AI504: Programming for Artificial Intelligence

Week 13: Graph Neural Networks

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr

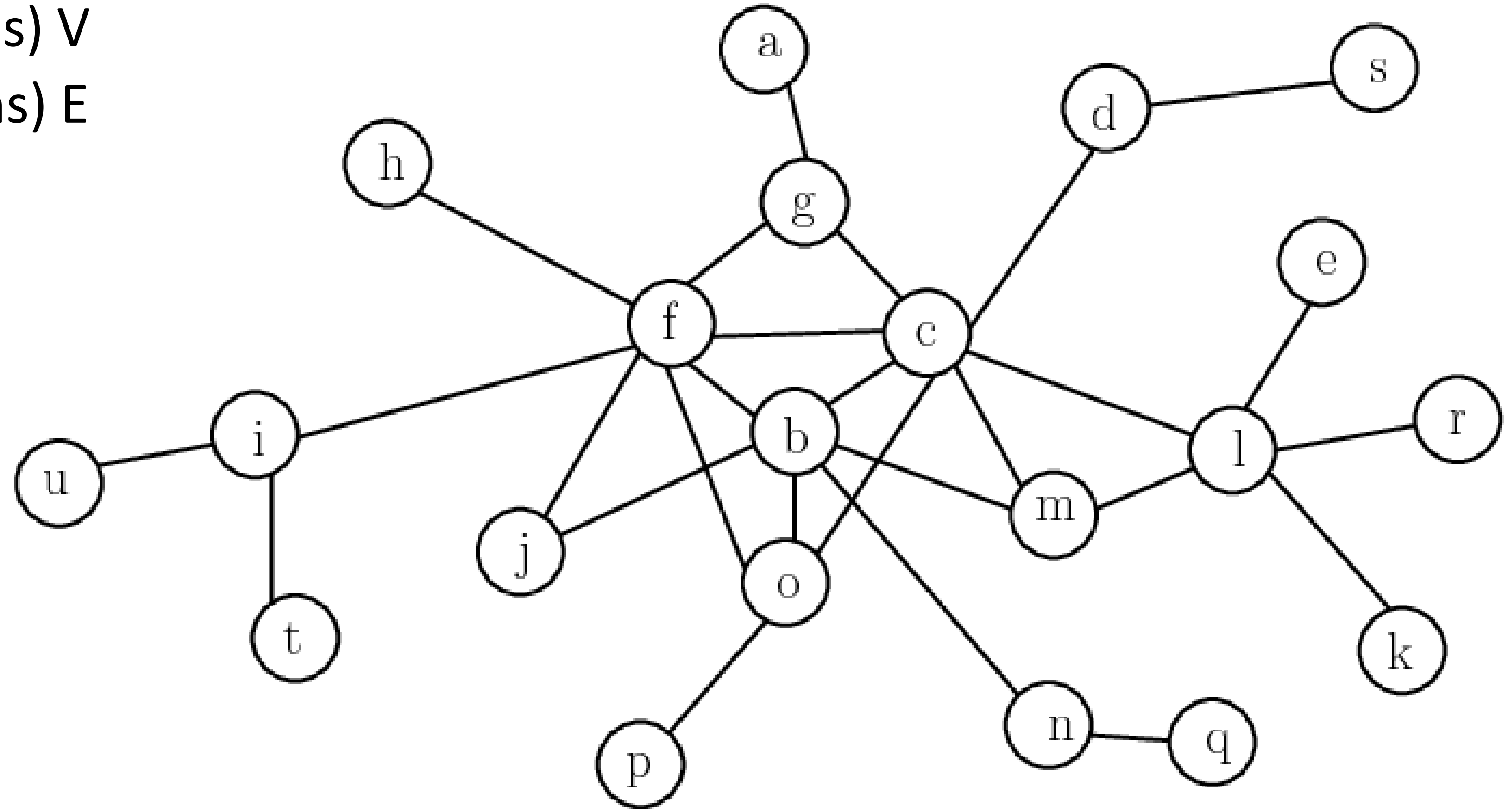
Index

- Graphs
- Graph Convolution
  - Relation with ConvNets
- Graph Neural Networks & Transformer
  - Self-attention VS graph convolution

Graphs

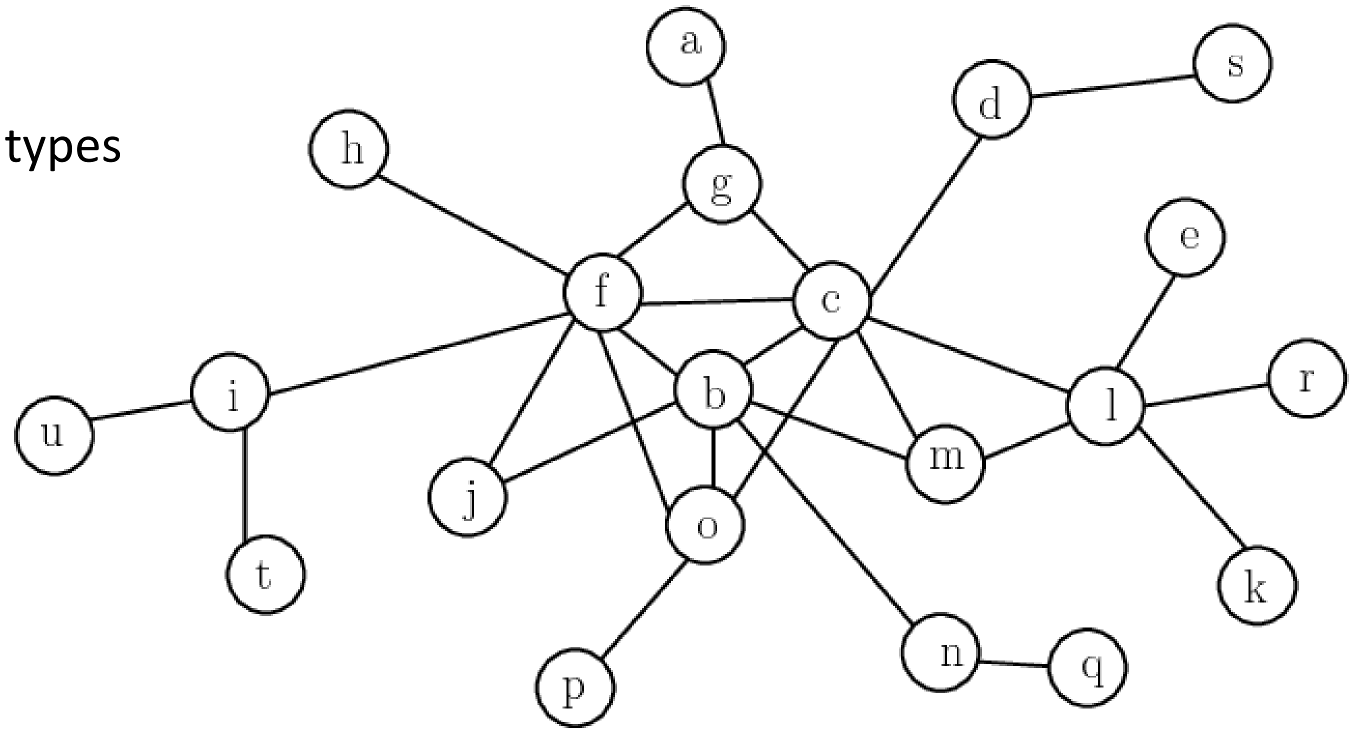
Graph

- Data consists of
  - Nodes (i.e. Vertices) V
  - Edges (i.e. reltaions) E



Graph

- Data consists of
  - Nodes (i.e. Vertices) V
    - Could have node-specific features
  - Edges (i.e. reltaions) E
    - Could be undirected or directed
    - There could be multiple relation types

