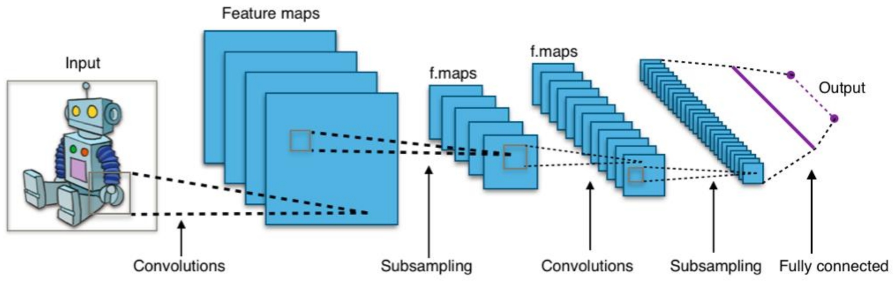
Graph
Convolutional Neural Networks

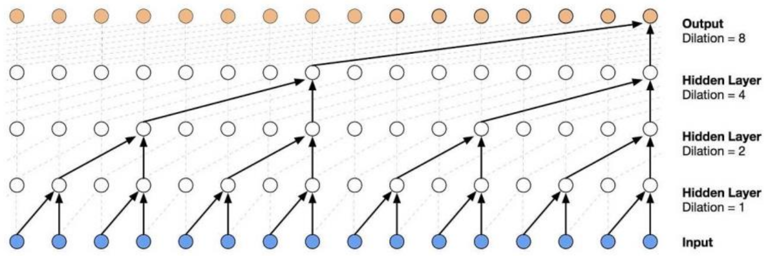
沈华伟
中国科学院计算技术研究所

Convolutional Neural Network

- Convolutional neural network (CNN) gains great success on Euclidean data, e.g., image, text, audio, and video
  - Image classification, object detection, machine translation



Convolutional neural networks on image



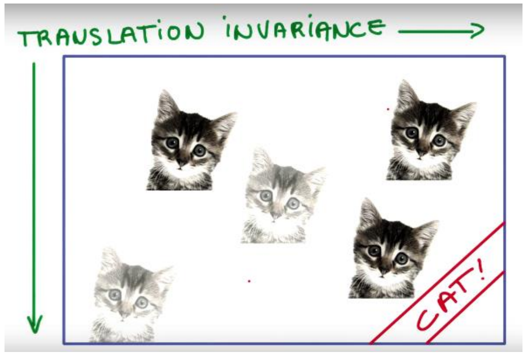
Temporal convolutional network

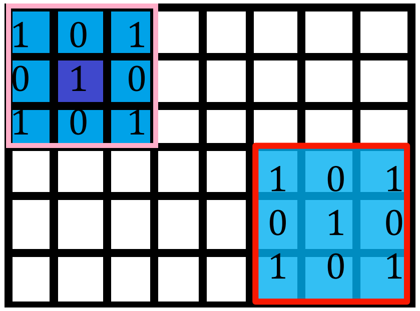
- The power of CNN lies in
  - its ability to learn local stationary structures, via localized convolution filter, and compose them to form multi-scale hierarchical patterns

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst. Geometric deep learning: going beyond Euclidean data. IEEE Signal Processing Magazine, 18-42, 2017.

Convolutional Neural Network

- **Localized convolutional filters are translation- or shift-invariant**
  - **Which are able to recognize identical features independently of their spatial locations**





X-Shape

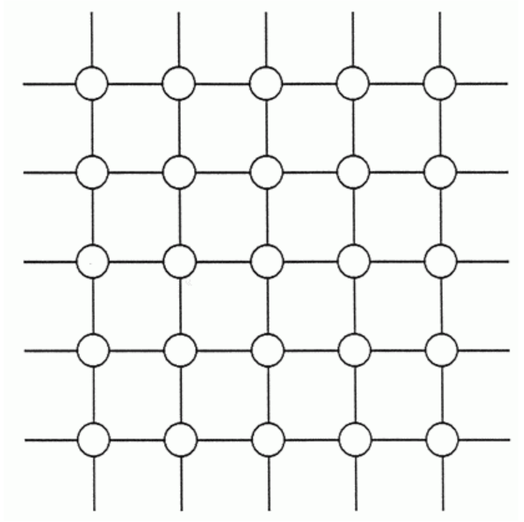
Template Matching

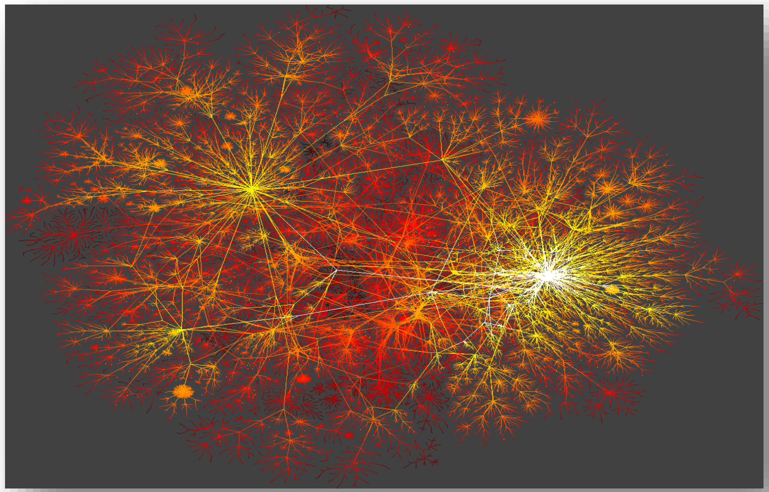
$$\begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix}$$

- **One interesting problem is how to generalize convolution to non-Euclidean domain, e.g., graph?**
  - **Irregular structure of graph poses challenges for defining convolution for graph data**

From CNN to graph CNN

- Convolution is well defined in Euclidean data, grid-like network
- Not straightforward to define convolution on irregular network, widely observed in real world


Grid-like network

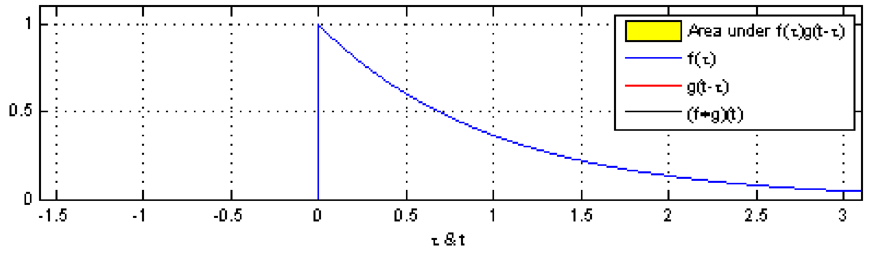

Irregular networks

Convolution

- Convolution is a mathematical operation on two functions, f and g, to produce a third function h.
  - Defined as the integral, in continuous case, or sum, in discrete case, of the product of the two functions after one is reversed and shifted.

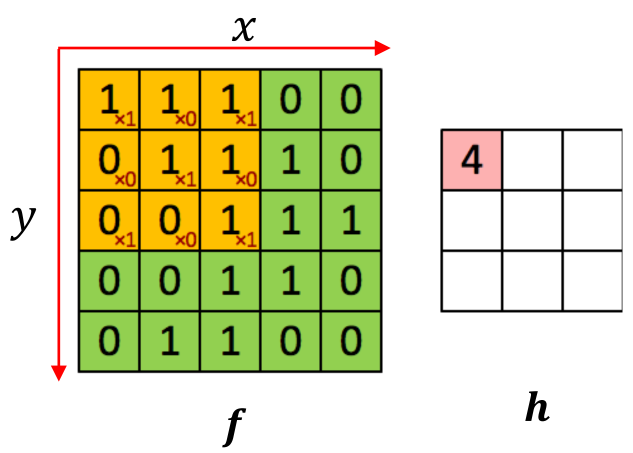
Continuous case

$$h(t) = (f * g)(t) \overset{\text{def}}{=} \int f(t-\tau)g(t)\,d\tau$$



Discrete case

$$h(x,y) = (f * g)(x,y) \overset{\text{def}}{=} \sum_{m,n} f(x-m, y-n)g(m,n)$$



$g=$

$g(1,1)$ 1	$g(0,1)$ 0	$g(-1,1)$ 1
$g(1,0)$ 0	$g(0,0)$ 1	$g(-1,0)$ 0
$g(1,-1)$ 1	$g(0,-1)$ 0	$g(-1,-1)$ 1

Existing methods to define convolution

- **Spectral methods: define convolution in spectral domain**
  - Convolution is defined via graph Fourier transform and convolution theorem.
  - The main challenge is that convolution filter defined in spectral domain is not localized in vertex domain.

- **Spatial methods: define convolution in the vertex domain**
  - Convolution is defined as a weighted average function over all vertices located in the neighborhood of target vertex.
  - The main challenge is that the size of neighborhood varies remarkably across nodes, e.g., power-law degree distribution.

Spectral methods for graph convolutional neural networks

Spectral methods

- **Given a graph $G = (V, E, W)$**
  - **V is node set with $n = |V|$, E is edge set, and $W \in R^{n \times n}$ is the weighted adjacency matrix**
  - **Each node is associated with d features, and $X \in R^{n \times d}$ is the feature matrix of nodes, each column of X is a signal defined over nodes**

- **Graph Laplacian**
  - **$L = D - W$, where is a diagonal matrix with $D_{ii} = \sum_j W_{ij}$**
  - **Normalized graph Laplacian**

$$L = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

where I is the identity matrix.

Graph Fourier Transform

- **Fourier basis of graph G**
  - **The complete set of orthonormal eigenvectors $\{u_l\}_{l=1}^{n}$ of L, ordered by its non-negative eigenvalues $\{\lambda_l\}_{l=1}^{n}$**
  - **Graph Laplacian could be diagonalized as**

$$L = U \Lambda U^T$$

where $U = [u_1, \cdots, u_n]$, and $\Lambda = \text{diag}([\lambda_1, \cdots, \lambda_n])$

- **Graph Fourier transform**
  - **Graph Fourier transform of a signal $x \in R^n$ is defined as**

$$\hat{x} = U^T x$$

  - **Graph Fourier inverse transform is**

$$x = U\hat{x}$$

Define convolution in spectral domain

- **Convolution theorem**
  - **The Fourier transform of a convolution of two signals is the point-wise product of their Fourier transforms**

- **According to convolution theorem, given a signal x as input and the other signal y as filter, graph convolution $*_G$ could be written as**

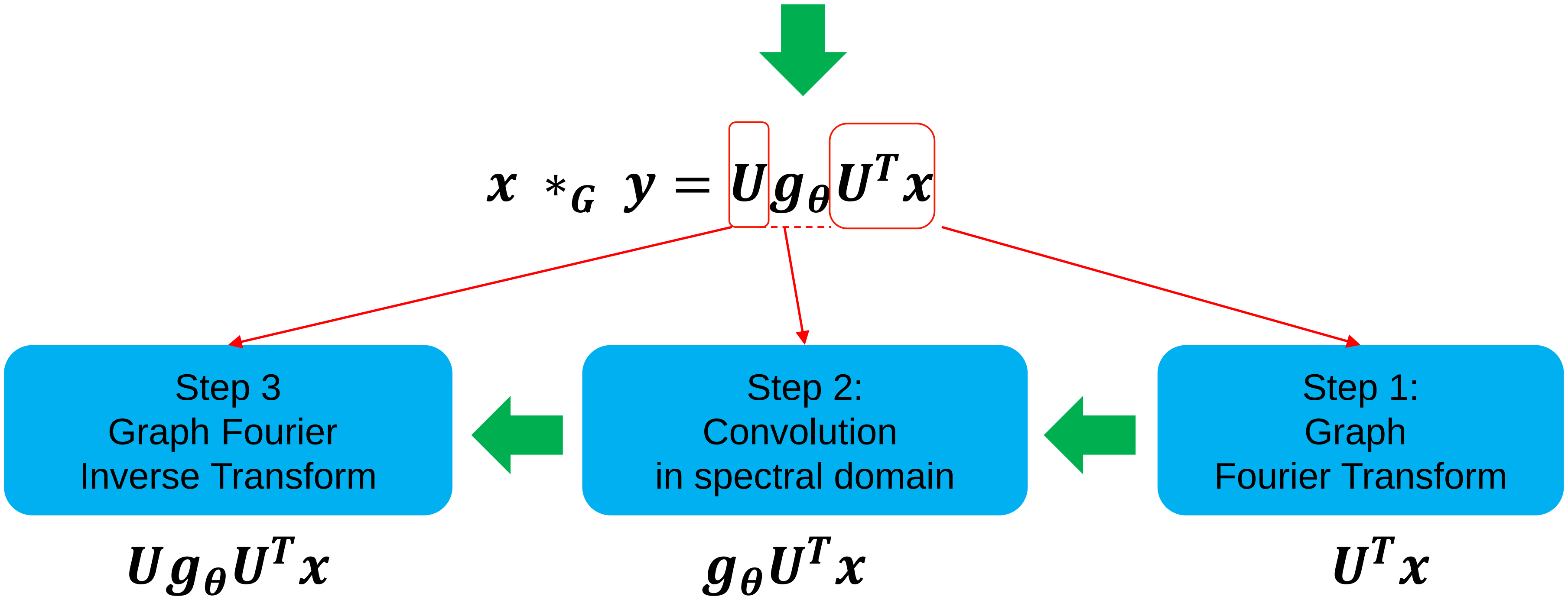
$$x *_G y = U\left(\left(\boldsymbol{U}^{T}\boldsymbol{x}\right)\odot\left(\boldsymbol{U}^{T}y\right)\right)$$

Here, the convolution filter in spectral domain is $\boldsymbol{U}^{T}y$.

