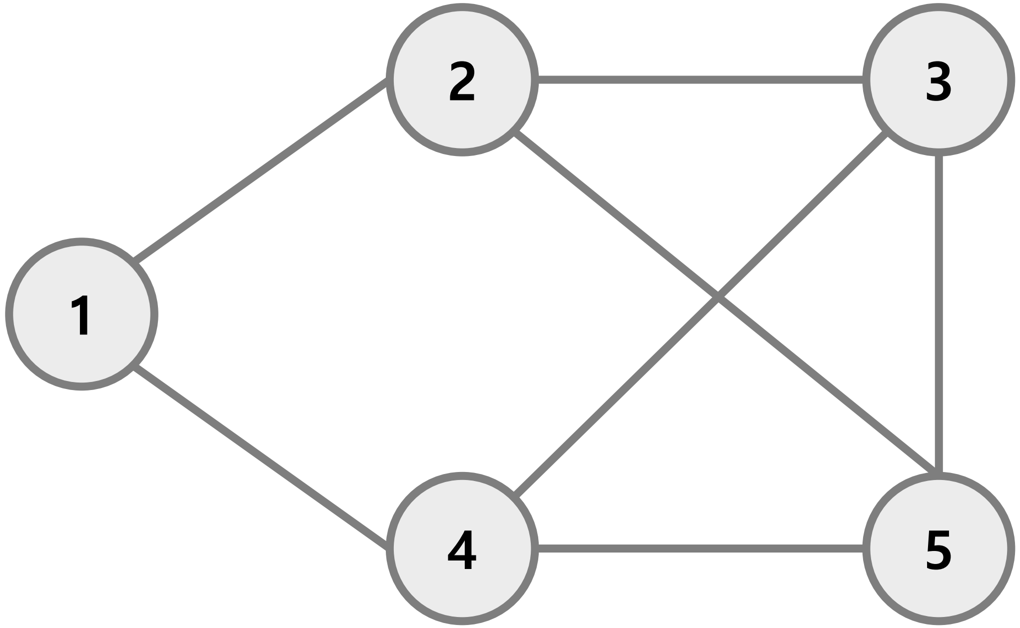
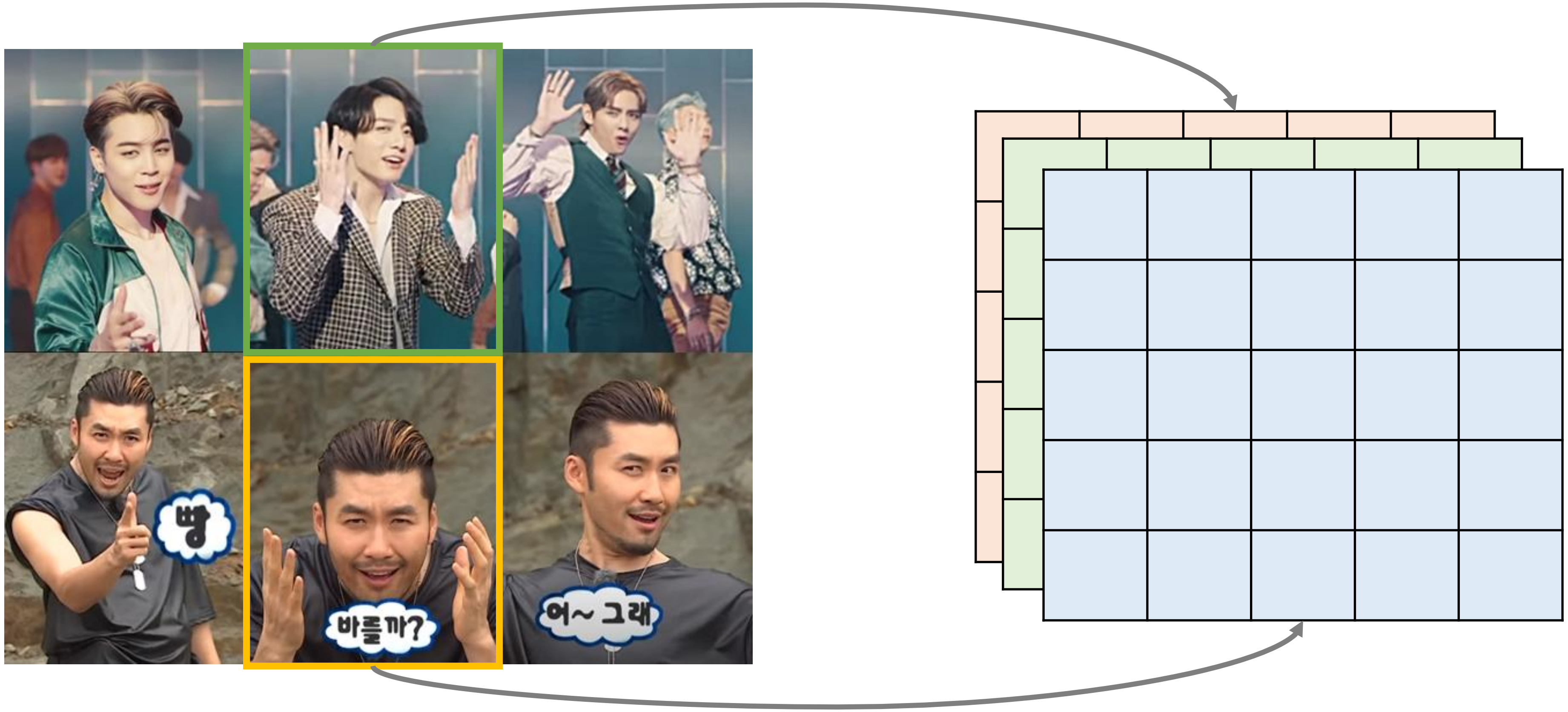
- Graph Data Structure
  - Node = Vertex
    - ✓ Represent elements of a system
  - Edge
    - ✓ Relationship or interaction between nodes



03 | Graph Neural Networks

Data Mining Quality Analytics

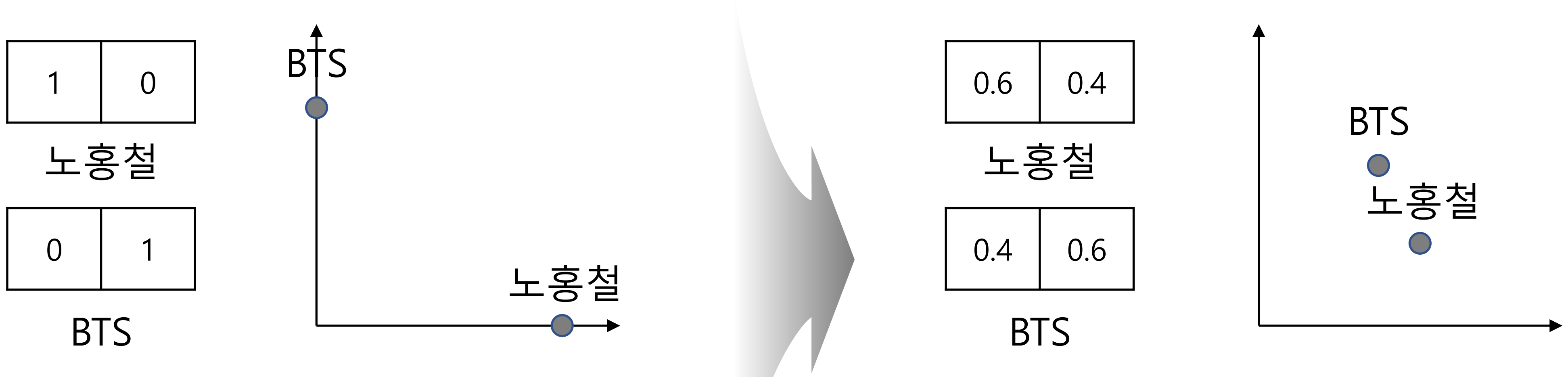
- Graph Data Structure
  - Image data: Euclidean space