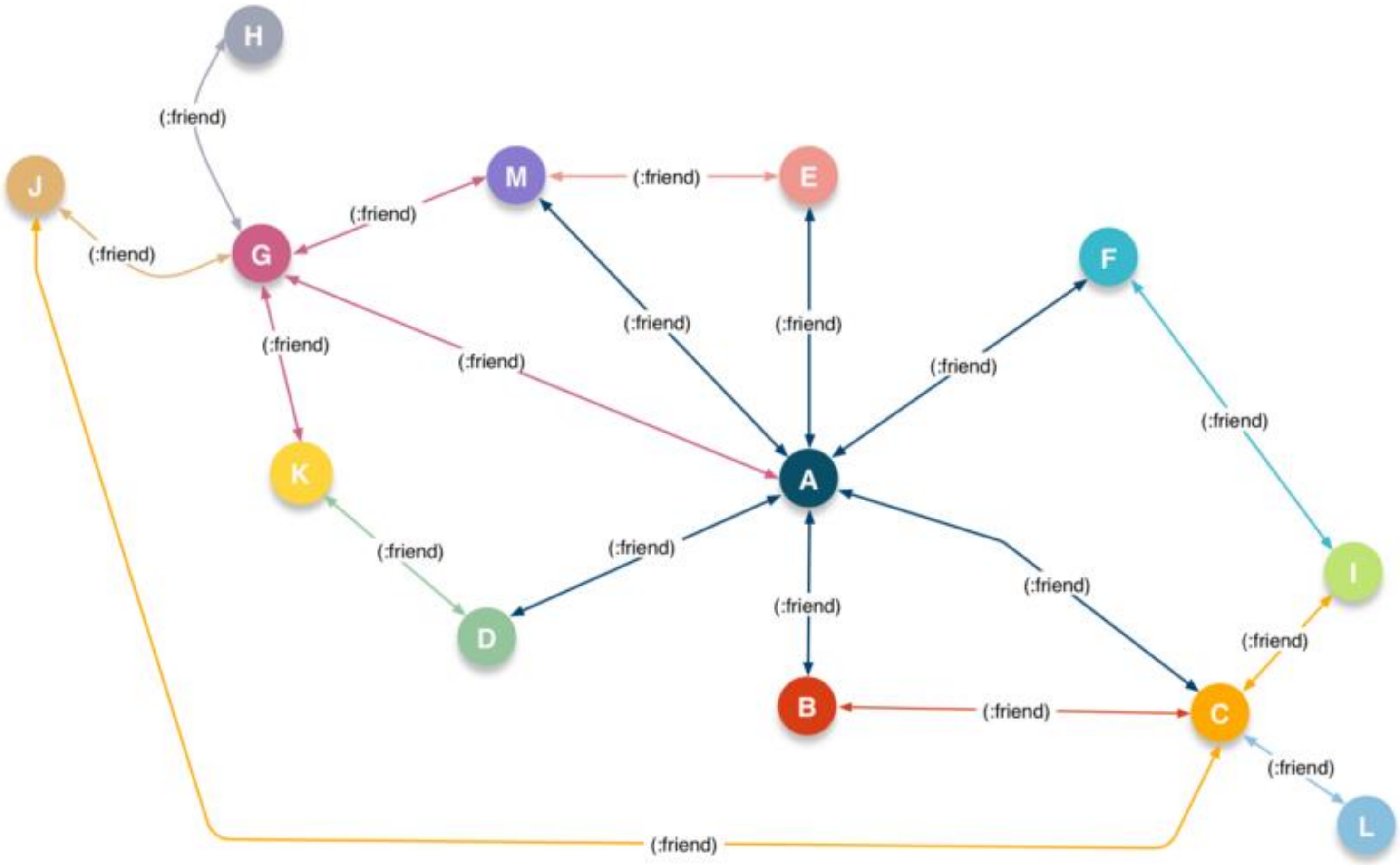


Graph

- Data consists of
  - Nodes (i.e. Vertices) V
  - Edges (i.e. reltaions) E
- Many datasets are graphs
  - Facebook friends
  - Web documents
  - Road networks
  - Chemical compounds
  - Knowledge bases

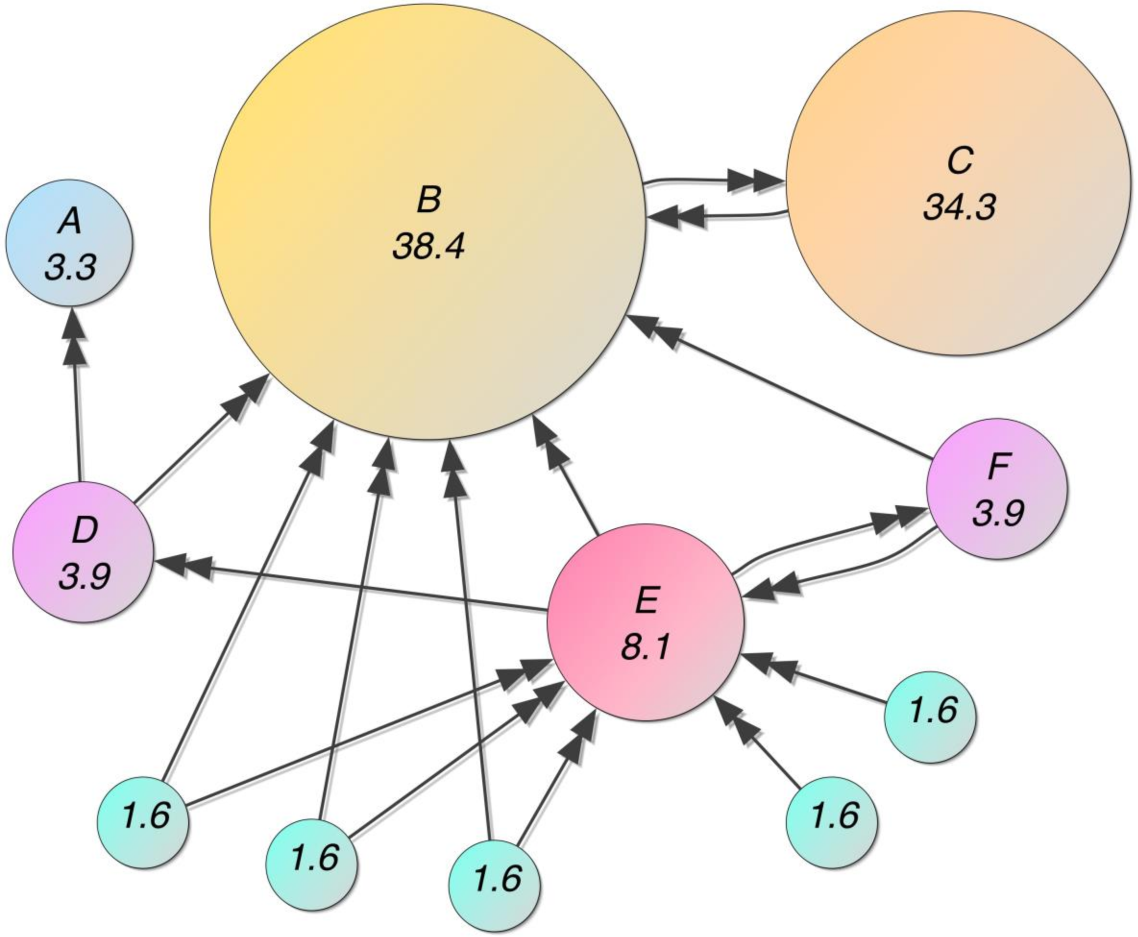
Graph Data Examples

- Friends Network
  - Nodes are unspecific
  - Undirected edges
- Social Network Analysis
  - Hot topic since SNS
  - Can find influencers
  - Can recommend friends



