#hutom_ai

Graph Neural Network (GNN)

2021-03-03 | JiHyun Lee

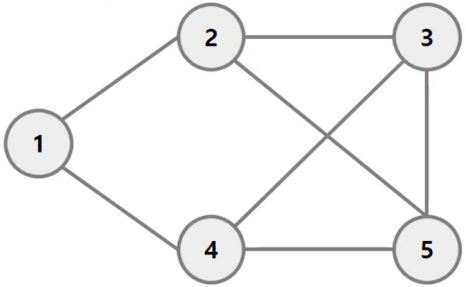
hutom

Content

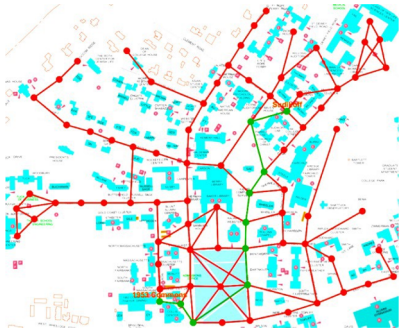
1. Graph Neural Network (GNN) Overview
    1) The graph neural network model (2009)

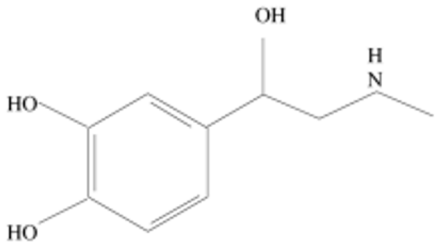
1. Graph Attention Network (GAT)
    1) Attention Overview
    2) Graph Attention Networks (2018)

- Graph Data Structure
  - Node와 Edge로 구성되어 있는 자료구조
    - Node = Vertex
      - 그래프 시스템의 하나의 원소값
    - Edge
      - 노드 간 관계의 여부
      - 데이터와 데이터 간의 상관관계 (데이터의 다양한 특징을 함축함)



  - 그래프 적용
    - GPS Navigation
    - Drugs or Not (by probability)
    - Social networks (e.g. Facebook uses a graph for suggesting friends)
    - Recommendations: Amazon/Netflix uses graphs to make suggestions for products/movies

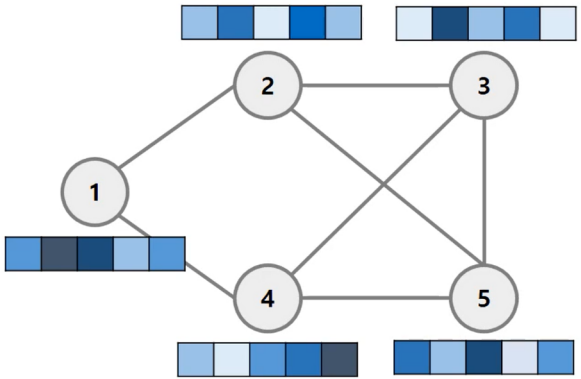




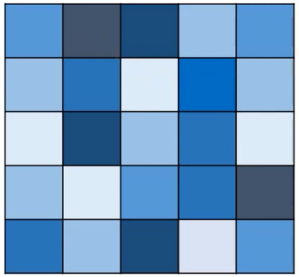
- Graph Data Structure
  - Non-Euclidean space
    - Image data, text data는 Euclidean space에 표현될 수 있는 것과 다르게 graph data는 Euclidean space에 표현하기 어려움.

- Matrix Representation of Graph
  - Node-Feature Matrix
  - Adjacency Matrix
    - Undirected graph
    - Directed graph
    - Directed graph + Identity matrix
    - Weighted directed graph

- Matrix Representation of Graph
  - **Node-Feature Matrix**
    - 노드의 특성 표현
    - 노드 하나하나가 가지는 특성

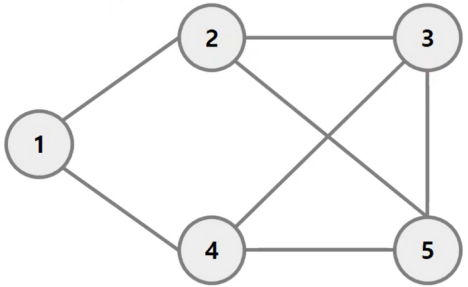


[Node-Feature Matrix]



- Matrix Representation of Graph
  - Adjacency Matrix
    - 노드 간 관계를 의미하는 edge를 matrix 형태로 표현
    - edge 방향성 여부에 따라
      - Undirected graph
      - Directed graph

