DDA4210 Advanced Machine Learning
Lecture 06 Graph Neural Networks

Jicong Fan

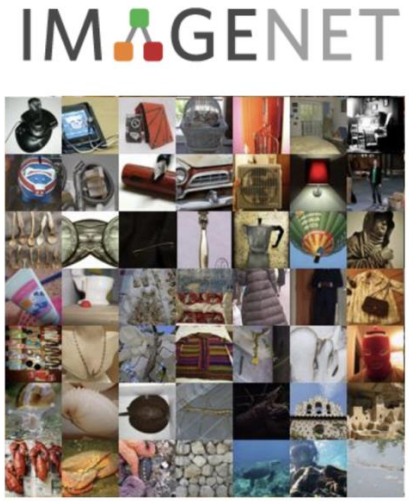
School of Data Science, CUHK-Shenzhen

February 29, 2023

Overview

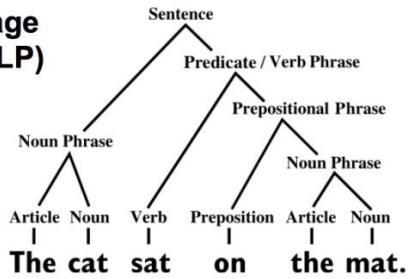
Traditional Neural Networks

Traditional neural networks: MLP, CNN, RNN, Transformer



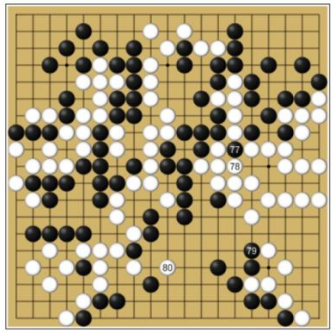
Speech data

Natural language processing (NLP)



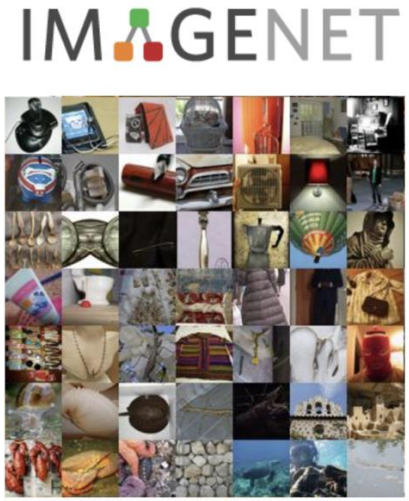
...

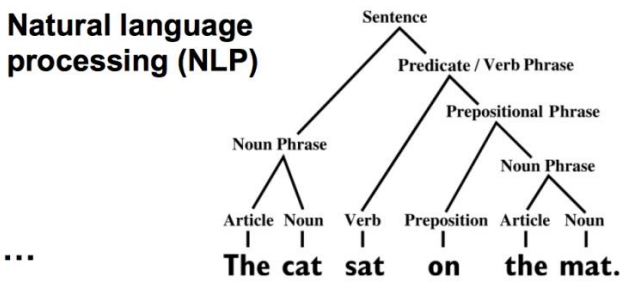
Grid games

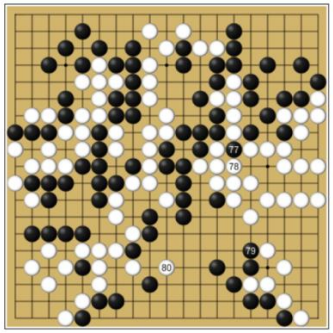


Traditional Neural Networks

Traditional neural networks: MLP, CNN, RNN, Transformer







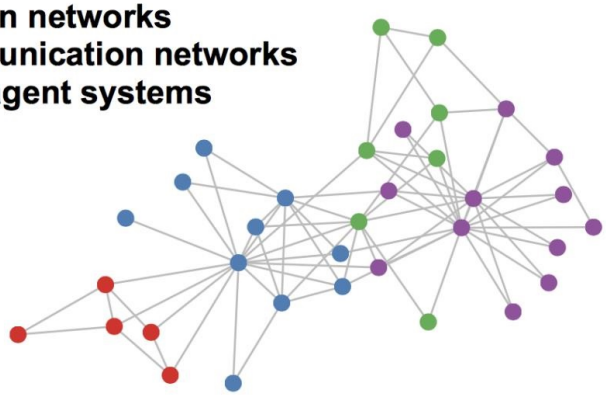
- Strength: strong feature representation ability

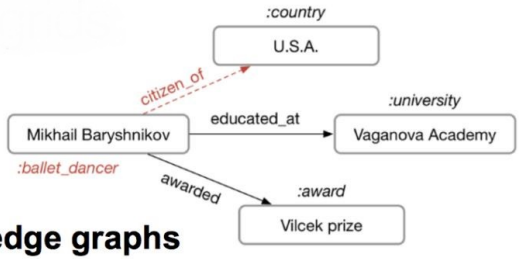
- Limitation: not applicable to non-Euclidean data

Graph-Structured Data

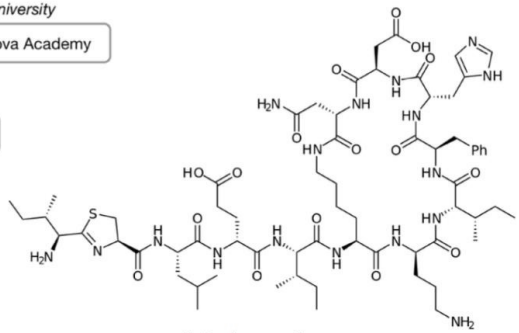
A lot of real-world data do not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems

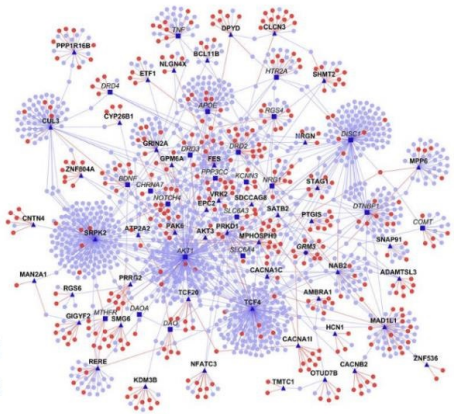




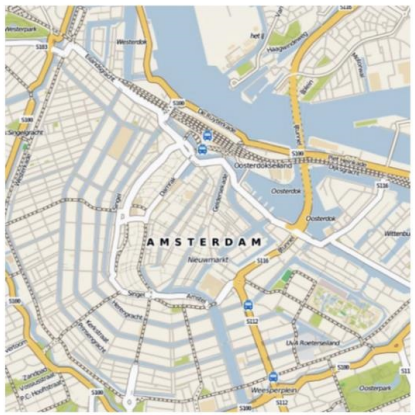
Knowledge graphs



Molecules



Protein interaction networks



Road maps

Standard CNN and RNN architectures don’t work on these data

Graph Data and Related Tasks

- Graph data
  - $G = (V, E)$
  - Vertices/nodes $V = \{v_1, v_2, \ldots, v_n\}$, edges/links $E = \{e_1, e_2, \ldots, e_l\}$
  - Affinity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
  - Feature matrix of nodes $\mathbf{X} \in \mathbb{R}^{n \times d}$ (may not exist)

Graph Data and Related Tasks

- Graph data
  - $G = (V, E)$
  - Vertices/nodes $V = \{v_1, v_2, \ldots, v_n\}$, edges/links $E = \{e_1, e_2, \ldots, e_l\}$
  - Affinity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
  - Feature matrix of nodes $\mathbf{X} \in \mathbb{R}^{n \times d}$ (may not exist)
- Tasks
  - Node embedding/representation
    - Given a graph G, represent each node as a vector, i.e., $(\mathbf{A}, \mathbf{X}) \to \mathbf{Z} \in \mathbb{R}^{n \times k}$
  - Graph embedding/representation
    - Given a set of graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$, represent each graph as a vector, i.e., $(\mathbf{A}_i, \mathbf{X}_i) \to \mathbf{g}_i \in \mathbb{R}^k$

Graph Data and Related Tasks

- Graph data
  - $G = (V, E)$
  - Vertices/nodes $V = \{v_1, v_2, \ldots, v_n\}$, edges/links $E = \{e_1, e_2, \ldots, e_l\}$
  - Affinity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
  - Feature matrix of nodes $\mathbf{X} \in \mathbb{R}^{n \times d}$ (may not exist)
- Tasks
  - Node embedding/representation
    - Given a graph G, represent each node as a vector, i.e., $(\mathbf{A}, \mathbf{X}) \to \mathbf{Z} \in \mathbb{R}^{n \times k}$
  - Graph embedding/representation
    - Given a set of graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$, represent each graph as a vector, i.e., $(\mathbf{A}_i, \mathbf{X}_i) \to \mathbf{g}_i \in \mathbb{R}^k$
  - Node classification: $v_i \to y_i, i = 1, \ldots, n$
  - Graph classification: $G_i \to y_i, i = 1, \ldots, N$
  - Link prediction, node or graph clustering, etc

Node and graph embeddings are crucial for node and graph classifications!

Traditional Embedding Methods

- Node embedding/representation
  - Laplacian embedding [Belkin&Niyogi 2003]
  - Deepwalk [Perozzi et al. 2014]
  - LINE [Tang et al. 2015]
  - node2vec [Grover&Leskovec 2016]

Traditional Embedding Methods

- Node embedding/representation
  - Laplacian embedding [Belkin&Niyogi 2003]
  - Deepwalk [Perozzi et al. 2014]
  - LINE [Tang et al. 2015]
  - node2vec [Grover&Leskovec 2016]
- Graph embedding/representation
  - Methods based on node embeddings
  - Graph kernels [Gartner et al 2003; Kriege et al. 2020]

Note that there are more methods for node and graph embeddings

Graph Neural Networks

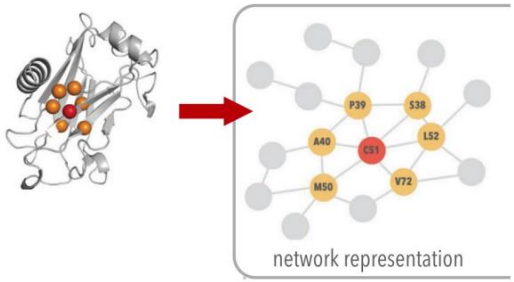
Graph neural networks (GNNs) are NNs that operate on graph-structured data.

Why do we need GNNs?

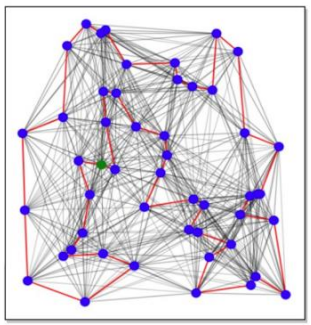
- To capture correlation inside sample.
- To capture correlation across samples.

GNN Example

To capture correlation inside sample



Protein structure [1]

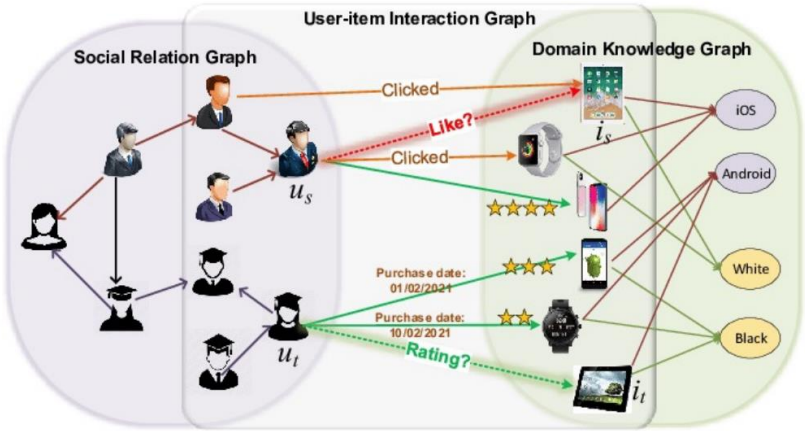


Travelling salesman problem [2]

[1] Vladimir Gligorijevic. et al. Structure-based protein function prediction using graph convolutional networks. 2021
[2] Chaitanya K. Joshi. et al. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. 2019

GNN Example

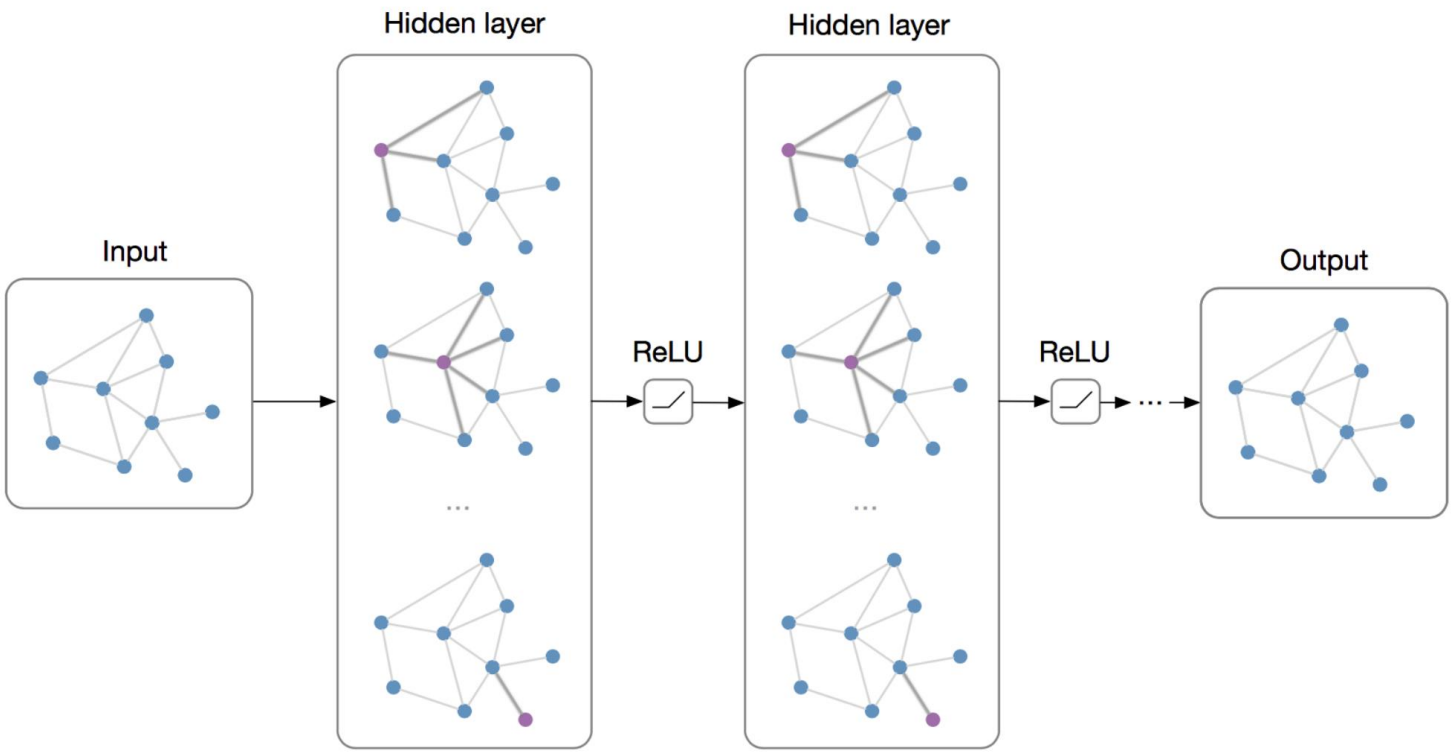
To capture correlation across samples



Recommender system [1]
E.g., Pinterest [2]

[1] Shoujin Wang. et al. Graph Learning based Recommender Systems: A Review. 2021
[2] Rex Ying. et al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. 2018

Graph Neural Network



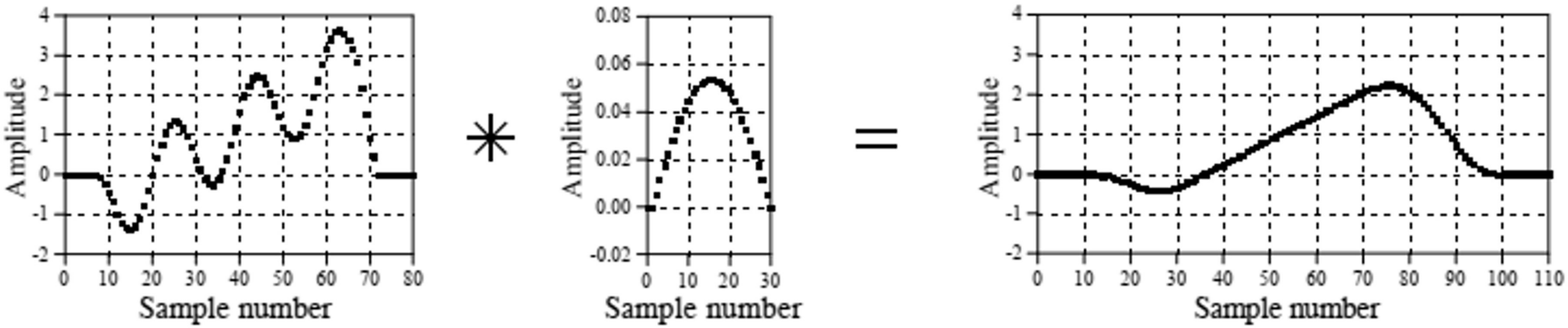
Main Idea: Pass massages between pairs of nodes and agglomerate
Alternative Interpretation: Pass massages between nodes to refine node (and possibly edge) representations

Convolution in signal processing

Convolution is a mathematical operation on two functions (f and g) that produces a third function ($h = f * g$).

- 1-D convolution

$$y[t] = \sum_{\tau=0}^{|k|-1} k[\tau]x[t+\tau]$$

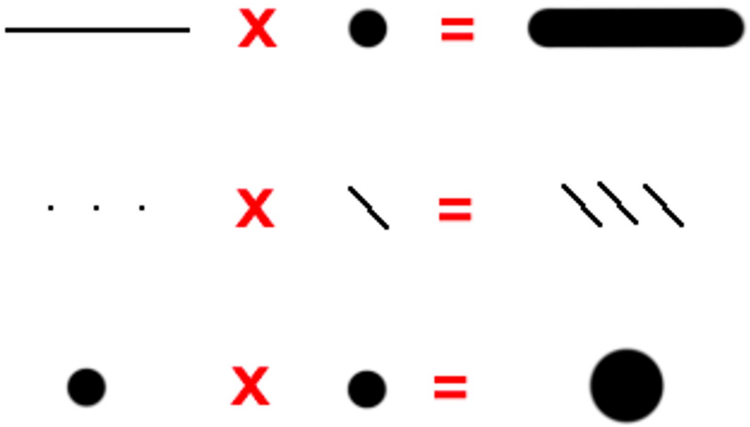


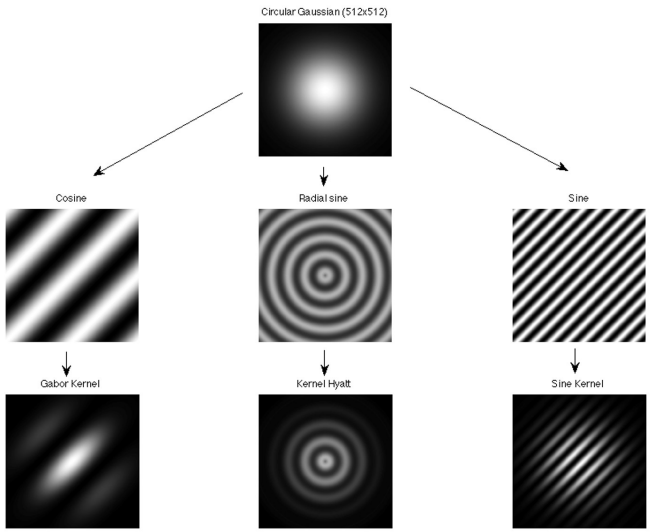
Convolution in signal processing

Convolution is a mathematical operation on two functions (f and g) that produces a third function ($h = f * g$).

- 2-D convolution

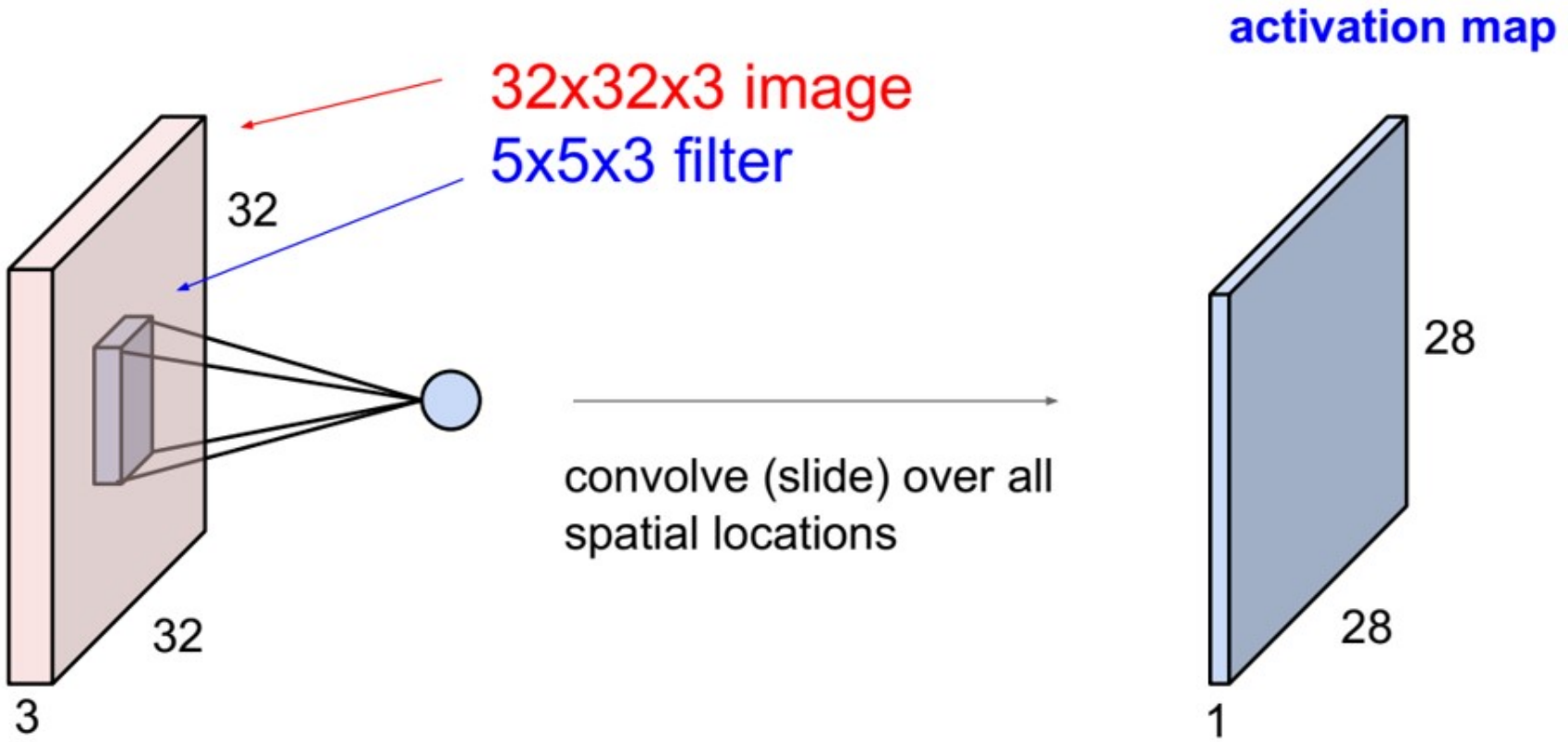
$$y[s,t] = \sum_{\tau=0}^{h-1} \sum_{\gamma=0}^{w-1} k[\tau,\gamma] x[s+\tau, t+\gamma]$$





Convolution in CNN

Convolution of image and filter



Convolution on graph

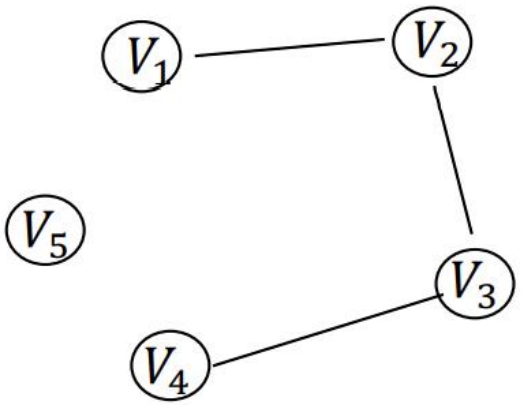
Convolution of a graph G and a feature matrix $\mathbf{H}^{(l)} \in \mathbb{R}^{n \times d_l}$

$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$$

- σ: activation function, e.g., ReLU and Sigmoid
- $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$: parameter matrix
- $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\tilde{\mathbf{D}} = \text{diag}(\sum_i \tilde{\mathbf{A}}_{i1}, \ldots, \sum_i \tilde{\mathbf{A}}_{in})$
- $\mathbf{H}^{(l+1)} \in \mathbb{R}^{n \times d_{l+1}}$: output of l-th GCN layer
- $\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{n \times d}$

Example of GCN layer

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2},\ \tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I},\ \tilde{\mathbf{D}} = \mathrm{diag}(\textstyle\sum_i \tilde{\mathbf{A}}_{i1}, \ldots, \sum_i \tilde{\mathbf{A}}_{in})$$



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tilde{\mathbf{A}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{\mathbf{D}}^{-1/2} = \begin{bmatrix} 0.71 & 0. & 0. & 0. & 0. \\ 0. & 0.58 & 0. & 0. & 0. \\ 0. & 0. & 0.58 & 0. & 0. \\ 0. & 0. & 0. & 0.71 & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}, \hat{\mathbf{A}} = \begin{bmatrix} 0.5 & 0.41 & 0. & 0. & 0. \\ 0.41 & 0.33 & 0.33 & 0. & 0. \\ 0. & 0.33 & 0.33 & 0.41 & 0. \\ 0. & 0. & 0.41 & 0.5 & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

Example of GCN layer

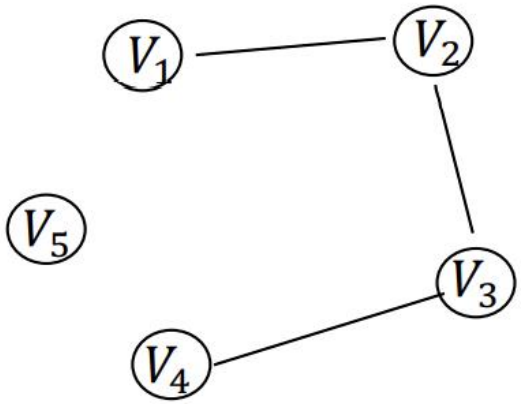
$$\hat{\mathbf{A}} = \begin{bmatrix} 0.5 & 0.41 & 0. & 0. & 0. \\ 0.41 & 0.33 & 0.33 & 0. & 0. \\ 0. & 0.33 & 0.33 & 0.41 & 0. \\ 0. & 0. & 0.41 & 0.5 & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