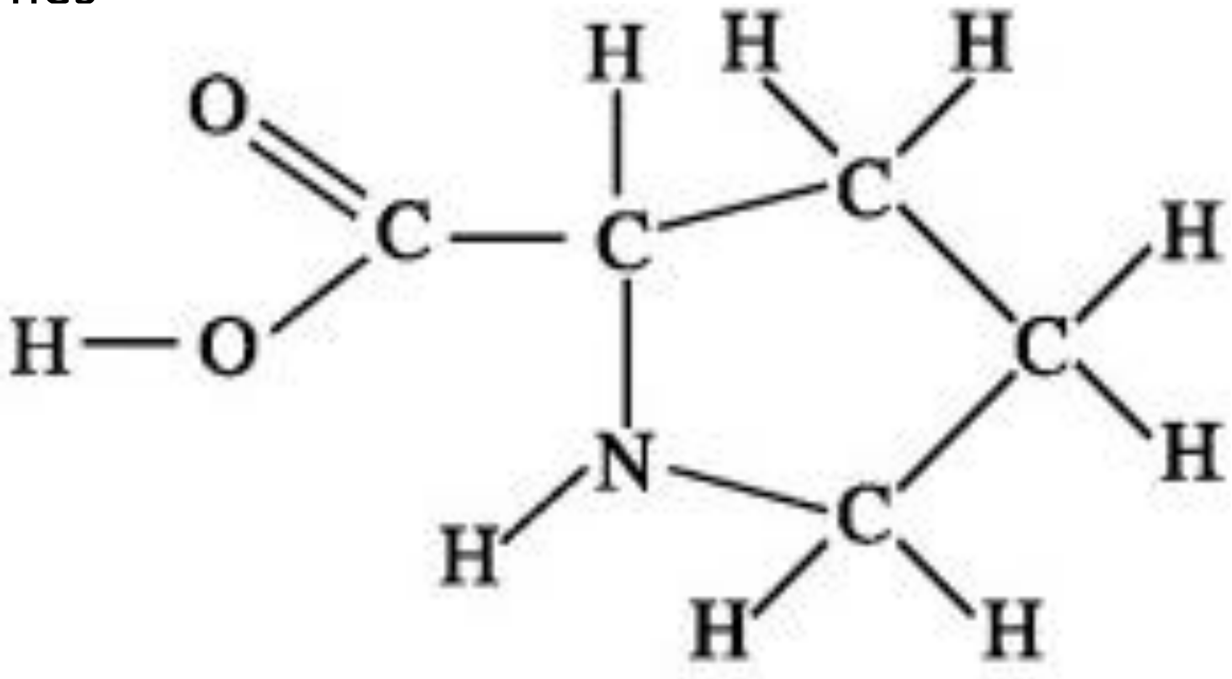
Graph Data Example

- Web documents
- PageRank
  - Made Google Search possible
  - Unspecific node
    - Each webpage is just a node
  - Directed edges
    - Outgoing links are edges
  - Calculated based on random walk.
    - The more incoming links,
      the more valuable a webpage is!



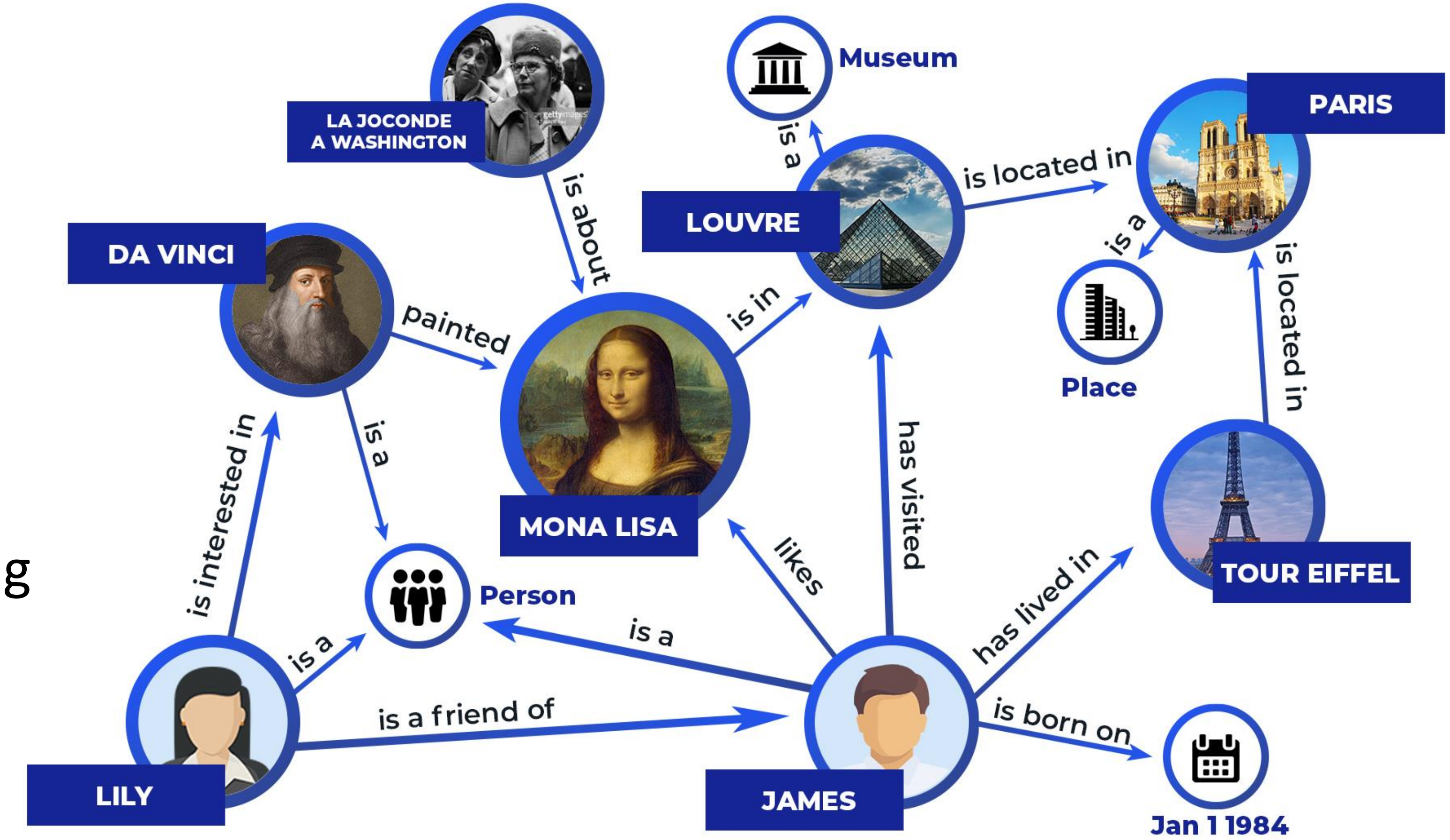
Graph Data Example

- Chemical structures
- Unspecific duplicate nodes
    - Same C (carbon) is used multiple times
- Undirected multi-type edges
    - Single bond, double bond
- Hot topic these days
    - Drug development
    - Toxicity prediction



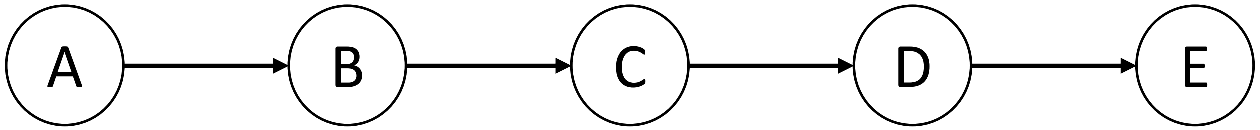
Graph Data Example

- Knowledge Base
- Multiple node types
  - Entity, value
- Directed, multi-type edges
  - is_a, parent_of, located_at
- Structured knowledge
  - Popular topic
  - Knowledge grounded reasoning