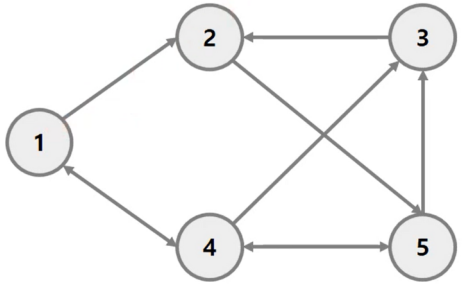
  - **Adjacency Matrix: Undirected graph**
    - edge 방향성이 없는 그래프
    - 특정 노드가 다른 노드와 인접한지 아닌지 (관계가 있는지 없는지) 유무만, binary 형태로 표현
    - N by N square matrix
    - Symmetric



[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

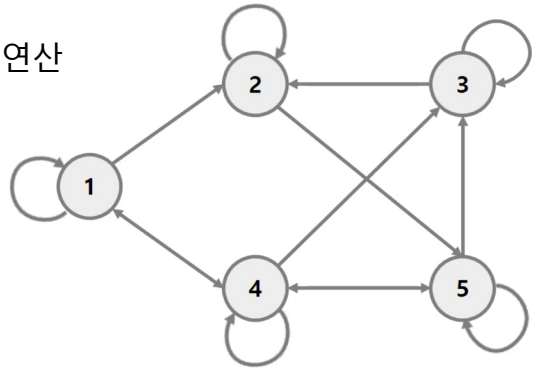
- Matrix Representation of Graph
  - **Adjacency Matrix: Directed graph**
    - edge 방향성이 있는 그래프
    - N by N square matrix
    - Asymmetric



[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

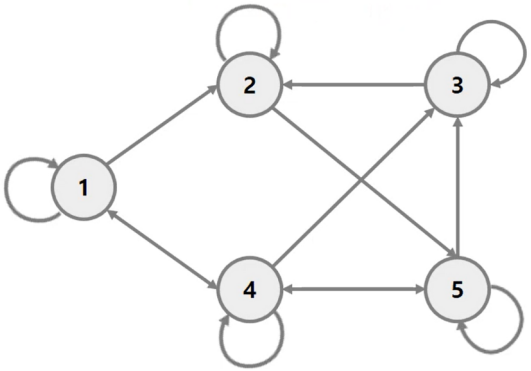
  - **Adjacency Matrix: Directed graph + Identity matrix**
    - **Adjacency Matrix + Identity Matrix**
    - self loop 존재
    - 모든 self loop를 고려해서 matrix 연산



[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
    - node와 node간의 관계를 weight로 표현
    - 단순히 binary 값으로 표현되는 것이 아니라, 숫자 값들이 위치한 matrix 형태



[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}

Graph Neural Network Overview

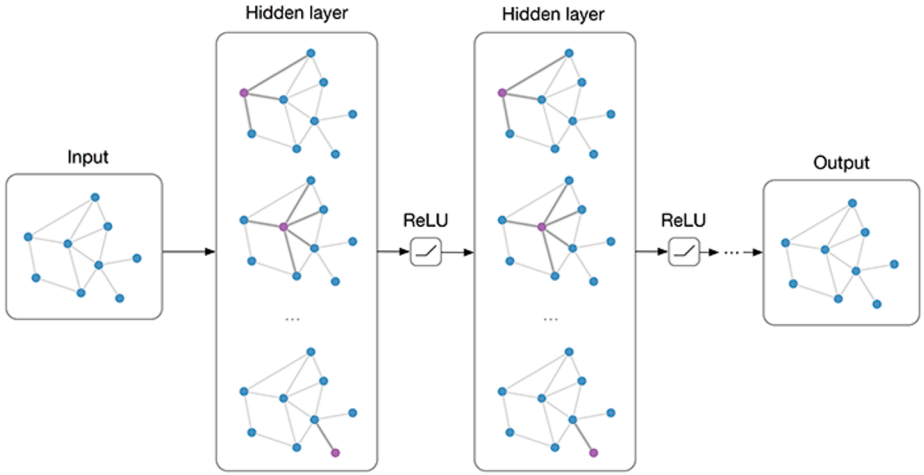
- Graph Neural Network task
  - **Input: Graph**
  - **Output: Label**
    1. Node Classification (Node Level)
       - Node embedding을 통해 점 분류
       - 일반적으로 그래프의 일부만 레이블 된 상황에서 semi-supervised learning
       - e.g. Reddit 게시물
    1. Link Prediction (Edge Level)
       - 그래프의 점들 사이의 관계를 파악하고, 두 점 사이에 얼마나 연관성이 있을지 예측.
       - e.g. 페이스북 친구 추천, 넷플릭스 영상 추천 (영화와 유저가 점이고, 유저가 영화를 봤으면 선으로 연결. 선으로 연결되지 않은 영화, 유저 쌍 중에 연결될 가능성이 높은 쌍을 찾아서 유저가 영화를 감상할 가능성이 높다고 예측할 수 있음)
    1. Graph Classification (Graph Level)
       - 그래프 전체를 여러가지 카테고리로 분류
       - e.g. 분자 구조가 그래프의 형태로 제공되고, 이를 분류하는 문제 등.