[3 X W X H]
dimension

03 | Graph Neural Networks

- Graph Data Structure
  - Text data: Euclidean space



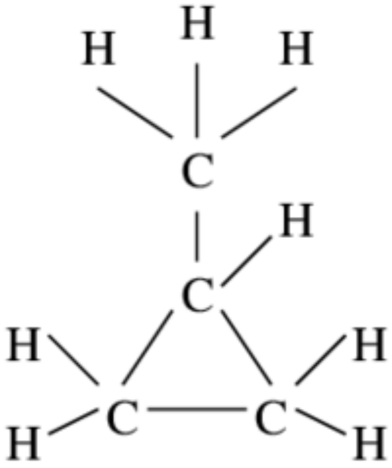
One-hot encoding

Distributed representation

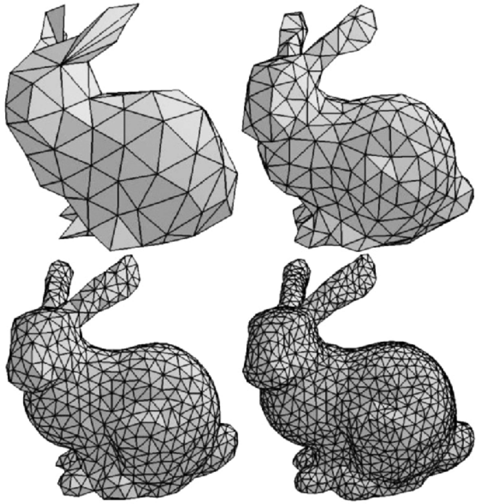
Data Mining Quality Analytics

- Graph Data Structure
  - Graph data: Non-Euclidean Space

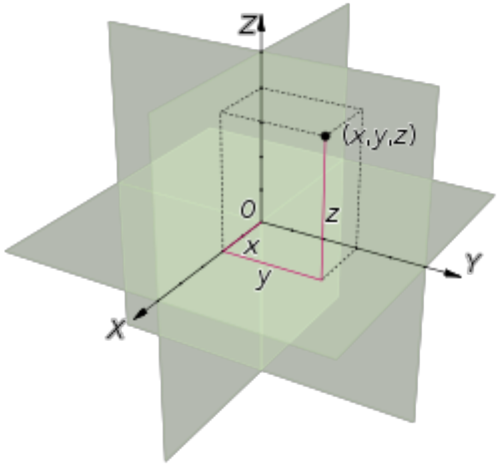
Social Network



Molecular Graph

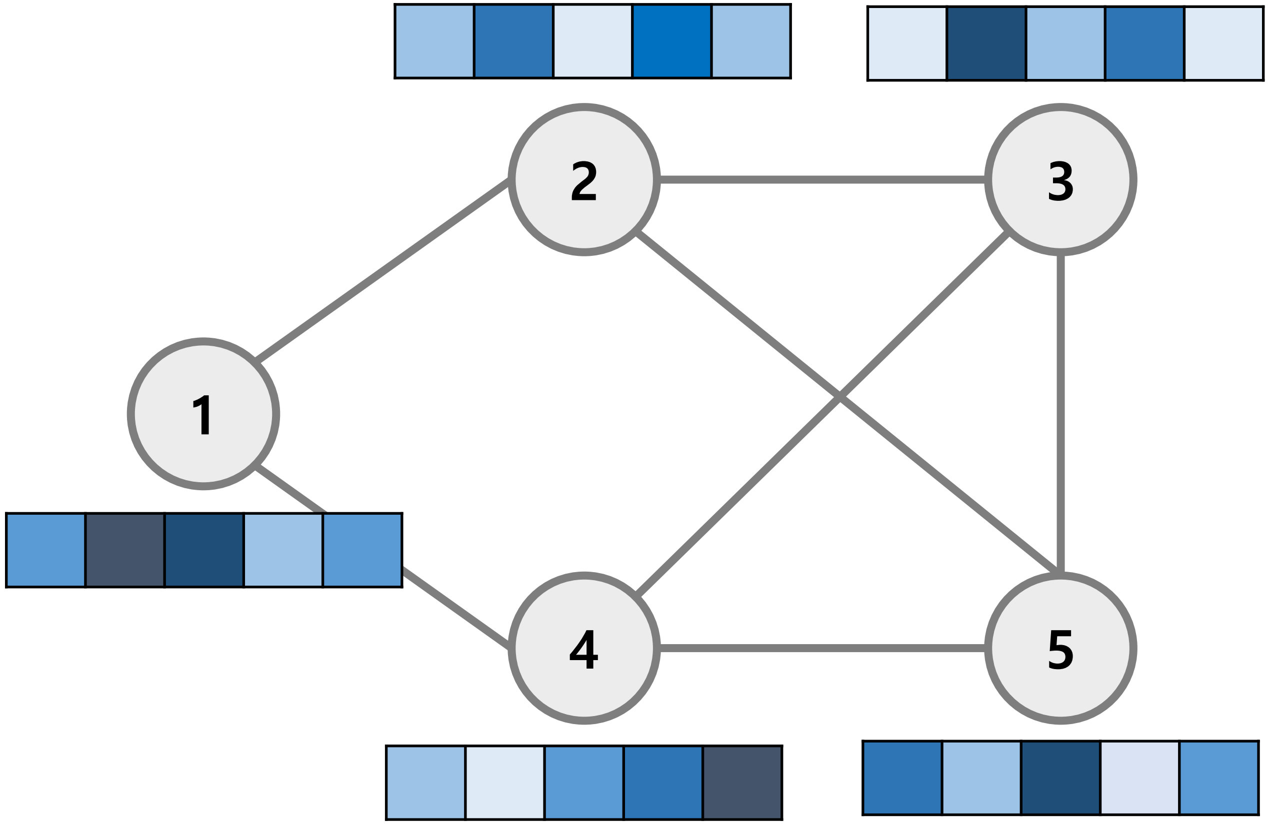


3D Mesh

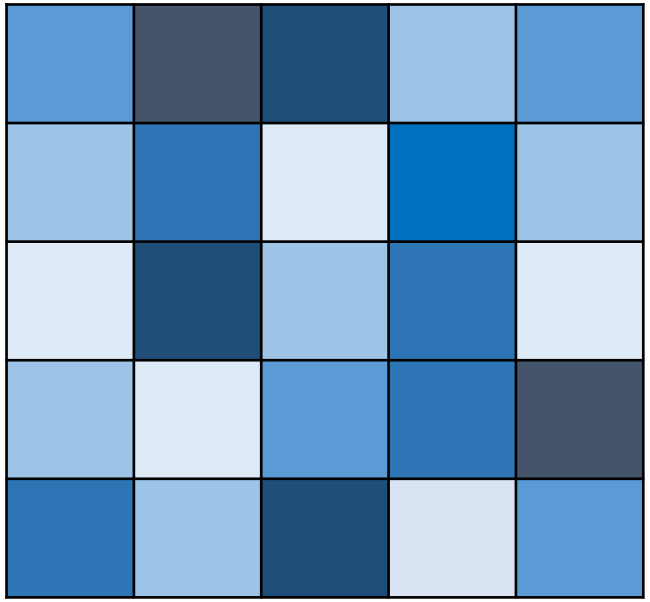


Euclidean Space

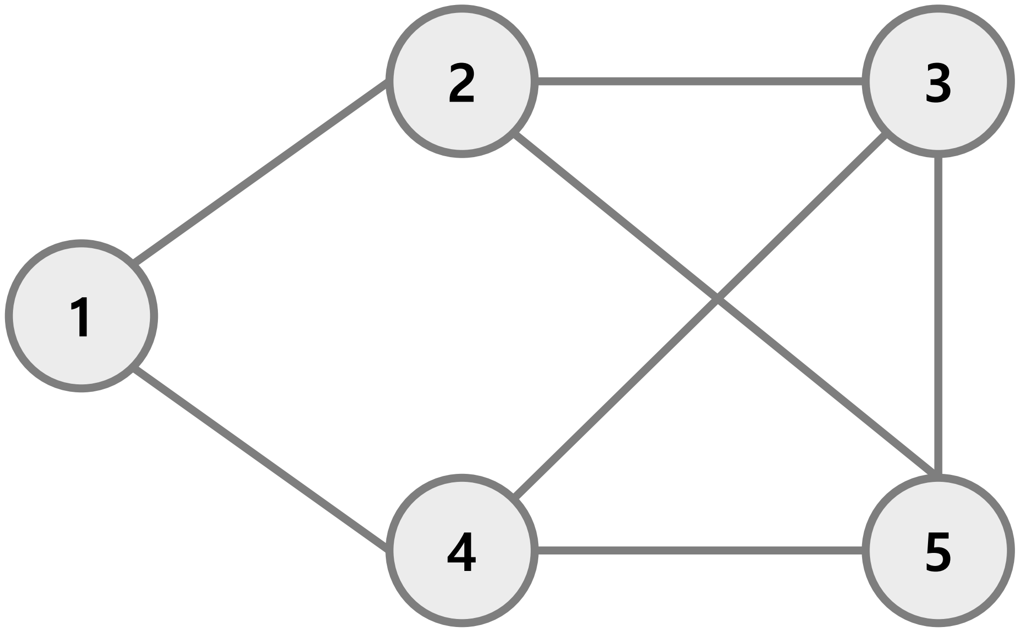
- Matrix Representation of Graph
  - Node-Feature matrix
    - ✓ N by D dimension



[Node-Feature Matrix]



- Matrix Representation of Graph
  - Adjacency matrix: Undirected graph
    - ✓ N by N square matrix
    - ✓ Symmetric

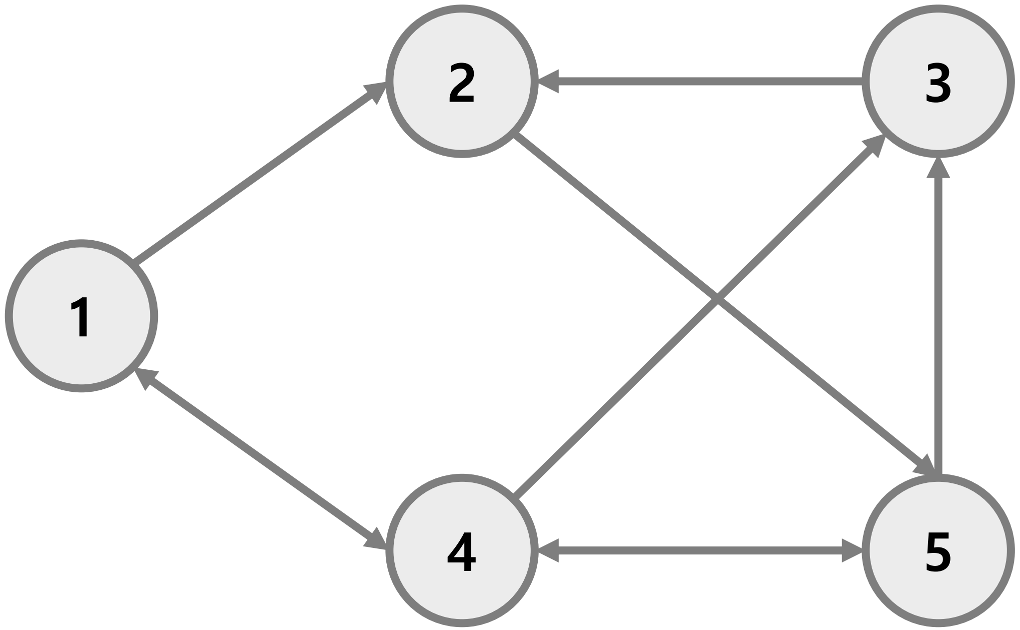


[Adjacency Matrix]

0	1	0	1	0
1	0	1	0	1
0	1	0	1	1
1	0	1	0	1
0	1	1	1	0

Data Mining Quality Analytics

- Matrix Representation of Graph
  - Adjacency matrix: Directed graph
    - ✓ N by N
    - ✓ Asymmetric

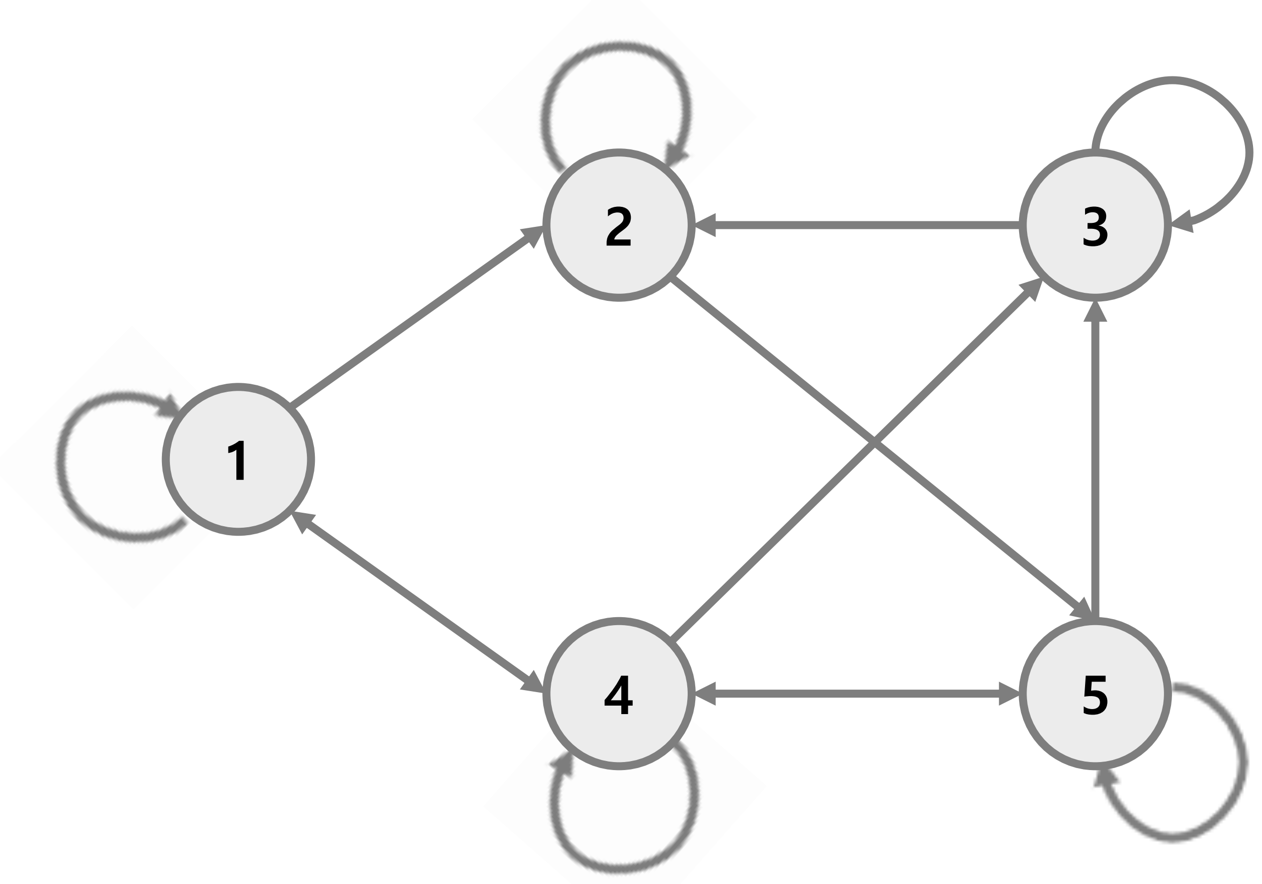


[Adjacency Matrix]