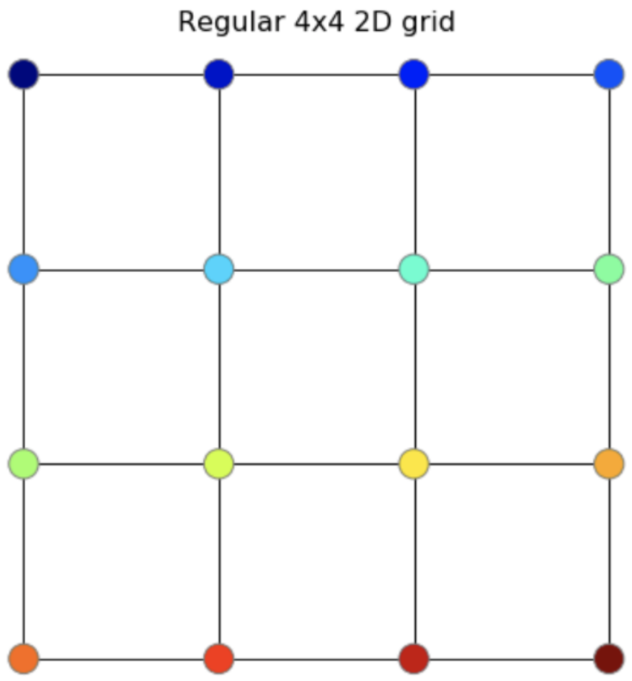
Generally Speaking…

- Everything is a graph
- Sequences are a special case of graphs (i.e. directed chains)



Generally Speaking…

- Everything is a graph
- Images are a special case of a graph (i.e. undirected grids)



Graph Convolution

Graph Representation

- How can we represent graphs?
- Images can be represented by ConvNets
  - 128x128 RGB image ➔ ResNet ➔ 2048-dimensional feature vector
- Text can be represented by RNN / BERT
  - 20 token text ➔ RNN/BERT ➔ 20 contextualized embedding
- Graph?
  - Graph ➔ ? ➔ ?

Graph Representation

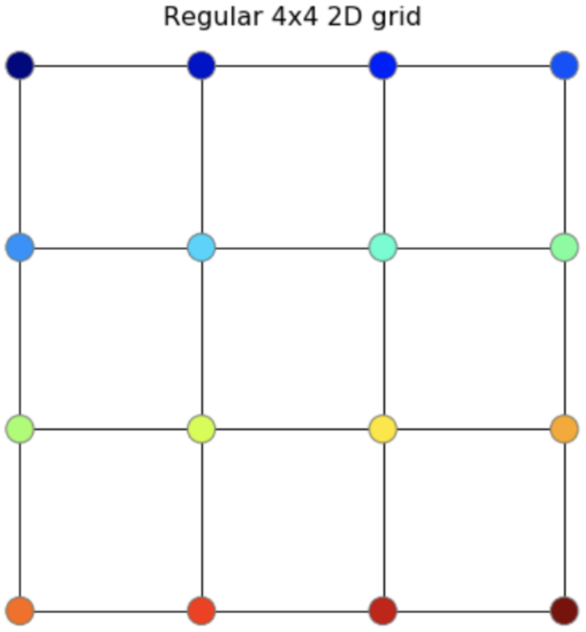
- How can we represent graphs?
- Images can be represented by ConvNets
  - 128x128 RGB image ➔ ResNet ➔ 2048-dimensional feature vector
- Text can be represented by RNN / BERT
  - 20 token text ➔ RNN/BERT ➔ 20 contextualized embedding
- Graph?
  - Graph with |V| nodes and |E| edges ➔ ? ➔ ?

Graph Representation

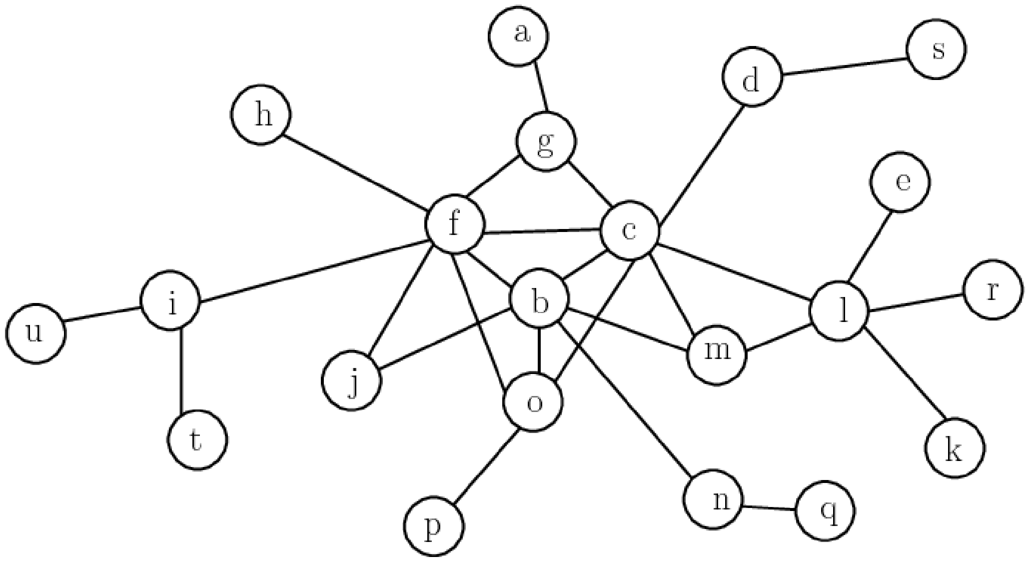
- How can we represent graphs?
- Images can be represented by ConvNets
  - 128x128 RGB image ➔ ResNet ➔ 2048-dimensional feature vector
- Text can be represented by RNN / BERT
  - 20 token text ➔ RNN/BERT ➔ 20 contextualized embedding
- Graph?
  - |V| nodes and |E| edges ➔ ? ➔ |V| embeddings (and |E| embeddings)

Why not ConvNet?

- Convolution filters assume same number of neighbors
  - General graphs assume no such thing…

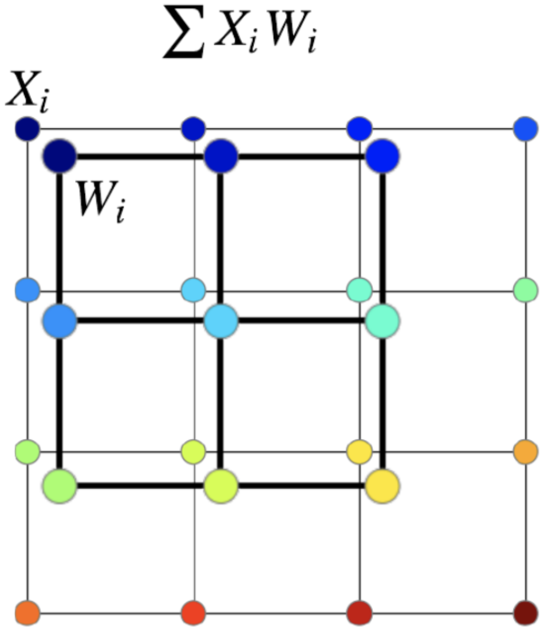


VS



Why not ConvNet?