- Graph Neural Network 가 각광받기 시작하면서, image나 text를 그래프로 표현하여 신경망을 학습하기도 함.

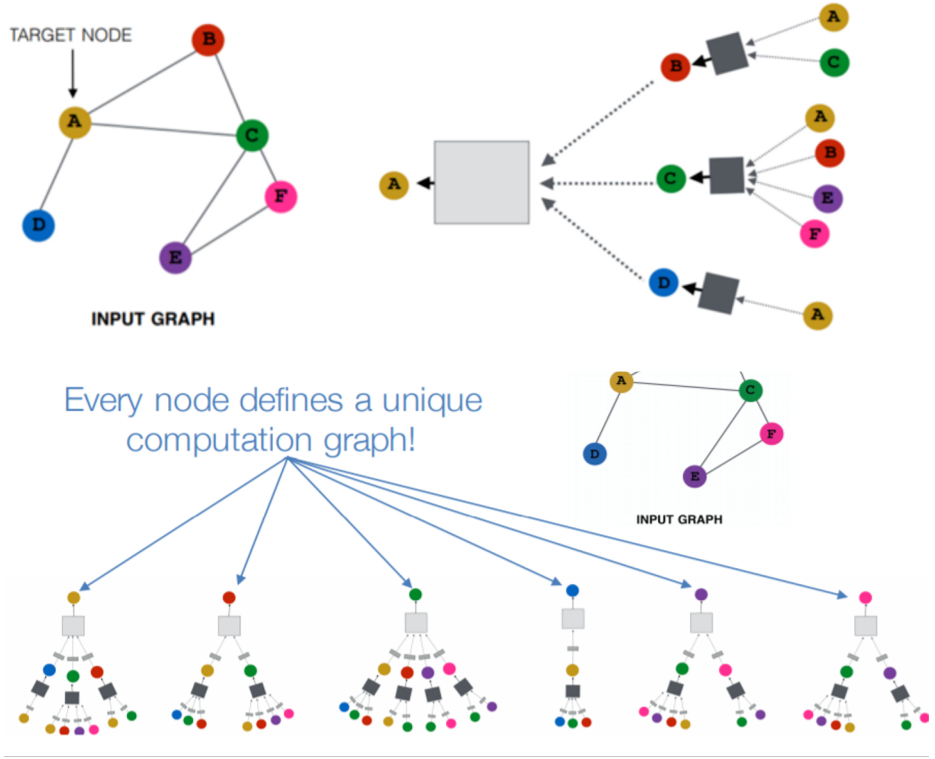
- Graph Neural Network

$$F(\text{ Graph }) = \text{embedding}$$

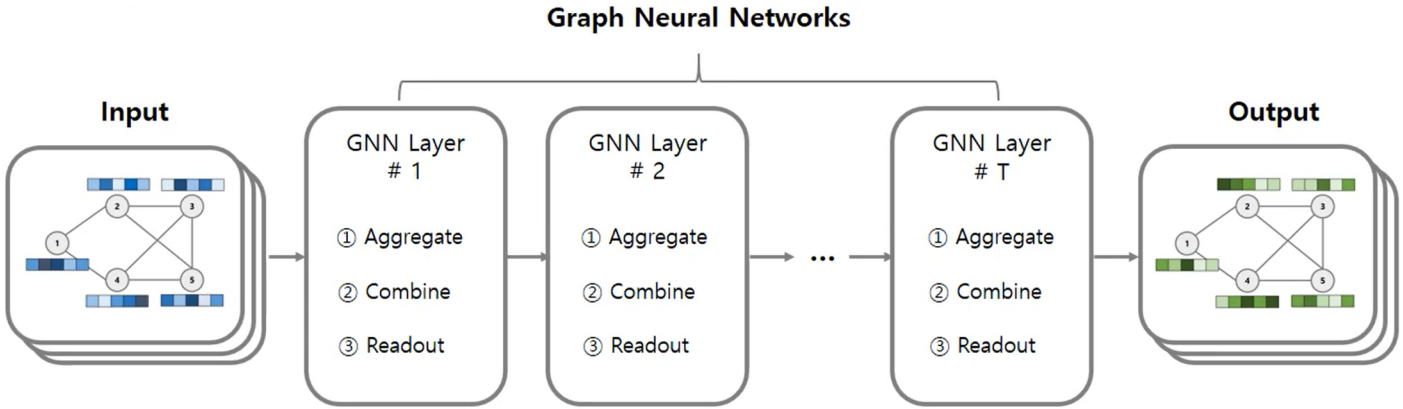
  - 새로운 그래프가 들어왔을 때, 이를 embedding 할 수 있는 함수 F 를 찾는 것이 목표



- Graph Neural Network
  - Key Idea: **Generate node embeddings based on local neighborhoods.**
  - Intuition: **Nodes aggregate information from their neighbors using neural networks**

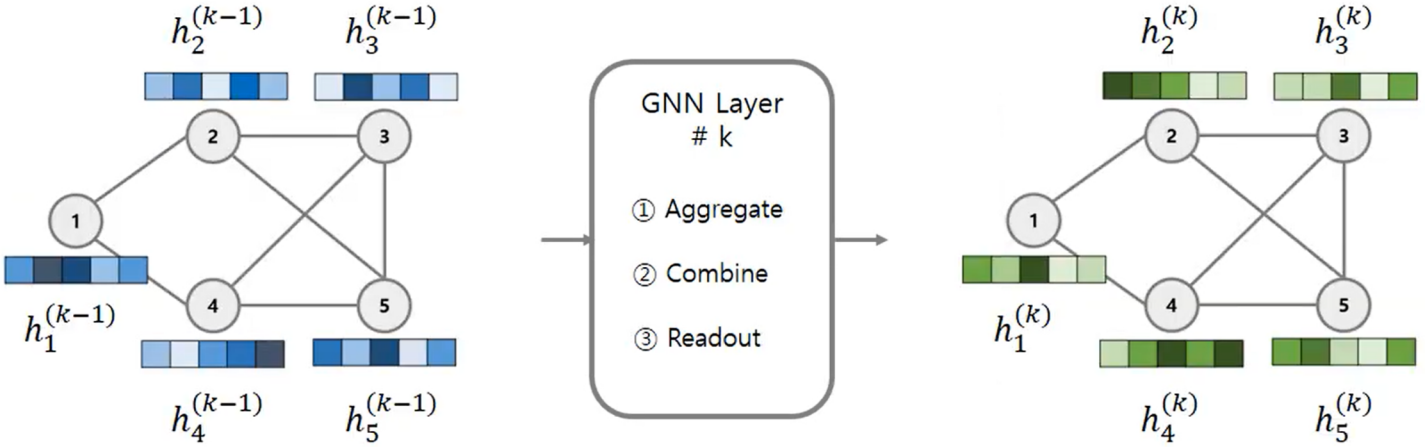


- Graph Neural Network
  - Key Idea: **Generate node embeddings based on local neighborhoods.**
  - Intuition: **Nodes aggregate information from their neighbors using neural networks**