0	1	0	1	0
0	0	0	0	1
0	1	0	0	0
1	0	1	0	1
0	0	1	1	0

Data Mining Quality Analytics

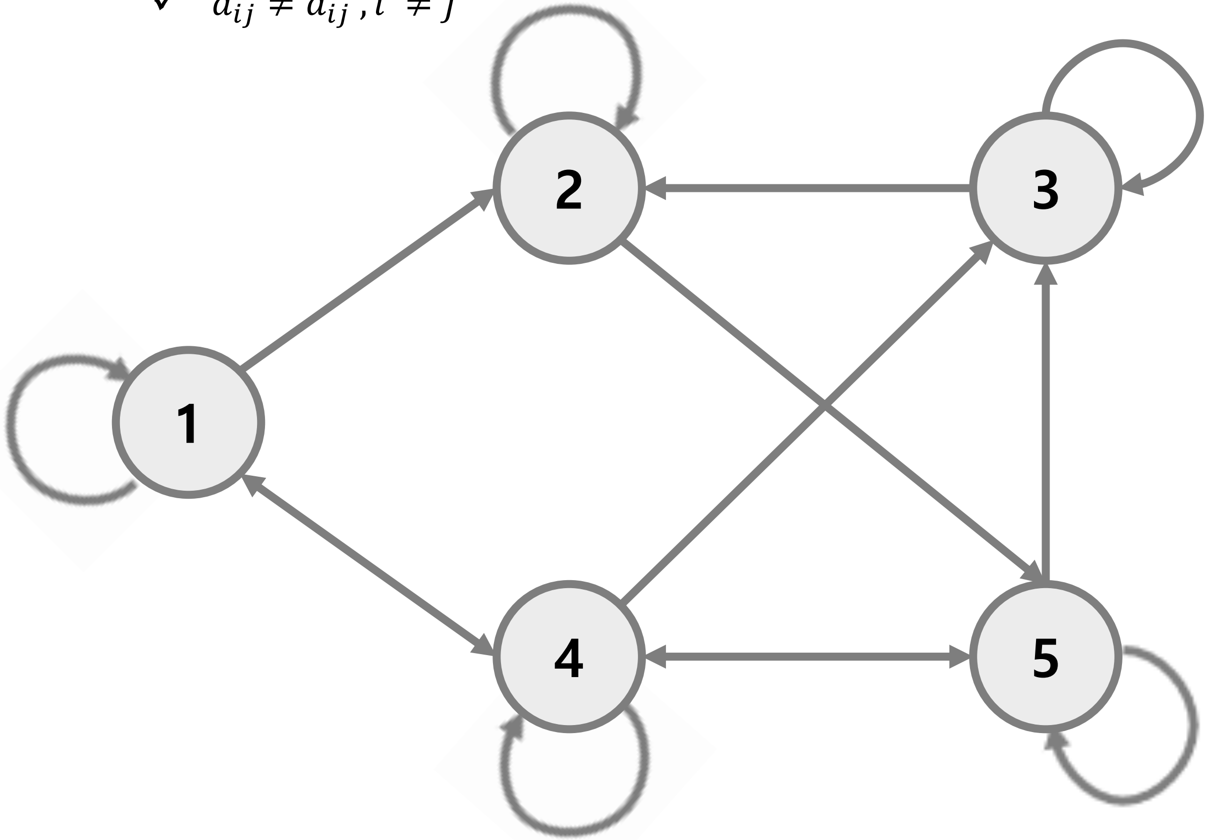
- Matrix Representation of Graph
    - Adjacency matrix: Directed graph
        - ✓ Adjacency matrix + Identity matrix



[Adjacency Matrix]

1	1	0	1	0
0	1	0	0	1
0	1	1	0	0
1	0	1	1	1
0	0	1	1	1

- Matrix Representation of Graph
  - Adjacency matrix: Weighted directed graph
    - ✓ Edge information
    - ✓ $a_{ij} \neq a_{ij}, i \neq j$

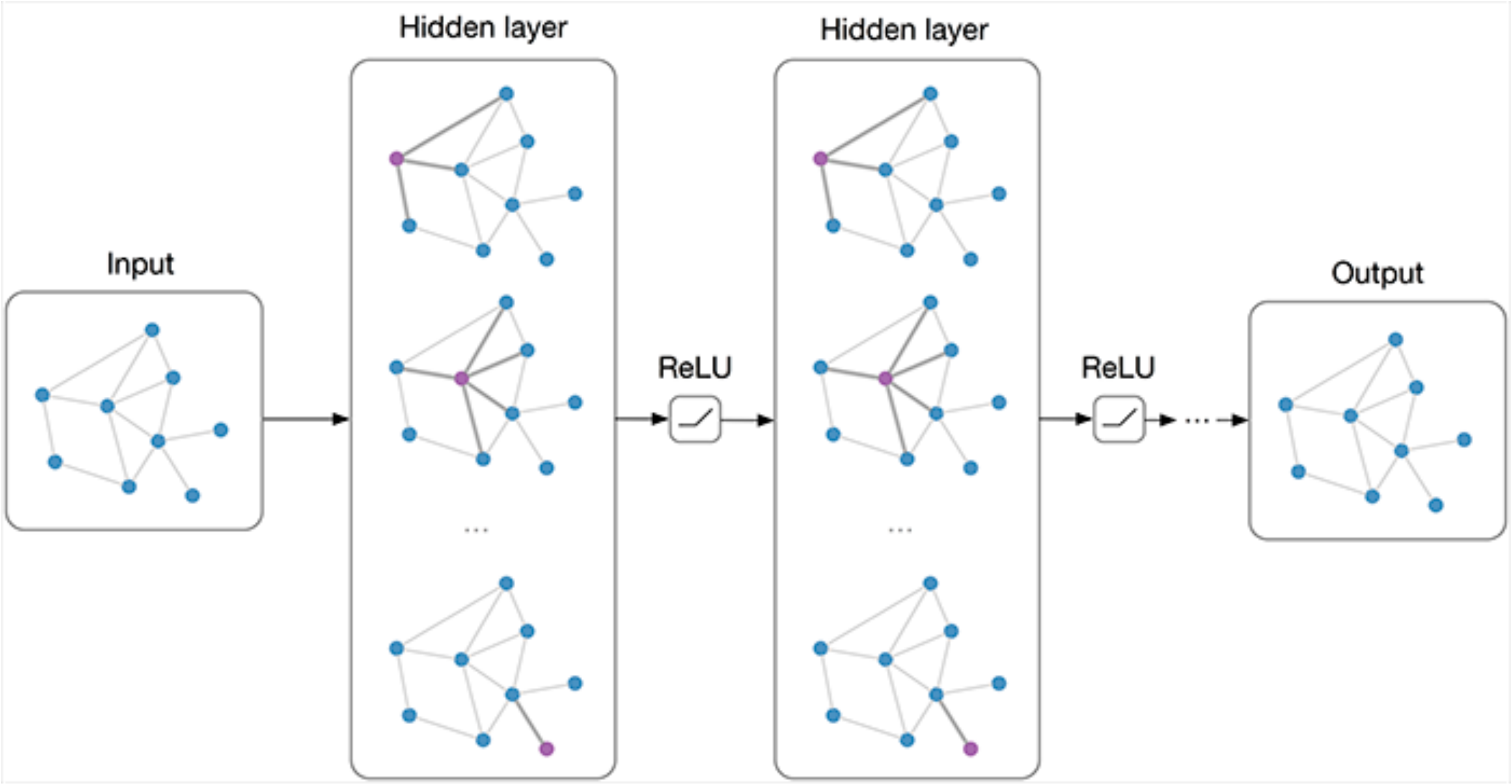


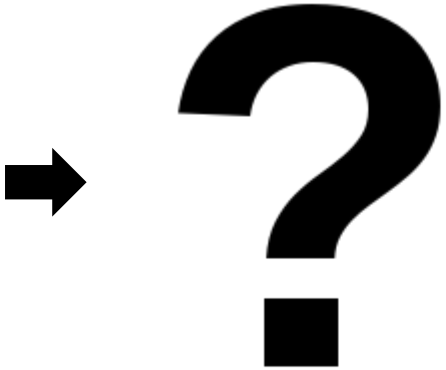
[Adjacency Matrix]

a_{11}	a_{12}	0	a_{14}	0
0	a_{22}	0	0	a_{25}
0	a_{32}	a_{33}	0	0
a_{41}	0	a_{43}	a_{44}	a_{45}
0	0	a_{53}	a_{54}	a_{55}

Data Mining Quality Analytics

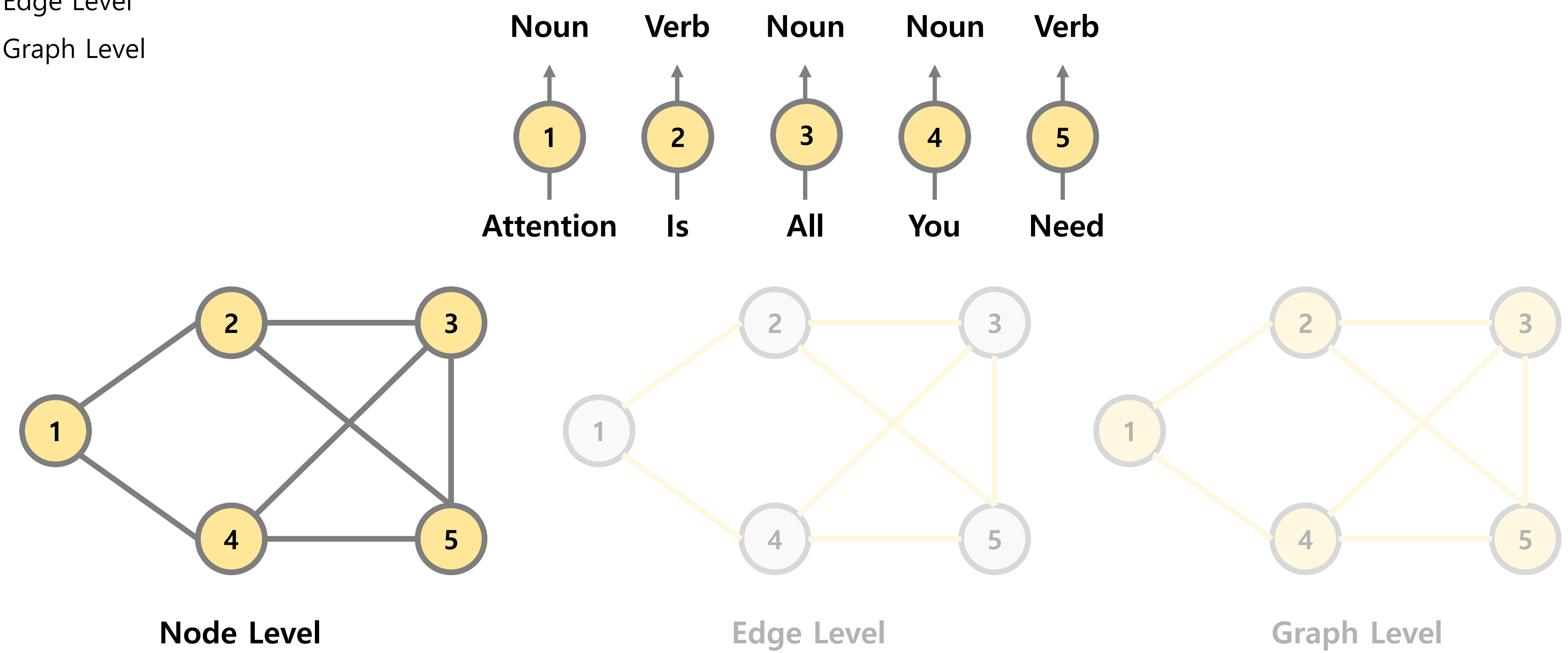
- Graph Neural Networks
  - Neural Networks for learning the structures of graphs