- Convolution filters assume same number of neighbors
  - General graphs assume no such thing…
- But we can use the core principle of ConvNet filters
  - Aggregate features from the local neighbors



Graph Convolution Principle

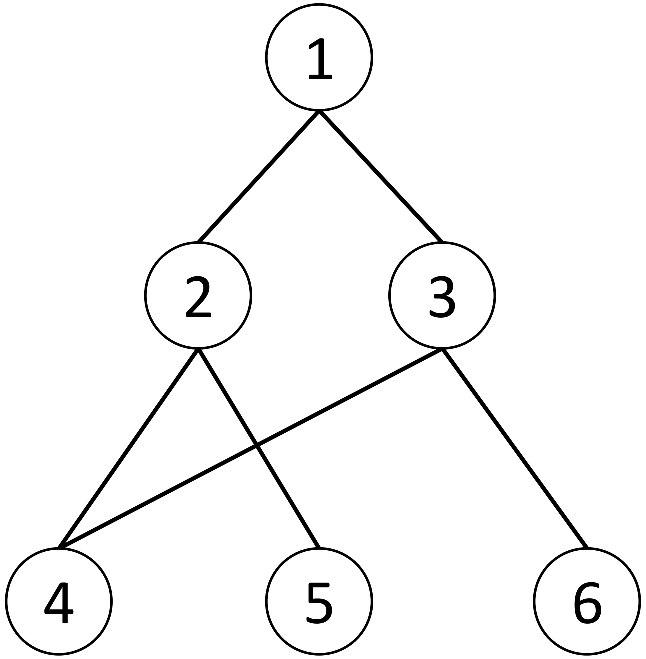
Given a graph G = (V, E),

- At each node v_i, aggregate all neighbors' features
  - $\mathbf{a}_i = \sum_{v_j \in \mathcal{A}_i} f(v_j)$, where $\mathcal{A}_i$: Set of nodes connected to v_i
- Combine v_i's feature with neightbors' features
  - $\mathbf{h}_i = g(f(v_i), \mathbf{a}_i)$
- $\mathbf{h}_i$ ➔ Representation of node v_i

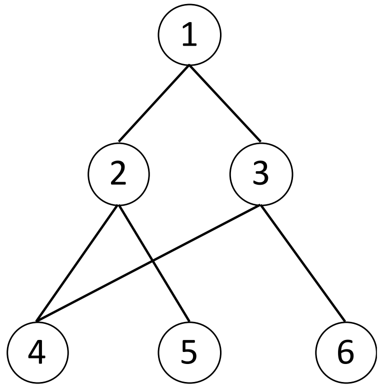
Graph Convolution Equation

- A: Adjacency matrix

0	**1**	**1**	0	0	0
1	0	0	**1**	**1**	0
1	0	0	**1**	0	**1**
0	**1**	**1**	0	0	0
0	**1**	0	0	0	0
0	0	**1**	0	0	0



Graph Convolution Equation



- A: Adjacency matrix
- X: Node index
- W: Node embedding vector

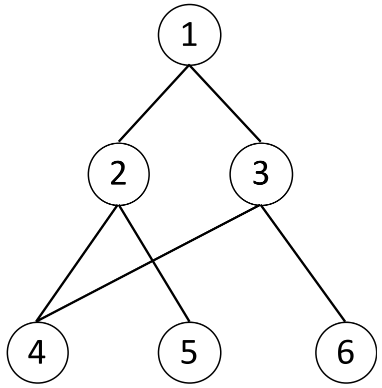
0	**1**	**1**	0	0	0
1	0	0	**1**	**1**	0
1	0	0	**1**	0	**1**
0	**1**	**1**	0	0	0
0	**1**	0	0	0	0
0	0	**1**	0	0	0

A

$f(v_1)$
$f(v_2)$
$f(v_3)$
$f(v_4)$
$f(v_5)$
$f(v_6)$

XW

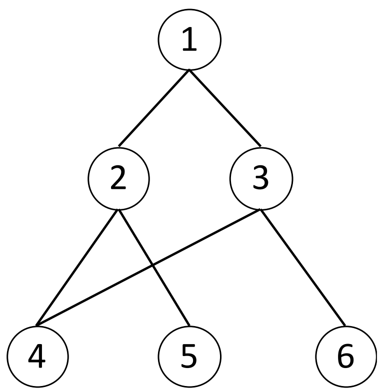
Graph Convolution Equation



- AXW

$$\underset{\mathbf{A}}{\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}} \times \underset{\mathbf{XW}}{\begin{bmatrix} f(v_1) \\ f(v_2) \\ f(v_3) \\ f(v_4) \\ f(v_5) \\ f(v_6) \end{bmatrix}} = \underset{\mathbf{AXW}}{\begin{bmatrix} f(v_2) + f(v_3) \\ f(v_1) + f(v_4) + f(v_5) \\ f(v_1) + f(v_4) + f(v_6) \\ f(v_2) + f(v_3) \\ f(v_2) \\ f(v_3) \end{bmatrix}}$$

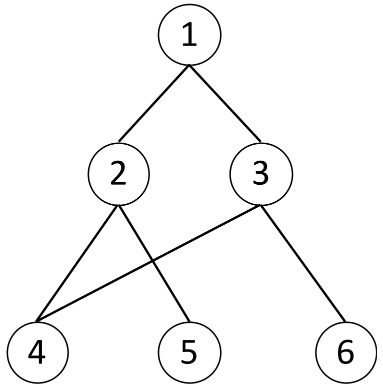
Graph Convolution Equation



- AXW ➔ Is this the node representations H?

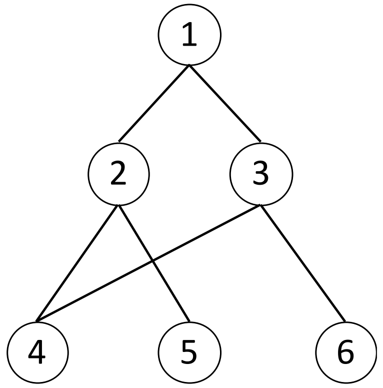
$$
\underbrace{\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \times \underbrace{\begin{bmatrix} f(v_1) \\ f(v_2) \\ f(v_3) \\ f(v_4) \\ f(v_5) \\ f(v_6) \end{bmatrix}}_{\mathbf{XW}} = \underbrace{\begin{bmatrix} f(v_2) + f(v_3) \\ f(v_1) + f(v_4) + f(v_5) \\ f(v_1) + f(v_4) + f(v_6) \\ f(v_2) + f(v_3) \\ f(v_2) \\ f(v_3) \end{bmatrix}}_{\mathbf{AXW}}
$$

Graph Convolution Equation



- No, AXW is just neighbor aggregation.
- We need the combination step!

$$
\underset{\mathbf{A}}{\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}}
\times
\underset{\mathbf{XW}}{\begin{bmatrix} f(v_1) \\ f(v_2) \\ f(v_3) \\ f(v_4) \\ f(v_5) \\ f(v_6) \end{bmatrix}}
=
\underset{\mathbf{AXW}}{\begin{bmatrix} f(v_2) + f(v_3) \\ f(v_1) + f(v_4) + f(v_5) \\ f(v_1) + f(v_4) + f(v_6) \\ f(v_2) + f(v_3) \\ f(v_2) \\ f(v_3) \end{bmatrix}}
$$

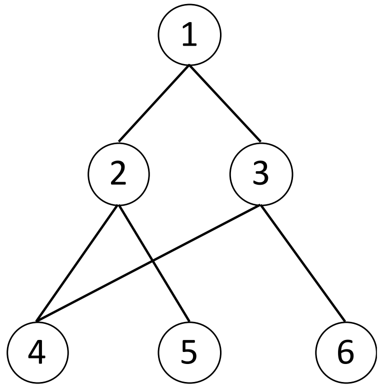


Graph Convolution Equation

- New A’ = A + I
  - I: Identity matrix

$$
\underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{A' = A + I}}
\times
\underbrace{\begin{bmatrix} f(v_1) \\ f(v_2) \\ f(v_3) \\ f(v_4) \\ f(v_5) \\ f(v_6) \end{bmatrix}}_{\mathbf{XW}}
=
\underbrace{\begin{bmatrix} f(v_1) + f(v_2) + f(v_3) \\ f(v_1) + f(v_2) + f(v_4) + f(v_5) \\ f(v_1) + f(v_3) + f(v_4) + f(v_6) \\ f(v_2) + f(v_3) + f(v_4) \\ f(v_2) + f(v_5) \\ f(v_3) + f(v_6) \end{bmatrix}}_{\mathbf{A'XW}}
$$