Define convolution in spectral domain

- **Graph convolution in spectral domain**
  - **Let $U^T y = [\theta_0, \cdots, \theta_{n-1}]^T$ and $g_\theta = \mathrm{diag}([\theta_0, \cdots, \theta_{n-1}])$, we have**

$$\boldsymbol{x} *_G \boldsymbol{y} = \boldsymbol{U}\left((\boldsymbol{U}^T\boldsymbol{x}) \odot (\boldsymbol{U}^T\boldsymbol{y})\right)$$



$\boldsymbol{x} *_G \boldsymbol{y} = \boxed{\boldsymbol{U}}\boldsymbol{g_\theta}\boxed{\boldsymbol{U}^T\boldsymbol{x}}$
Step 3
Graph Fourier
Inverse Transform
Step 2:
Convolution
in spectral domain
Step 1:
Graph
Fourier Transform
$\boldsymbol{U}\boldsymbol{g_\theta}\boldsymbol{U}^T\boldsymbol{x}$
$\boldsymbol{g_\theta}\boldsymbol{U}^T\boldsymbol{x}$
$\boldsymbol{U}^T\boldsymbol{x}$

Spectral Graph CNN

- **Spectral Graph CNN**

$$x_{k+1,j} = h\left(\sum_{i=1}^{f_k} U \boxed{F_{k,i,j}} U^T \boxed{x_{k,i}}\right)$$

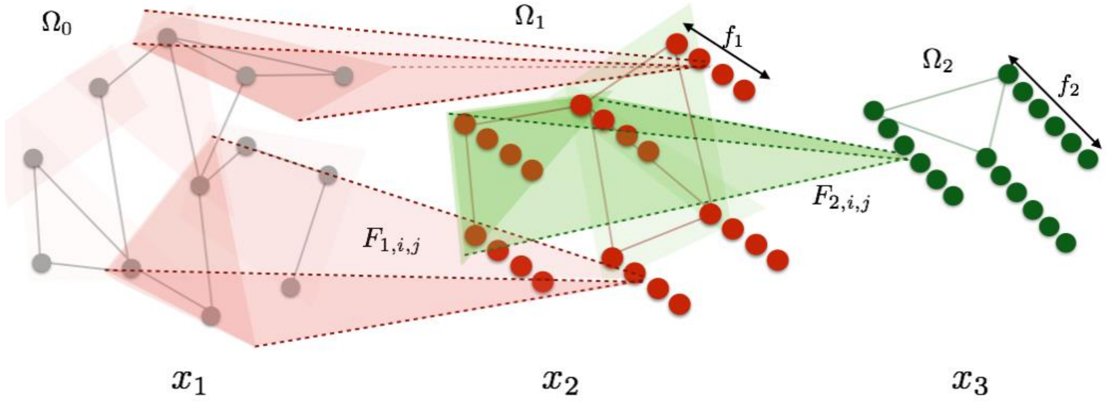
$j = 1, \cdots, f_{k+1}$

Signals in the k-th layer

Filter in the k-th layer

Graph Fourier Transform
$\hat{x} = U^T x$

Graph Fourier Inverse Transform
$x = U\hat{x}$



Shortcomings of Spectral graph CNN

- **Requiring eigen-decomposition of Laplacian matrix**
  - **Eigenvectors are explicitly used in convolution**
- **High Computational cost**
  - **Multiplication with graph Fourier basis $\boldsymbol{U}$ is $\boldsymbol{O}\left(\boldsymbol{n}^2\right)$**
- **Not localized in vertex domain**

ChebyNet: parameterizing filter

- **Parameterizing convolution filter via polynomial approximation**

$$\boldsymbol{g_\theta} = \mathbf{diag}([\boldsymbol{\theta_0}, \cdots, \boldsymbol{\theta_{n-1}}])$$

$$\Downarrow$$

$$g_\beta(\Lambda) = \sum_{k=0}^{K-1} \beta_k \Lambda^k \qquad \Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n)$$

- **ChebyNet**

$$\boldsymbol{x} *_{\boldsymbol{G}} \boldsymbol{y} = \boldsymbol{U} g_\beta(\Lambda) \boldsymbol{U}^{\boldsymbol{T}} \boldsymbol{x} = \sum_{\boldsymbol{k}=\mathbf{0}}^{\boldsymbol{K}-\mathbf{1}} \beta_k L^k \boldsymbol{x}$$

The number of free parameters reduces from n to K

M. Defferrard, X. Bresson, P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. NeuralPS, 2016.

ChebyNet vs. Spectral Graph CNN

- **Eigen-decomposition is not required**

- **Computational cost is reduced from $O(n^2)$ to $O(|E|)$**

$$\boldsymbol{x} *_G \boldsymbol{y} = \boldsymbol{U} g_\beta(\Lambda) \boldsymbol{U}^T \boldsymbol{x} = \sum_{k=0}^{K-1} \beta_k L^k \boldsymbol{x}$$

- **Convolution is localized in vertex domain**
  - **Convolution is strictly localized in a ball of radius K, i.e., K hops from the central vertex**

Is this method good enough? What could we do more?

M. Defferrard, X. Bresson, P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. NeuralPS, 2016.

Our method:
Graph Wavelet Neural Network
(ICLR 2019)

Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, Xueqi Cheng. Graph wavelet neural network. ICLR 2019.

Graph wavelet neural network

- **ChebyNet achieves localized convolutional via restricting the space of graph filters as a polynomial function of eigenvalue matrix $\boldsymbol{\Lambda}$**

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

- **We focus on the Fourier basis to achieve localized graph convolution**

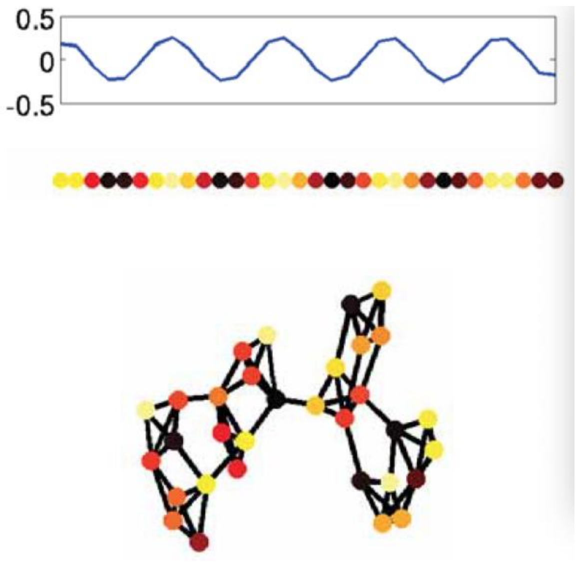
$$\boldsymbol{x} *_G \boldsymbol{y} = \boxed{\boldsymbol{U}} \boldsymbol{g}_\theta \boxed{\boldsymbol{U}^T} \boldsymbol{x}$$

- **We propose to replace Fourier basis with wavelet basis.**

Fourier vs. Wavelet

Fourier Basis

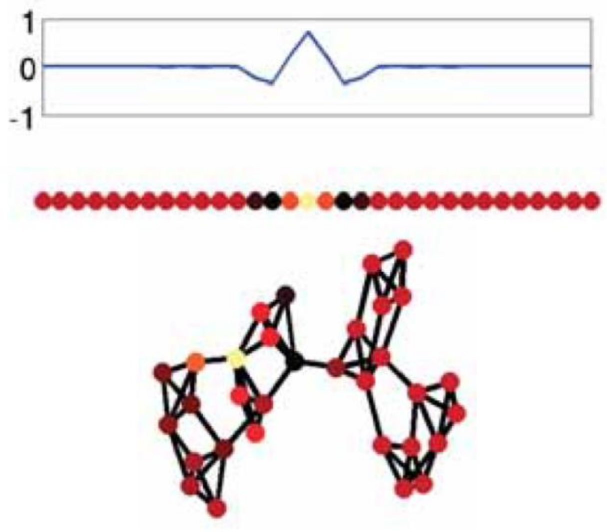
- Dense
- Not localized
- High Computational cost



Fourier basis : U

Wavelet Basis

- Sparse
- Localized
- Low Computational cost



Wavelet basis : $\psi_s = Ue^{\lambda s}U^T$

Graph wavelet neural network

- **Graph Wavelet Neural Network**
  - **Replace graph Fourier transform with graph wavelet transform**

Graph Fourier transform

$$\hat{x} = U^T x$$

Inverse Fourier transform

$$x = U\hat{x}$$

Graph Wavelet transform

$$x^* = {\psi_s}^{-1} x$$

Inverse Wavelet transform

$$x = \psi_s x^*$$

Graph wavelet neural network (GWNN)

- **Graph convolution via wavelet transform**

$$\boldsymbol{x} *_{\mathcal{G}} \boldsymbol{y} = \boldsymbol{U}\left((\boldsymbol{U}^\top \boldsymbol{y}) \odot (\boldsymbol{U}^\top \boldsymbol{x})\right),$$

Replacing basis

$$\boldsymbol{x} *_{\mathcal{G}} \boldsymbol{y} = \psi_s((\psi_s^{-1} \boldsymbol{y}) \odot (\psi_s^{-1} \boldsymbol{x}))$$

- **Graph wavelet neural network**

$$x_{k+1,j} = h\left(\sum_{i=1}^{p} U F_{k,i,j} U^T x_{k,i}\right) \Rightarrow x_{k+1,j} = h\left(\sum_{i=1}^{p} \psi_s F_{k,i,j} \psi_s^{-1} x_{k,i}\right)$$

$j = 1, \cdots, q$

Parameter complexity : $O(n * p * q)$

Reducing parameter complexity

- **Key idea:**
  - **Detaching graph convolution from feature transformation**

$$x_{k+1,j} = h\left(\sum_{i=1}^{p} \psi_s F_{k,i,j} \psi_s^{-1} x_{k,i}\right) \qquad j = 1, \cdots, q$$

$\boldsymbol{T} \in \boldsymbol{R}^{q\times p}$
with $\boldsymbol{p} * \boldsymbol{q}$ parameters

F_k is a diagonal matrix
with $\boldsymbol{n}$ parameters

$$\underbrace{y_{k,j} = \sum_{i=1}^{p} T_{ji} x_{k,i} \quad + \quad x_{k+1,j} = h\left(\psi_s F_k \psi_s^{-1} y_{k,j}\right)}$$

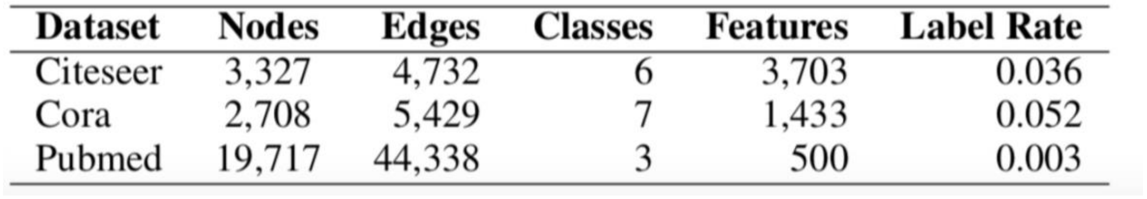
Feature transformation

Graph convolution

The number of parameters reduces from $\boldsymbol{O}(\boldsymbol{n} * \boldsymbol{p} * \boldsymbol{q})$ to $\boldsymbol{O}(\boldsymbol{n} + \boldsymbol{p} * \boldsymbol{q})$

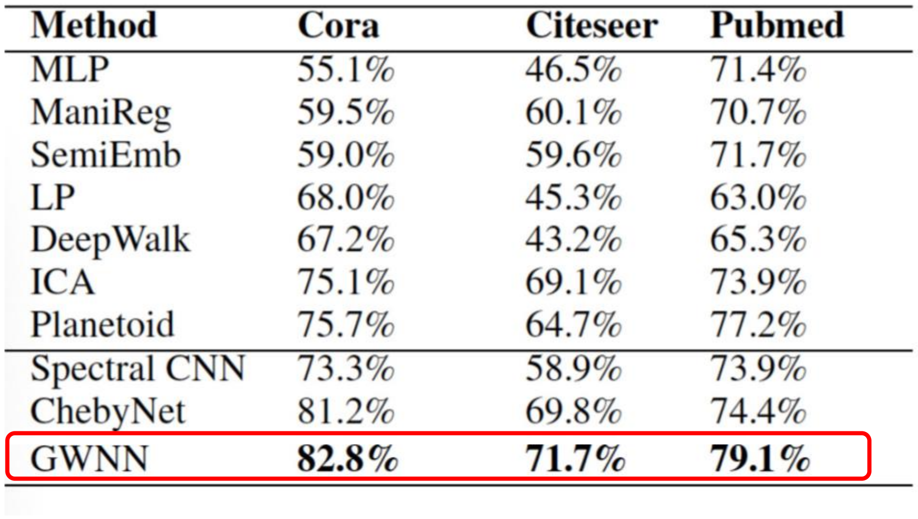
GWNN vs. ChebyNet

- **Benchmark datasets**



Dataset	Nodes	Edges	Classes	Features	Label Rate
Citeseer	3,327	4,732	6	3,703	0.036
Cora	2,708	5,429	7	1,433	0.052
Pubmed	19,717	44,338	3	500	0.003

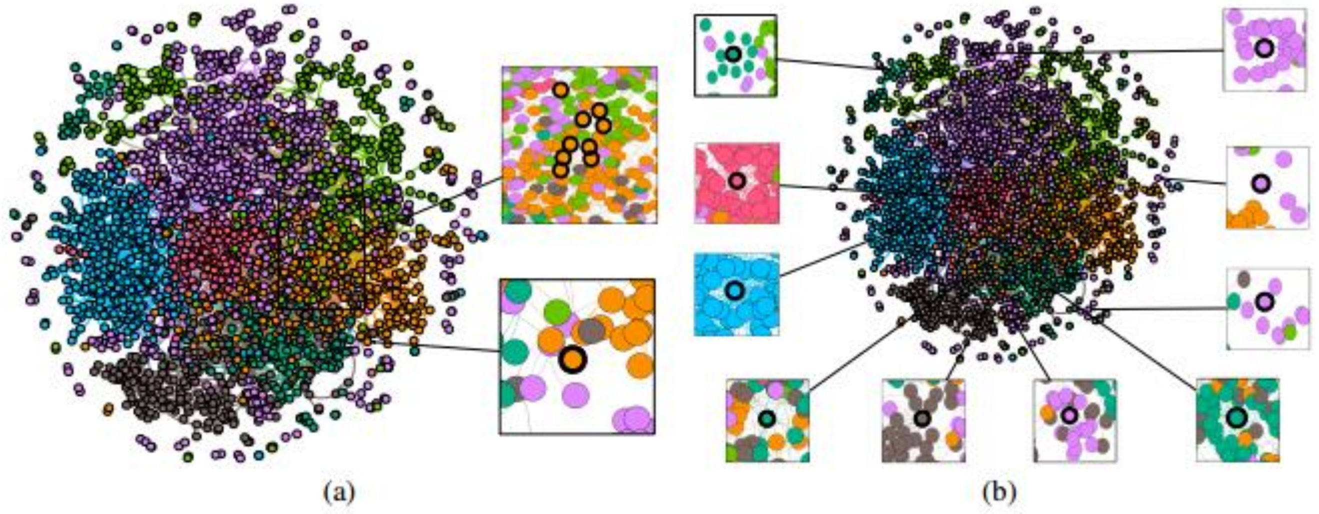
- **Results at the task of node classification**



Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg	59.5%	60.1%	70.7%
SemiEmb	59.0%	59.6%	71.7%
LP	68.0%	45.3%	63.0%
DeepWalk	67.2%	43.2%	65.3%
ICA	75.1%	69.1%	73.9%
Planetoid	75.7%	64.7%	77.2%
Spectral CNN	73.3%	58.9%	73.9%
ChebyNet	81.2%	69.8%	74.4%
GWNN	**82.8%**	**71.7%**	**79.1%**

Graph wavelet neural network

- **Each Graph wavelet offers us a local view, i.e., from a center node, about the proximity for each pair of nodes**

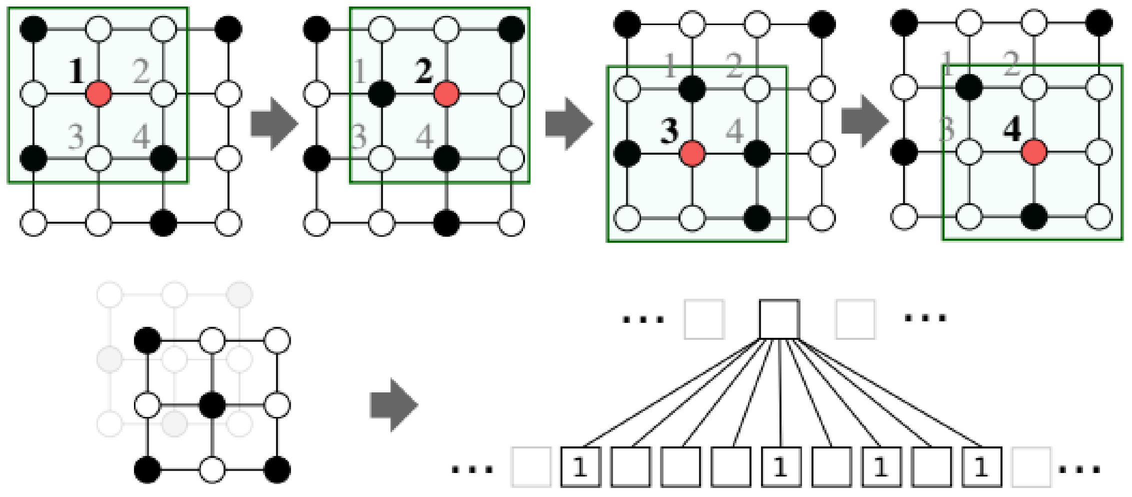


Wavelet offers us a better basis for defining graph convolutional networks in spectral domain

Spatial methods for graph convolutional neural networks

Spatial Methods for Graph CNN

- By analogy
  - What can we learn from the architecture of standard convolutional neural network?



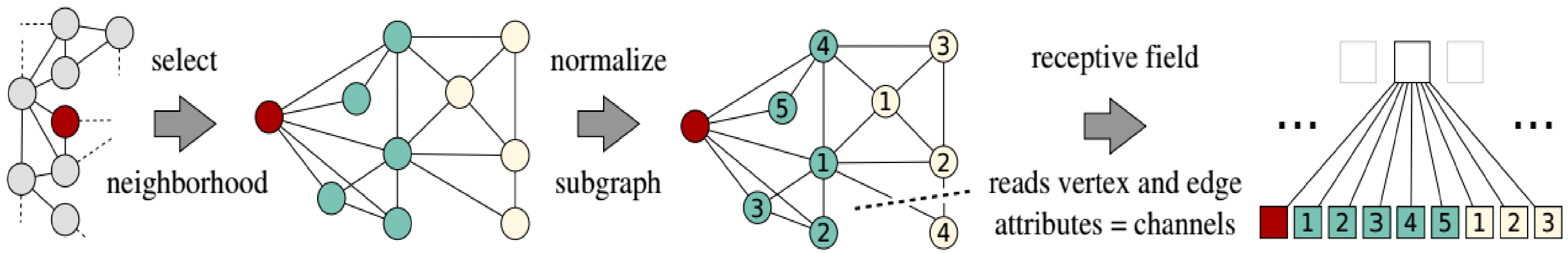
1. Determine Neighborhood

2. Impose an order in neighborhood

3. Parameter sharing

Spatial Methods for Graph CNN

- By analogy
  - For each node, select the fixed number of nodes as its neighboring nodes, according to certain proximity metric
  - Impose an order according to the proximity metric
  - Parameter sharing



1. Determine Neighborhood

2. Impose an order in neighborhood

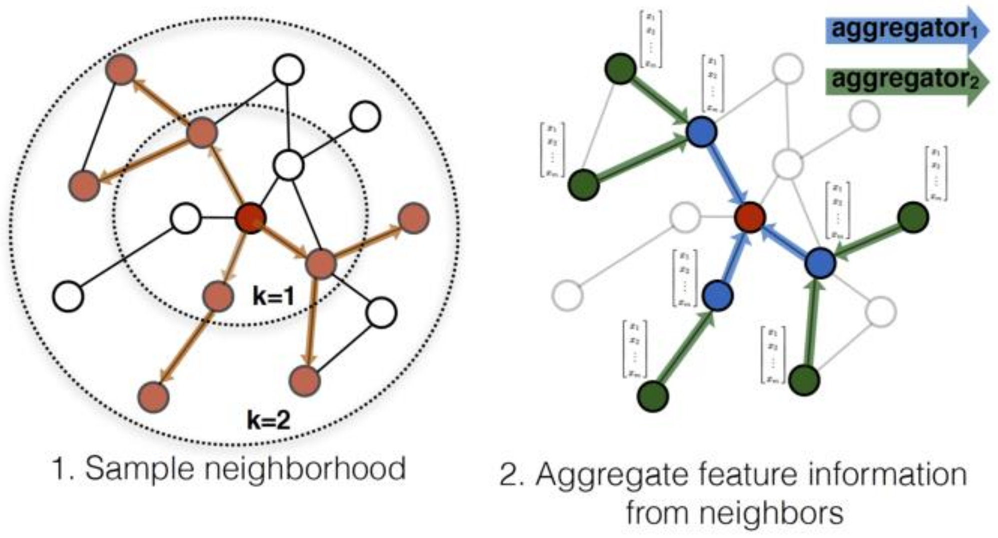
3. Parameter sharing

Spatial Methods for Graph CNN

- **GraphSAGE**
  - **Sampling neighbors**
  - **Aggregating neighbors**

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$



GraphSAGE: Inductive Learning

General framework of graph neural networks: Aggregate the information of neighboring nodes to update the representation of center node

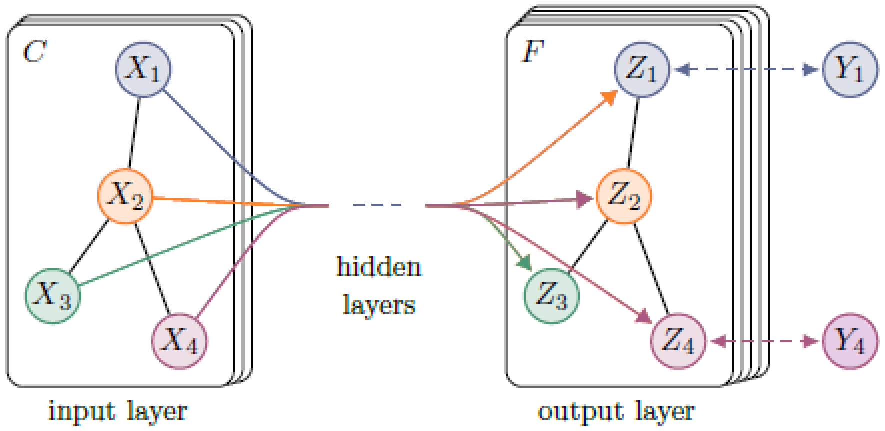
W. L. Hamilton, R. Ying, J. Leskovec. Inductive Representation Learning on Large Graphs. NeuralPS 2017

Spatial Methods for Graph CNN

- **GCN: Graph Convolution Network**
  - **Aggregating information from neighborhood via a normalized Laplacian matrix**
  - **Shared parameters are from feature transformation**
  - **A reduced version of ChebyNet**

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}X\boxed{W^{(0)}}\right)\boxed{W^{(1)}}\right)$$

Parameter for feature Transformation



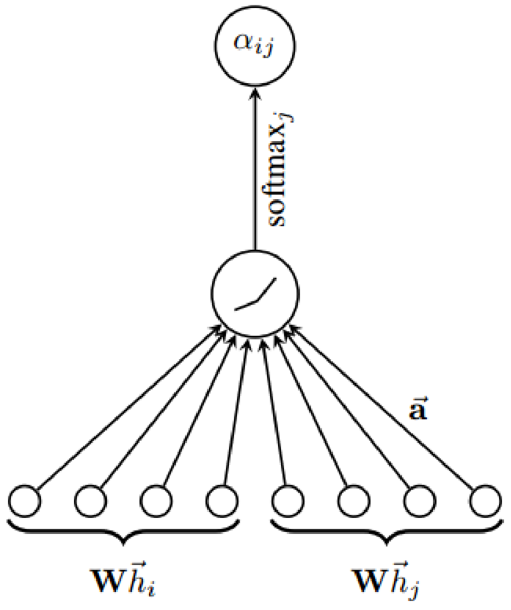
Spatial Methods for Graph CNN

- **GAT: Graph Attention Network**
  - **Learning the aggregation matrix, i.e., Laplacian matrix in GCN, via attention mechanism**
  - **Shared parameters contain two parts**
    - **Parameters for feature transformation**
    - **Parameters for attention**

Parameter for feature Transformation

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\boxed{\mathbf{W}}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\boxed{\vec{\mathbf{a}}^T}[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

Parameter of Attention mechanism



Attention Mechanism in GAT

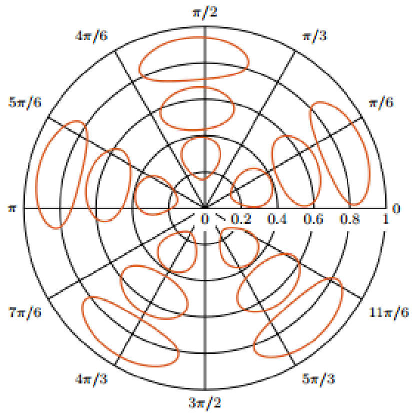
P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio. Graph Attention Networks. NeuralPS 2018.

Spatial Methods for Graph CNN

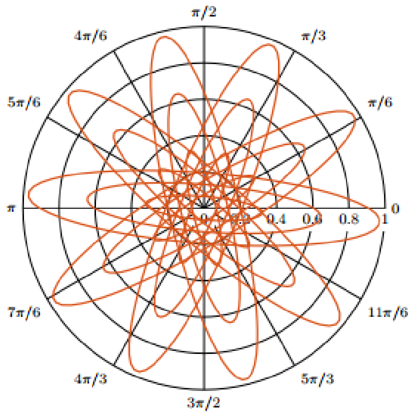
- **MoNet: A general framework for spatial methods**
  - **Define multiple kernel functions, parameterized or not, to measure the similarity between target node and other nodes**
  - **Convolution kernels are the weights of these kernel functions**

Convolution kernel

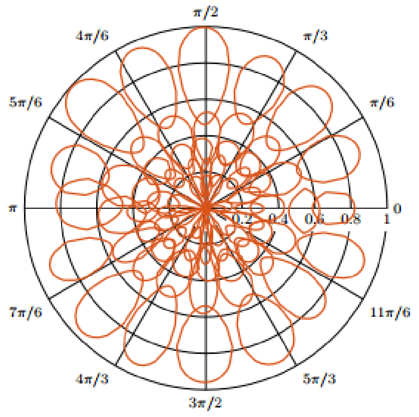
$$(f \star g)(x) = \sum_{j=1}^{J} \boxed{g_j} D_j(x) f$$



$\exp(-\frac{1}{2}(\mathbf{u}-\bar{\mathbf{u}}_j)^\top \begin{pmatrix} \bar{\sigma}_\rho^2 & \\ & \bar{\sigma}_\theta^2 \end{pmatrix}^{-1} (\mathbf{u}-\bar{\mathbf{u}}_j))$



$\exp(-\frac{1}{2}\mathbf{u}^\top \mathbf{R}_{\bar{\theta}_j} \begin{pmatrix} \bar{\alpha} & \\ & 1 \end{pmatrix} \mathbf{R}_{\bar{\theta}_j}^\top \mathbf{u})$



$\exp(-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1}(\mathbf{u}-\boldsymbol{\mu}_j))$

F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. CVPR 2017.

Our method:

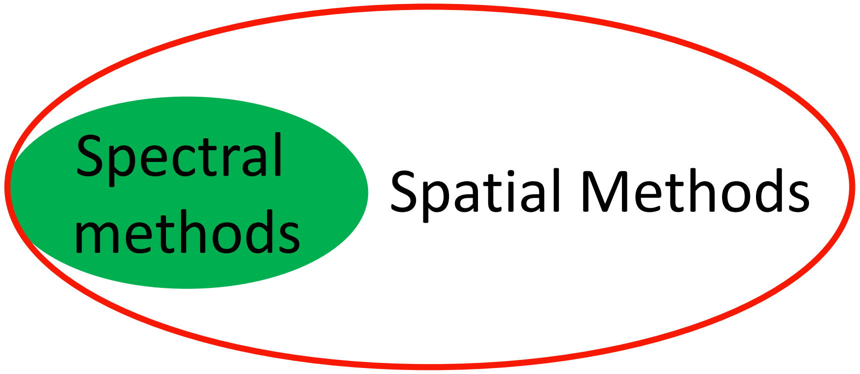
Graph Convolutional Networks using Heat Kernel for Semi-supervised Learning

（IJCAI 2019）

Bingbing Xu, Huawei Shen, Qi Cao, Keting Cen, Xueqi Cheng. Graph Convolutional Networks using Heat Kernel for Semi-supervised Learning, IJCAI 2019.

Spectral methods vs. Spatial methods

- **Connections**
  - **Spectral methods are special cases of spatial methods**



$$(f \star g)(x) = \sum_{j=1}^{J} g_j \boxed{D_j(x)} f$$

Kernel function: Characterizing the similarity or distance among nodes

- **Difference**
  - **Spectral methods define kernel functions via an explicit space transformation, i.e., projecting into spectral space**
  - **Spatial methods directly define kernel functions**

Spectral methods: Recap

- **Spectral CNN**

$$y = Ug_\theta U^T x = (\theta_1 \boxed{u_1 u_1^T} + \theta_2 \boxed{u_2 u_2^T} + \cdots + \theta_n \boxed{u_n u_n^T})x$$

- **ChebyNet**

$$y = (\theta_0 I + \theta_1 L + \theta_2 L^2 + \cdots + \theta_{K-1} L^{K-1})x$$

- **GCN**

$$y = \theta(I - L)x$$

Question:

Why GCN with less parameters performs better than ChebyNet?

Graph Signal Processing: filter

- **Smoothness of a signal x over graph is measured by**

$$x^T L x = \sum_{(u,v)\in E} A_{uv}\left(\frac{x_u}{\sqrt{d_u}} - \frac{x_v}{\sqrt{d_v}}\right)^2$$

$\lambda_i = u_i^T L u_i$ can be viewed as the frequency of u_i

- **Basic filters**
  - $u_i u_i^T (1 \leq i \leq n)$ **are a set of basic filters**
  - **For a graph signal x, the basic filter $u_i u_i^T$ only allows the component with frequency λ_i passes**

$$x = \alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_n u_n,$$
$$u_i u_i^T x = \alpha_i u_i$$

Combined filters: High-pass vs. Low-pass

- **Combined filters**
  - **A linear combination of basic filters**

$$\theta_1 u_1 u_1^T + \theta_2 u_2 u_2^T + \cdots + \theta_n u_n u_n^T$$

  - $\boldsymbol{L^k}$ **is a combined filter with the coefficients** $\{\lambda_i^k\}_{i=1}^n$
  - $\boldsymbol{L^k}$ **assign high coefficients to basic filters with high-frequency, i.e.,** $\boldsymbol{L^k}$ **is a high-pass filter**

- **GCN only consider $k = 0$ and $k = 1$, avoiding the boosting effect to basic filters with high-frequency**
  - **Behaving as a low-pass combined filter**
  - **Explaining why GCN performs better than ChebyNet**

Our method: GraphHeat

- **Low-pass combined filters**
  - $\{e^{-skL}\}$**, where** s **is scaling parameter, and k is order**
  - e^{-sL} **is heat kernel over graph, which defines the similarity among nodes via heat diffusion over graph**

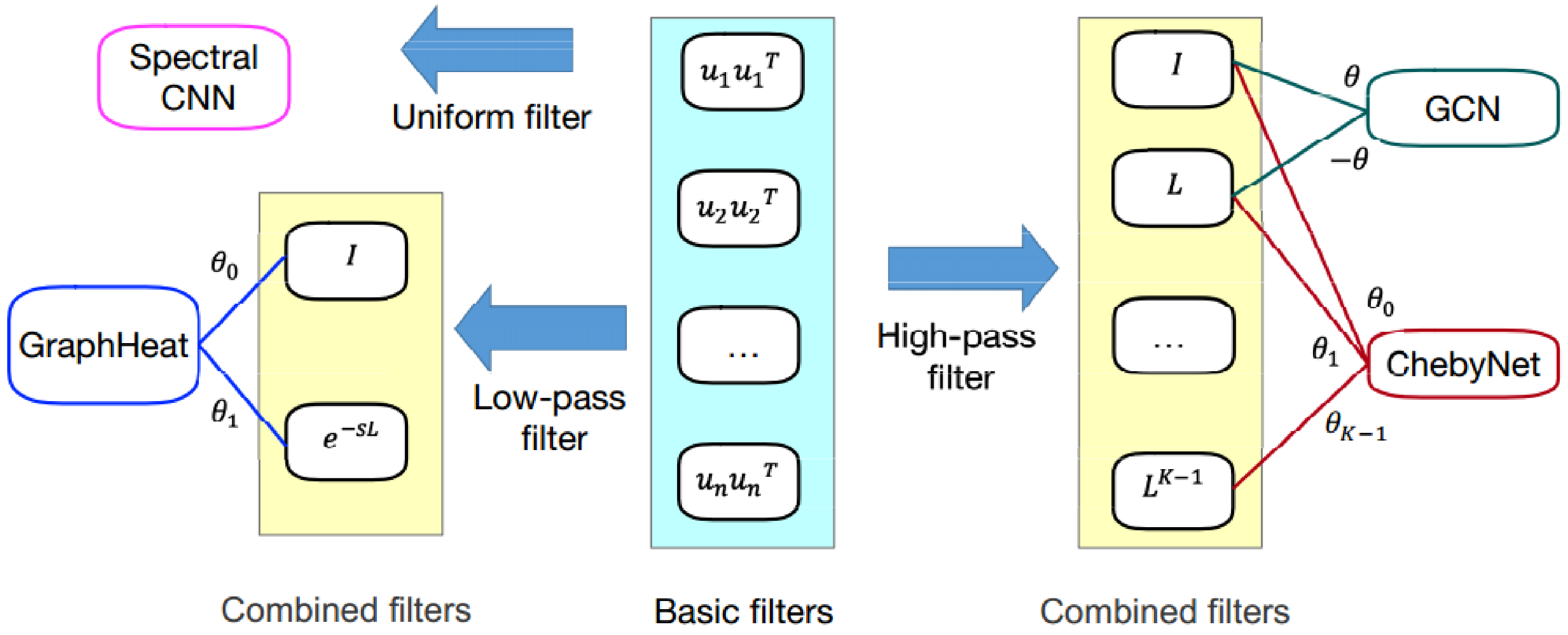
$$e^{-sL} = Ue^{-s\Lambda}U^T, \Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n)$$

  - **The basic filter** $u_i u_i^T (1 \leq i \leq n)$ **has the coefficient** $e^{-s\lambda_i}$**, suppressing signals with high-frequency**

B. Xu, H. Shen, Q. Cao, K. Cen, X. Cheng. Graph Convolutional Networks using Heat Kernel for Semi-supervised Learning, IJCAI 2019.

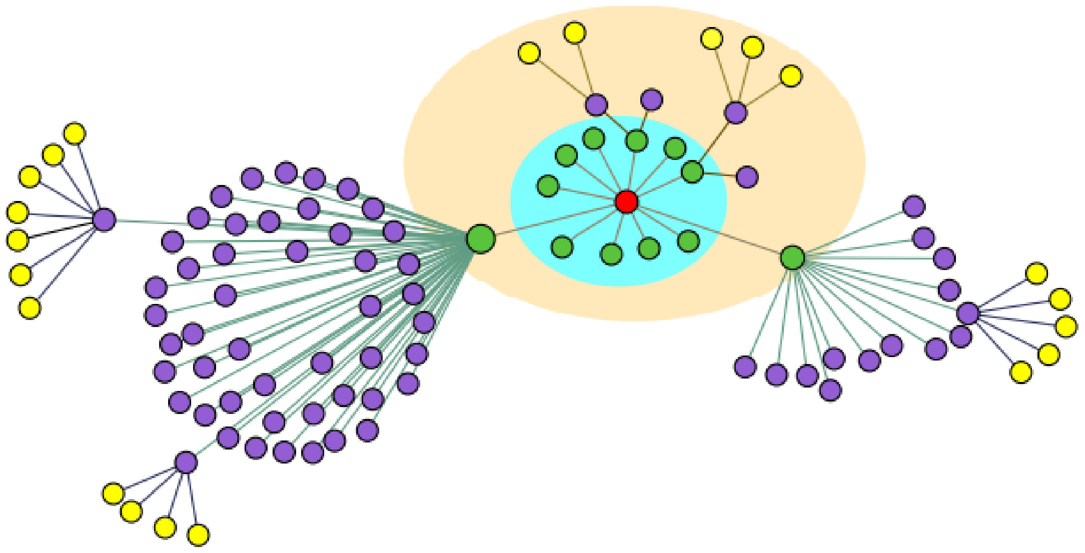
GraphHeat vs. baseline methods

- **Compared with baseline methods**



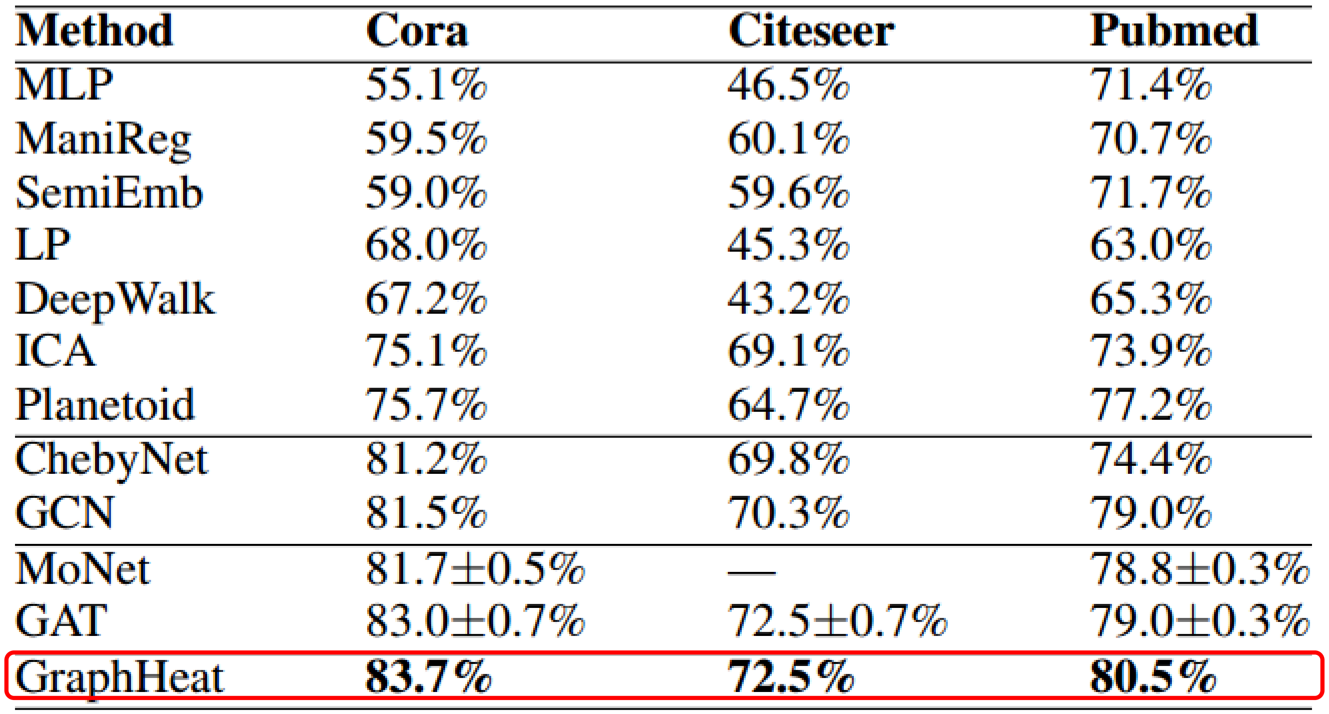
GraphHeat vs. baseline methods

- Neighborhood
  - GCN and ChebyNet determine neighborhood according to the hops away from center node, i.e., in an order-style
    - Nodes in different colors
  - GraphHeat determines neighborhood according to the similarity function by heat diffusion over graph
    - Nodes in different circles



Experimental results

- **Results at the task of node classification**



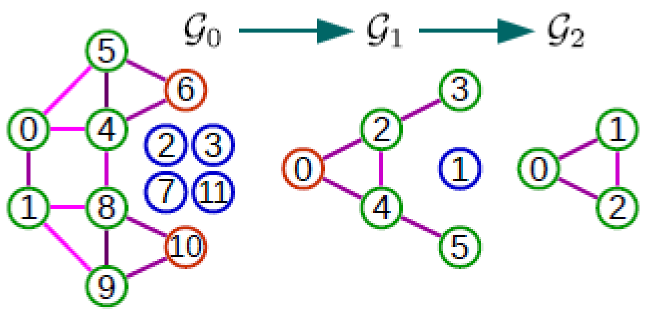
Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg	59.5%	60.1%	70.7%
SemiEmb	59.0%	59.6%	71.7%
LP	68.0%	45.3%	63.0%
DeepWalk	67.2%	43.2%	65.3%
ICA	75.1%	69.1%	73.9%
Planetoid	75.7%	64.7%	77.2%
ChebyNet	81.2%	69.8%	74.4%
GCN	81.5%	70.3%	79.0%
MoNet	81.7±0.5%	—	78.8±0.3%
GAT	83.0±0.7%	72.5±0.7%	79.0±0.3%
GraphHeat	**83.7%**	**72.5%**	**80.5%**

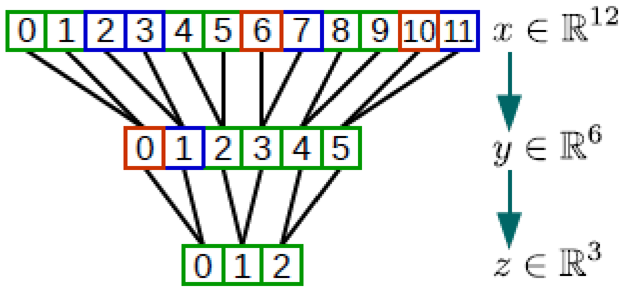
GraphHeat achieves state-of-the-art performance on the task of node classification on the three benchmark datasets

Graph Pooling

Graph Pooling via graph coarsening

- **Graph coarsening**
  - **Merging nodes into clusters and take each cluster as a super node**



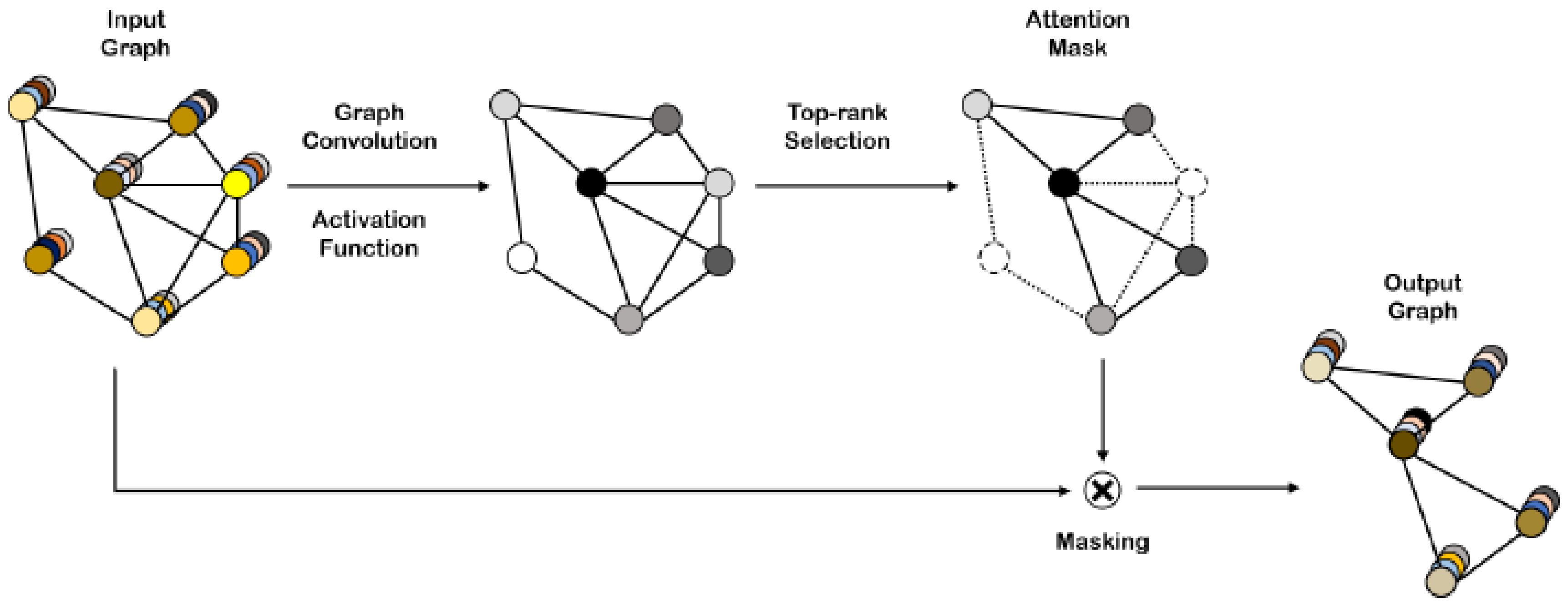


  - **Node merging could be done a priori or during the training process of graph convolutional neural networks, e.g, DiffPooling**

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling, NeuralPS 2018

Graph pooling via node selection

- **Node selection**
  - **Learn a metric to quantify the importance of nodes and select several nodes according to the learned metric**

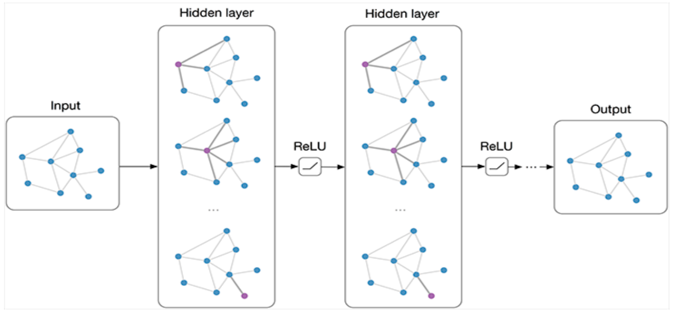


J. Lee, I. Lee, J. Kang. Self-attention Graph Pooling, ICML 2019.

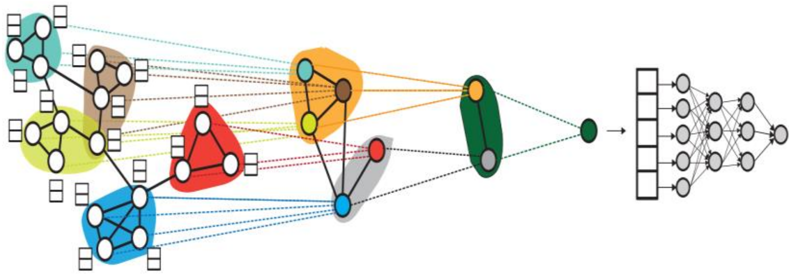
Expressive Power of Graph Neural Networks

Graph Neural Networks

- **Graph neural networks (GNNs) gained remarkable success**
  - **Achieving state-of-the-art empirical performance in node classification, link prediction, and graph classification.**



Node classification

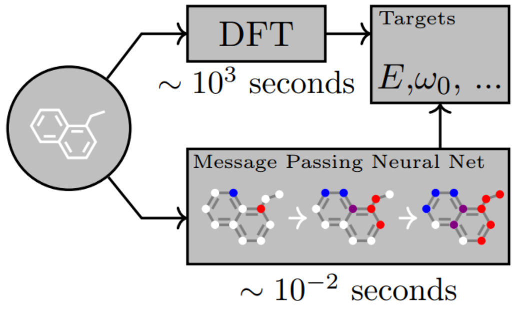


Graph classification

- **The design of new GNNs is mostly based on**
  - **empirical intuition, heuristics and experimental trial-and-error.**
- **We lack theoretical understanding of the properties and limitations of GNNs.**
  - **One fundamental problem is the expressive power of GNNs**

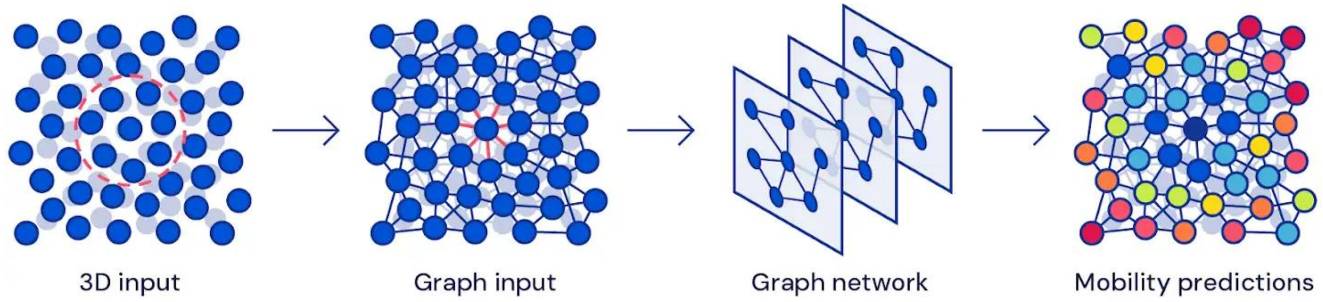
Power of GNNs: an empirical view

- **GNNs for quantum chemistry**
  - **Predict the quantum properties of molecules using GNNs**
  - **Traditional methods, i.e., DFT (Density Functional Theory), is computationally expensive**



- ✓ Predict DFT to within **chemical accuracy** on 11 out of 13 targets with molecule topology and spatial information as input
- ✓ Predict DFT to within chemical accuracy on 5 out of 13 targets while operating on the topology of the molecule alone

- **Modeling dynamics of glassy systems using GNNs**



About Expressive Power

- **Number of possible distinct occurrences**
  - Expressive power of n bits is 2^n
    - For example, 5 bits has the ability to distinguish 32 distinct states
    - Given no more than 32 bottles of water and one of them is poisonous, at most 5 mice are required to identify which bottle is poisonous.

- **Approximation capability**
  - Expressive power of 1-layer perceptron
    - 1-layer perceptron is not a universal approximator
    - For example, XOR cannot be approximated by 1-layer perceptron.
  - Multi-layer perceptron offers us a universal approximator
    - Universal approximation theorem
    - Expressive power scales exponentially with the number of layers

K. Hornik, M. Stinchcombe, H. White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.
M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, J. S. Dickstein. On the Expressive Power of Deep Neural Networks. ICML, 2017.

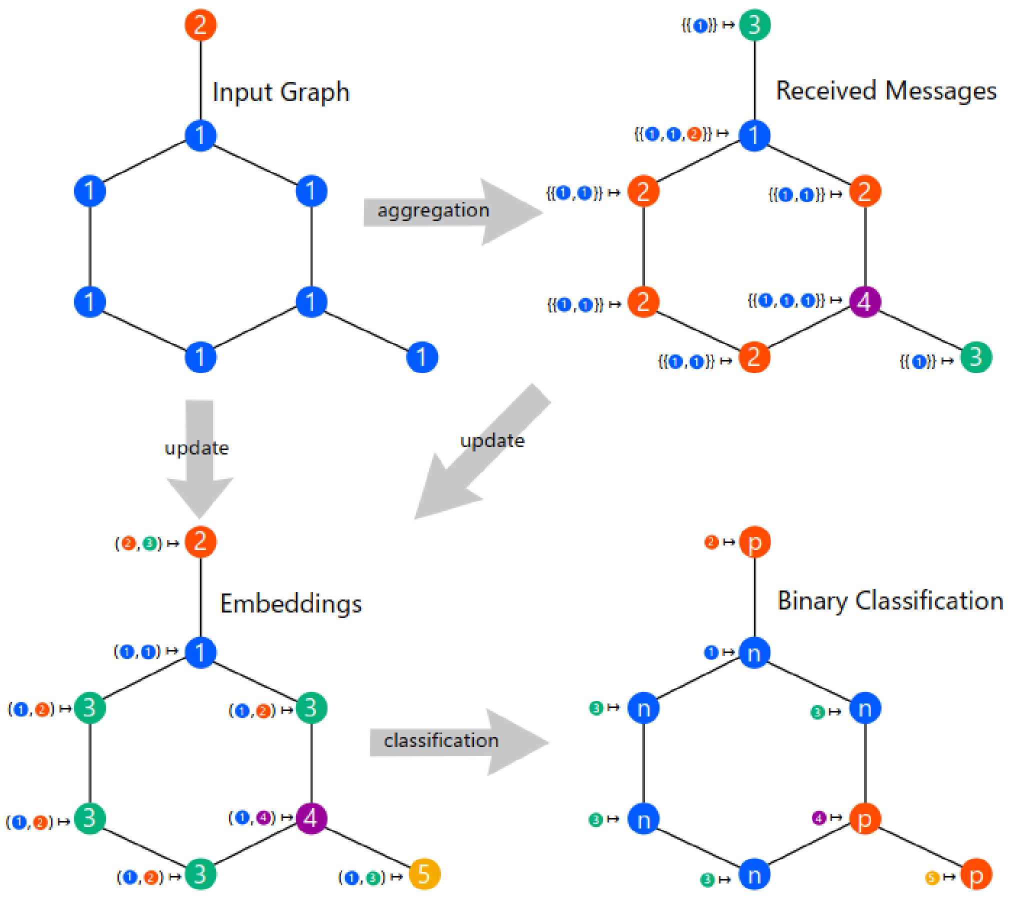
Two typical tasks for GNNs

- **Denote a graph with $G = (V, E, W, X)$**
  - V is the node set with $n = |V|$, E is the edge set, and $W \in R^{n\times n}$ is the weighted adjacency matrix
  - Each node is associated with d features, and $X \in R^{n\times d}$ is the feature matrix of nodes

- **Two typical tasks for graph**
  - **Node classification**: Each node $v \in V$ has a label y_v, and the goal is to learn a representation vector h_v such that v's label can be predicted as $y_v = f(h_v)$
  - **Graph classification**: Given a set of graphs $\{G_1, \cdots, G_N\}$ with labels $\{y_1, \cdots, y_N\}$, the goal is to learn a representation vector h_G for each graph G such that we can predict the label of graph as $y_G = g(h_G)$

GNNs for node classification

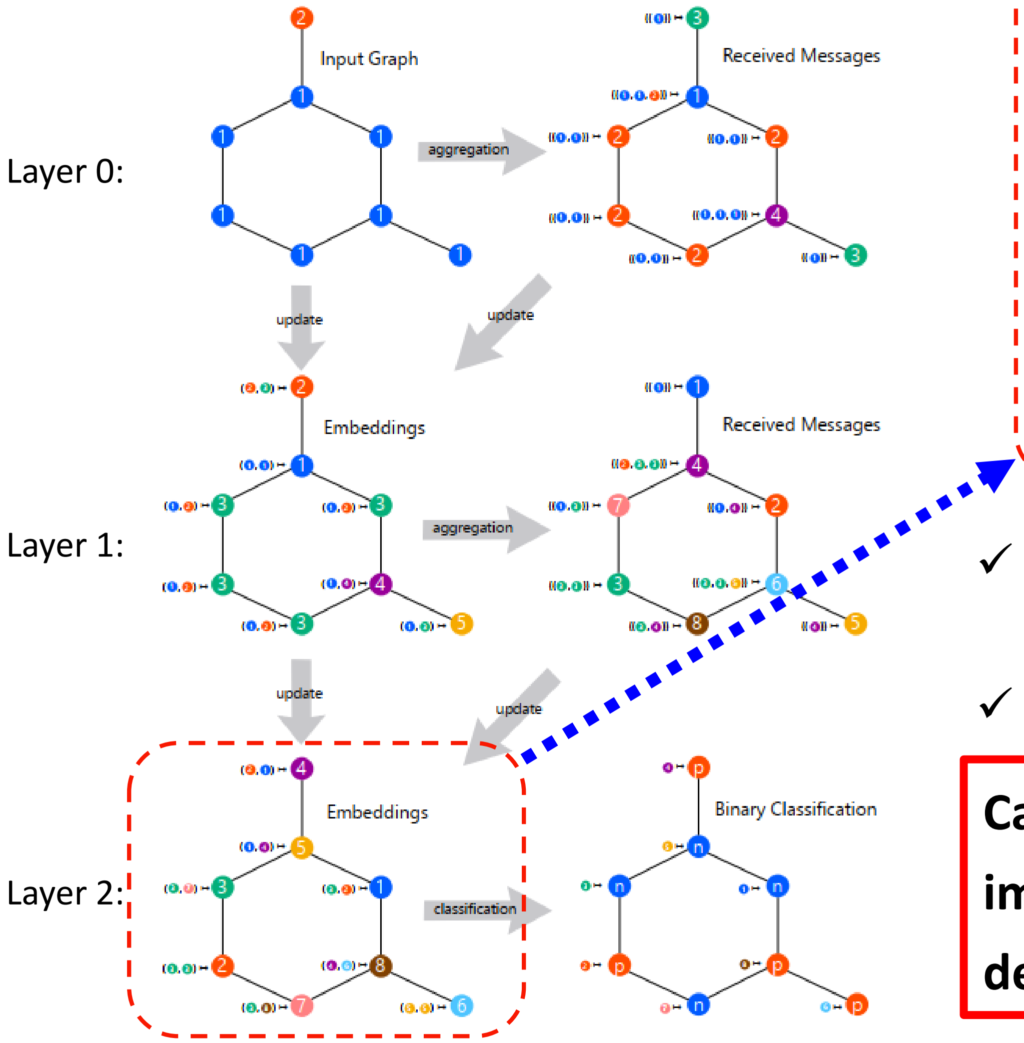
- **Example: 1-layer GCN**

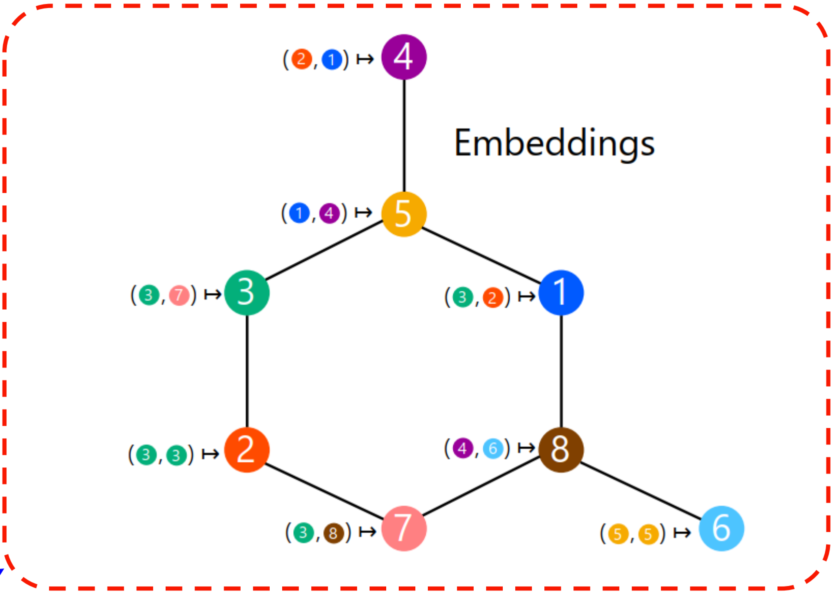


Limited expressive power of 1-layer GCN: it cannot fully distinguish all nodes

GNNs for node classification

- **Example: 2-layer GCN**





- ✓ **2-layer GCN can fully distinguish all nodes in this toy example.**
- ✓ **Depth of GNNs matters.**

Can the expressive power of GNNs be improved infinitely via improving their depth? Is there a bound?

GNNs for graph classification

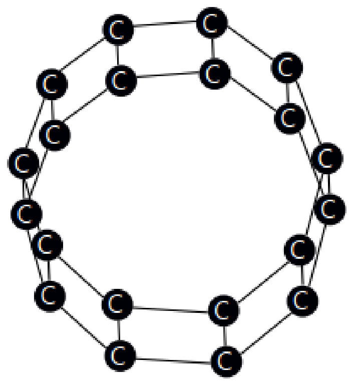
- GNN is a function: $G \rightarrow R^d$
  - Mapping a graph, where node feature is from a countable space, into a real-valued vector with infinite expressivity

- The expressive power of GNNs lies in their capability to distinguish different graphs
  - Different graphs should have different representations in R^d

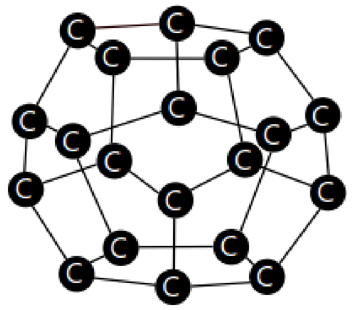
- Two key factors
  - Node Feature
    - Feature transformation with neural networks, e.g., MLP
  - + Graph Structure
    - Graph isomorphism

GNNs for graph classification

- Examples to show the limited expressive power of GNNs



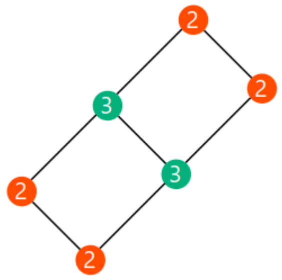
(a) Decaprismane.

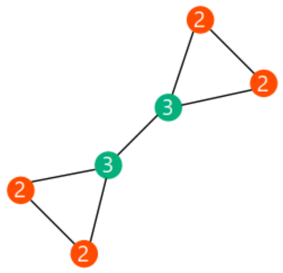


(b) Dodecahedrane.

✓ Regular graphs

✓ Identical node labels





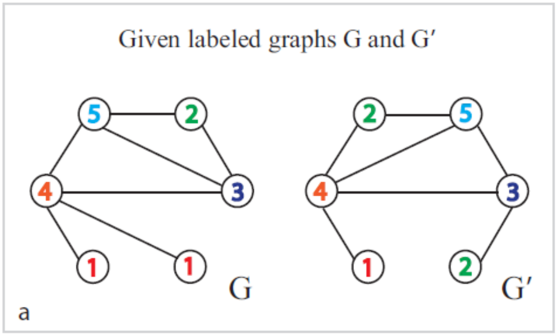
✓ Regular graphs

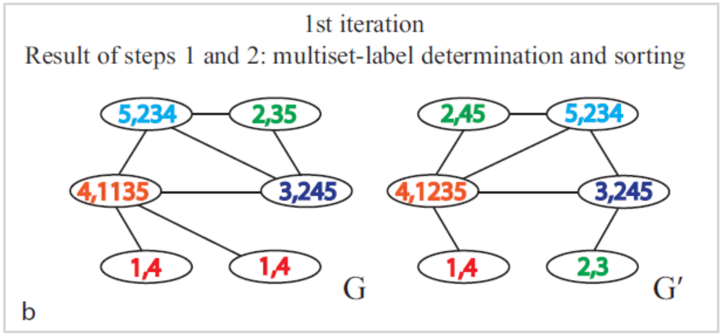
✓ Node labels are not identical

Weisfeiler-Lehman Isomorphism Test

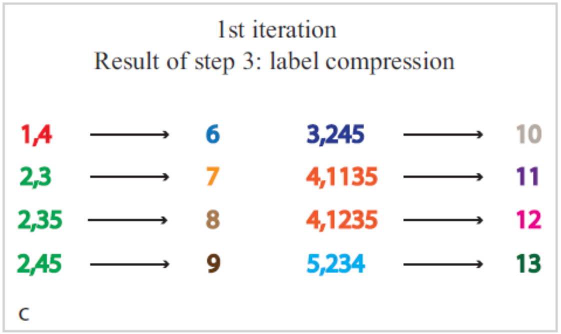
- **WL test is widely used to judge whether two graphs, labeled or unlabeled, are topologically identical, or how they are topologically similar.**

Example: similarity measurement.





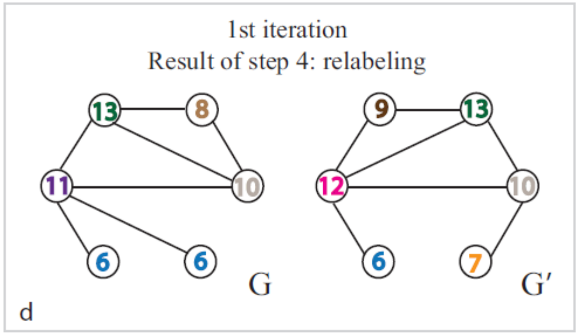
1st iteration
Result of steps 1 and 2: multiset-label determination and sorting
G
G′
b

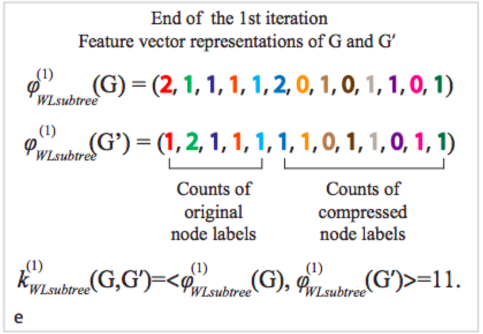

1st iteration

Result of step 3: label compression

1,4 →	6	3,245 →	10
2,3 →	7	4,1135 →	11
2,35 →	8	4,1235 →	12
2,45 →	9	5,234 →	13

c




End of the 1st iteration

Feature vector representations of G and G′

$$\varphi^{(1)}_{WLsubtree}(G) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

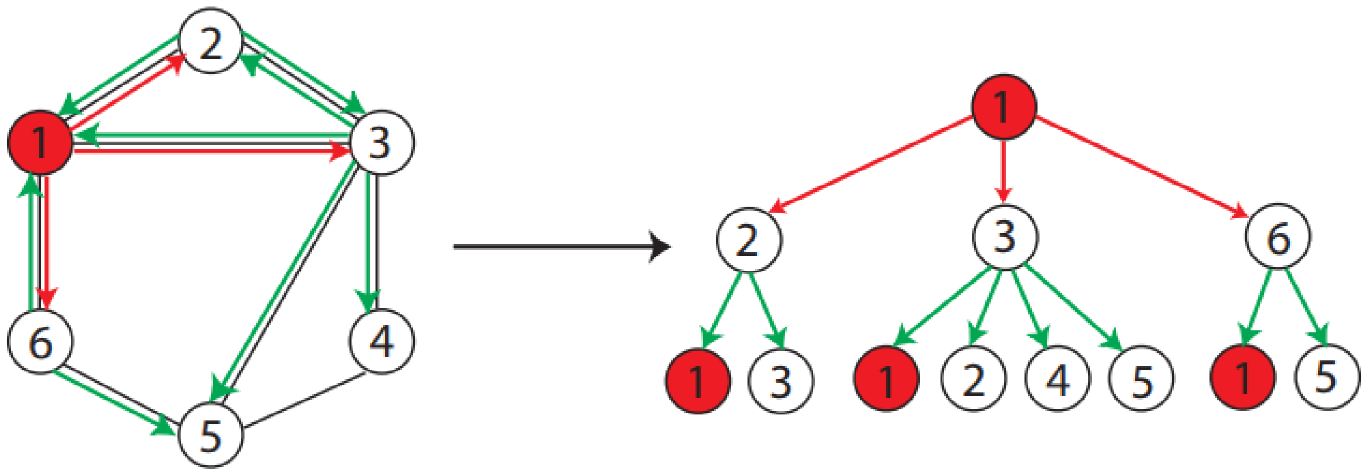
$$\varphi^{(1)}_{WLsubtree}(G') = (\underbrace{1, 2, 1, 1, 1}_{\text{Counts of original node labels}}, \underbrace{1, 1, 0, 1, 1, 0, 1, 1}_{\text{Counts of compressed node labels}})$$

$$k^{(1)}_{WLsubtree}(G,G') = \langle \varphi^{(1)}_{WLsubtree}(G), \varphi^{(1)}_{WLsubtree}(G') \rangle = 11.$$

e

WL Test: Subtree-based Graph Kernel

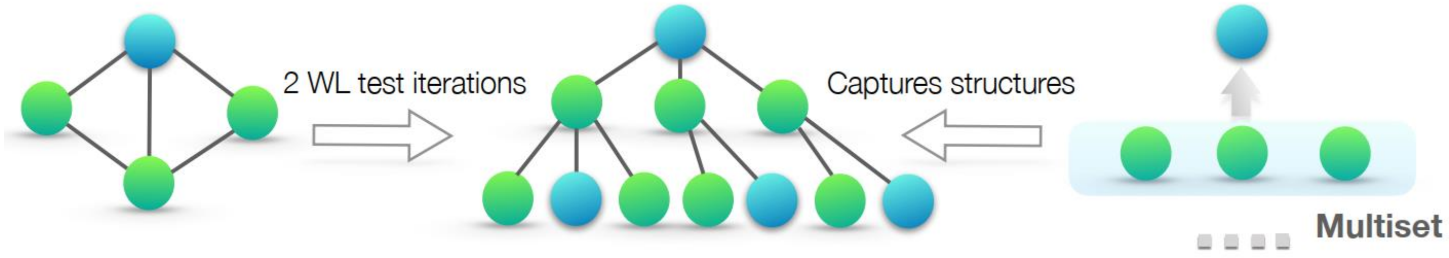
- **A node's label at the k-th iteration of WL test represents a subtree structure of height k rooted at the node**



For WL test, graph features are essentially counts of different rooted subtrees in the graph.

Connection Between GNN and WL Test

- GNNs recursively update each node's feature vector to capture network structure and features of neighboring nodes, i.e., rooted subtree – following the practice of WL test.
- Features of neighboring nodes form a multiset
  - The same element can appear multiple times since different nodes can have identical feature vectors.
  - A multiset is denoted as a 2-tuple $X = (S, m)$, where S is the underlying set with distinct elements, and $m: S \to \mathbb{N}_{\geq 1}$ gives the multiplicity of elements.