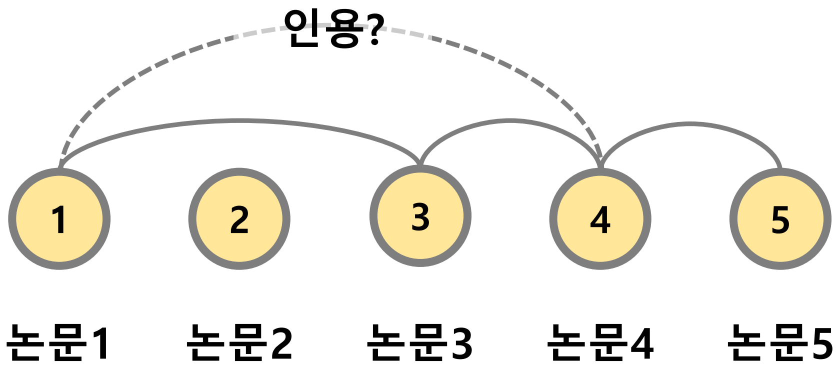


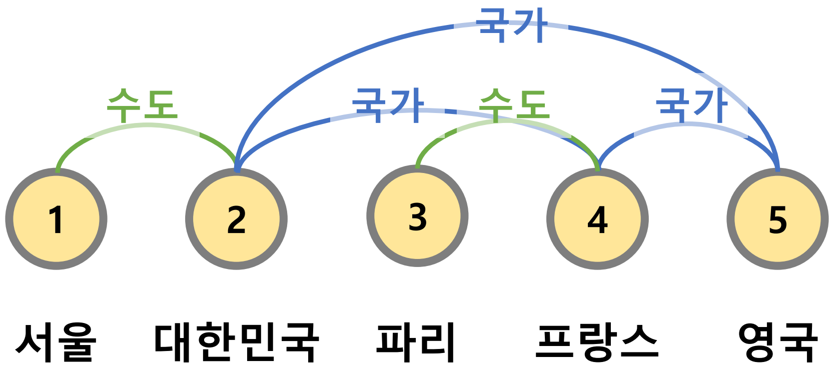
Data Mining Quality Analytics

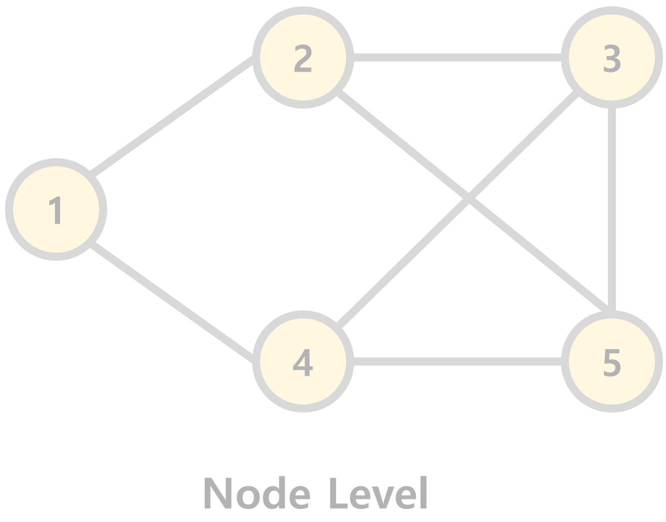
- GNN Tasks
  - Node Level
  - Edge Level
  - Graph Level

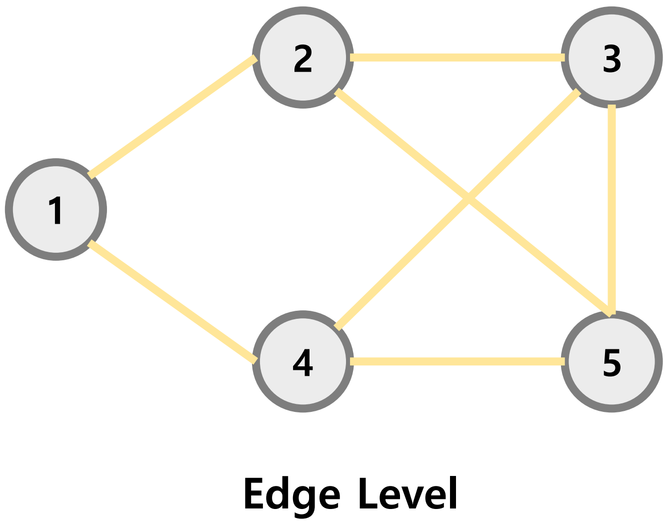


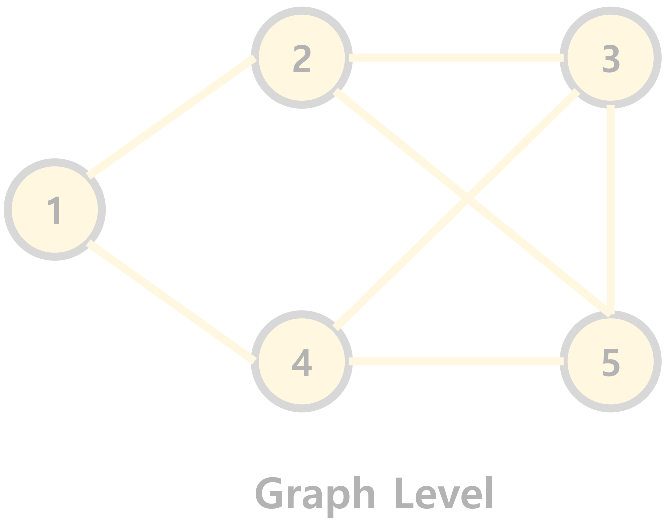
- GNN Tasks
  - Node Level
  - Edge Level
  - Graph Level



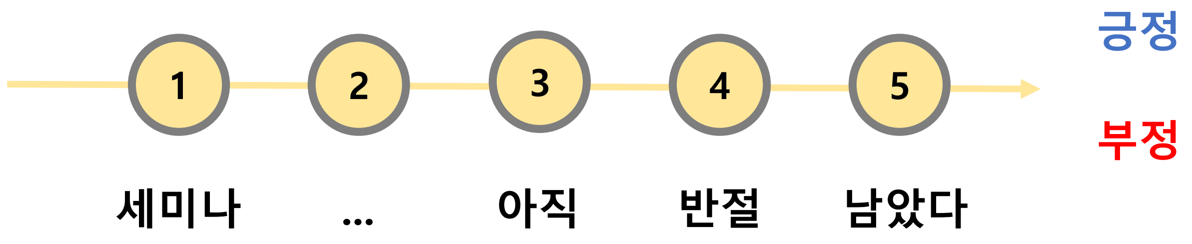


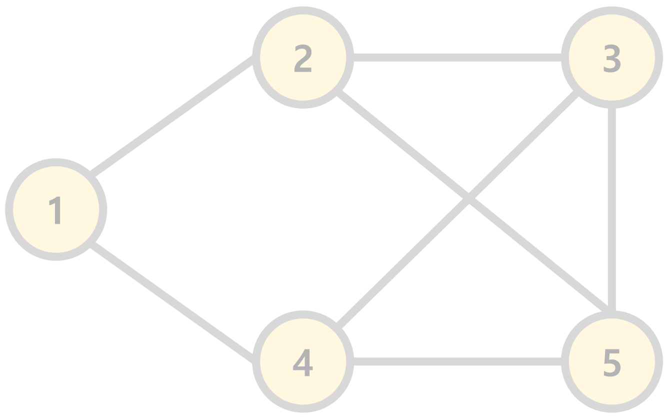




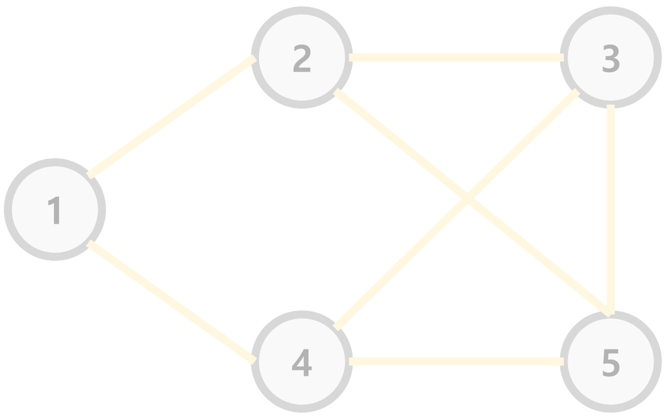


- GNN Tasks
  - Node Level
  - Edge Level
  - Graph Level

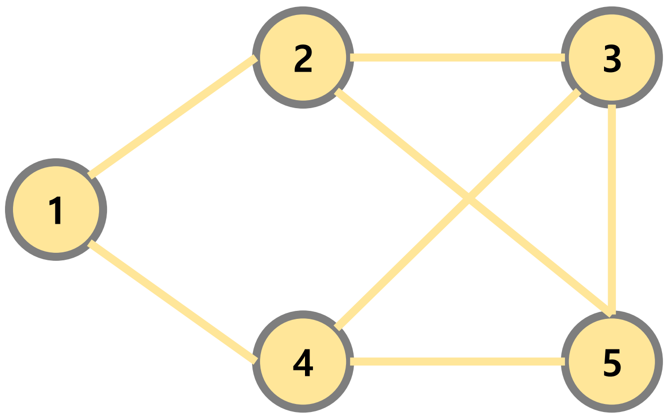




Node Level



Edge Level



Graph Level

03 | Graph Neural Networks

- Graph Task
  - Computer Vision
  - Natural Language Processing
  - Etc.

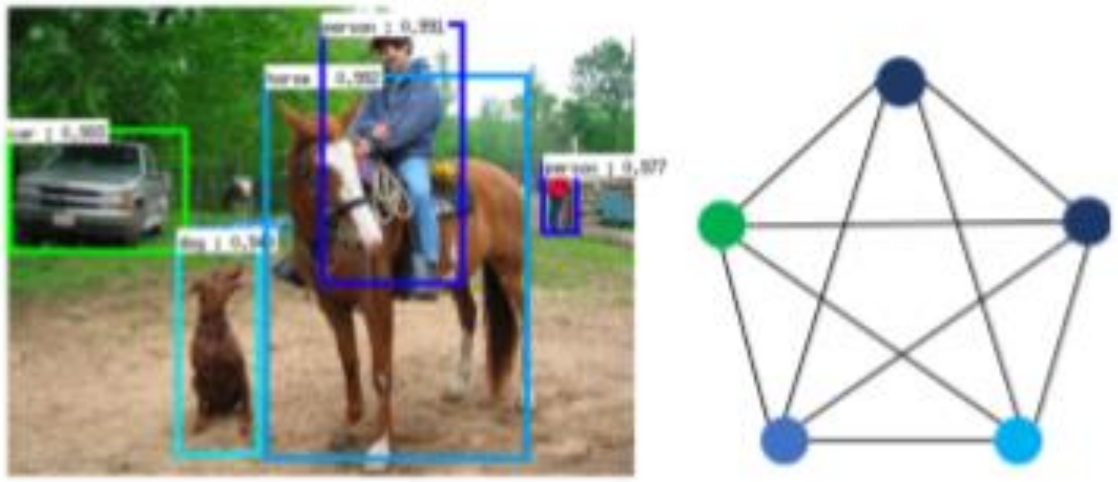
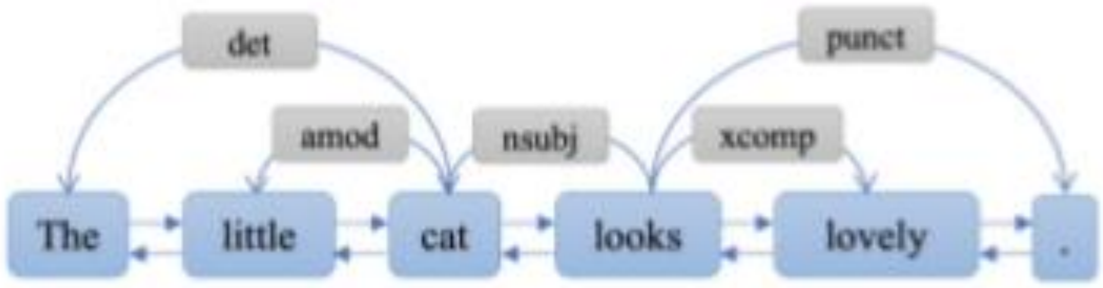


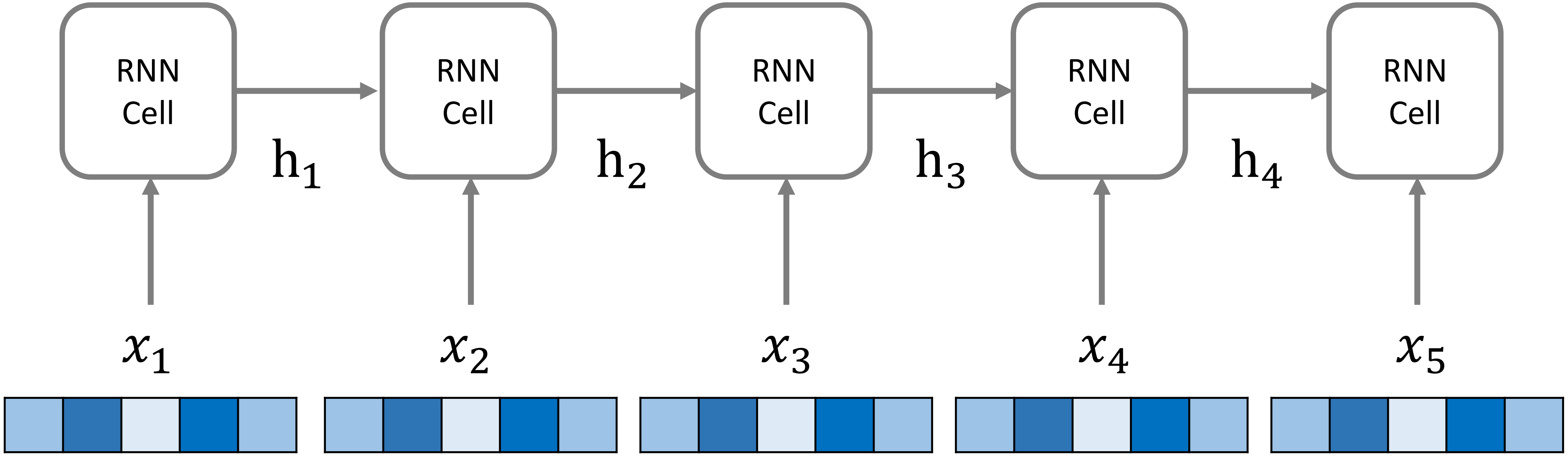
Image Graph



Text Graph

Zhou, Jie, et al. "Graph neural networks: A review of methods and applications." *arXiv preprint arXiv:1812.08434* (2018).

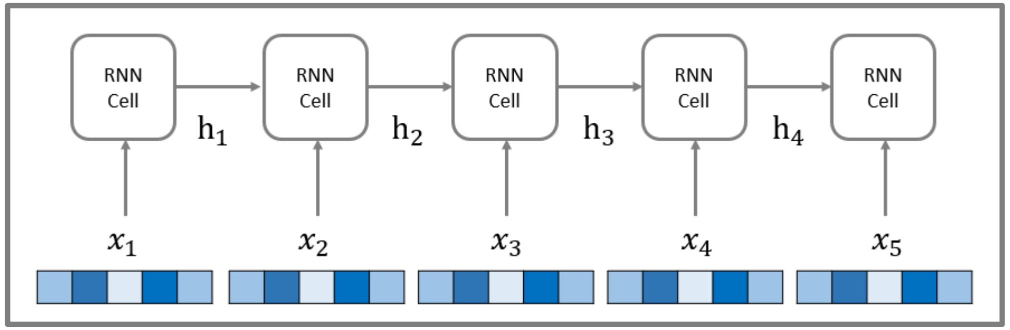
- GNN Learning Process
  - RNN learning process

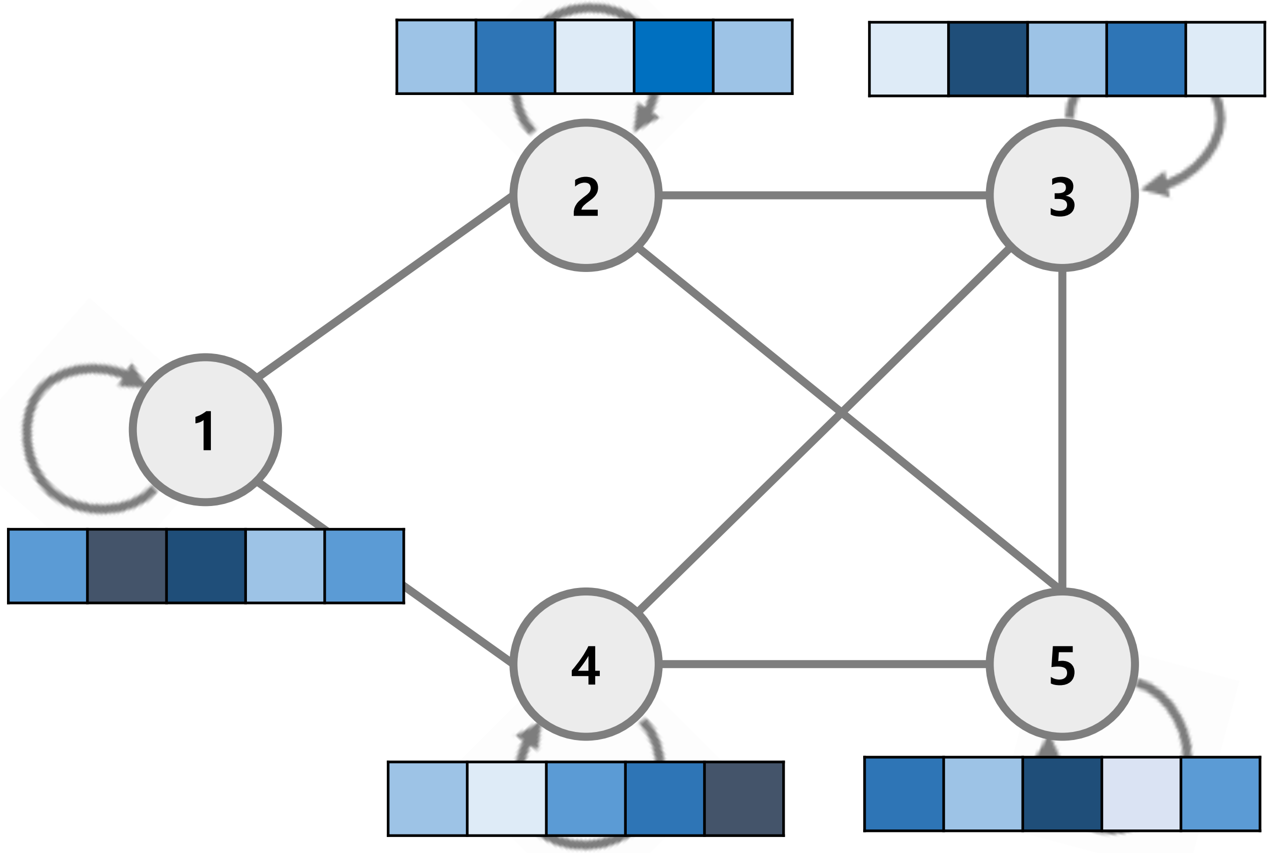


$h_t = combine(h_{t-1}, x_t)$ $\qquad$ $combine \in \{RNN, LSTM, GRU\}$

Data Mining Quality Analytics

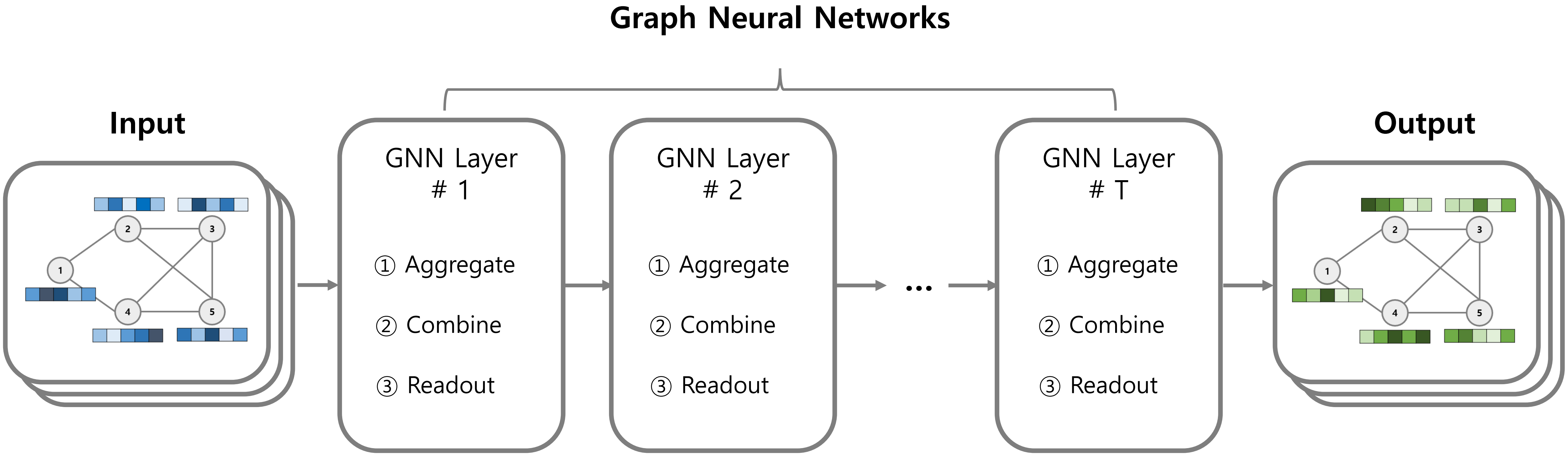
- GNN Learning Process
  - Graph: different with sequential data
    - ✓ No sequences(ordering)
    - ✓ Various graph structures
    - ✓ Multiple in-edge per node
  - Considerations for encoding graphs
    - ✓ Information passes along edges
    - ✓ Information passes in parallel
    - ✓ Target nodes affected by multiple nodes





Data Mining Quality Analytics

- GNN Layer
  - Node feature update reflecting graph structures
    - ① Aggregate / Massage passing
    - ② Combine / Update
    - ③ Readout



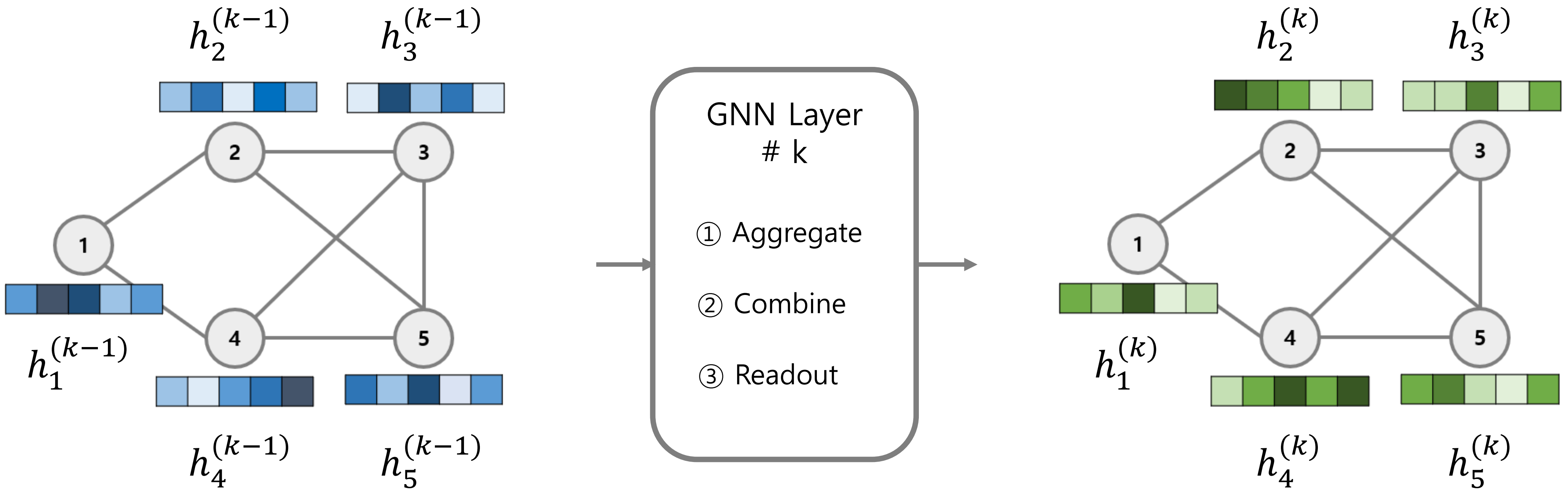
Data Mining Quality Analytics

- GNN Notation
  - $h_v^{(k)}$: hidden embedding node v at kth GNN layer
  - $v = target\ node$
  - $N(v) = neightbor\ nodes\ of\ v$
  - $u = neightbor\ node \in N(v)$

$$graph\ = G(A, X)$$

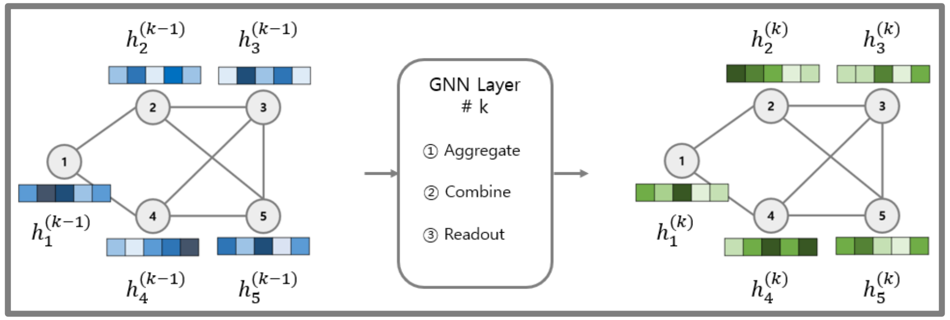
$$X = Node - Feature\ Matrix$$

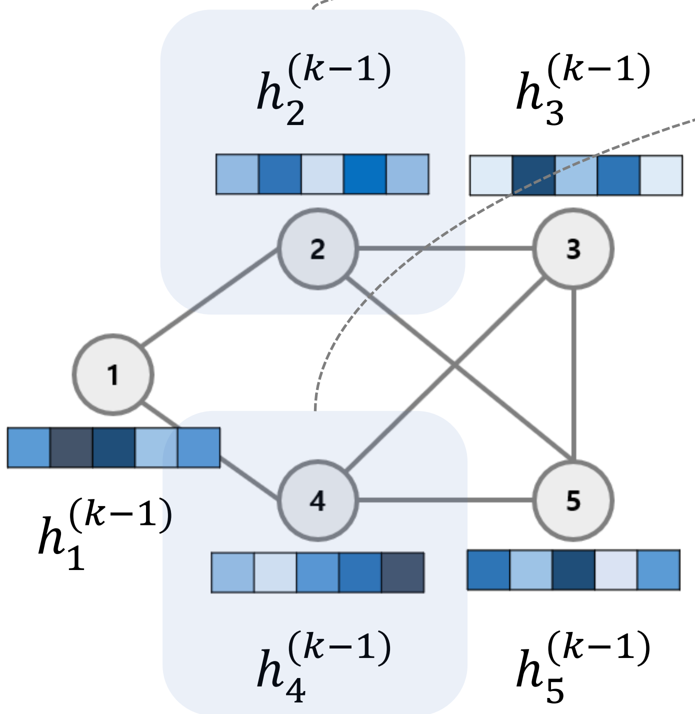
$$A = Adjacency\ Matrx$$



Data Mining Quality Analytics

- GNN: ① Aggregate
  - 타겟 노드의 이웃 노드들의 k-1 시점의 hidden state를 결합

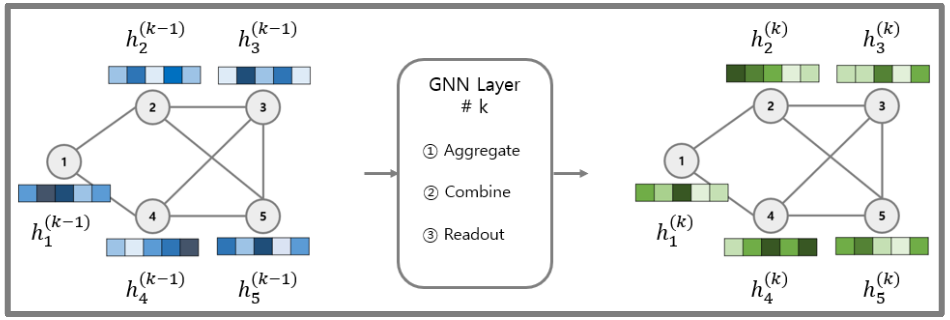


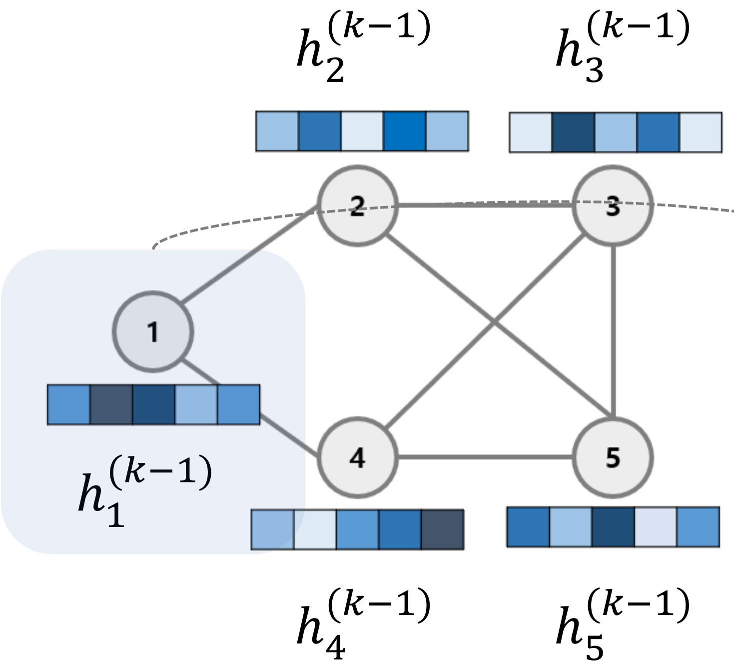


$$a_1^{(k-1)} = aggregate^k(\{ \quad , \quad \})$$

$$a_v^{(k-1)} = aggregate^k(\{h_u^{(k-1)}, \forall u \in N(v)\})$$

- GNN: ② Combine
  - k-1 시점 target node의 hidden state와 aggregated information을 사용하여 k 시점의 target node의 hidden state를 update





$$a_1^{(k-1)} = aggregate^k(\{ \quad , \quad \})$$

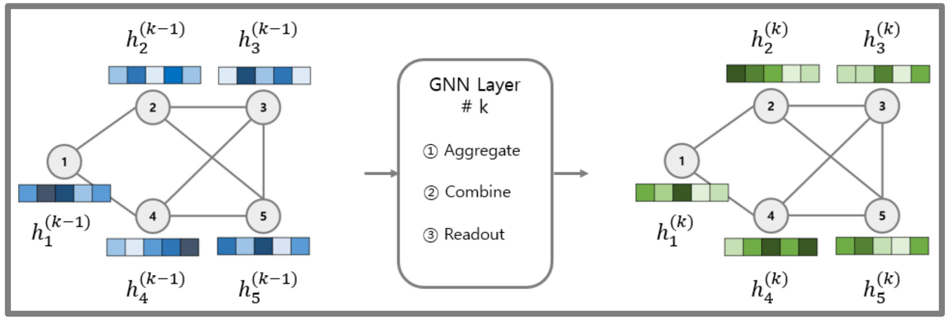
$$h_1^{(k)} = combine^k(a_1^{(k-1)}, \quad)$$

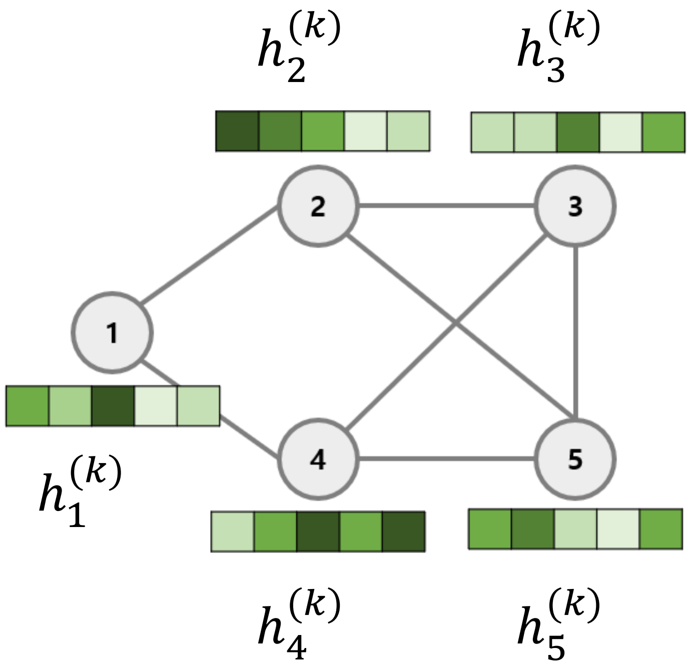
$$=$$

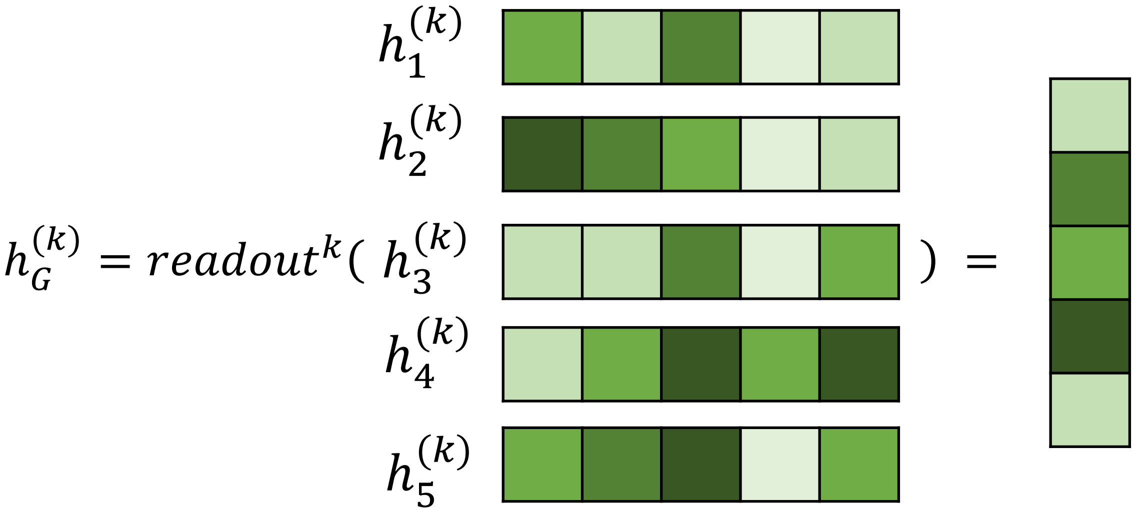
$$h_v^{(k)} = combine^k(a_u^{(k-1)}, h_v^{(k-1)})$$

Data Mining Quality Analytics

- GNN: ③ Readout
  - K 시점의 모든 Node들의 hidden state를 결합하여 graph의 hidden state 생성
  - Graph level classification

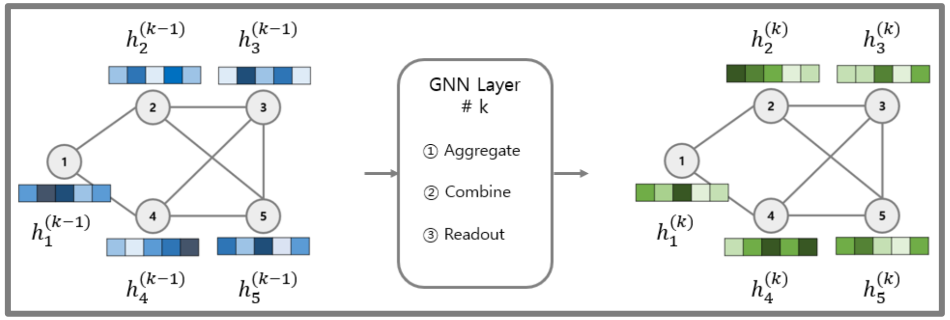


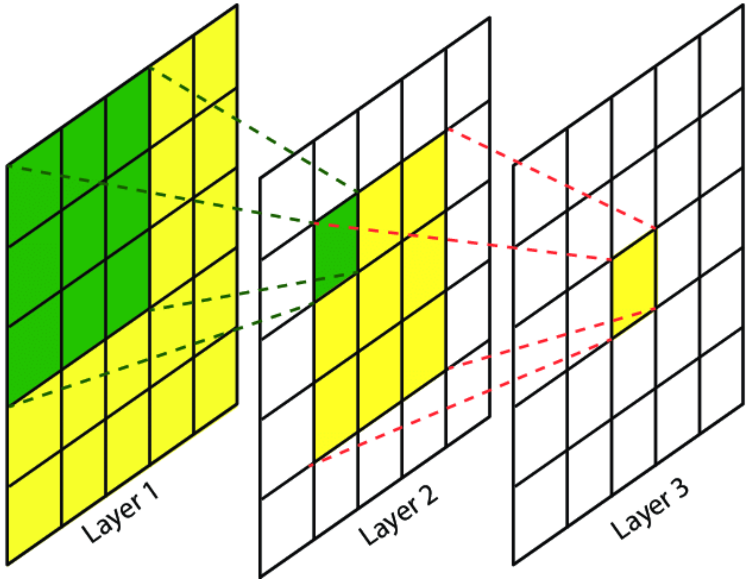




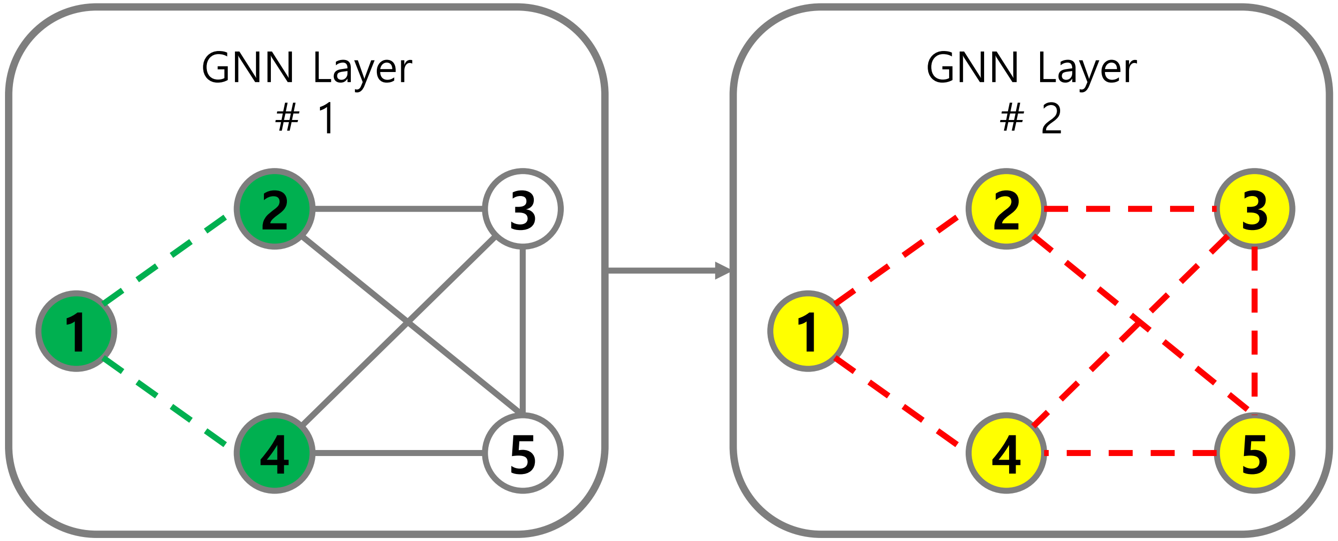
$$h_G^{(k)} = readout^k(h_v^{(k)}, \forall v \in G)$$

- Stacking GNN Layer
  - CNN: Increase the receptive filed
  - GNN: Increase hop of the graph





Convolutional Neural Networks



Graph Neural Networks

- GNN: Summary
  - Aggregate
    - ✓ $a_v^{(k-1)} = aggregate^k(\{h_{\mathrm{u}}^{(k-1)},\ \forall u \in\ N(v)\})$
  - Combine
    - ✓ $h_v^{(k)} = combine^k(a_u^{(k-1)}, h_v^{(k-1)})$
  - Readout
    - ✓ For graph level task
    - ✓ $h_G^{(k)} = readout^k(h_v^{(k)}, \forall v \in G)$
  - Stacking GNN Layer
    - ✓ Increase hop of the graph

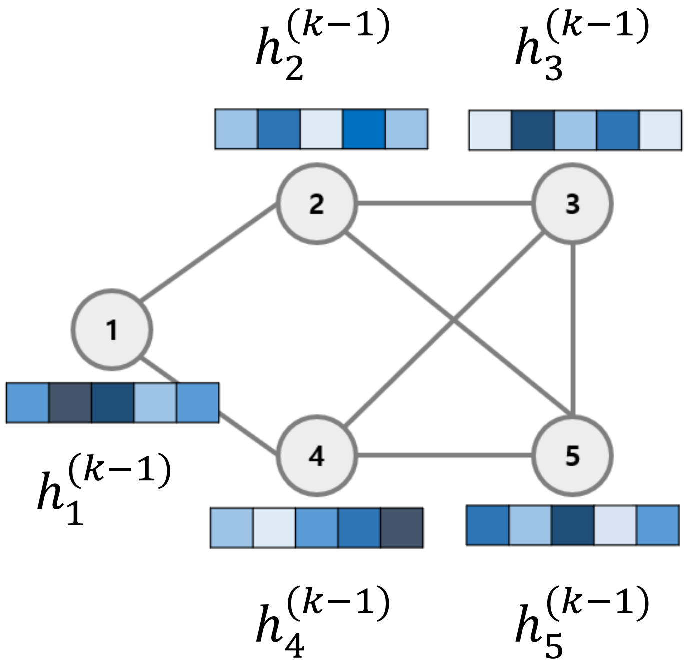
- GNN Variants
  - Aggregate / Combine function의 정의에 따라 다양한 방식의 모델이 존재함
  - Differentiable function

Name	Variant	Aggregator	Updater
Spectral Methods	ChebNet	$\mathbf{N}_k = \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}$	$\mathbf{H} = \sum_{k=0}^{K} \mathbf{N}_k \mathbf{\Theta}_k$
	1^{st}-order model	$\mathbf{N}_0 = \mathbf{X}$ $\mathbf{N}_1 = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}_0\mathbf{\Theta}_0 + \mathbf{N}_1\mathbf{\Theta}_1$
	Single parameter	$\mathbf{N} = (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
	GCN	$\mathbf{N} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
Non-spectral Methods	Convolutional networks in [33]	$\mathbf{h}_{\mathcal{N}_v}^t = \mathbf{h}_v^{t-1} + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{h}_v^t = \sigma(\mathbf{h}_{\mathcal{N}_v}^t \mathbf{W}_L^{\mathcal{N}_v})$
	DCNN	Node classification: $\mathbf{N} = \mathbf{P}^*\mathbf{X}$ Graph classification: $\mathbf{N} = 1_N^T\mathbf{P}^*\mathbf{X}/N$	$\mathbf{H} = f(\mathbf{W}^c \odot \mathbf{N})$
	GraphSAGE	$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t\left(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\}\right)$	$\mathbf{h}_v^t = \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \Vert \mathbf{h}_{\mathcal{N}_v}^t]\right)$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \Vert \mathbf{W}\mathbf{h}_k]))}{\sum_{j\in\mathcal{N}_v}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \Vert \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\sum_{k\in\mathcal{N}_v}\alpha_{vk}\mathbf{W}\mathbf{h}_k\right)$ Multi-head concatenation: $\mathbf{h}_{\mathcal{N}_v}^t = \Vert_{m=1}^{M}\sigma\left(\sum_{k\in\mathcal{N}_v}\alpha_{vk}^m\mathbf{W}^m\mathbf{h}_k\right)$ Multi-head average: $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{k\in\mathcal{N}_v}\alpha_{vk}^m\mathbf{W}^m\mathbf{h}_k\right)$	$\mathbf{h}_v^t = \mathbf{h}_{\mathcal{N}_v}^t$
Gated Graph Neural Networks	GGNN	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k\in\mathcal{N}_v}\mathbf{h}_k^{t-1} + \mathbf{b}$	$\mathbf{z}_v^t = \sigma(\mathbf{W}^z\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^z\mathbf{h}_v^{t-1})$ $\mathbf{r}_v^t = \sigma(\mathbf{W}^r\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^r\mathbf{h}_v^{t-1})$ $\widetilde{\mathbf{h}_v^t} = \tanh(\mathbf{W}\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}))$ $\mathbf{h}_v^t = (1-\mathbf{z}_v^t)\odot\mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^t}$
Graph LSTM	Tree LSTM (Child sum)	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k\in\mathcal{N}_v}\mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{U}^i\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma\left(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}^f\mathbf{h}_k^{t-1} + \mathbf{b}^f\right)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{U}^o\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{U}^u\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k\in\mathcal{N}_v}\mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
	Tree LSTM (N-ary)	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{l=1}^{K}\mathbf{U}_l^i\mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v k}^{tf} = \sum_{l=1}^{K}\mathbf{U}_{kl}^f\mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{l=1}^{K}\mathbf{U}_l^o\mathbf{h}_{vl}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{l=1}^{K}\mathbf{U}_l^u\mathbf{h}_{vl}^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v k}^{tf} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{l=1}^{K}\mathbf{f}_{vl}^t \odot \mathbf{c}_{vl}^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
	Graph LSTM in [34]	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{k\in\mathcal{N}_v}\mathbf{U}_{m(v,k)}^i\mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{k\in\mathcal{N}_v}\mathbf{U}_{m(v,k)}^o\mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{k\in\mathcal{N}_v}\mathbf{U}_{m(v,k)}^u\mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f\mathbf{h}_k^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k\in\mathcal{N}_v}\mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$

Zhou, Jie, et al. "Graph neural networks: A review of methods and applications." *arXiv preprint arXiv:1812.08434* (2018).