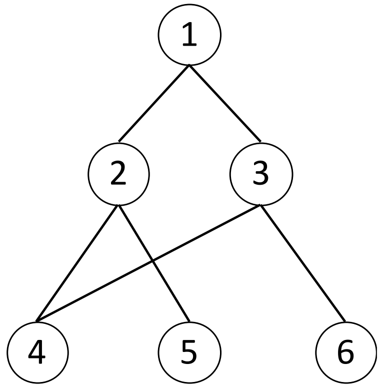
Graph Convolution Equation



- A'XW performs neighbor aggregation and combination
  - Combination function *g* is just simple summation.
- A'XW can be node representations H!

$$\underset{\mathbf{A' = A + I}}{\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} \end{bmatrix}} \times \underset{\mathbf{XW}}{\begin{bmatrix} f(v_1) \\ f(v_2) \\ f(v_3) \\ f(v_4) \\ f(v_5) \\ f(v_6) \end{bmatrix}} = \underset{\mathbf{A'XW}}{\begin{bmatrix} h_1 = f(v_1) + f(v_2) + f(v_3) \\ h_2 = f(v_1) + f(v_2) + f(v_4) + f(v_5) \\ h_3 = f(v_1) + f(v_3) + f(v_4) + f(v_6) \\ h_4 = f(v_2) + f(v_3) + f(v_4) \\ h_5 = f(v_2) + f(v_5) \\ h_6 = f(v_3) + f(v_6) \end{bmatrix}}$$

Graph Convolution Equation



- But there is another problem:
- Scales of node features differ by the number of neighbors!
  - h_2 can be twice as large as h_5!

1	**1**	**1**	0	0	0
1	**1**	0	**1**	**1**	0
1	0	**1**	**1**	0	**1**
0	**1**	**1**	**1**	0	0
0	**1**	0	0	**1**	0
0	0	**1**	0	0	**1**

A’ = A + I

$\times$

$f(v_1)$
$f(v_2)$
$f(v_3)$
$f(v_4)$
$f(v_5)$
$f(v_6)$

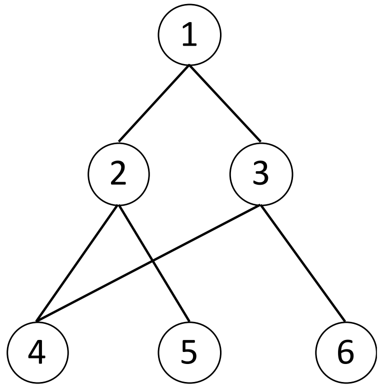
XW

$=$

$h_1 = f(v_1) + f(v_2) + f(v_3)$
$h_2 = f(v_1) + f(v_2) + f(v_4) + f(v_5)$
$h_3 = f(v_1) + f(v_3) + f(v_4) + f(v_6)$
$h_4 = f(v_2) + f(v_3) + f(v_4)$
$h_5 = f(v_2) + f(v_5)$
$h_6 = f(v_3) + f(v_6)$

A’XW

Graph Convolution Equation



- D: Degree matrix
  - Diagonal matrix where $d_{i,I}$ = number of edges
- $D^{-1}A'$: Normalized adjacency matrix

.33	**.33**	**.33**	0	0	0
.25	**.25**	0	**.25**	**.25**	0
.25	0	**.25**	**.25**	0	**.25**
0	**.33**	**.33**	**.33**	0	0
0	**.5**	0	0	**.5**	0
0	0	**.5**	0	0	**.5**

$D^{-1}A'$

×

$f(v_1)$
$f(v_2)$
$f(v_3)$
$f(v_4)$
$f(v_5)$
$f(v_6)$

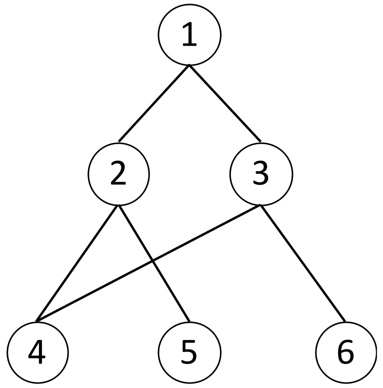
XW

=

$h_1 = 0.33 * (f(v_1) + f(v_2) + \dots)$
$h_2 = 0.25 * (f(v_1) + f(v_2) + \dots)$
$h_3 = 0.25 * (f(v_1) + f(v_3) + \dots)$
$h_4 = 0.33 * (f(v_2) + f(v_3) + \dots)$
$h_5 = 0.5 * (f(v_2) + f(v_5))$
$h_6 = 0.5 * (f(v_3) + f(v_6))$

$D^{-1}A'XW$

Graph Convolution Equation



- One more thing:
  - $D^{-1}A'XW$ ➔ All linear operations
- Need non-linearity

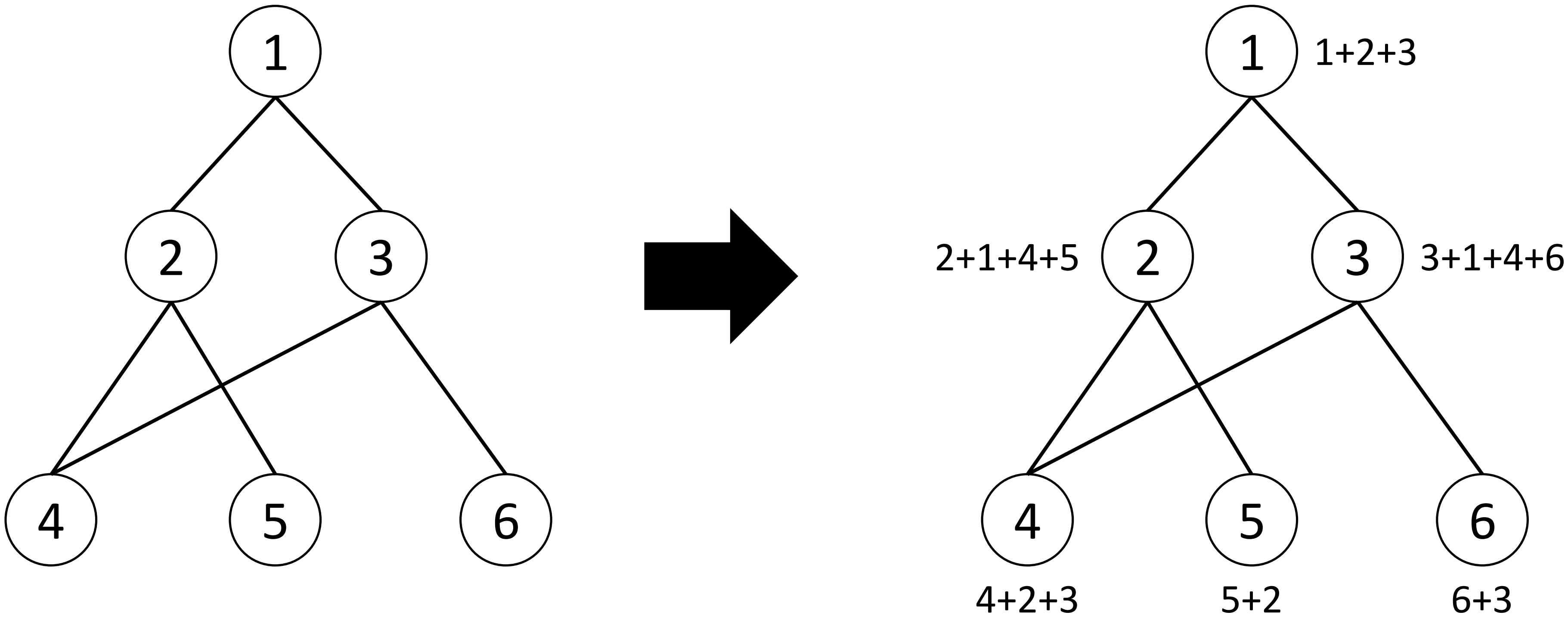
$$\sigma\left(\underbrace{\begin{bmatrix} .33 & .33 & .33 & 0 & 0 & 0 \\ .25 & .25 & 0 & .25 & .25 & 0 \\ .25 & 0 & .25 & .25 & 0 & .25 \\ 0 & .33 & .33 & .33 & 0 & 0 \\ 0 & .5 & 0 & 0 & .5 & 0 \\ 0 & 0 & .5 & 0 & 0 & .5 \end{bmatrix}}_{\mathbf{D^{-1}A'}} \times \underbrace{\begin{bmatrix} f(v_1) \\ f(v_2) \\ f(v_3) \\ f(v_4) \\ f(v_5) \\ f(v_6) \end{bmatrix}}_{\mathbf{XW}}\right) = \sigma\left(\underbrace{\begin{bmatrix} h_1 = 0.33 * (f(v_1) + f(v_2) + \ldots) \\ h_2 = 0.25 * (f(v_1) + f(v_2) + \ldots) \\ h_3 = 0.25 * (f(v_1) + f(v_3) + \ldots) \\ h_4 = 0.33 * (f(v_2) + f(v_3) + \ldots) \\ h_5 = 0.5 * (f(v_2) + f(v_5)) \\ h_6 = 0.5 * (f(v_3) + f(v_6)) \end{bmatrix}}_{\mathbf{D^{-1}A'XW}}\right)$$

