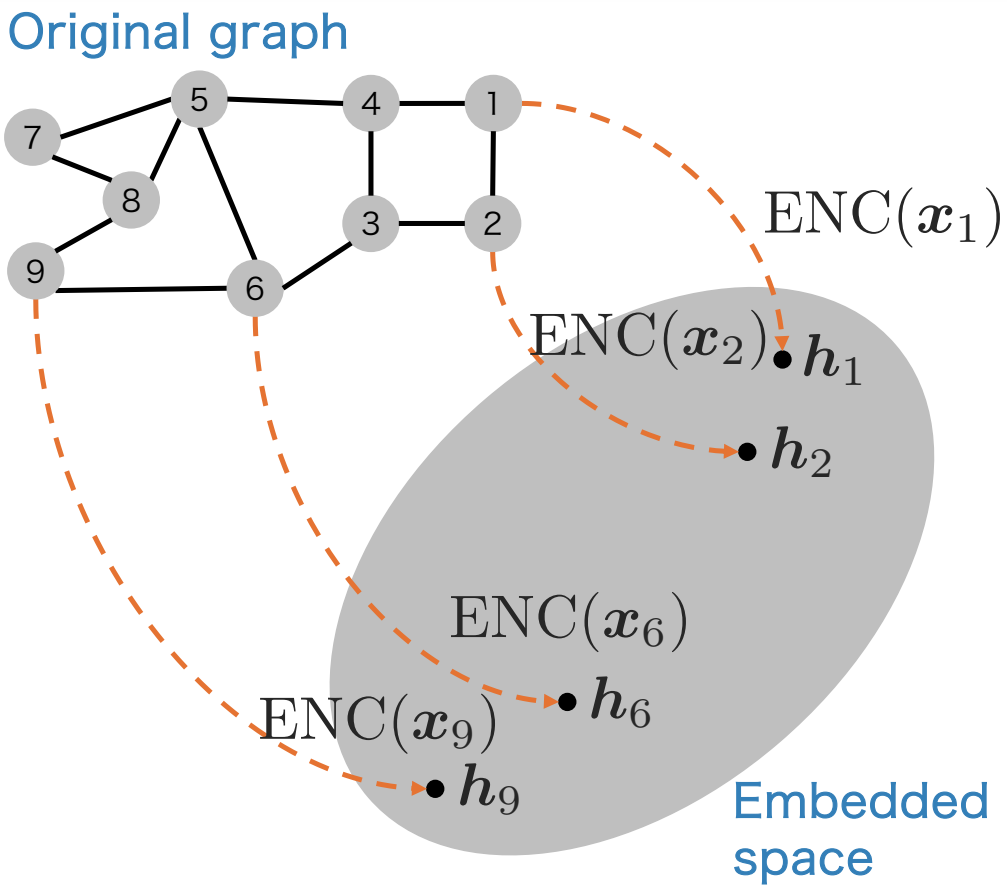
Graph stores the adjacent information between samples.

- K nearest neighbor graph $(A_{ij} = 0, 1)$
- Weighted graph $(A_{ij} = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2)$



What is GNN?

It is an **embedding method** for graph.



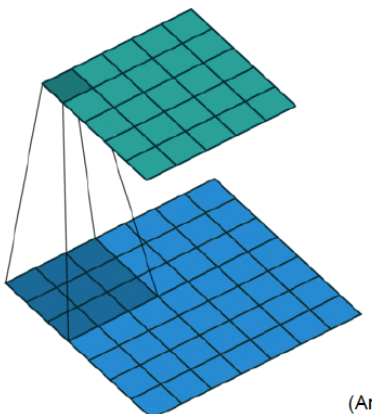
Convolution

Convolutional Neural Networks (CNN)

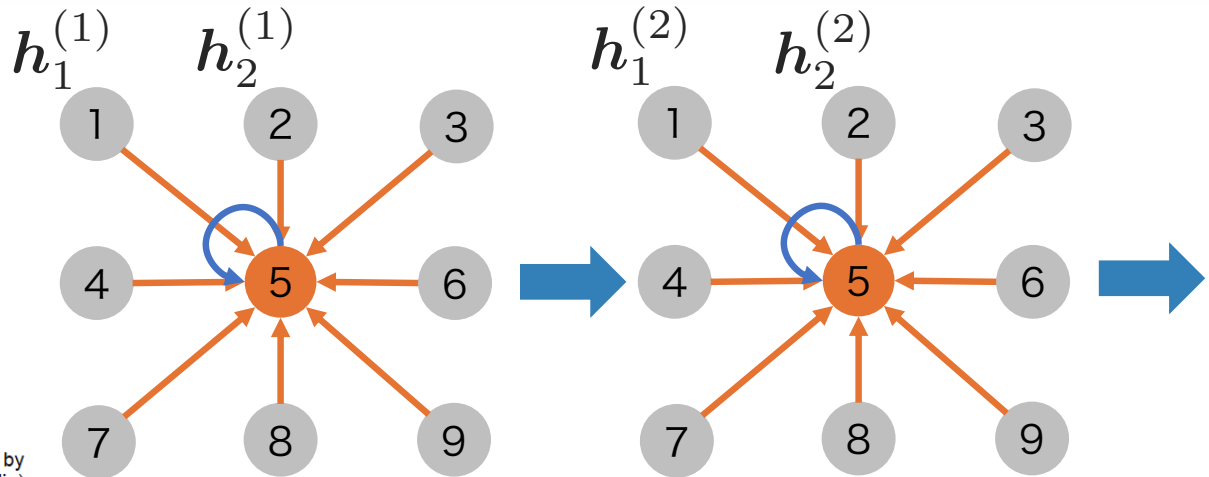
- Aggregate neighbor pixels.
- Apply $\sigma(\cdot)$ and propagate the information to higher layers.

e.g., Convolution for 5th pixel.

$$\boldsymbol{h}_5^{(\ell+1)} = \sigma\left(\boldsymbol{W}_1^{(\ell)}\boldsymbol{h}_1^{(\ell)} + \boldsymbol{W}_2^{(\ell)}\boldsymbol{h}_2^{(\ell)} + \ldots + \boldsymbol{W}_9^{(\ell)}\boldsymbol{h}_9^{(\ell)}\right)$$

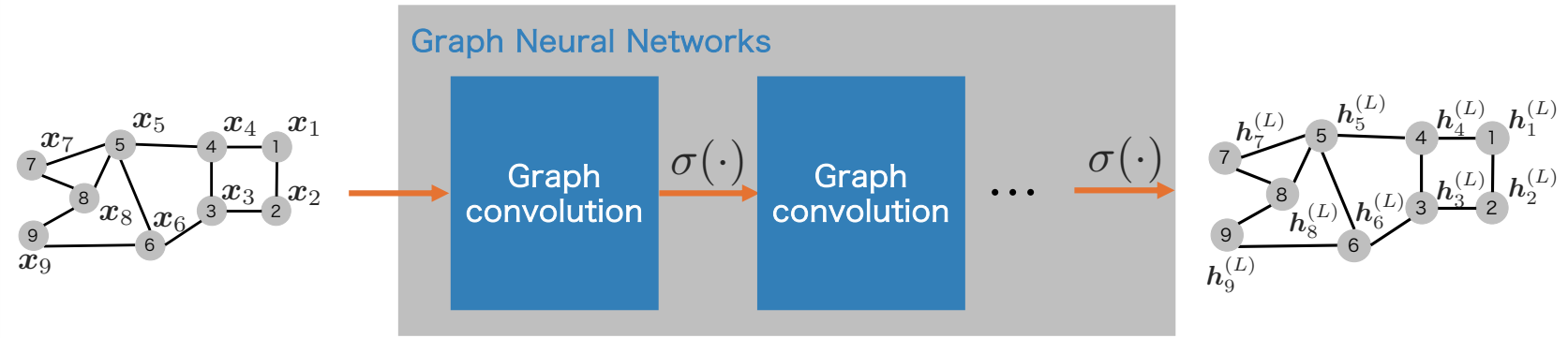


(Animation by Vincent Dumoulin)



Graph Neural Networks (GNN)

GNN: **Neural network with graph convolution**.



Task $\boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{h}_i^{(L)}, \boldsymbol{W} \in \mathbb{R}^{K \times d_L}$

- Node classification : $\mathrm{softmax}(\boldsymbol{z}_i)$
- Graph classification : $\mathrm{softmax}(\sum_i \boldsymbol{z}_i)$
- Link prediction : $p(a_{ij}) = \sigma(\boldsymbol{z}_i^\top \boldsymbol{z}_j)$

GNN (Node classification)

We compute probability by using **an additional layer** ($\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K)^\top \in \mathbb{R}^{K \times d_L}$) to the graph embedding $\boldsymbol{h}_i^{(L)}$

$$\boldsymbol{z}_i = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_i^{(L)}) = \begin{pmatrix} \frac{\exp(\boldsymbol{w}_1^\top \boldsymbol{h}_i^{(L)})}{\sum_{k'=1}^K \exp(\boldsymbol{w}_{k'}^\top \boldsymbol{h}_i^{(L)})} \\ \vdots \\ \frac{\exp(\boldsymbol{w}_K^\top \boldsymbol{h}_i^{(L)})}{\sum_{k'=1}^K \exp(\boldsymbol{w}_{k'}^\top \boldsymbol{h}_i^{(L)})} \end{pmatrix}$$

Loss function (Training data: $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, $\boldsymbol{A} \in \mathbb{R}^{n \times n}$)

$$\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K y_{k,i} \log(z_{k,i})$$

$\boldsymbol{z}_i = (z_{1,i}, \ldots, z_{K,i})^\top$, $\boldsymbol{y}_i$ is an one-hot vector.

GNN (Graph classification)

We transform the output of graph convolution $\boldsymbol{h}_i^{(L)}$ by an additional layer and **aggregate them** to compute probability.

$$\boldsymbol{z}_i = \mathrm{softmax}(\boldsymbol{W} \sum_i \boldsymbol{h}_i^{(L)}) = \begin{pmatrix} \frac{\exp(\boldsymbol{w}_1^\top \sum_i \boldsymbol{h}_i^{(L)})}{\sum_{k'=1}^K \exp(\boldsymbol{w}_{k'}^\top \sum_i \boldsymbol{h}_i^{(L)})} \\ \vdots \\ \frac{\exp(\boldsymbol{w}_K^\top \sum_i \boldsymbol{h}_i^{(L)})}{\sum_{k'=1}^K \exp(\boldsymbol{w}_{k'}^\top \sum_i \boldsymbol{h}_i^{(L)})} \end{pmatrix}$$

Loss function (Training data: $\{(\boldsymbol{X}_i, \boldsymbol{y}_i, \boldsymbol{A}_i)\}_{i=1}^n$)

$$\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K y_{k,i} \log(z_{k,i})$$

$\boldsymbol{z}_i = (z_{1,i}, \ldots, z_{K,i})^\top$, $\boldsymbol{y}_i$ is a one-hot vector.

GNN (Link prediction)

The output of an GNN: $\boldsymbol{z}_i = \boldsymbol{h}_i^{(L)}$

Generative model

$$p(\boldsymbol{A}|\boldsymbol{Z}) = \prod_{i=1}^{n}\prod_{i=1}^{n} p(a_{ij}|\boldsymbol{z}_i, \boldsymbol{z}_j),$$
$$p(a_{ij} = 1|\boldsymbol{z}_i, \boldsymbol{z}_j) = \sigma(\boldsymbol{z}_i^\top \boldsymbol{z}_j)$$

$\sigma(\cdot)$ is the sigmoid function.

Loss function ($\boldsymbol{X} \in \mathbb{R}^{d\times n}, \boldsymbol{A} \in \mathbb{R}^{n\times n}$)

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}[\log p(\boldsymbol{A}|\boldsymbol{Z})] - \mathrm{KL}[q(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})\|p(\boldsymbol{Z})]$$

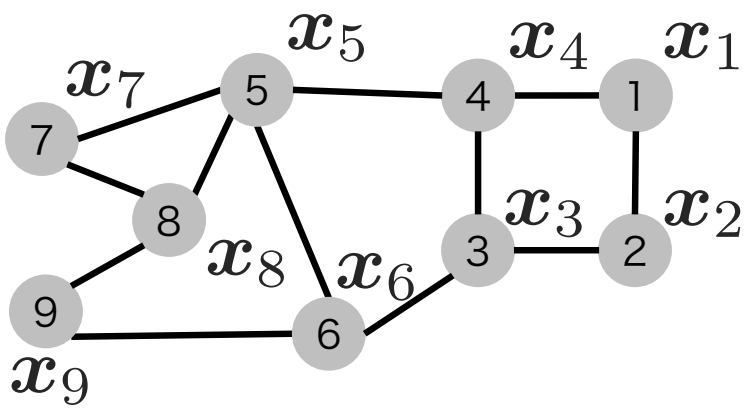
$\mathrm{KL}[q(\cdot)\|p(\cdot)]$ is the KL-divergence.

Graph Convolutional Networks (GCN)

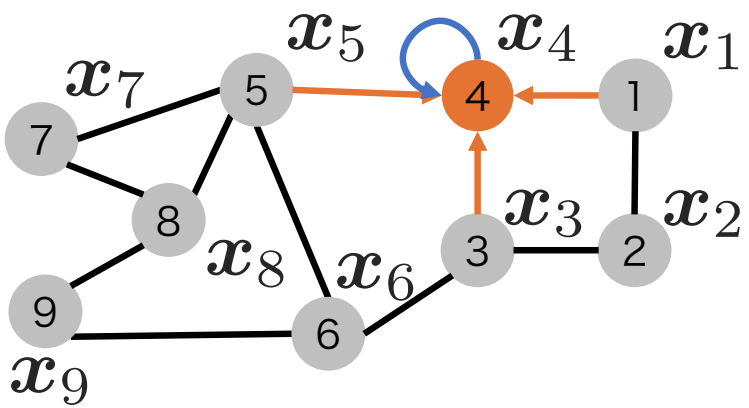
Graph convolution

$$\boldsymbol{h}_i^{(\ell+1)} = \sigma\left(\boldsymbol{W}_0^{(\ell)}\boldsymbol{h}_i^{\ell} + \sum_{j:a_{i,j}\neq 0}\frac{1}{c_{i,j}}\boldsymbol{W}_1^{\ell}\boldsymbol{h}_j^{(\ell)}\right)$$

Attributed graph



Graph convolution



Graph Convolutional Networks (GCN)

Graph convolution

$$\boldsymbol{h}_i^{(\ell+1)} = \sigma\left(\boldsymbol{W}_0^{(\ell)}\boldsymbol{h}_i^{\ell} + \sum_{j:a_{i,j}\neq 0}\frac{1}{c_{i,j}}\boldsymbol{W}_1^{\ell}\boldsymbol{h}_j^{(\ell)}\right)$$

Definition of variables

- $\boldsymbol{A} \in \mathbb{R}^{n\times n}$: Adjacency matrix
- $\boldsymbol{W}_1^{(\ell)} \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$: Transformation matrix
- $\boldsymbol{h}_i^{(\ell)} \in \mathbb{R}^{m_\ell}$: i-th sample's intermediate representation of the ℓ-th layer
- $\sigma(\cdot)$: Activation function

Explain about graph convolution through **graph Fourier transform**.

Fourier transform

Transform real signal to frequencies.

Discrete Fourier transform

$$X(k) = \sum_{i=0}^{n-1} x(k) e^{\frac{-j2\pi ki}{n}} = \sum_{i=0}^{n-1} x(k) w_n^{(ki)}$$

Matrix representation

$$\begin{pmatrix} X(0) \\ X(1) \\ \vdots \\ X(n-1) \end{pmatrix} = \begin{pmatrix} w_n^{(0)} & w_n^{(0)} & \cdots & w_n^{(0)} \\ w_n^{(1)} & w_n^{(2)} & \cdots & w_n^{(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_n^{(0)} & w_n^{(n-1)} & \cdots & w_n^{(1)} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ x(n-1) \end{pmatrix}$$

Low frequency component tends to be constant.

Graph Fourier transform

Definition Eigenvalue decomposition of the (normalized) graph Laplacian.

$$\bar{\boldsymbol{L}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$$

Normalized graph Laplacian

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

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A},$$

where

$$\boldsymbol{U} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n)$$

$$\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n), 0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$$

Graph Fourier transform

Eigenvalue decomposition (1st eigenvalue)

Since we have $\boldsymbol{L}\mathbf{1}_n = 0\mathbf{1}_n$ and $\boldsymbol{L}$ is a PSD matrix, $\frac{1}{\sqrt{n}}\mathbf{1}_n$ is the corresponding eigenvector.

Moreover, we can write the eigenvalue decomposition as

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(u_i - u_j)^2 \\ \text{s.t.} \quad & \boldsymbol{u}^\top \boldsymbol{u} = 1. \end{aligned}$$

Thus, the minimum is achieved when
$u_1 = u_2 = \ldots = u_n =$ Const. with its eigenvector $\frac{1}{\sqrt{n}}\mathbf{1}_n$. (i.e., **Low pass filter**)

Graph Fourier transform

Eigenvalue decompsition (2nd eigenvalue)

$$\min_{\boldsymbol{u}} \quad \boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(u_i - u_j)^2$$

$$\text{s.t.} \quad \boldsymbol{u}^\top \boldsymbol{u} = 1, \boldsymbol{u}^\top \boldsymbol{u}_1 = 0.$$

$\boldsymbol{u}_1 = \frac{1}{n}\mathbf{1}_n$ is the first eigenvector.

Interpretation

We want to have an eigenvector that is orthogonal to the 1st eigenvector $\boldsymbol{u}_1$ and minimizes $\boldsymbol{u}^\top \boldsymbol{L}\boldsymbol{u}$.

That is, eigenvectors with small eigenvalues are **low pass filters** and eigenvectors with high eigenvalues are **high pass filters**.

Graph convolutions (Graph Fourier)

We apply (graph) Fourier transform to the input and then apply filters to the transformed vector. Then, we apply inverse (graph) Fourier transform to the filtered signal.

Graph Fourier transform

$$\widetilde{\boldsymbol{x}} = \boldsymbol{U}^\top \boldsymbol{x}$$

Convolution (**multiplication in frequency domain.**)

$$\boldsymbol{\Theta}\widetilde{\boldsymbol{x}} = \boldsymbol{\Theta}\boldsymbol{U}^\top \boldsymbol{x}$$

Inverse Graph Fourier transform

$$\boldsymbol{x}_{\text{filtered}} = \boldsymbol{U}\boldsymbol{\Theta}\boldsymbol{U}^\top \boldsymbol{x}$$

$\boldsymbol{\Theta}$ is estimated from training data set.

Graph convlution (ChebyNet)

In graph Fourier transform, it needs to solve eigenvalue decomposition $O(n^3)$.

ChebyNet Compute filter **without** eigenvalue decomposition

$$\boldsymbol{\Theta}(\boldsymbol{\Lambda}) \approx \sum_{k=1}^{K} \theta_k T_k \left(\frac{2}{\lambda_{\text{max}}} \boldsymbol{\Lambda} - \boldsymbol{I}_n \right)$$

θ_k are the model parameters, $T_k(\Lambda)$ is the k-th Chebychev polinomial.

$$T_k(\boldsymbol{\Lambda}) = 2\boldsymbol{\Lambda} T_{k-1}(\boldsymbol{\Lambda}) - T_{k-2}(\boldsymbol{\Lambda})(k > 2), T_0(\boldsymbol{\Lambda}) = \boldsymbol{I}_n, T_1(\boldsymbol{\Lambda}) = \boldsymbol{\Lambda}$$

$$\boldsymbol{x}_{\text{filtered}} = \boldsymbol{U}\boldsymbol{\Theta}\boldsymbol{U}^\top \boldsymbol{x} \approx \left[\sum_{k=1}^{K} \theta_k T_k \left(\frac{2}{\lambda_{\text{max}}} \boldsymbol{L} - \boldsymbol{I}_n \right) \right] \boldsymbol{x}$$

Graph convolution (GCN)

We use the normalized graph Laplacian ($\lambda_{\text{max}} = 2$) and use the 1st order ChebyNet.

$$
\begin{aligned}
\boldsymbol{x}_{\text{filtered}} &= \boldsymbol{U}\left[\theta_0 \boldsymbol{I}_n + \theta_1(\boldsymbol{\Lambda} - \boldsymbol{I}_n)\right]\boldsymbol{U}^\top \boldsymbol{x} \\
&= \left[\theta_0 \boldsymbol{U}\boldsymbol{U}^\top + \theta_1(\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top - \boldsymbol{I}_n)\right]\boldsymbol{x} \\
&= \left[\theta_0 \boldsymbol{I}_n + \theta_1(\boldsymbol{I}_n - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{\frac{1}{2}} - \boldsymbol{I}_n)\right]\boldsymbol{x} \\
&= \theta_0 \boldsymbol{x} - \theta_1 \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{x}
\end{aligned}
$$

Thus, this can be written more generally as

$$
\boldsymbol{h}_i^{(\ell+1)} = \sigma\left(\boldsymbol{W}_0^{(\ell)}\boldsymbol{h}_i^{\ell} + \sum_{j:a_{i,j}\neq 0} \frac{1}{c_{i,j}} \boldsymbol{W}_1^{\ell}\boldsymbol{h}_j^{(\ell)}\right)
$$

Graph convolution (matrix form)

We can write the graph convolution by matrix multiplications.

Without normalization

$$\boldsymbol{H}^{(\ell+1)} = \sigma\left(\boldsymbol{W}_1^{(\ell)} \boldsymbol{H}^{(\ell)} \boldsymbol{A}\right)$$

With normalization

$$\boldsymbol{H}^{(\ell+1)} = \sigma\left(\boldsymbol{W}_1^{(\ell)} \boldsymbol{H}^{(\ell)} \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}\right)$$

There exist many variants.

GNN (Node classification)

We compute probability by using **an additional layer** ($\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K)^\top \in \mathbb{R}^{K \times d_L}$) to the graph embedding $\boldsymbol{h}_i^{(L)}$

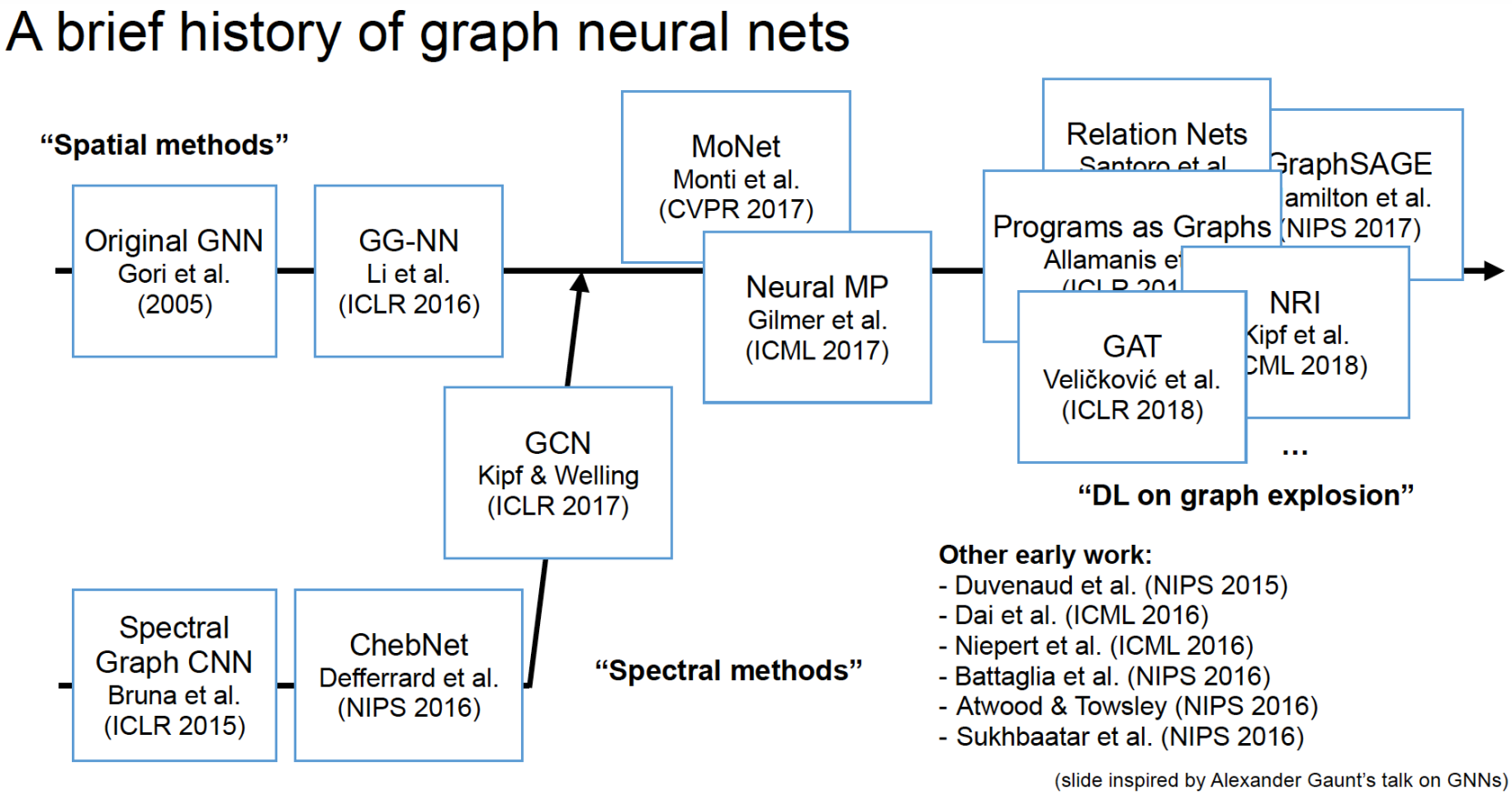
$$\boldsymbol{z}_i = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_i^{(L)}) = \begin{pmatrix} \frac{\exp(\boldsymbol{w}_1^\top \boldsymbol{h}_i^{(L)})}{\sum_{k'=1}^K \exp(\boldsymbol{w}_{k'}^\top \boldsymbol{h}_i^{(L)})} \\ \vdots \\ \frac{\exp(\boldsymbol{w}_K^\top \boldsymbol{h}_i^{(L)})}{\sum_{k'=1}^K \exp(\boldsymbol{w}_{k'}^\top \boldsymbol{h}_i^{(L)})} \end{pmatrix}$$

Loss function (Training data: $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, $\boldsymbol{A} \in \mathbb{R}^{n \times n}$)

$$\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K y_{k,i} \log(z_{k,i})$$

$\boldsymbol{z}_i = (z_{1,i}, \ldots, z_{K,i})^\top$, $\boldsymbol{y}_i$ is an one-hot vector.

Various types of GNNs



http://tkipf.github.io/misc/SlidesCambridge.pdf

Summary

- Graph data
- Introduction to graph neural networks
- Graph Fourier transform
- Graph convolution
- Graph Convolutional Networks (GCN)