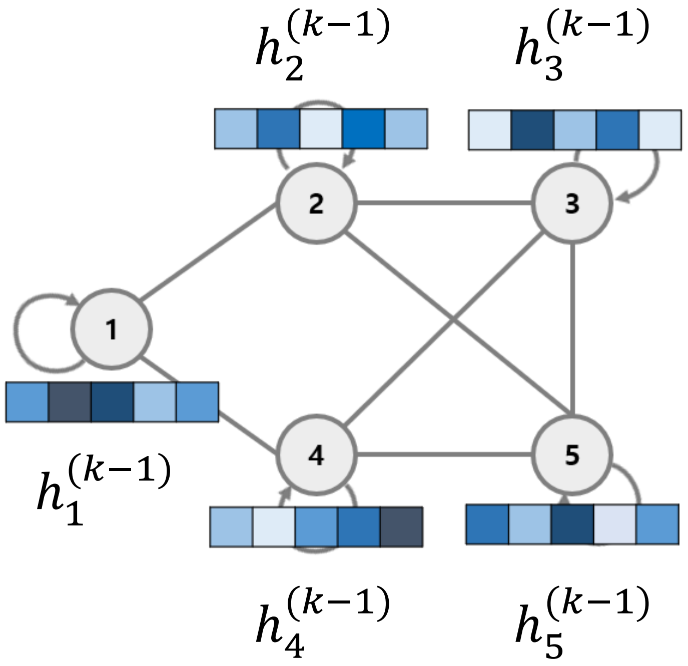
Data Mining Quality Analytics

- GNN
  - Aggregate
    - ✓ $a_v^{(k-1)} = \sum_{u \in N(v)} h_{\mathrm{u}}^{(k-1)}$
  - Combine
    - ✓ $h_v^{(k)} = Relu\left(\mathrm{W_{self}} h_v^{(k-1)} + \mathrm{W_{neigh}} a_v^{(k-1)}\right)$
  - Matrix form
    - ✓ $H^{(t)} = Relu(H^{(t-1)} W_{self} + AH^{(t-1)} W_{neigh} a_v^{(k-1)})$



F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," IEEE TNN 2009, vol. 20, no. 1, pp. 61–80, 2009.

- GNN (Self-loop)
  - When aggregating, self information is needed
  - $W_{\text{self}} = W_{\text{neigh}} = W$
  - Aggregate
    - ✓ $a_v^{(k-1)} = \sum_{u \in N(v)} h_u^{(k-1)} + h_v^{(k-1)}$
  - Combine
    - ✓ $h_v^{(k)} = Relu(Wa_v^{(k-1)})$
    - ✓ $h_v^{(k)} = Relu(W \sum_{u \in N(v) \cup \{v\}} h_u^{(k-1)})$
  - Matrix form
    - ✓ $H^{(t)} = Relu((A + I)H^{(t-1)}W)$



F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," IEEE TNN 2009, vol. 20, no. 1, pp. 61–80, 2009.

Data Mining Quality Analytics

- Graph Convolutional Networks (2016)
  - Amsterdam Univ., CIFAR
  - If graph size is too large
    - ✓ Unstable and sensitive to node degrees
    - ✓ Degree = # of neighbor nodes for each node
  - Normalized aggregate function
    - ✓ $a_v^{(k-1)} = \sum_{u \in N(v) \cup \{v\}} \frac{h_u^{(k-1)}}{\sqrt{|N(v)||N(u)|}}$
  - Combine
    - ✓ $h_v^{(k)} = Relu(Wa_v^{(k-1)})$
  - Matrix form
    - ✓ $H^{(t)} = Relu((D^{-\frac{1}{2}}(A+I)\, D^{-\frac{1}{2}} H^{(t-1)} W)$

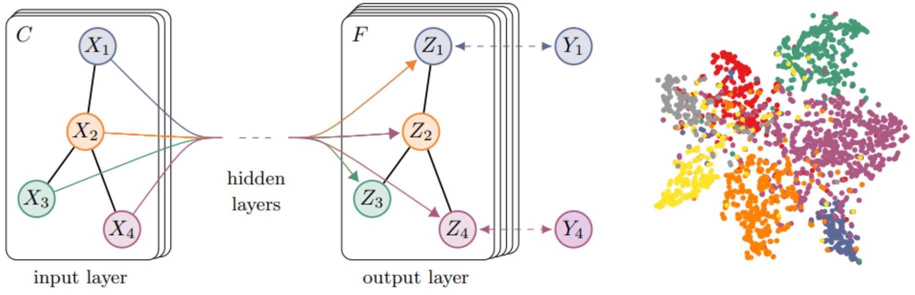
SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

ABSTRACT

We present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. We motivate the choice of our convolutional architecture via a localized first-order approximation of spectral graph convolutions. Our model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on citation networks and on a knowledge graph dataset we demonstrate that our approach outperforms related methods by a significant margin.



(a) Graph Convolutional Network (b) Hidden layer activations

Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).

- Gated Graph Neural Networks (2016)
  - Stacking deep layers lead overfitting / vanishing gradient
  - ICLR, Toronto Univ., Microsoft
  - Aggregate
    - ✓ $a_v^{(k-1)} = \sum_{u\in N(v)} \sum h_u^{(k-1)}$
  - Combine
    - ✓ $h_v^{(k)} = GRU(h_v^{(k-1)}, a_v^{(k-1)})$
  - Matrix form
    - ✓ $H^{(t)} = GRU(((A+I)\mathrm{W},\ H^{(t-1)})$

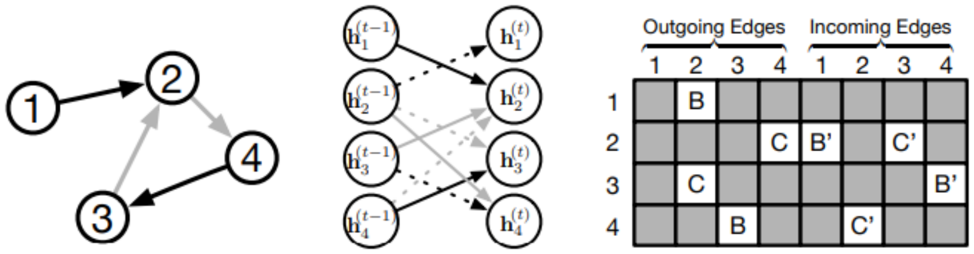
Gated Graph Sequence Neural Networks

Yujia Li* & Richard Zemel
Department of Computer Science, University of Toronto
Toronto, Canada
{yujiali,zemel}@cs.toronto.edu

Marc Brockschmidt & Daniel Tarlow
Microsoft Research
Cambridge, UK
{mabrocks,dtarlow}@microsoft.com

Abstract

Graph-structured data appears frequently in domains including chemistry, natural language semantics, social networks, and knowledge bases. In this work, we study feature learning techniques for graph-structured inputs. Our starting point is previous work on Graph Neural Networks (Scarselli et al., 2009), which we modify to use gated recurrent units and modern optimization techniques and then extend to output sequences. The result is a flexible and broadly useful class of neural net-



(a) (b) (c) $\mathbf{A} = \left[\mathbf{A}^{(\mathrm{out})}, \mathbf{A}^{(\mathrm{in})}\right]$

Li, Yujia, et al. "Gated graph sequence neural networks." *arXiv preprint arXiv:1511.05493* (2015).

- GraphSAGE (2017)
  - NIPS, Stanford Univ.
  - Consider node importance or ordering
  - Aggregate
    - Mean aggregate
      - ✓ $a_v^{(k-1)} = \sum_{u\in\ N(v)} \frac{h_{\mathrm{u}}^{(k-1)}}{|N(u)|}$
    - LSTM aggregate (Random permutation of neighbors)
      - ✓ $a_v^{(k-1)} = LSTM(\sum\{\mathrm{W}_{\mathrm{agg}} h_{\mathrm{u}}^{(k-1)}, \forall u \in\ N(v)\})$
    - Pooling aggregate
      - ✓ $a_v^{(k-1)} = Pool(\{\mathrm{W}_{\mathrm{pool}} h_{\mathrm{u}}^{(k-1)}, \forall u \in\ N(v)\})$
      - ✓ $Pool = element\text{-}wise\ mean\ or\ max$
  - Combine
    - ✓ $h_v^{(k)} = Relu(W[h_v^{(k-1)}, a_v^{(k-1)}])$
  - Residual connection / Skip connection

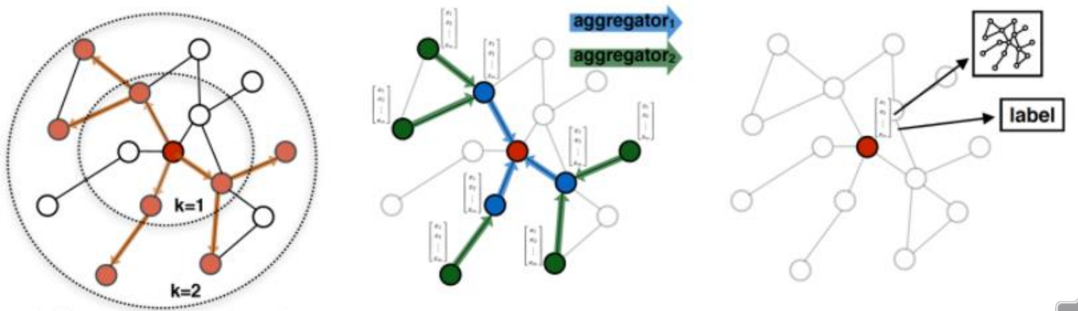
Inductive Representation Learning on Large Graphs

William L. Hamilton* wleif@stanford.edu
Rex Ying* rexying@stanford.edu
Jure Leskovec jure@cs.stanford.edu

Department of Computer Science
Stanford University
Stanford, CA, 94305

Abstract

Low-dimensional embeddings of nodes in large graphs have proved extremely useful in a variety of prediction tasks, from content recommendation to identifying protein functions. However, most existing approaches require that all nodes in the graph are present during training of the embeddings; these previous approaches are inherently *transductive* and do not naturally generalize to unseen nodes. Here we



Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." *Advances in neural information processing systems*. 2017.

03 | Graph Neural Networks

- GNN Variants
  - Challenges
    - ✓ Self-loop (Vanilla GNN)
    - ✓ Node degrees (GCN)
    - ✓ Node importance (GraphSAGE)
    - ✓ Overfitting / Vanishing gradient (GGNN)

Attention is All You Need!