Graph Convolution Equation

- $\boldsymbol{H} = \sigma(\boldsymbol{D}^{-1}\boldsymbol{A}'\boldsymbol{X}\boldsymbol{W})$
  - $\boldsymbol{A}' = \boldsymbol{A} + \boldsymbol{I}$
  - $\boldsymbol{D}$ = Degree matrix
  - σ = Non-linear activation
  - $\boldsymbol{W}$ = Learnable parameters

Graph Convolution Equation

- $\boldsymbol{H}^{(\mathbf{1})} = \sigma(\boldsymbol{D}^{-1}\boldsymbol{A}'\boldsymbol{X}\boldsymbol{W})$
- This is aggregating neighbors just 1-hop away



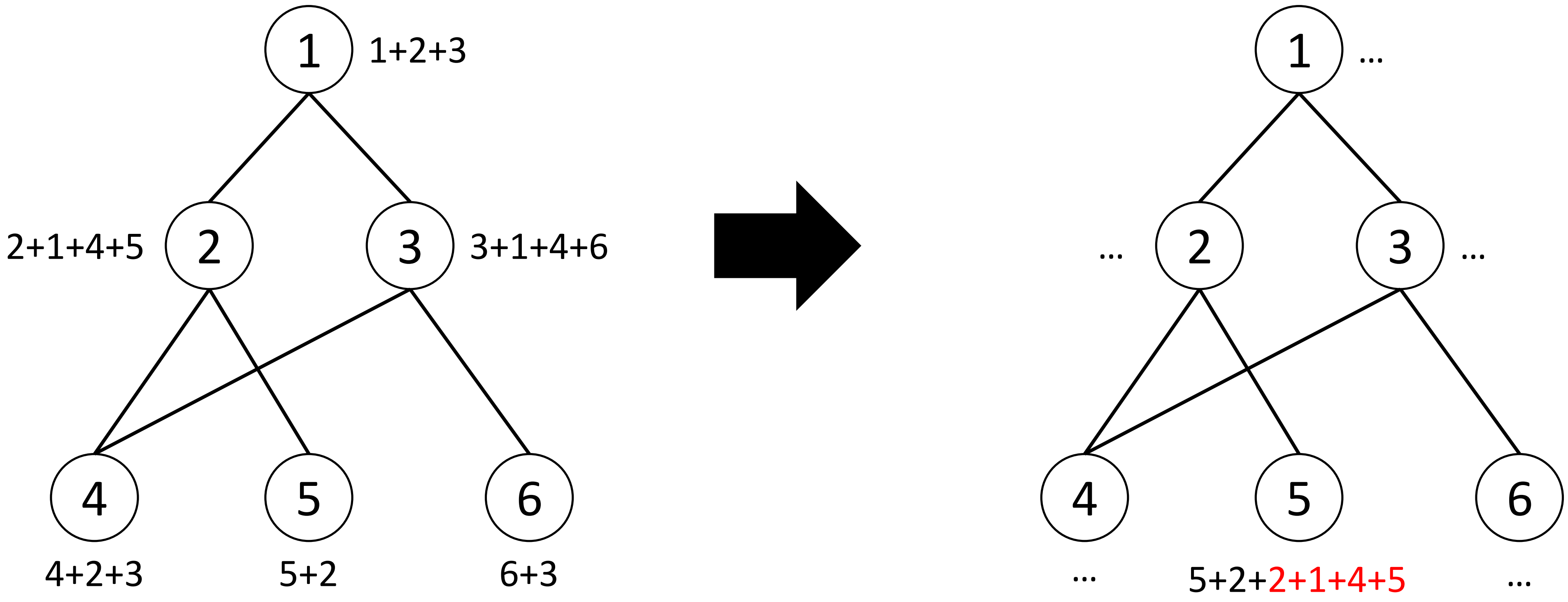
Graph Convolution Equation

- $\boldsymbol{H}^{(\mathbf{1})} = \sigma(\boldsymbol{D}^{-1}\boldsymbol{A}'\boldsymbol{X}\boldsymbol{W})$
- This is aggregating neighbors just 1-hop away
- How do we aggregate neighbors 2-hops away?

Graph Convolution Equation

- How do we aggregate neighbors 2-hops away?

➔ $\boldsymbol{H}^{(2)} = \sigma\left(\boldsymbol{D}^{-1}\boldsymbol{A}'\boldsymbol{H}^{(1)}\boldsymbol{W}^{(2)}\right)$



Graph Convolution Equation

- How do we aggregate neighbors k-hops away?

➔ $\boldsymbol{H}^{(\boldsymbol{k})} = \sigma\left(\boldsymbol{D}^{-1}\boldsymbol{A}'\boldsymbol{H}^{(\boldsymbol{k}-\mathbf{1})}\boldsymbol{W}^{(\boldsymbol{k})}\right)$