Except for the diagonals, $\hat{\mathbf{A}}$ has the same pattern of non-zero entries with $\mathbf{A}$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \\ \mathbf{x}_5 \end{bmatrix} \Rightarrow \hat{\mathbf{A}}\mathbf{X} = \begin{bmatrix} 0.5\mathbf{x}_1 + 0.41\mathbf{x}_2 \\ 0.41\mathbf{x}_1 + 0.33\mathbf{x}_2 + 0.33\mathbf{x}_3 \\ 0.33\mathbf{x}_2 + 0.33\mathbf{x}_3 + 0.41\mathbf{x}_4 \\ 0.41\mathbf{x}_4 + 0.5\mathbf{x}_5 \\ \mathbf{x}_5 \end{bmatrix}$$

Example of GCN layer

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \\ \mathbf{x}_5 \end{bmatrix} \Rightarrow \hat{\mathbf{A}}\mathbf{X} = \begin{bmatrix} 0.5\mathbf{x}_1 + 0.41\mathbf{x}_2 \\ 0.41\mathbf{x}_1 + 0.33\mathbf{x}_2 + 0.33\mathbf{x}_3 \\ 0.33\mathbf{x}_2 + 0.33\mathbf{x}_3 + 0.41\mathbf{x}_4 \\ 0.41\mathbf{x}_4 + 0.5\mathbf{x}_5 \\ \mathbf{x}_5 \end{bmatrix}$$



Convolution is just weighted sum of a node's feature and its neighbors' features, aka message passing and aggregation

$$\mathbf{H}^{(1)} = \sigma(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)})$$
$$\mathbf{H}^{(2)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(1)}\mathbf{W}^{(2)})$$
$$\vdots$$
$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l+1)})$$
$$\vdots$$

Why GCNs work

In essence, GCN layer is an approximated spectral convolution.
Consider a signal $\mathbf{x} \in \mathbb{R}^n$ (each node has a scalar) and a filter g_θ (e.g. $g_\theta(\Lambda) = \mathrm{diag}(\theta)$) parameterized by $\theta \in \mathbb{R}^n$ in Fourier domain. $\mathbf{x}$ is filtered by g_θ as

$$
\begin{aligned}
g_\theta * \mathbf{x} &\overset{(1)}{=} g_\theta(\mathbf{L})\mathbf{x} = \mathbf{U} g_\theta(\Lambda)\mathbf{U}^\top \mathbf{x} \\
&\overset{(2)}{\approx} \mathbf{U}\left(\sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})\right)\mathbf{U}^\top \mathbf{x} = \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{x} \\
&\overset{(3)}{\approx} \theta_0 \mathbf{x} - \theta_1 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{x} \\
&\overset{(4)}{\approx} \theta(\mathbf{I}_N + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x} \\
&\overset{(5)}{\approx} \theta\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{x}
\end{aligned}
$$

- In (1) $\mathbf{U}$: eigenvectors of $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = \mathbf{U}\Lambda\mathbf{U}^\top$ (time consuming!)
- (2) uses K-th order Chebyshev polynomials, $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - \mathbf{I}$, $\tilde{\mathbf{L}} = \frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}$. The Chebyshev polynomials are recursively defined as $T_k(a) = 2aT_{k-1}(a) - T_{k-2}(a)$, with $T_0(a) = 1$ and $T_1(a) = a$.

Why GCNs work

In essence, GCN layer is an approximated spectral convolution.
Consider a signal $\mathbf{x} \in \mathbb{R}^n$ (each node has a scalar) and a filter g_θ (e.g. $g_\theta(\Lambda) = \mathrm{diag}(\theta)$) parameterized by $\theta \in \mathbb{R}^n$ in Fourier domain. $\mathbf{x}$ is filtered by g_θ as

$$
\begin{aligned}
g_\theta * \mathbf{x} &\overset{(1)}{=} g_\theta(\mathbf{L})\mathbf{x} = \mathbf{U} g_\theta(\Lambda)\mathbf{U}^\top \mathbf{x} \\
&\overset{(2)}{\approx} \mathbf{U}\left(\sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})\right)\mathbf{U}^\top \mathbf{x} = \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{x} \\
&\overset{(3)}{\approx} \theta_0 \mathbf{x} - \theta_1 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{x} \\
&\overset{(4)}{\approx} \theta(\mathbf{I}_N + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x} \\
&\overset{(5)}{\approx} \theta\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{x}
\end{aligned}
$$

- (3) sets $K = 1$ and $\lambda_{\max} \approx 2$
- (4) assumes $\theta_0 + \theta_1 = 0$
- (5) uses the renormalization trick $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$

Why GCNs work

In essence, GCN layer is an approximated spectral convolution.
Consider a signal $\mathbf{x} \in \mathbb{R}^n$ (each node has a scalar) and a filter g_θ (e.g. $g_\theta(\Lambda) = \mathrm{diag}(\theta)$) parameterized by $\theta \in \mathbb{R}^n$ in Fourier domain. $\mathbf{x}$ is filtered by g_θ as

$$
\begin{aligned}
g_\theta * \mathbf{x} &\overset{(1)}{=} g_\theta(\mathbf{L})\mathbf{x} = \mathbf{U} g_\theta(\Lambda)\mathbf{U}^\top \mathbf{x} \\
&\overset{(2)}{\approx} \mathbf{U}\left(\sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})\right)\mathbf{U}^\top \mathbf{x} = \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{x} \\
&\overset{(3)}{\approx} \theta_0 \mathbf{x} - \theta_1 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{x} \\
&\overset{(4)}{\approx} \theta(\mathbf{I}_N + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x} \\
&\overset{(5)}{\approx} \theta \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{x}
\end{aligned}
$$

Note that $g_\theta * \mathbf{x} \approx \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{x}\theta = \hat{\mathbf{A}}\mathbf{x}\theta$. The form $\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}$ in GCN is the generalization of the formula. More details can be found in [Kipf and Welling 2017; Defferrard et al. 2016].

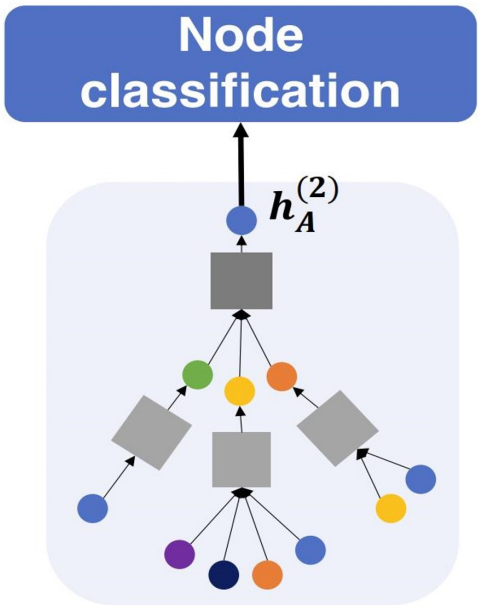
Why GCNs work

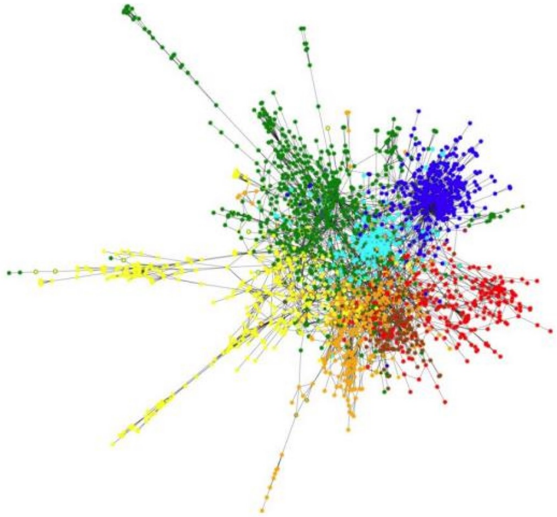
- Commonly used architecture

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \text{softmax}(\hat{\mathbf{A}}\ \text{ReLu}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)})$$

- Why only two layers?
  - Deep GCNs do not perform well.
  - An intuitive explanation is, graph convolution can be viewed as information exchange between neighbors, and if we keep doing this, all nodes' features will become more and more similar.
  - Graph Laplacian $\hat{\mathbf{A}}$ has a smoothing effect. [Li et al. 2018] proved that if we apply the graph Laplacian enough times, all nodes' features will converge to the same value. Hence the name over-smoothing.
  - There are still some deep GCNs, with modified architectures.

GCN: node classification





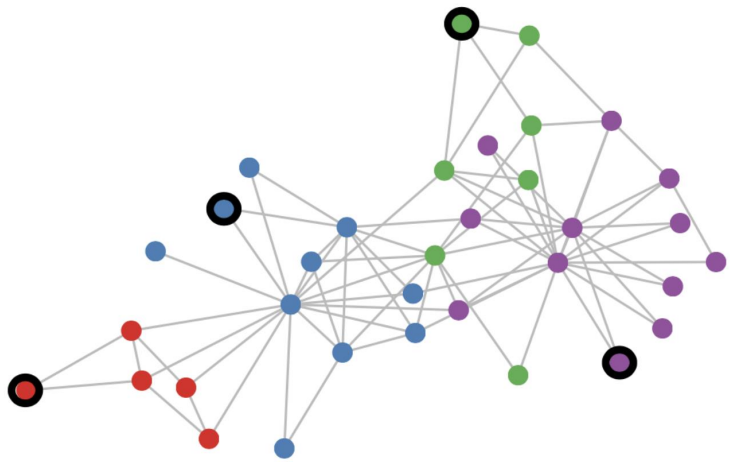
reddit

amazon

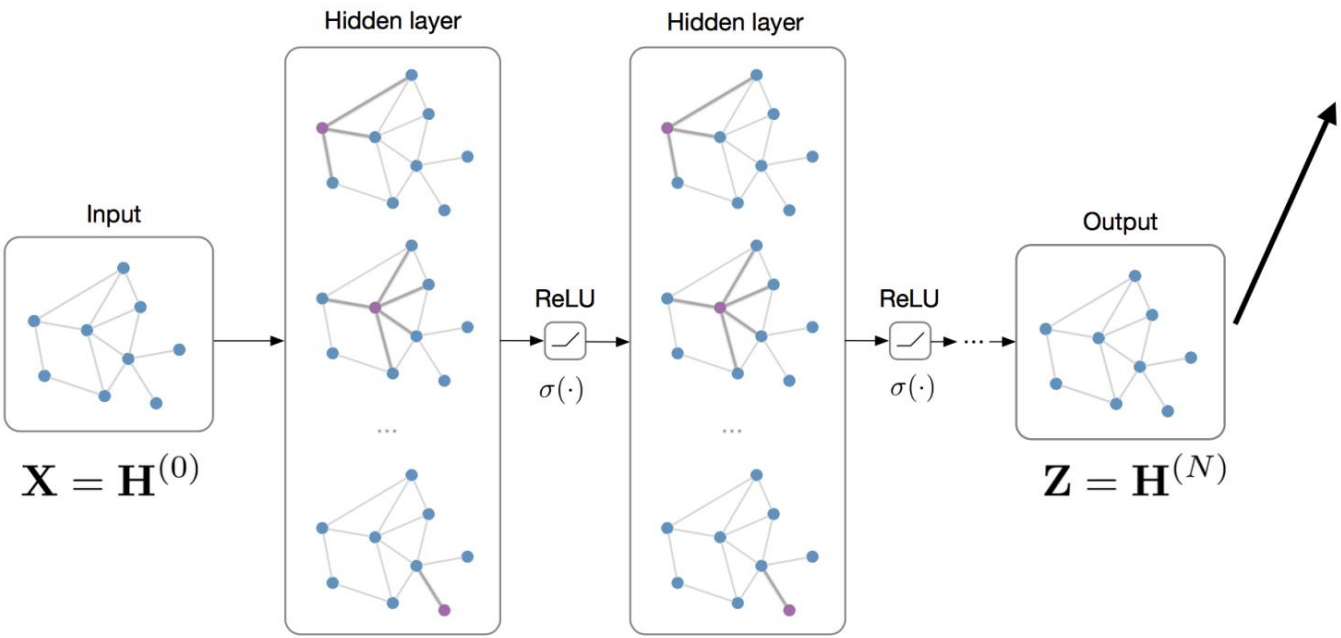
- Classify **papers** into topics on **citation networks**
- Classify **posts** into subgroups on **Reddit networks**
- Classify **products** into categories on **Amazon co-purchase graphs**

GCN: node classification

- **Setting**: some nodes are labeled (black circle), all other nodes are unlabeled
  - $\mathcal{Y}_L$: set of labeled node indices
  - $\mathbf{Y} \in \{0,1\}^{L \times K}$: label matrix
  - $\mathbf{X} \in \mathbb{R}^{n \times d}$: feature matrix
  - $\hat{\mathbf{A}}$: preprocessed adjacency matrix
- **Task**: predict node labels of unlabeled nodes



GCN: node classification

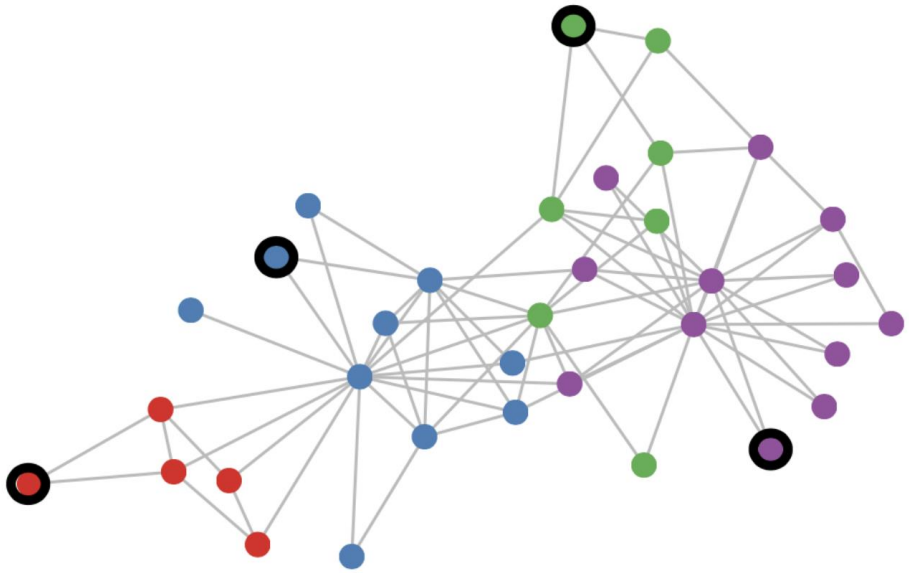


Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

GCN: node classification



- Output of GCN:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \mathrm{softmax}(\hat{\mathbf{A}}\ \mathrm{ReLu}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)})$$

- Objective function (semi-supervised):

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}_L} \sum_{k=1}^{K} Y_{ik} \ln Z_{ik}$$

GCN: node classification

Experiments

- Datasets [Yang et al. 2016]

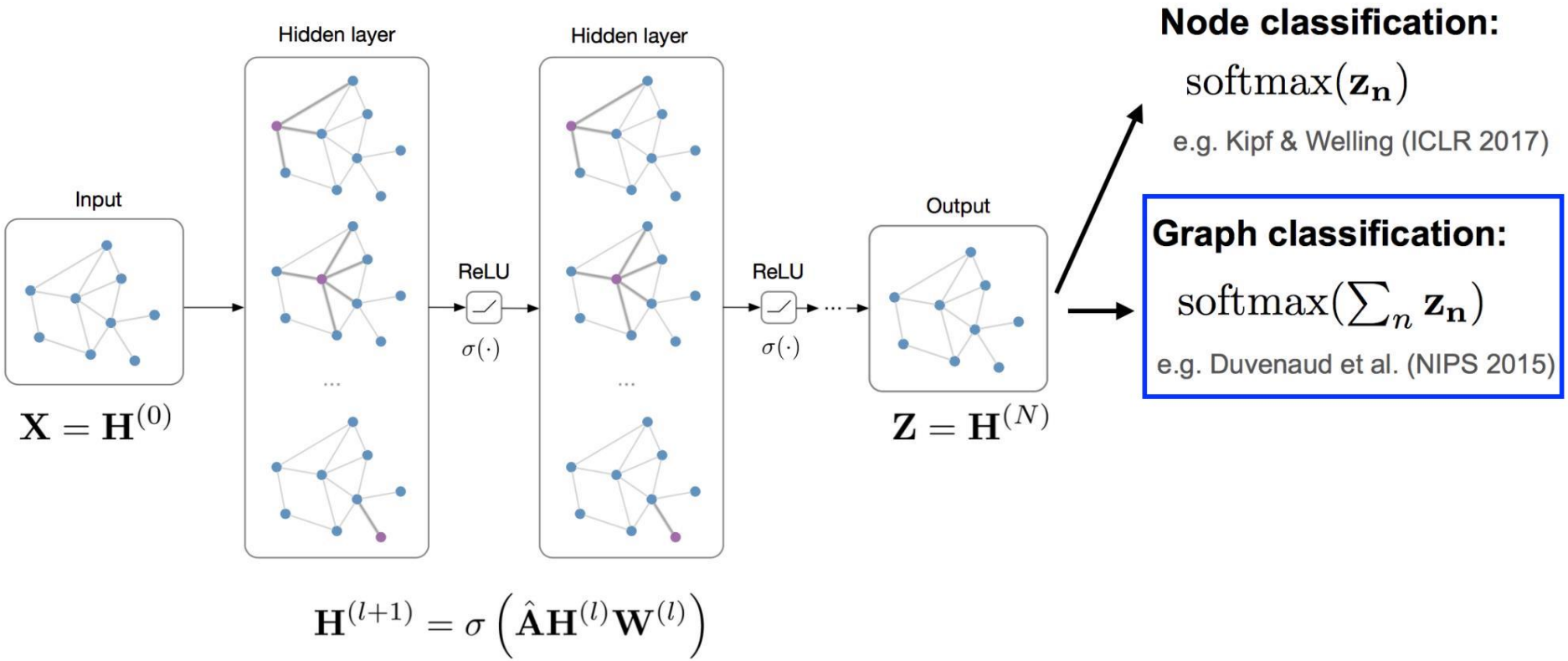
Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

- Classification accuracy [Kipf & Welling 2017]

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: graph classification

Task: given a set of graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_j, \ldots\}$ with $\{\mathbf{X}_j \in \mathbb{R}^{n_j \times d}, \hat{\mathbf{A}}_j \in \mathbb{R}^{n_j \times n_j}\}$, train a model to classify them into K classes.



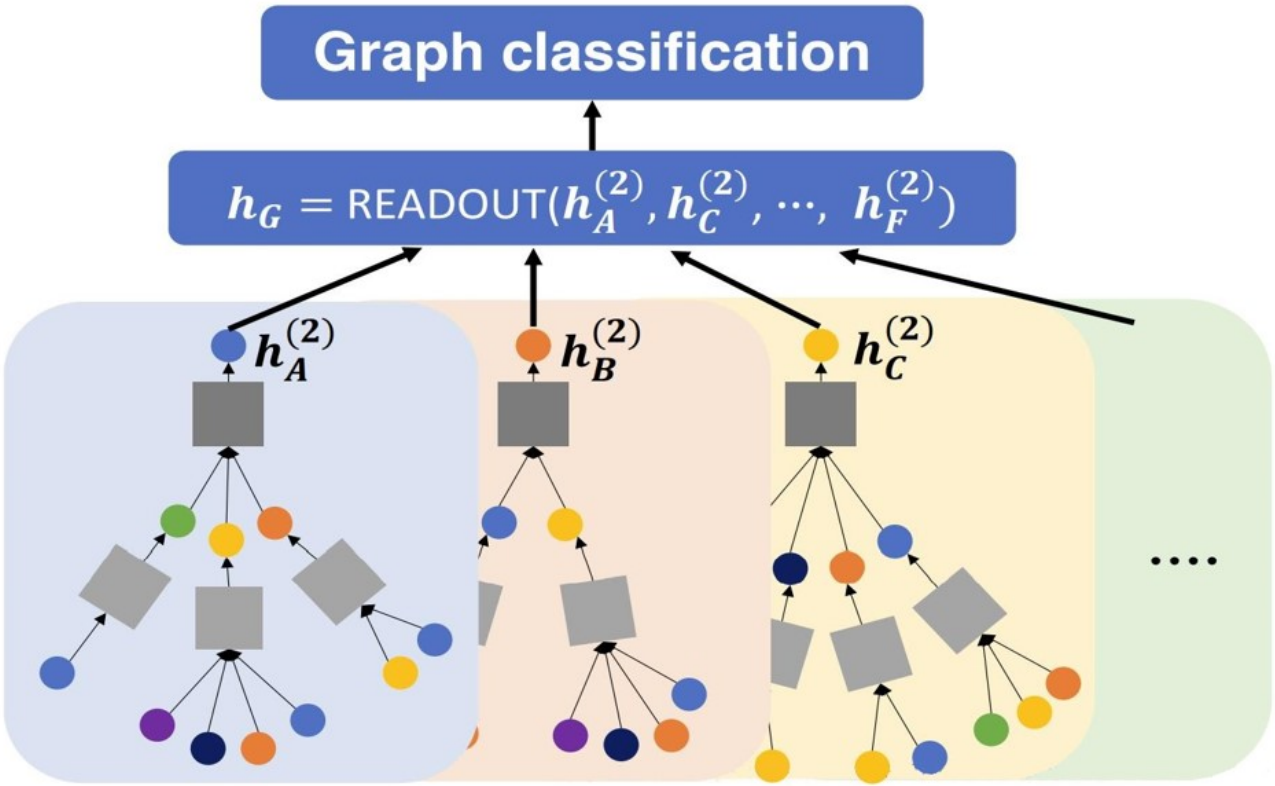
How to define the feature vector of a graph?

GCN: graph classification

READOUT function: compute graph feature from nodes' features

$$\mathbf{h}_G = \text{READOUT}(\{\mathbf{h}_v\}_{v \in \mathcal{V}})$$

E.g.: sum, average, min/max pooling of node embeddings



GCN: graph classification

READOUT function using different ways

- Sum: $\mathbf{h}_G = \sum_{i=1}^{n_G} \mathbf{h}_i$
- Average: $\mathbf{h}_G = \frac{1}{N_G} \sum_{i=1}^{n_G} \mathbf{h}_i$
- Min/Max: $\mathbf{h}_G = \min / \max([\mathbf{h}_1; \ldots; \mathbf{h}_{n_G}])$

Which one is better?

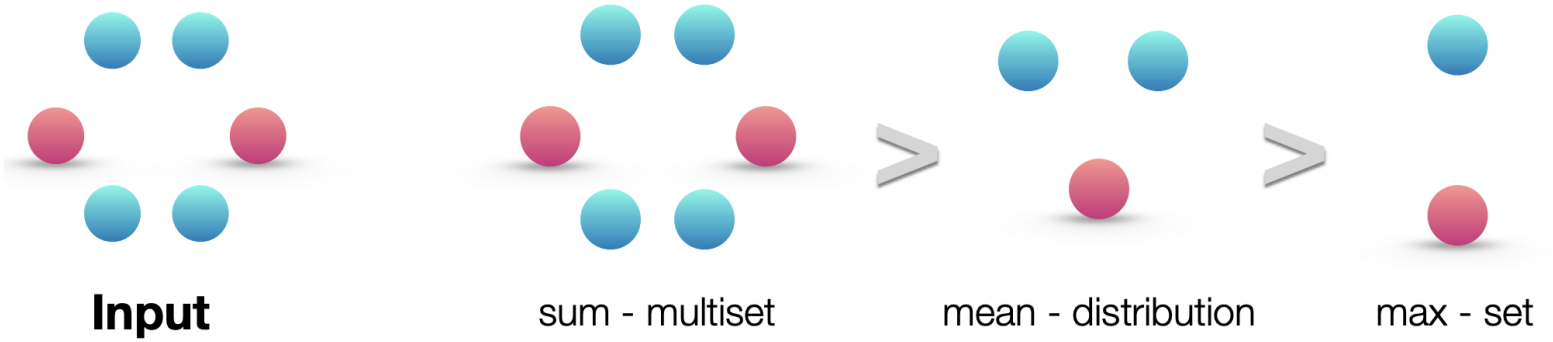
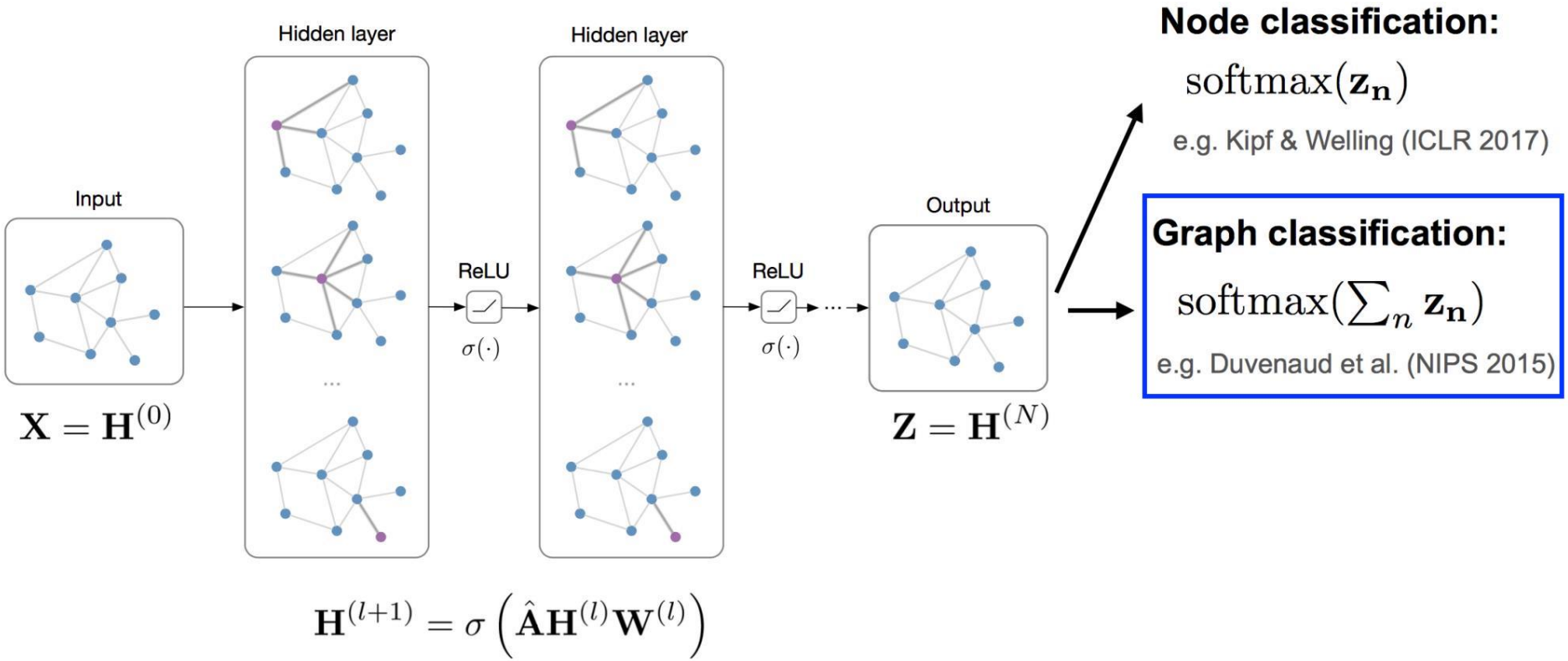


Figure 2: **Ranking by expressive power for sum, mean and max aggregators over a multiset.** Left panel shows the input multiset, *i.e.*, the network neighborhood to be aggregated. The next three panels illustrate the aspects of the multiset a given aggregator is able to capture: sum captures the full multiset, mean captures the proportion/distribution of elements of a given type, and the max aggregator ignores multiplicities (reduces the multiset to a simple set).

Image from: Xu et al. How powereful are graph neural networks? ICLR 2019.

GCN: graph classification

Task: given a set of graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_j, \ldots\}$ with $\{\mathbf{X}_j \in \mathbb{R}^{n \times d}, \hat{\mathbf{A}}_j \in \mathbb{R}^{n \times n}\}$, train a model to classify them into K classes.



*In this chart, feature of graph is computed as the sum of the features of its nodes.

Objective function (supervised): $\mathcal{L} = -\sum_j \sum_{k=1}^{K} Y_{jk} \ln Z_{jk}$

GCN: graph classification

Experiments: graph classification accuracy (%) of different GNNs with different readout functions

	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	$\mathbf{86.0 \pm 1.8}$ *
	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	$\mathbf{92.6 \pm 4.2}$ *	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**)	$\mathbf{75.1 \pm 5.1}$	$\mathbf{52.3 \pm 2.8}$	$\mathbf{92.4 \pm 2.5}$	$\mathbf{57.5 \pm 1.5}$	$\mathbf{80.2 \pm 1.9}$	$\mathbf{89.4 \pm 5.6}$	$\mathbf{76.2 \pm 2.8}$	$\mathbf{64.6 \pm 7.0}$	$\mathbf{82.7 \pm 1.7}$
	SUM–MLP (**GIN-ϵ**)	$\mathbf{74.3 \pm 5.1}$	$\mathbf{52.1 \pm 3.6}$	$\mathbf{92.2 \pm 2.3}$	$\mathbf{57.0 \pm 1.7}$	$\mathbf{80.1 \pm 1.9}$	$\mathbf{89.0 \pm 6.0}$	$\mathbf{75.9 \pm 3.8}$	63.7 ± 8.2	$\mathbf{82.7 \pm 1.6}$
	SUM–1-LAYER	74.1 ± 5.0	$\mathbf{52.2 \pm 2.4}$	90.0 ± 2.7	55.1 ± 1.6	$\mathbf{80.6 \pm 1.9}$	$\mathbf{90.0 \pm 8.8}$	$\mathbf{76.2 \pm 2.6}$	63.1 ± 5.7	82.0 ± 1.5
	MEAN–MLP	73.7 ± 3.7	$\mathbf{52.3 \pm 3.1}$	50.0 ± 0.0	20.0 ± 0.0	79.2 ± 2.3	83.5 ± 6.3	75.5 ± 3.4	$\mathbf{66.6 \pm 6.9}$	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

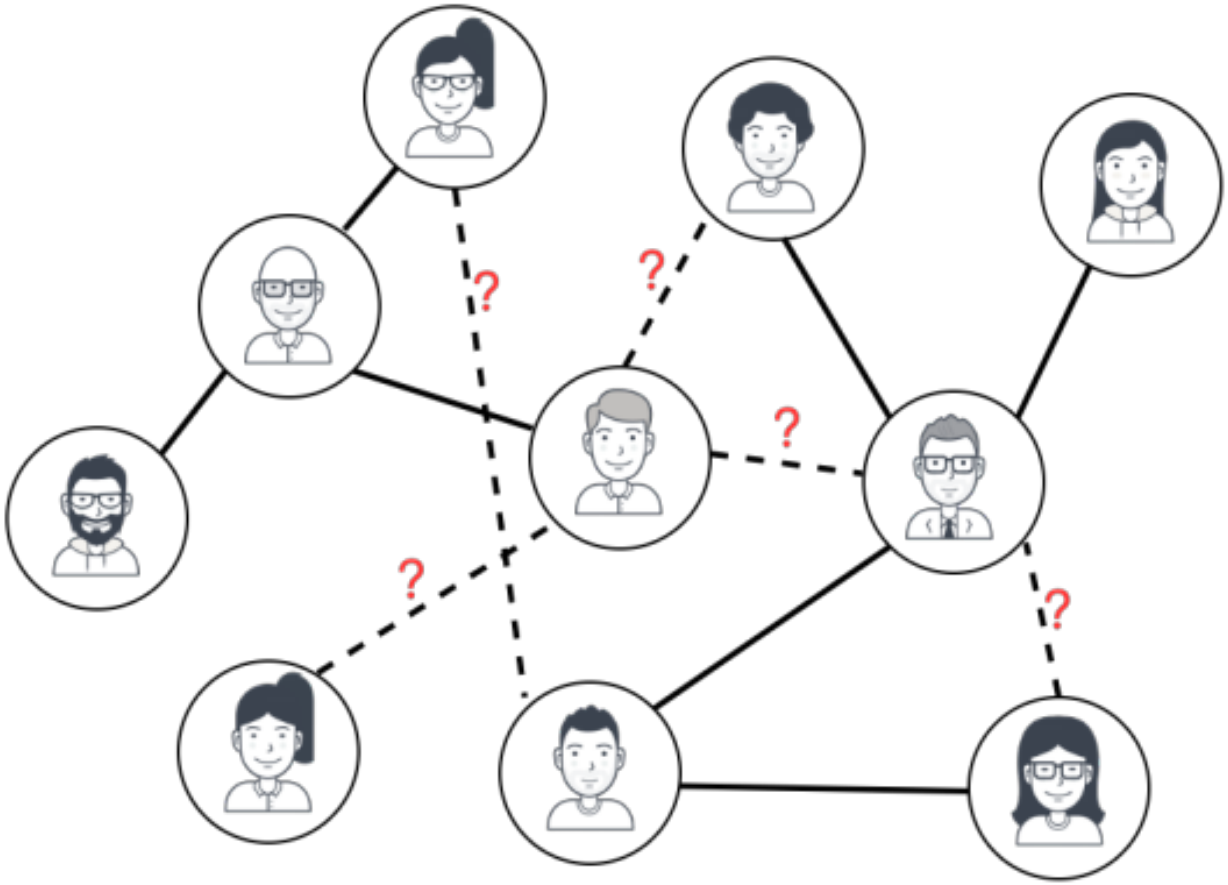
Table from: Xu et al. How powereful are graph neural networks? ICLR 2019.

GCN: link prediction

Link prediction: given a graph $G = (V, E)$, predict new edges, i.e.,

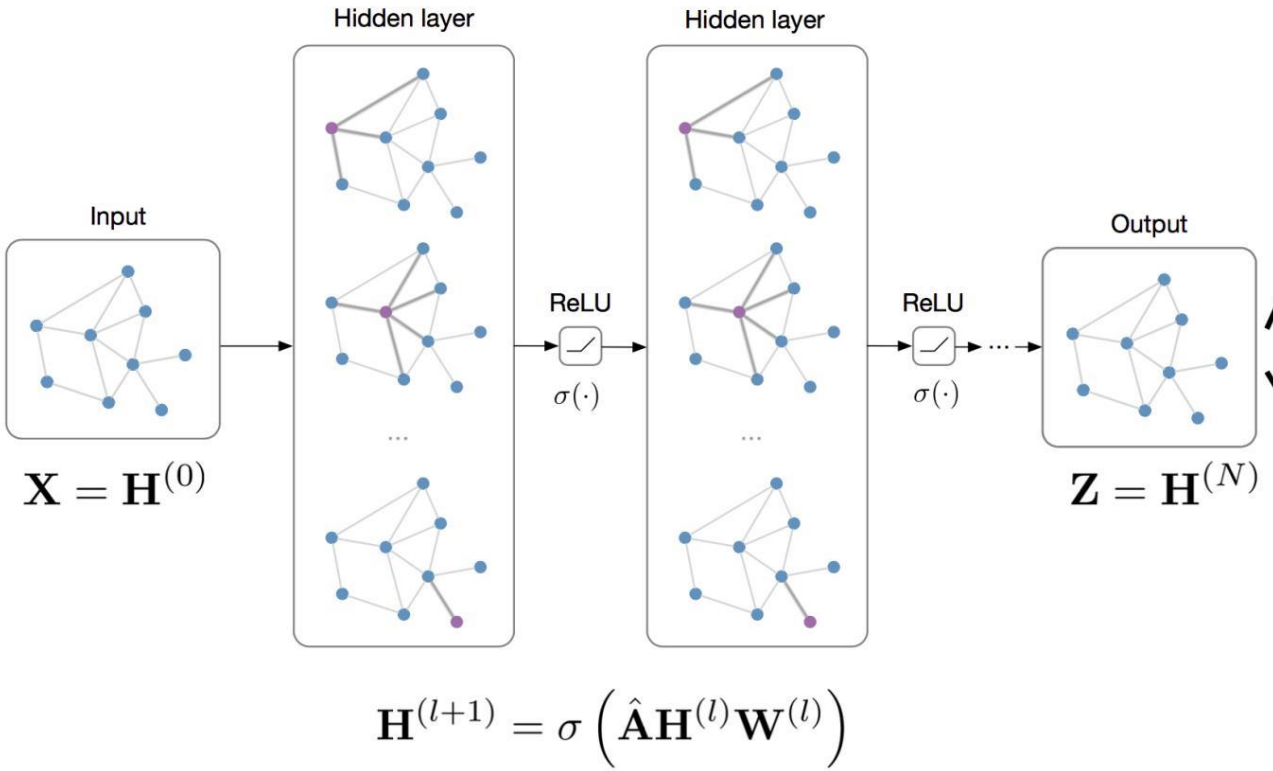
$$E = (e_1, \ldots, e_l) \longrightarrow \tilde{E} = (e_1, \ldots, e_l, e_{l+1}, \ldots, e_{l+m})$$

Applications: recommendation system, knowledge graph mining, etc



GCN: link prediction

Task: given a graph G with $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\hat{\mathbf{A}}$, predict the potential edges of G



Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$\mathrm{softmax}(\sum_n \mathbf{z_n})$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

Objective function: $\mathcal{L} = -\sum_{(i,j)\in\Omega} A_{ij} \ln \sigma(\mathbf{z}_i^\top \mathbf{z}_j)$

GCN: link prediction

Experiments: link prediction task in citation networks
Datasets: Cora, Citeseer, and Pubmed
Evaluation metrics: AUC and AP

Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

Table from: Kipf and Welling. Variational Graph Auto-Encoders. 2016.

1 Introduction

2 Graph Convolutional Network (GCN)
- Architecture of GCN
- Applications of GCN

3 Other GNNs
- GraphSAGE
- GAT

4 Conclusions

GraphSAGE (optional)

- Limitations of GCN
  - Require that all nodes are presented in the training stage
  - Do transductive learning but not inductive learning

GraphSAGE (optional)

- Limitations of GCN
  - Require that all nodes are presented in the training stage
  - Do transductive learning but not inductive learning

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Hamilton et al. Inductive Representation Learning on Large Graphs. NeurIPS 2017.

Graph Attention Network (GAT)

- Self-attention: $e_{ij} = a\left(\mathbf{W}\mathbf{h}_i, \mathbf{W}\mathbf{h}_j\right)$
  - $\mathbf{h}_i$ and $\mathbf{h}_j$ are the d-dimensional features of nodes i and j
  - $\mathbf{W} \in \mathbb{R}^{d' \times d}$, $a : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}$
  - $\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$, it is a normalized e_{ij}
  - $\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]))}$, $\|$ is the concatenation operation. Here a is a single-layer feedforward neural network.

Graph Attention Network (GAT)

- Self-attention: $e_{ij} = a\left(\mathbf{W}\mathbf{h}_i, \mathbf{W}\mathbf{h}_j\right)$
  - $\mathbf{h}_i$ and $\mathbf{h}_j$ are the d-dimensional features of nodes i and j
  - $\mathbf{W} \in \mathbb{R}^{d' \times d}$, $a : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}$
  - $\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$, it is a normalized e_{ij}
  - $\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]))}$, $\|$ is the concatenation operation. Here a is a single-layer feedforward neural network.

- Compute the next layer

$$\mathbf{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right)$$

or with multi-head attention $\mathbf{h}'_i = \|_{m=1}^{M} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(m)} \mathbf{W}^{(m)} h_j\right)$

Compare GAT with GCN: What are the differences?

Velickovic et al. Graph Attention Networks. ICLR 2018.

Graph Attention Network

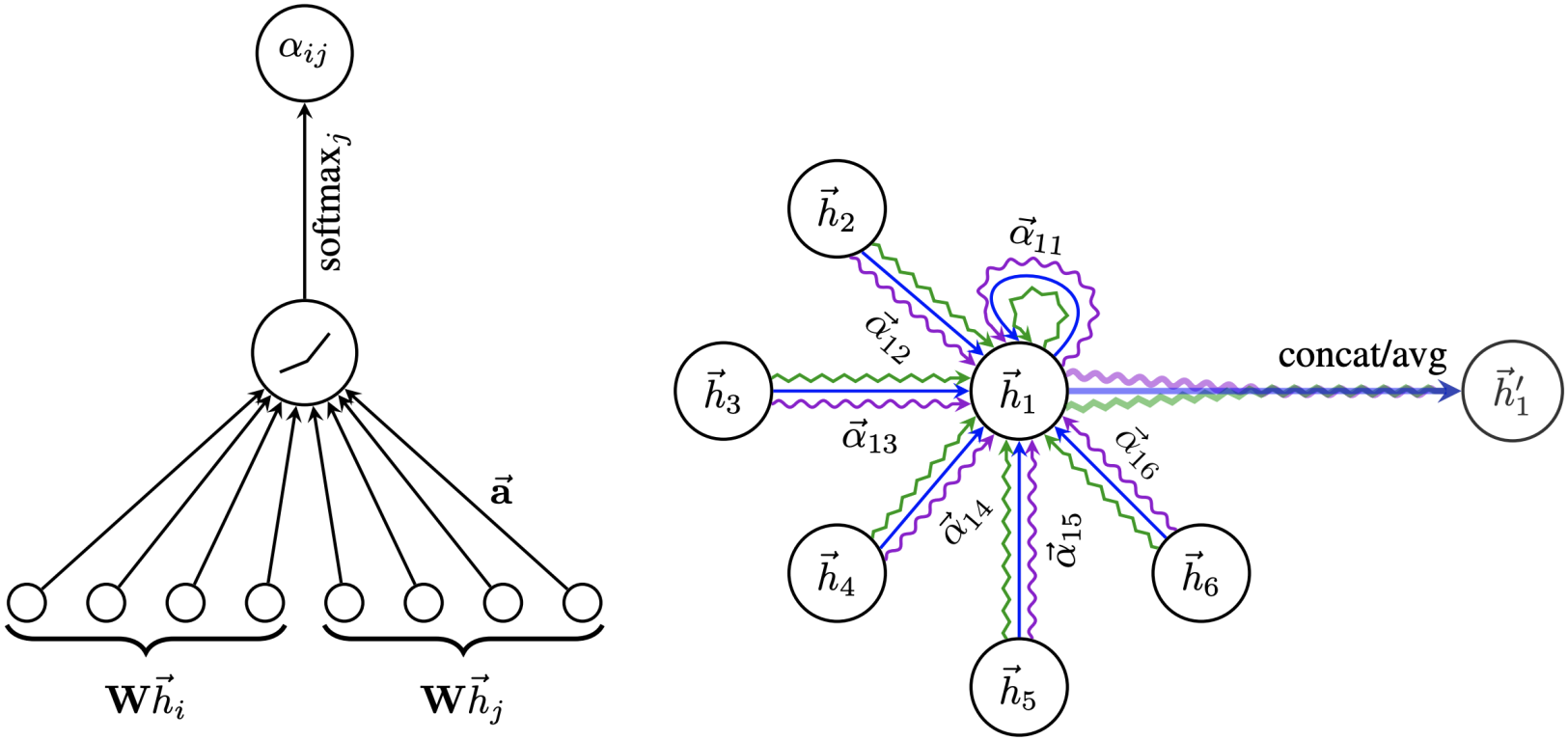


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Velickovic et al. Graph Attention Networks. ICLR 2018.

1. Introduction

2. Graph Convolutional Network (GCN)
   - Architecture of GCN
   - Applications of GCN

3. Other GNNs
   - GraphSAGE
   - GAT

4. Conclusions

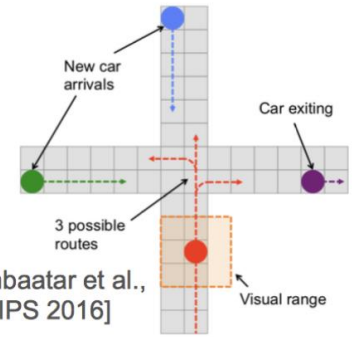
Conclusions

- Deep learning on graphs works and is very effective!
- Exciting area: lots of new applications and extensions (hard to keep up)

Relational reasoning

[Santoro et al., NIPS 2017]

Multi-Agent RL



[Sukhbaatar et al., NIPS 2016]

GCN for recommendation on 16 billion edge graph!

[Leskovec lab, Stanford]

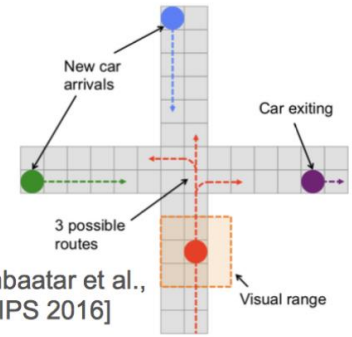
Conclusions

- Deep learning on graphs works and is very effective!
- Exciting area: lots of new applications and extensions (hard to keep up)

Relational reasoning

[Santoro et al., NIPS 2017]

Multi-Agent RL



[Sukhbaatar et al., NIPS 2016]

GCN for recommendation on 16 billion edge graph!

[Leskovec lab, Stanford]

- Open problems
  - Theory
  - Learning on large, evolving data
  - Multi-modal and cross-model learning (e.g., sequence2graph)

Learning outcomes

- Understand the motivation of GCN
- Understand the architectures of GCN and GAT
- Know the applications of GNN
- Be able to conduct some experiments (e.g. node classification) using GNN