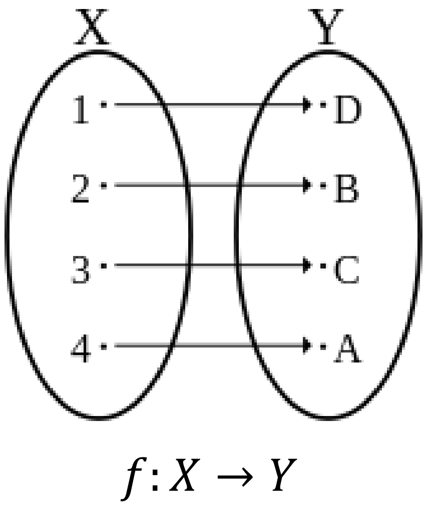


Graph Rooted subtrees GNN aggregation

Connection Between GNN and WL Test

- **WL test provides an upper bound for the expressive power of the aggregation-based GNNs.**
  - **A maximally powerful GNN never map two different neighborhoods to the same representation, i.e., the aggregation function over multiset is injective.**

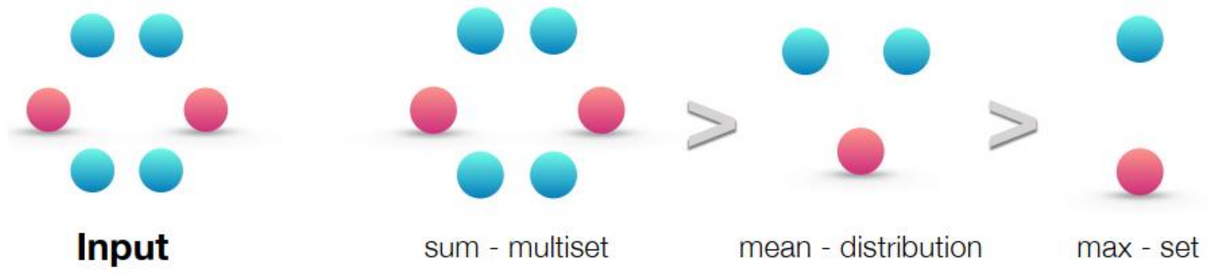
For popular GNNs, like GCN and GraphSAGE, their aggregation functions are inherently not injective.



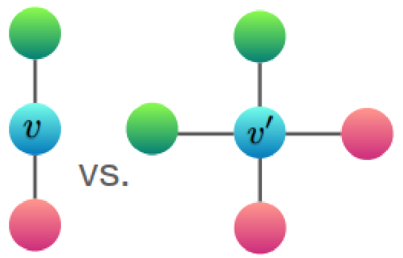
Injective function

Expressivity Limit of GCN and GraphSAGE

- Aggregation function **is not injective**
  - 1-layer perceptron is not sufficient
    - 1-layer perceptron is not a universal approximator
  - Mean and max pooling is not injective
    - Mean pooling learns distributions
    - Max pooling learns sets with distinct elements



Comparing expressive power of sum, mean, max pooling



Max and mean both fail

Graph Isomorphism Network

- **Basic idea: compose a universal injective aggregation function over a node and the multiset of its neighbors**

$$h_v^{(k)} = \phi\left(h_v^{(k-1)}, f\left(\left\{h_u^{(k-1)} : \forall u \in \mathcal{N}(v)\right\}\right)\right)$$

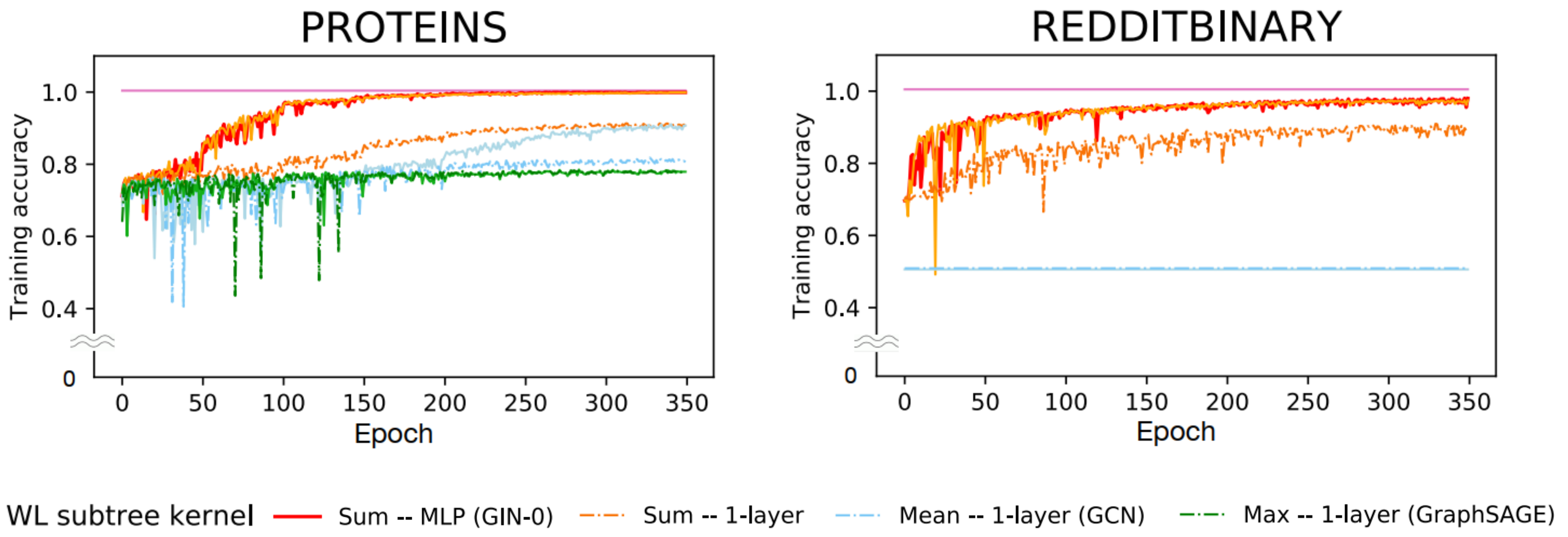
- **Graph isomorphism network**

$$h_v^{(k)} = \underbrace{\boxed{\mathrm{MLP}^{(k)}}}_{①}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \underbrace{\boxed{\sum}}_{②}{}_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$$

① Multiple-layer perceptron offers universal approximator.
② Sum pooling offers injective condition.

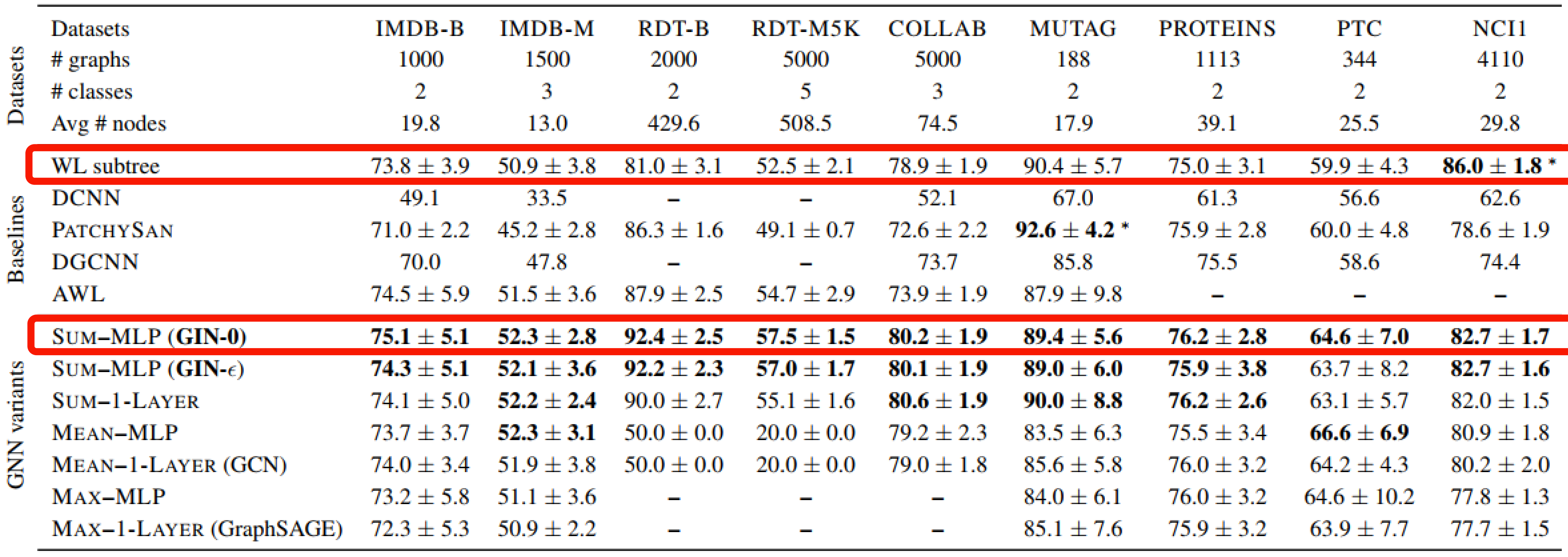
Experimental Validation

- **Validating expressive power or representation capability**
  - **Metric: training accuracy**
  - **Task: graph classification**
  - **Datasets: bioinformatics and social networks**



Performance on Graph Classification

- **Does the high expressive power of GNN imply good performance on down-stream task, e.g., graph classification?**
  - **Metric: test accuracy**



	Datasets	IMDB-B	IMDB-M	RDT-B	RDT-M5K	COLLAB	MUTAG	PROTEINS	PTC	NCI1
Datasets	# graphs	1000	1500	2000	5000	5000	188	1113	344	4110
	# classes	2	3	2	5	3	2	2	2	2
	Avg # nodes	19.8	13.0	429.6	508.5	74.5	17.9	39.1	25.5	29.8
Baselines	WL subtree	73.8 ± 3.9	50.9 ± 3.8	81.0 ± 3.1	52.5 ± 2.1	78.9 ± 1.9	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	**86.0 ± 1.8** *
	DCNN	49.1	33.5	–	–	52.1	67.0	61.3	56.6	62.6
	PATCHYSAN	71.0 ± 2.2	45.2 ± 2.8	86.3 ± 1.6	49.1 ± 0.7	72.6 ± 2.2	**92.6 ± 4.2** *	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
	DGCNN	70.0	47.8	–	–	73.7	85.8	75.5	58.6	74.4
	AWL	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 2.5	54.7 ± 2.9	73.9 ± 1.9	87.9 ± 9.8	–	–	–
GNN variants	SUM–MLP (GIN-0)	**75.1 ± 5.1**	**52.3 ± 2.8**	**92.4 ± 2.5**	**57.5 ± 1.5**	**80.2 ± 1.9**	**89.4 ± 5.6**	**76.2 ± 2.8**	**64.6 ± 7.0**	**82.7 ± 1.7**
	SUM–MLP (GIN-ϵ)	**74.3 ± 5.1**	**52.1 ± 3.6**	**92.2 ± 2.3**	**57.0 ± 1.7**	**80.1 ± 1.9**	**89.0 ± 6.0**	**75.9 ± 3.8**	63.7 ± 8.2	**82.7 ± 1.6**
	SUM–1-LAYER	74.1 ± 5.0	**52.2 ± 2.4**	90.0 ± 2.7	55.1 ± 1.6	**80.6 ± 1.9**	**90.0 ± 8.8**	**76.2 ± 2.6**	63.1 ± 5.7	82.0 ± 1.5
	MEAN–MLP	73.7 ± 3.7	**52.3 ± 3.1**	50.0 ± 0.0	20.0 ± 0.0	79.2 ± 2.3	83.5 ± 6.3	75.5 ± 3.4	**66.6 ± 6.9**	80.9 ± 1.8
	MEAN–1-LAYER (GCN)	74.0 ± 3.4	51.9 ± 3.8	50.0 ± 0.0	20.0 ± 0.0	79.0 ± 1.8	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0
	MAX–MLP	73.2 ± 5.8	51.1 ± 3.6	–	–	–	84.0 ± 6.1	76.0 ± 3.2	64.6 ± 10.2	77.8 ± 1.3
	MAX–1-LAYER (GraphSAGE)	72.3 ± 5.3	50.9 ± 2.2	–	–	–	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5

High expressive power does not always bring good performance
Note that: low expressive power always implies bad performance

Conclusions

- WL test provides an upper bound for the expressive power of the aggregation-based GNNs.

- Graph isomorphism network is composed as a maximally powerful aggregation-based GNN.
  - 1-layer perceptron $\rightarrow$ Multiple-layer perceptron
  - Mean, max pooling $\rightarrow$ Sum pooling

K. Xu, W. Hu, J. Leskovec, S. Jegelka. How powerful are graph neural networks? ICLR, 2019.

Is expressive power necessary?

- Expressive power offers us a theoretical guide for understanding the capability of GNNs.
  - Whether, and to what degree, are GNNs universal approximitor to functions mapping graphs to real-valued vector?
    - *For graph classification, No!!!*
    - *For node classification, it is almost.*

- For specific tasks, it is not practically necessary to seek high expressive power for performance improvement
  - This partly explains why GCN, GraphSAGE works well although their expressive power is less than GIN.
  - What we really need is a universal function that can map similar objects (nodes or graphs) to close representations, facilitating down-stream tasks.

Thank you for your attentions!