Graph Convolution Variations

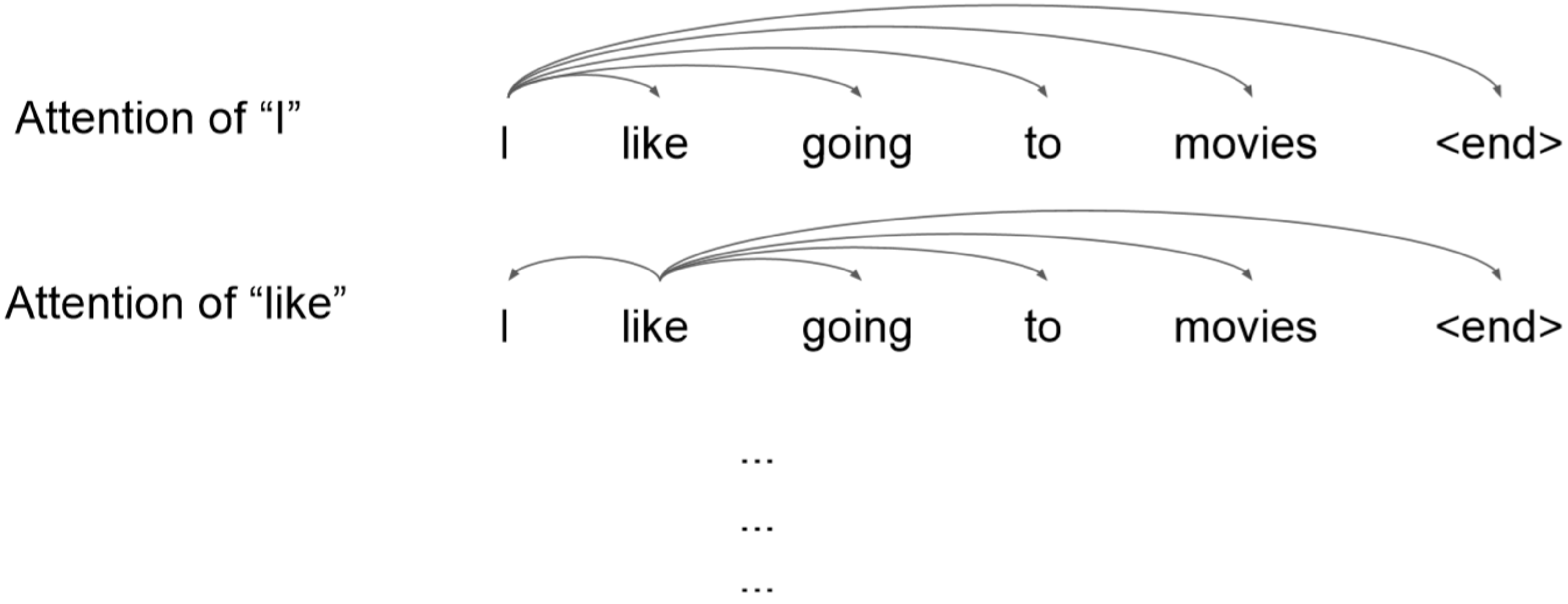
- Different normalization
  - $\boldsymbol{H}^{(\boldsymbol{k})} = \sigma\left(\boldsymbol{D}^{-1/2}\boldsymbol{A}'\boldsymbol{D}^{-1/2}\boldsymbol{H}^{(\boldsymbol{k-1})}\boldsymbol{W}^{(\boldsymbol{k})}\right)$
  - Motivated by spectral graph convolution
    - Whole theory regarding graph laplacian…
- Different Combination step
  - Instead of summation $g(f(v_i), \mathbf{a}_i)$ = $f(v_i) + \mathbf{a}_i$,
  - Use linear layer $g(f(v_i), \mathbf{a}_i)$ = W · $[f(v_i); \mathbf{a}_i]$
- Nodes become RNNs
  - More sophisticated way to accumulate N-hop information
- Many more variations ➔ Called Graph Neural Networks (GNN)

Semi-Supervised Classification with Graph Convolutional Networks (https://arxiv.org/pdf/1609.02907.pdf)
How Powerful are Graph Neural Networks? (https://openreview.net/pdf?id=ryGs6iA5Km)

GNN & Transformer

Attention is All You Need

- Vaswani et al. 2017
- Let’s use only attentions to handle sequences.



Self-Attention

- $Attention(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = Softmax\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right)\boldsymbol{V}$

0.5	0.1	0.0	0.2	0.2	0.0
0.2	0.6	0.0	0.0	0.1	0.0
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

$Softmax\left(\frac{\boldsymbol{QK}^T}{\sqrt{d}}\right)$

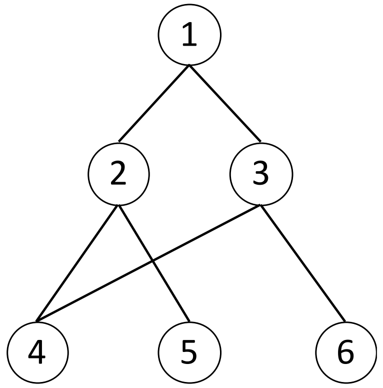
I
like
going
to
movies
<end>

$\boldsymbol{V}$

0.5*I + 0.1*like + 0.2*to + 0.2* movies
0.2*I + 0.6*like + 0.1*movies
...
...
...
...

$Attention(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})$

Graph Convolution



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

.33	**.33**	**.33**	0	0	0
.25	**.25**	0	**.25**	**.25**	0
.25	0	**.25**	**.25**	0	**.25**
0	**.33**	**.33**	**.33**	0	0
0	**.5**	0	0	**.5**	0
0	0	**.5**	0	0	**.5**

$D^{-1}A'$

$\times$

$f(v_1)$
$f(v_2)$
$f(v_3)$
$f(v_4)$
$f(v_5)$
$f(v_6)$

XW

$=$

$h_1 = 0.33 * (f(v_1) + f(v_2) + \ldots)$
$h_2 = 0.25 * (f(v_1) + f(v_2) + \ldots)$
$h_3 = 0.25 * (f(v_1) + f(v_3) + \ldots)$
$h_4 = 0.33 * (f(v_2) + f(v_3) + \ldots)$
$h_5 = 0.5 * (f(v_2) + f(v_5))$
$h_6 = 0.5 * (f(v_3) + f(v_6))$

$D^{-1}A'XW$

Self-Attention VS Graph Convolution

- Self-attention
  - Don’t know graph structure
  - Assum (implicitly) fully-connected graph
    - Learn edge weights during training
  - Learn node embeddings in a data-driven fashion
- Graph convolution
  - Prior knowledge on graph structure
  - Learn node embeddings based on the fixed adjacency matrix

Transformer & Graph Networks

- Graph Networks
  - $\mathbf{C}^{(j)} = \mathrm{MLP}^{(j)}(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{C}^{(j-1)}\mathbf{W}^{(j)})$
- Transformer
  - $\mathbf{C}^{(j)} = \mathrm{MLP}^{(j)}(\mathrm{softmax}(\frac{\mathbf{Q}^{(j)}\mathbf{K}^{(j)\top}}{\sqrt{d}})\mathbf{V}^{(j)})$

$$\mathbf{Q}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_Q^{(j)}, \quad \mathbf{K}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_K^{(j)}, \quad \mathbf{V}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_V^{(j)}$$

Transformer & Graph Networks

- Graph Networks
  - $\mathbf{C}^{(j)} = \mathrm{MLP}^{(j)}(\boxed{\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}}\mathbf{C}^{(j-1)}\mathbf{W}^{(j)})$

Adjacency Matrix ⟺ Self-Attention

- Transformer
  - $\mathbf{C}^{(j)} = \mathrm{MLP}^{(j)}(\boxed{\mathrm{softmax}(\frac{\mathbf{Q}^{(j)}\mathbf{K}^{(j)\top}}{\sqrt{d}})}\mathbf{V}^{(j)})$

$$\mathbf{Q}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_Q^{(j)}, \quad \mathbf{K}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_K^{(j)}, \quad \mathbf{V}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_V^{(j)}$$

Transformer & Graph Networks

- Graph Networks
  - $\mathbf{C}^{(j)} = \mathrm{MLP}^{(j)}(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\boxed{\mathbf{C}^{(j-1)}\mathbf{W}^{(j)}})$

Node Embedding $\Leftrightarrow$ Token Embedding

- Transformer
  - $\mathbf{C}^{(j)} = \mathrm{MLP}^{(j)}(\mathrm{softmax}(\frac{\mathbf{Q}^{(j)}\mathbf{K}^{(j)\top}}{\sqrt{d}})\mathbf{V}^{(j)})$

$$\mathbf{Q}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_Q^{(j)}, \quad \mathbf{K}^{(j)} = \mathbf{C}^{(j-1)}\mathbf{W}_K^{(j)}, \quad \mathbf{V}^{(j)} = \boxed{\mathbf{C}^{(j-1)}\mathbf{W}_V^{(j)}}$$

Transformer instead of GNN?

- Learn the edge weights with attention
  - Zero-out the probability of non-connected edges
    - Mask QK/sqrt(d) with negative infinity matrix
  - Graph Attention Network (https://arxiv.org/pdf/1710.10903.pdf)

- Learning the structure of graphs with attention
  - Use self-attention to learn the underlying graph structure
  - Start with a prior knowledge driven adjacency matrix, then gradually evolve with self-attention
    - Prior knowledge: conditional probability between pairs of nodes
  - Graph Convolutional Transformer (https://arxiv.org/pdf/1906.04716.pdf)

AI504: Programming for Artificial Intelligence

Week 13: Graph Neural Networks

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr