Basic Graph Neural Networks

Yukuo Cen

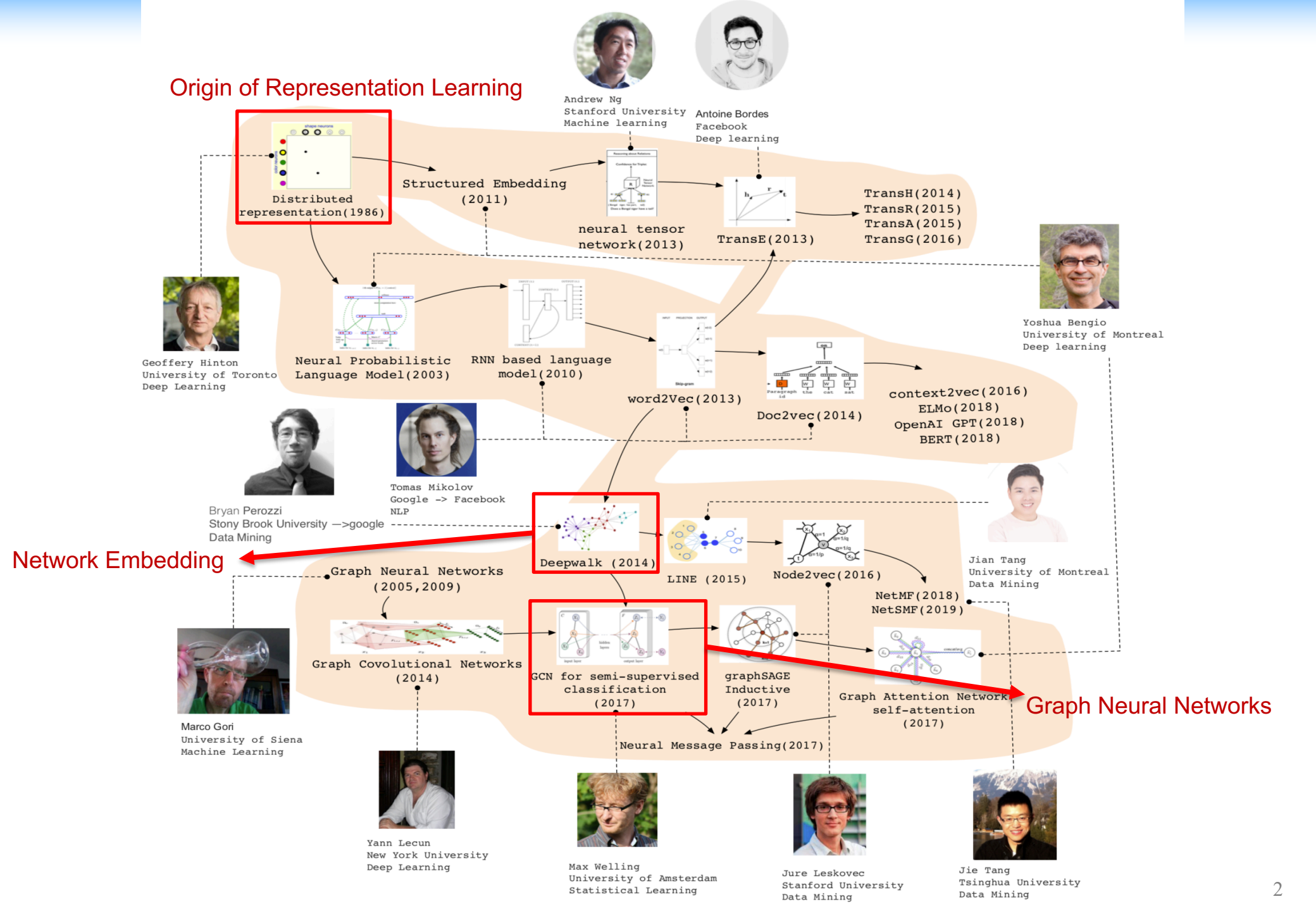
GNN Center, Zhipu AI

KEG, Tsinghua University

Advisors: Yuxiao Dong, Jie Tang

Course Link: https://cogdl.ai/gnn2022/

CogDL is publicly available at https://github.com/THUDM/cogdl


Origin of Representation Learning
Andrew Ng
Stanford University
Machine learning
Antoine Bordes
Facebook
Deep learning
Distributed representation(1986)
Structured Embedding (2011)
neural tensor network(2013)
TransE(2013)
TransH(2014)
TransR(2015)
TransA(2015)
TransG(2016)
Geoffery Hinton
University of Toronto
Deep Learning
Neural Probabilistic Language Model(2003)
RNN based language model(2010)
word2Vec(2013)
Doc2vec(2014)
context2vec(2016)
ELMo(2018)
OpenAI GPT(2018)
BERT(2018)
Yoshua Bengio
University of Montreal
Deep learning
Bryan Perozzi
Stony Brook University —>google
Data Mining
Tomas Mikolov
Google -> Facebook
NLP
Network Embedding
Deepwalk (2014)
LINE (2015)
Node2vec(2016)
NetMF(2018)
NetSMF(2019)
Jian Tang
University of Montreal
Data Mining
Graph Neural Networks (2005,2009)
Graph Covolutional Networks (2014)
GCN for semi-supervised classification (2017)
graphSAGE Inductive (2017)
Graph Attention Network self-attention (2017)
Graph Neural Networks
Marco Gori
University of Siena
Machine Learning
Neural Message Passing(2017)
Yann Lecun
New York University
Deep Learning
Max Welling
University of Amsterdam
Statistical Learning
Jure Leskovec
Stanford University
Data Mining
Jie Tang
Tsinghua University
Data Mining

Graph Neural Networks

- Input: an undirected weighted network $G = (V, E)$ with $|V| = n$ & $|E| = \mathrm{m}$
  - Adjacency matrix $\boldsymbol{A} \in \mathbb{R}_{+}^{n \times \mathrm{n}}$

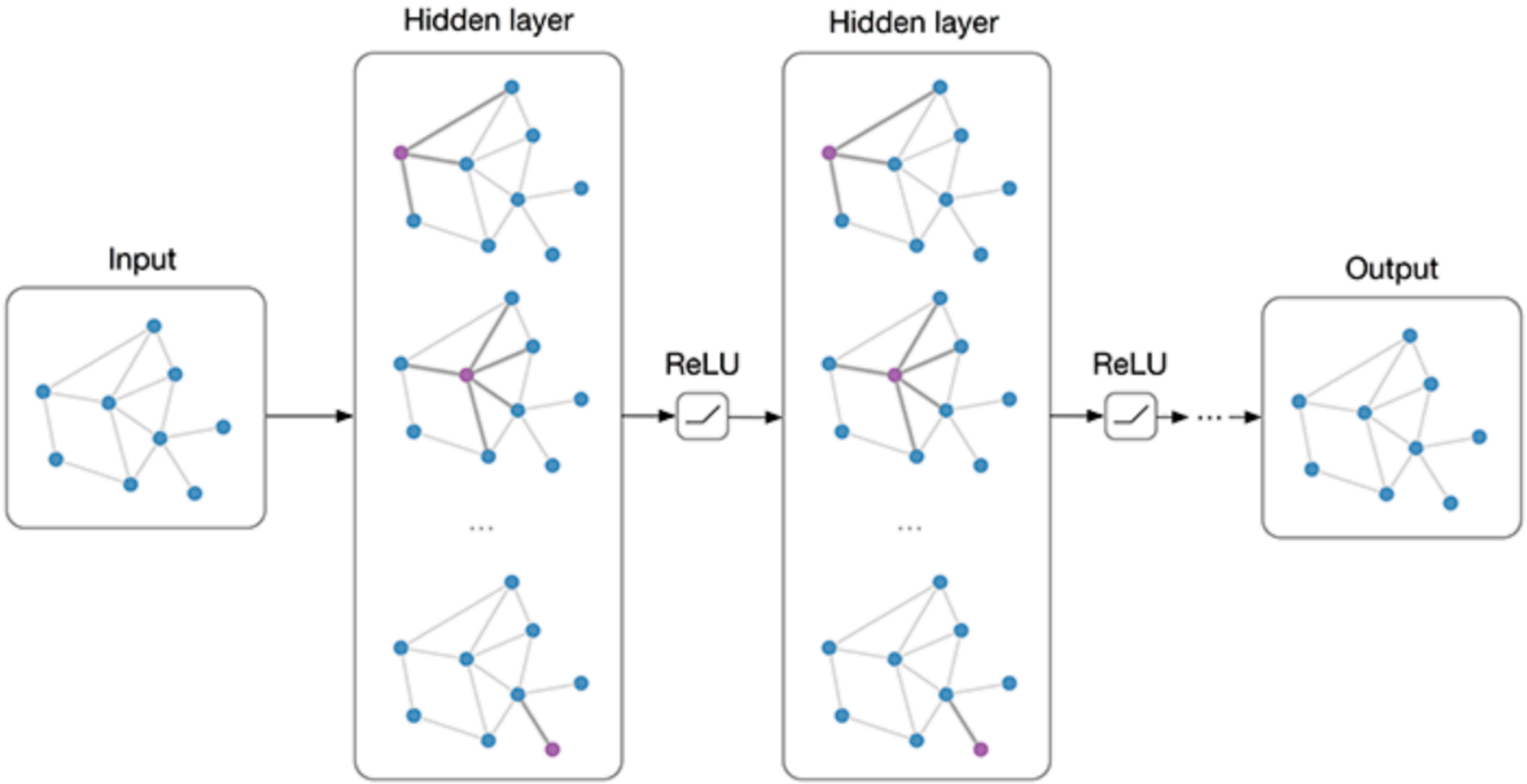
$$A_{i,j} = \begin{cases} a_{i,j} > 0 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$$

  - Degree matrix $\boldsymbol{D} = diag(d_1, d_2, \cdots, d_n)$
  - Node feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times F}$

– Output: for each node, its k-dimension latent feature representation vector $\mathbf{Z}$

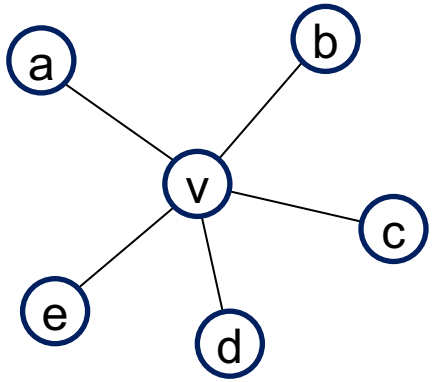
  - Latent feature embedding matrix $\mathbf{Z} \in \mathbb{R}^{n \times c}$

Graph Neural Networks



1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

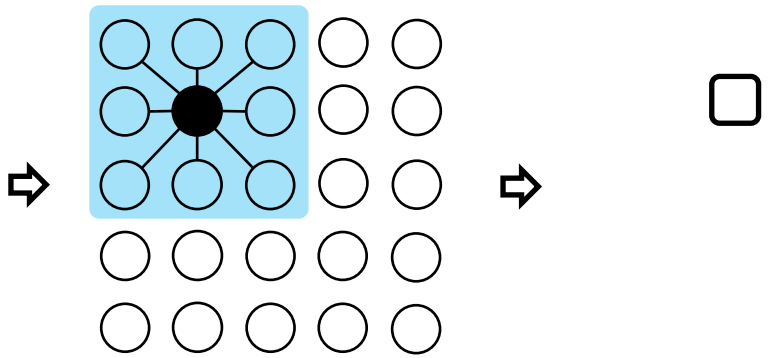
The Core of Graph Neural Networks



$$\boldsymbol{h}_v = f(\boldsymbol{h}_a, \boldsymbol{h}_b, \boldsymbol{h}_c, \boldsymbol{h}_d, \boldsymbol{h}_e)$$

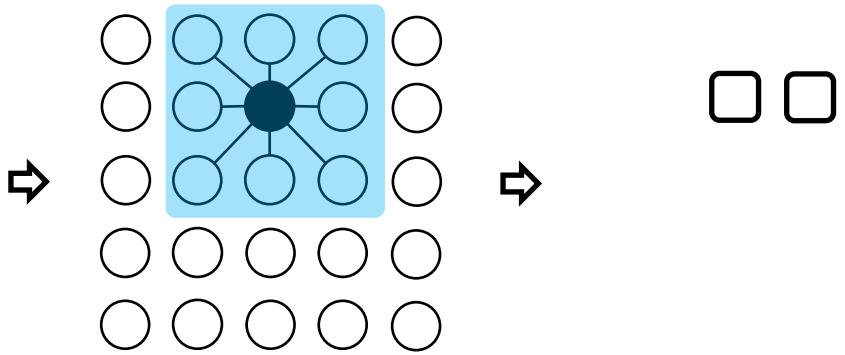
- **Neighborhood Aggregation:**
  - Aggregate neighbor information and pass into a neural network

Convolutional Neural Network



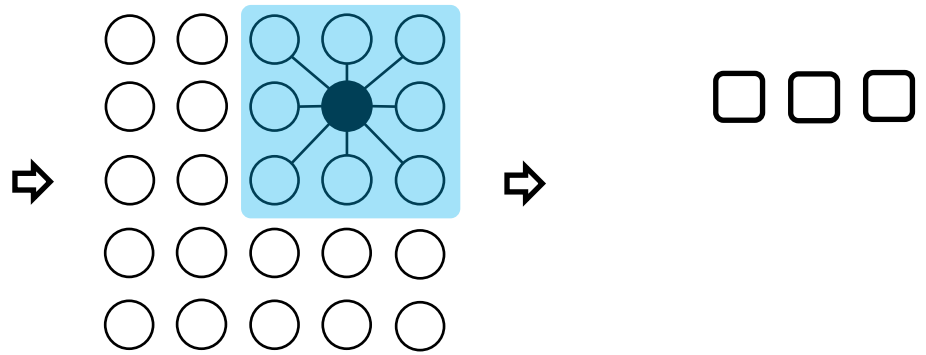
$$\sum WX$$

Convolutional Neural Network



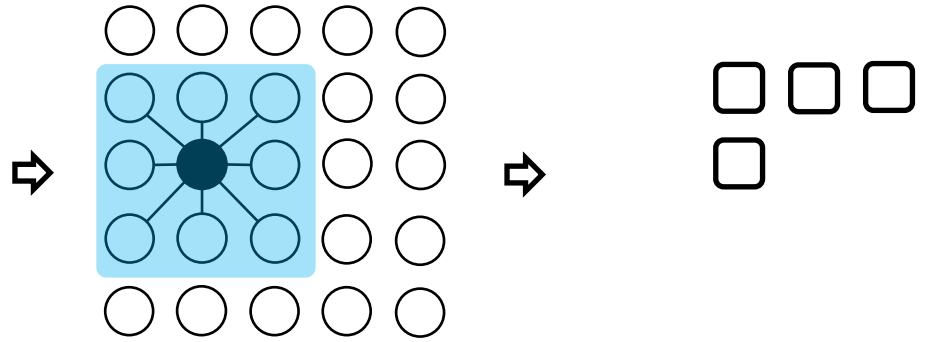
$$\sum WX$$

Convolutional Neural Network



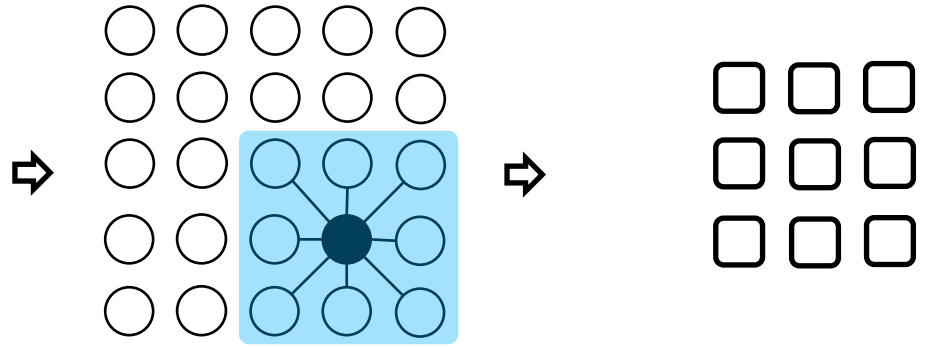
$$\sum WX$$

Convolutional Neural Network



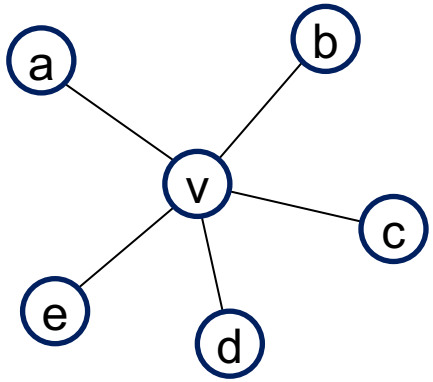
$$\sum WX$$

Convolutional Neural Network



$$\sum WX$$

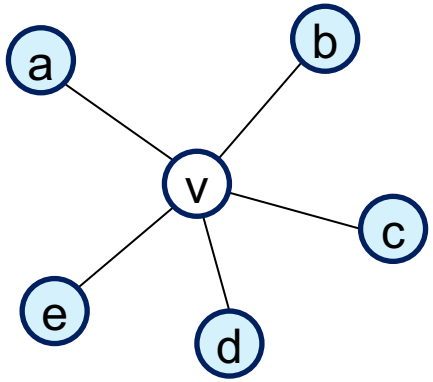
Graph Neural Networks



$$\boldsymbol{h}_a = f(\boldsymbol{h}_a, \boldsymbol{h}_b, \boldsymbol{h}_c, \boldsymbol{h}_d, \boldsymbol{h}_e)$$

- **Neighborhood Aggregation:**
  - Aggregate neighbor information and pass into a neural network
  - It can be viewed as a center-surround filter in CNN---graph convolutions!

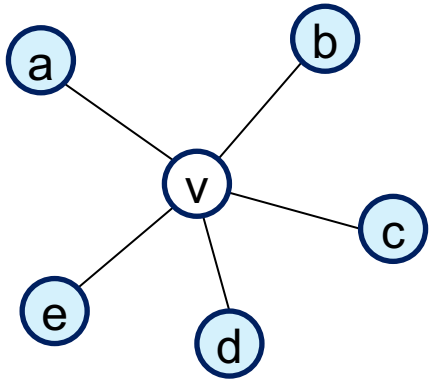
GNN: Graph Convolutional Networks



$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

GNN: Graph Convolutional Networks



parameters in layer k

Non-linear activation function (e.g., ReLU)

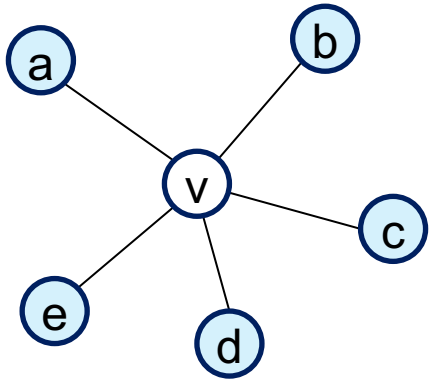
$$\boldsymbol{h}_v^k = \sigma\left(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

node v's embedding at layer k

the neighbors of node v

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

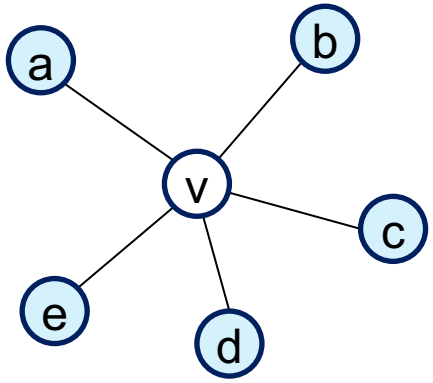
GNN: Graph Convolutional Networks



$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

GNN: Graph Convolutional Networks



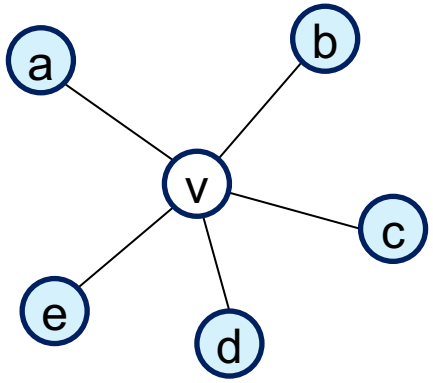
Aggregate from v's neighbors

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v)} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}} + \boldsymbol{W}^k \sum_{v} \frac{\boldsymbol{h}_v^{k-1}}{\sqrt{|N(v)||N(v)|}})$$

Aggregate from itself

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

GNN: Graph Convolutional Networks

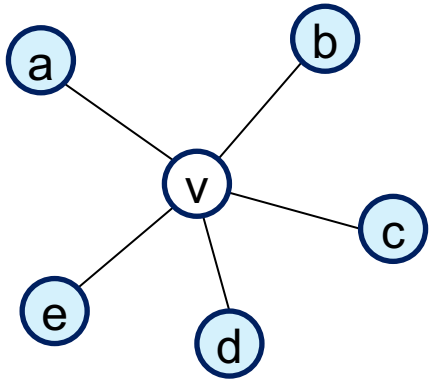


The same parameters for both its neighbors & itself

$$\boldsymbol{h}_v^k = \sigma\Big(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v)} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}} + \boldsymbol{W}^k \sum_{v} \frac{\boldsymbol{h}_v^{k-1}}{\sqrt{|N(v)||N(v)|}}\Big)$$

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

GNN: Graph Convolutional Networks



$$\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}$$

$$\boldsymbol{h}_v^k = \sigma\left(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v)} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}} + \boldsymbol{W}^k \sum_{v} \frac{\boldsymbol{h}_v^{k-1}}{\sqrt{|N(v)||N(v)|}}\right)$$

$$\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{I}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}$$

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

Spectral Analysis Behind GCNs

- Spectral graph convolution: ($x \in \mathbb{R}^n$)

$$g_\theta \star x = U g_\theta U^\top x$$

- Truncated chebyshev expansion:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda})$$

- Now have:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x$$

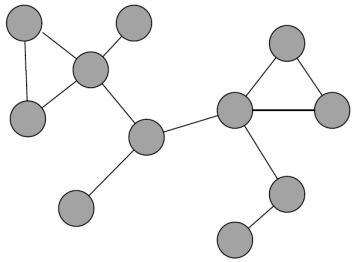
$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) x$$

- Finally, extend $x \in \mathbb{R}^n$ to $X \in \mathbb{R}^{n \times F}$ with C filters:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

GNN: Graph Convolutional Networks



$G = (V, E, \boldsymbol{A})$

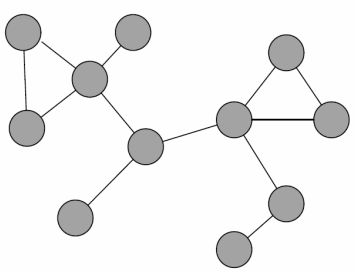
$\boldsymbol{H^0} = \boldsymbol{X}$

$$\Rightarrow \boldsymbol{H}^k = \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}\right) \Rightarrow \quad \boldsymbol{Z} = \boldsymbol{H}^K$$

Input

Output

GNN: Graph Convolutional Networks



$G = (V, E, \boldsymbol{A})$

$\boldsymbol{H}^0 = \boldsymbol{X}$

$$\Rightarrow \boldsymbol{H}^k = \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}\right) \Rightarrow \quad \boldsymbol{Z} = \boldsymbol{H}^K$$

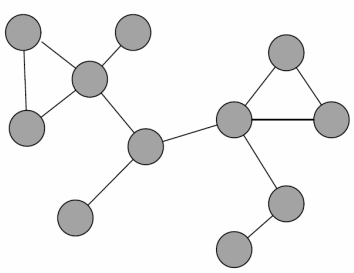
Input

Output

Model training

- The common setting is to have an end to end training framework with a supervised task
- That is, define a loss function over $\boldsymbol{Z}$

GNN: Graph Convolutional Networks



$G = (V, E, \boldsymbol{A})$

$\boldsymbol{H}^0 = \boldsymbol{X}$

$$\Rightarrow \boldsymbol{H}^k = \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}\right) \Rightarrow \quad \boldsymbol{Z} = \boldsymbol{H}^K$$

Input

Output

Benefits: Parameter sharing for all nodes

- |V| is irrelevant to #parameters
- Enable inductive learning for new nodes

GCN Performance

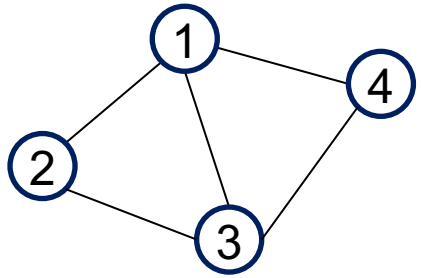
- 2-layer GCN: $\boldsymbol{Z} = \mathrm{softmax}\left(\tilde{\boldsymbol{A}}\,\sigma\left(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}_0\right)\boldsymbol{W}_1\right)$

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

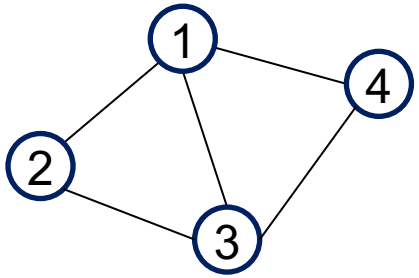
GCN: An Example

- $\widehat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$



- $\widehat{\boldsymbol{D}} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \widehat{\boldsymbol{D}}^{-\frac{1}{2}} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{3}} \end{bmatrix}$

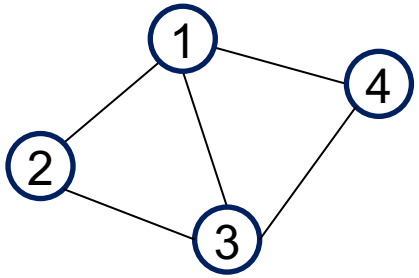
GCN: An Example



- $\widetilde{\boldsymbol{A}} = \widehat{\boldsymbol{D}}^{-\frac{1}{2}}\widehat{\boldsymbol{A}}\widehat{\boldsymbol{D}}^{-\frac{1}{2}} = \begin{bmatrix} \frac{1}{4} & \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{\sqrt{3}}{6} \\ \frac{\sqrt{3}}{6} & \frac{1}{3} & \frac{\sqrt{3}}{6} & 0 \\ \frac{1}{4} & \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{\sqrt{3}}{6} \\ \frac{\sqrt{3}}{6} & 0 & \frac{\sqrt{3}}{6} & \frac{1}{3} \end{bmatrix}$, where $\frac{\sqrt{3}}{6} \approx 0.2887$

GCN: An Example

- $\boldsymbol{H}^{(k)} = \sigma\left(\widetilde{\boldsymbol{A}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}_k\right)$

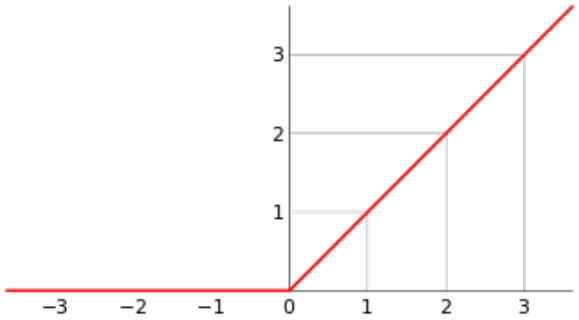


- $\boldsymbol{H}^{(0)} = \boldsymbol{X} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \boldsymbol{W}_1 = \begin{bmatrix} 1 & -0.5 \\ 0.5 & 1 \end{bmatrix}$

- $\boldsymbol{H}^{(0)}\boldsymbol{W}_1 = \begin{bmatrix} 1 & -0.5 \\ 0.5 & 1 \\ 1 & -0.5 \\ 1.5 & 0.5 \end{bmatrix}$

GCN: An Example

- $\widetilde{\boldsymbol{A}}\boldsymbol{H}^{(0)}\boldsymbol{W}_1 = \begin{bmatrix} \frac{1}{4} & \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{\sqrt{3}}{6} \\ \frac{\sqrt{3}}{6} & \frac{1}{3} & \frac{\sqrt{3}}{6} & 0 \\ \frac{1}{4} & \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{\sqrt{3}}{6} \\ \frac{\sqrt{3}}{6} & 0 & \frac{\sqrt{3}}{6} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 & -0.5 \\ 0.5 & 1 \\ 1 & -0.5 \\ 1.5 & 0.5 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{3}+\frac{1}{2} & \frac{\sqrt{3}}{4}-\frac{1}{4} \\ \frac{\sqrt{3}}{3}+\frac{1}{6} & -\frac{\sqrt{3}}{6}+\frac{1}{3} \\ \frac{\sqrt{3}}{3}+\frac{1}{2} & \frac{\sqrt{3}}{4}-\frac{1}{4} \\ \frac{\sqrt{3}}{3}+\frac{1}{2} & -\frac{\sqrt{3}}{6}+\frac{1}{6} \end{bmatrix}$
- $\boldsymbol{H}^{(1)} = ReLU\left(\widetilde{\boldsymbol{A}}\boldsymbol{H}^{(0)}\boldsymbol{W}_1\right) = \begin{bmatrix} \frac{\sqrt{3}}{3}+\frac{1}{2} & \frac{\sqrt{3}}{4}-\frac{1}{4} \\ \frac{\sqrt{3}}{3}+\frac{1}{6} & -\frac{\sqrt{3}}{6}+\frac{1}{3} \\ \frac{\sqrt{3}}{3}+\frac{1}{2} & \frac{\sqrt{3}}{4}-\frac{1}{4} \\ \frac{\sqrt{3}}{3}+\frac{1}{2} & 0 \end{bmatrix} = \begin{bmatrix} 1.0774 & 0.1830 \\ 0.7440 & 0.0447 \\ 1.0774 & 0.1830 \\ 1.0774 & 0 \end{bmatrix}$



GCN Training

- $Z = \text{softmax}\left(\boldsymbol{H}^{(2)}\right) = \text{softmax}\left(\widetilde{\boldsymbol{A}}\boldsymbol{H}^{(1)}\boldsymbol{W}_2\right)$
- $\boldsymbol{W}_2 = \begin{bmatrix} 0.5 & -0.5 \\ 1 & 0.5 \end{bmatrix}$
- $Z = \text{softmax}\left(\boldsymbol{H}^{(2)}\right) = \text{softmax}\left(\begin{bmatrix} 0.6366 & -0.4800 \\ 0.5556 & -0.3747 \\ 0.6366 & -0.4800 \\ 0.5962 & -0.4377 \end{bmatrix}\right)$

$$= \begin{bmatrix} 0.7534 & 0.2466 \\ 0.7171 & 0.2829 \\ 0.7534 & 0.2466 \\ 0.7377 & 0.2623 \end{bmatrix}$$

GCN Training

- $Y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$
- Objective function: cross entropy

$$L = -\frac{1}{4}\sum_{i=1}^{4}\sum_{j=1}^{2} Y_{ij} \ln Z_{ij} = 0.5334$$

- Update $\boldsymbol{W}_1, \boldsymbol{W}_2$ by back-propagation

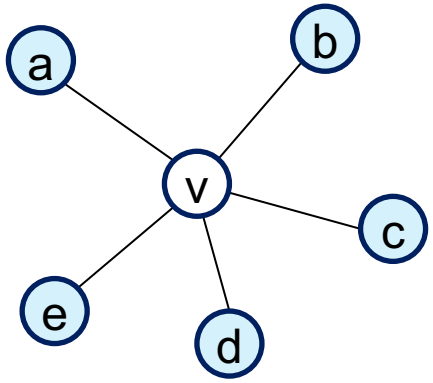
GNN: Graph Convolutional Networks

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

GCN is one way of neighbor aggregations

- **GraphSAGE**
- Graph Attention Networks
- … …

GraphSAGE



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

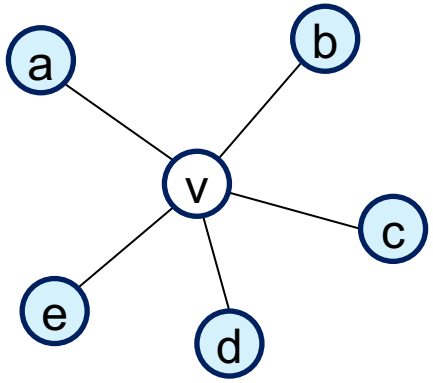
GraphSAGE

$$\boldsymbol{h}_v^k = \sigma([\boldsymbol{A}^k \cdot \mathrm{AGG}(\{\boldsymbol{h}_u^{k-1}, \forall u \in N(v)\}), \boldsymbol{B}^k \boldsymbol{h}_v^{k-1}])$$

Generalized aggregation: any differentiable function that maps set of vectors to a single vector

1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017

GraphSAGE



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

GraphSAGE

Instead of summation, it concatenates neighbor & self embeddings

$$\boldsymbol{h}_v^k = \sigma([\boldsymbol{A}^k \cdot \mathrm{AGG}(\{\boldsymbol{h}_u^{k-1}, \forall u \in N(v)\}), \boldsymbol{B}^k \boldsymbol{h}_v^{k-1}])$$

Generalized aggregation: any differentiable function that maps set of vectors to a single vector

1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017

GraphSAGE

$$\boldsymbol{h}_v^k = \sigma([\boldsymbol{A}^k \cdot \mathrm{AGG}\left(\{\boldsymbol{h}_u^{k-1}, \forall u \in N(v)\}\right), \boldsymbol{B}^k \boldsymbol{h}_v^{k-1}])$$

- Mean

$$\mathrm{AGG} = \sum_{u \in N(v)} \frac{\boldsymbol{h}_u^{k-1}}{|N(v)|}$$

- LSTM (random permutation of neighbors)

$$\mathrm{AGG} = \mathrm{LSTM}([h_u^{k-1}, \forall u \in \pi(N(v))])$$

- Max-pooling

$$\mathrm{AGG} = \max\left(\left\{\sigma\left(\boldsymbol{W}_{pool}\boldsymbol{h}_u^{k-1} + b\right), u \in N(v)\right\}\right)$$

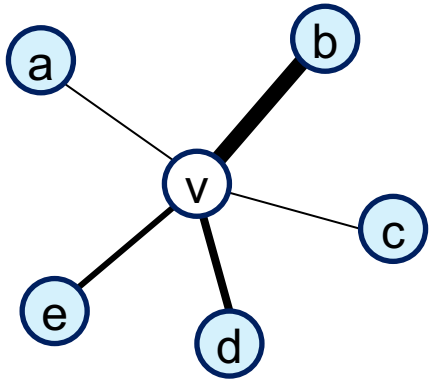
1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017

GraphSAGE Performance

- AGGs: GCN / mean / LSTM / max-pooling
- Supervised (Sup.), Unsupervised (Unsup.)

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

Graph Neural Networks



Realistically, neighbors are of different importance to each node

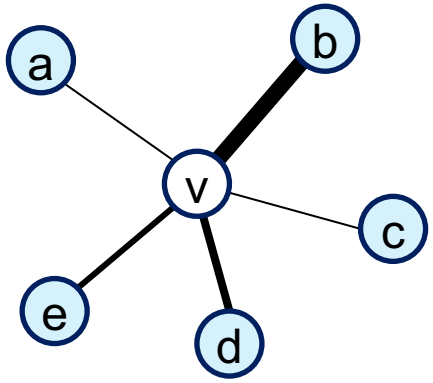
Graph Neural Networks

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

GCN is one way of neighbor aggregations

- GraphSAGE
- **Graph Attention Networks**
- … …

GNN: Graph Attention



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

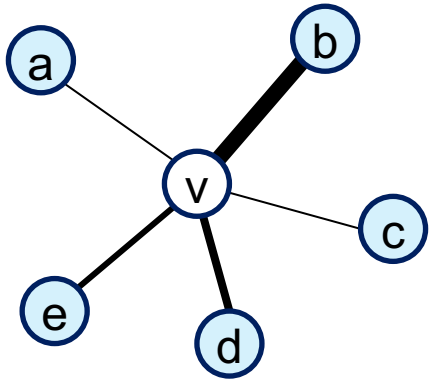
Graph Attention Networks

$$\boldsymbol{h}_v^k = \sigma(\sum_{u \in \mathrm{N}(v) \cup v} \alpha_{v,u} \boldsymbol{W}^k \boldsymbol{h}_u^{k-1})$$

Learned attention weights

1. Velickovic et al. Graph Attention Networks. ICLR 2018

GNN: Graph Attention



$$\alpha_{v,u} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_u]\right)\right)}{\sum_{u' \in N(v) \cup \{v\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_{u'}]\right)\right)}$$

Various ways to define attention!

1. Velickovic et al. Graph Attention Networks. ICLR 2018

GAT Performance

Transductive

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

Inductive

Method	PPI
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	**0.973** ± 0.002

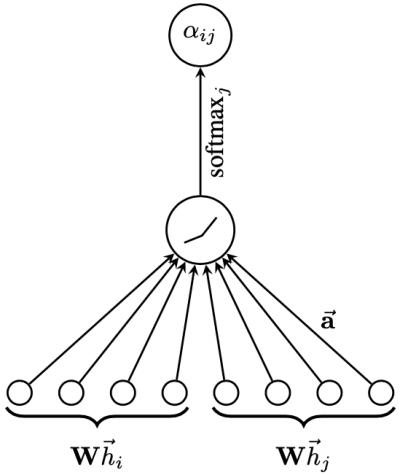
Sparse Operators in GNNs

- GCN（Sparse Maxtrix-Matrix Multiplication, SpMM）

$$\boldsymbol{H}^{(i+1)} = \boldsymbol{A}\boldsymbol{H}^{(i)}\boldsymbol{W}$$

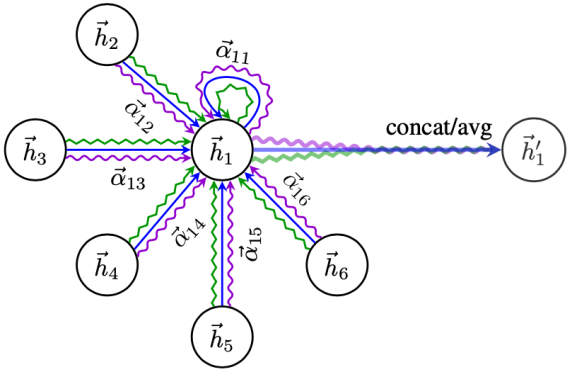
- GAT（ Edge-wise-softmax）

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$



- GAT（Multi-Head SpMM）

$$\boldsymbol{h}_i = CONCAT\left(\sigma\left(\sum_{j \in N_i} \alpha_{ij}^k \boldsymbol{W}^k \boldsymbol{h}_j\right)\right)$$



GCN/GAT Layer in CogDL

```
# GCN Layer
# graph: cogdl.data.Graph
# x: node featurs
# weight: parameters

h = torch.mm(x, weight)
h = spmm(graph, h)
out = torch.relu(h)
```

```
# GAT Layer
# graph: cogdl.data.Graph
# h: node featurs
# h_score: importance score of edge

edge_attention = mul_edge_softmax(graph, edge_score)
h = mh_spmm(graph, edge_attention, h)
out = torch.cat(h, dim=1)
```

$$H^{(i+1)} = AH^{(i)}\boldsymbol{W}^{(\mathrm{i})}$$

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})}$$

$$h_i = CONCAT\left(\sigma\left(\sum_{j\in N_i}\alpha_{ij}^{k}\boldsymbol{W}^{k}h_j\right)\right)$$

Implementation of GCN/GAT Layer

```python
class GCNLayer(nn.Module):
    """
    Simple GCN layer, similar to https://arxiv.o
    """

    def __init__(self, in_features, out_features,

    def reset_parameters(self):…

    def forward(self, graph, x):
        support = torch.mm(x, self.weight)
        out = spmm(graph, support)
```

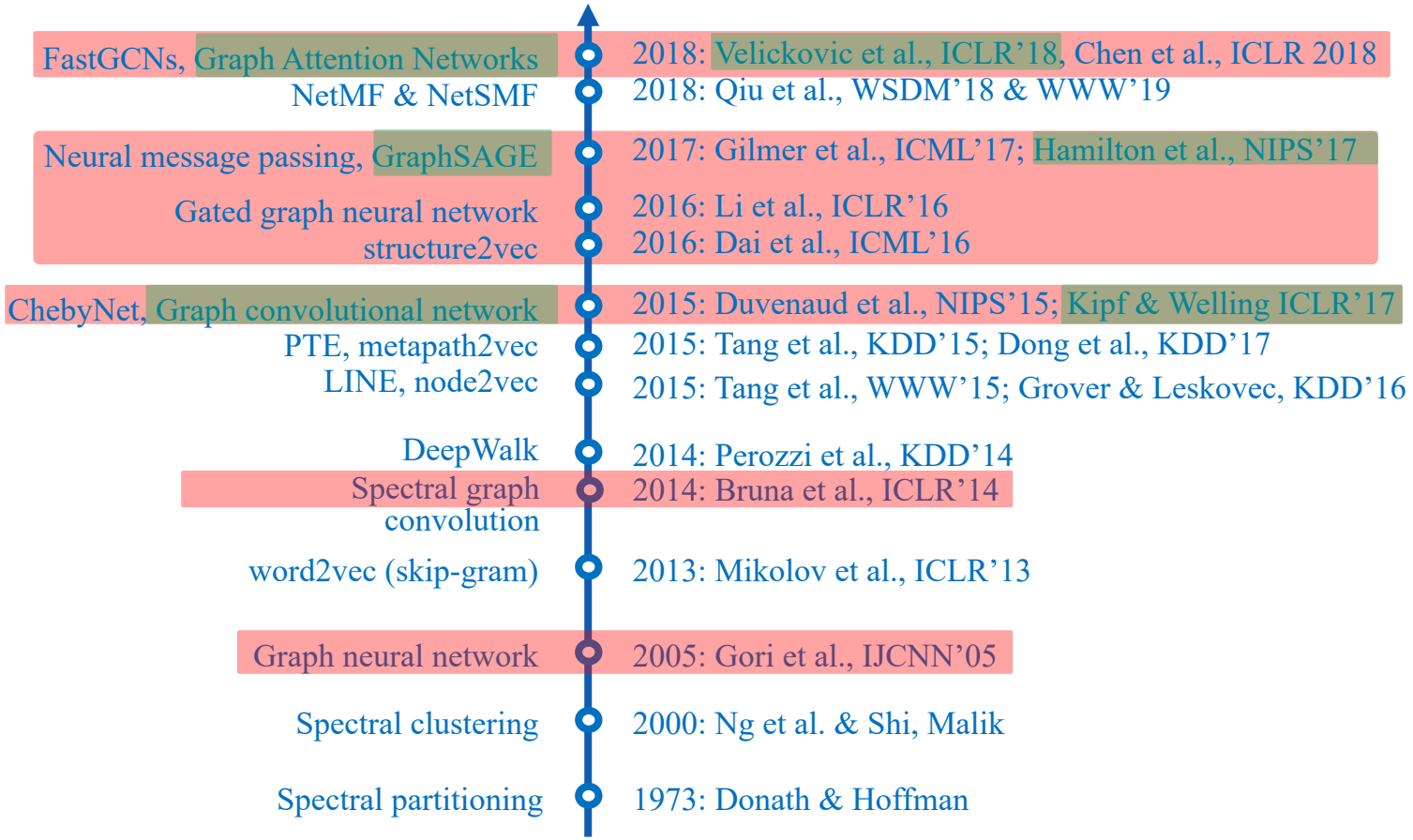
```python
class GATLayer(nn.Module):
    """
    Sparse version GAT layer, similar to https://arxiv.org/ab
    """

    def __init__(self, in_features, out_features, nhead=1, al

    def reset_parameters(self):…

    def forward(self, graph, x):
        h = torch.matmul(x, self.W)
        edge_score = self.compute_edge_score(graph, x, h)
        # edge_attention: E * H
        edge_attention = mul_edge_softmax(graph, edge_score)
        edge_attention = self.dropout(edge_attention)

        out = mh_spmm(graph, edge_attention, h)
```

$$H^{(i+1)} = AH^{(i)}\boldsymbol{W}^{(\mathrm{i})}$$

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})}$$

$$h_i = CONCAT\left(\sigma\left(\sum_{j\in N_i}\alpha_{ij}^{k}\boldsymbol{W}^{k}h_j\right)\right)$$

Network Representation Learning / Network Embedding



Homework 3: GCN Implementation

- Experiments on GCN:
  - Due by 31st July
  - Reproduce GCN computation introduced in slides
  - Implement a simple version of GCN model
  - Test GCN model on the cora dataset
  - Test CogDL's GCN for comparison
  - Give the analysis of the results

- Find the homework material from the course website: https://cogdl.ai/gnn2022/

Thank you !

Collaborators:

Zhenyu Hou, Yuxiao Dong, Jie Tang, et al. (**THU**)

Qingfei Zhao, Xinije Zhang, Peng Zhang, et al. (**Zhipu AI**)

Hongxiao Yang, Chang Zhou, et al. (**Alibaba**)

Yang Yang (**ZJU**)

Yukuo Cen, KEG, Tsinghua U.

https://github.com/THUDM/cogdl

Online Discussion Forum https://discuss.cogdl.ai/