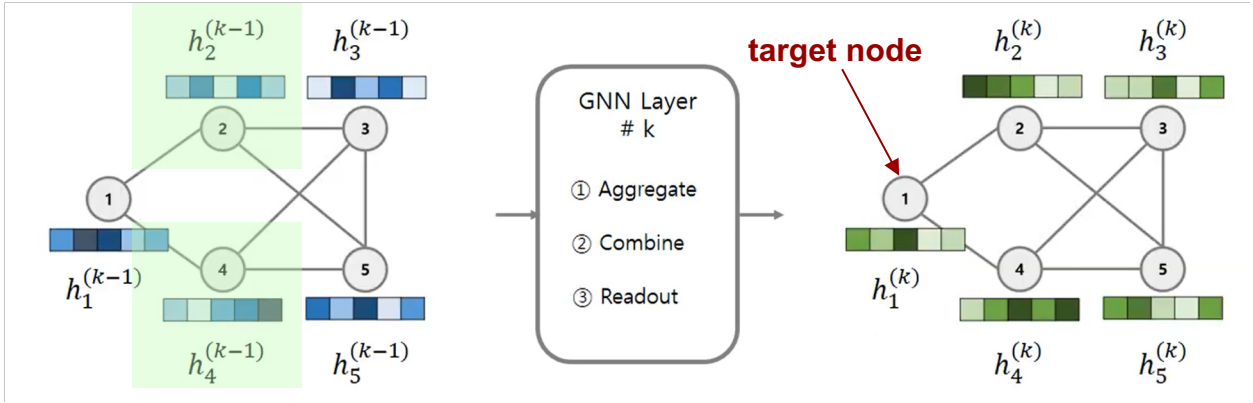


- GNN layer
  - Node feature update reflecting graph structures
    1. **Aggregate (Massage passing)**
    2. **Combine (Update)**
    3. **Readout**

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

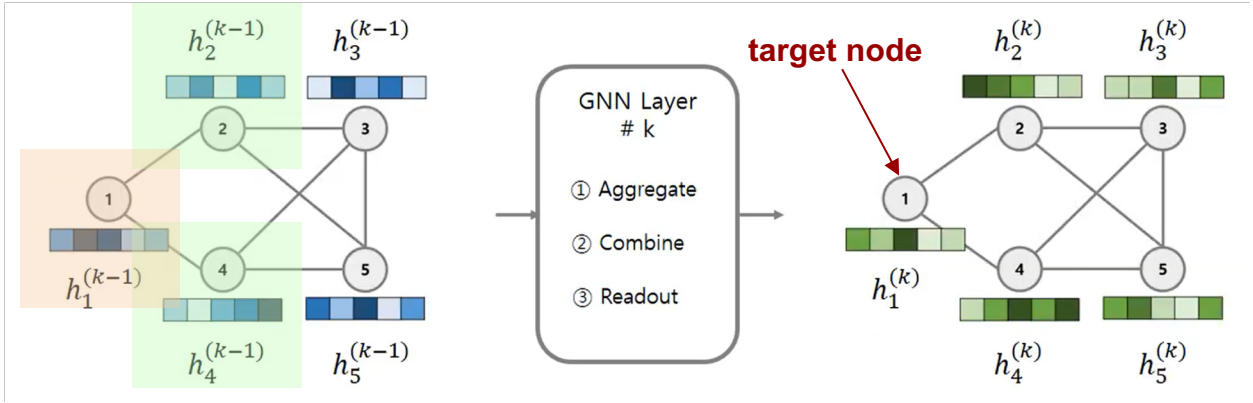


- GNN: ① Aggregate
  - 타겟 노드의 **이웃 노드들의 k-1 시점의 hidden state**를 결합



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

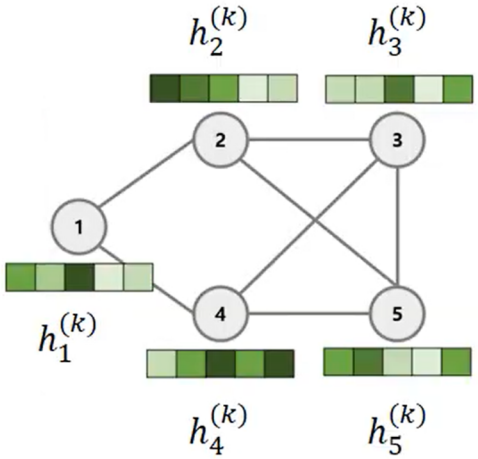
- GNN: ② Combine
  - k-1 시점 target node의 hidden state 와 ① aggregated information 을 사용하여 k 시점의 target node의 hidden state를 update



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

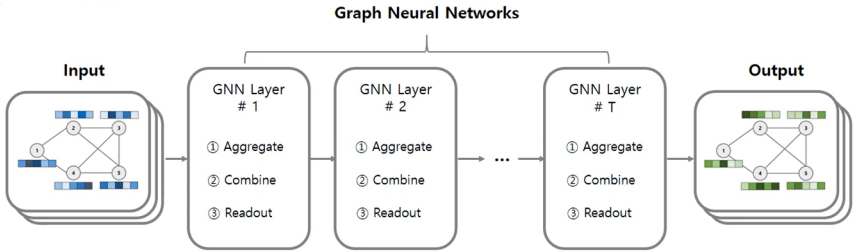
$$h_v^{(k)} = combine^k(a_u^{(k-1)}, h_v^{(k-1)})$$

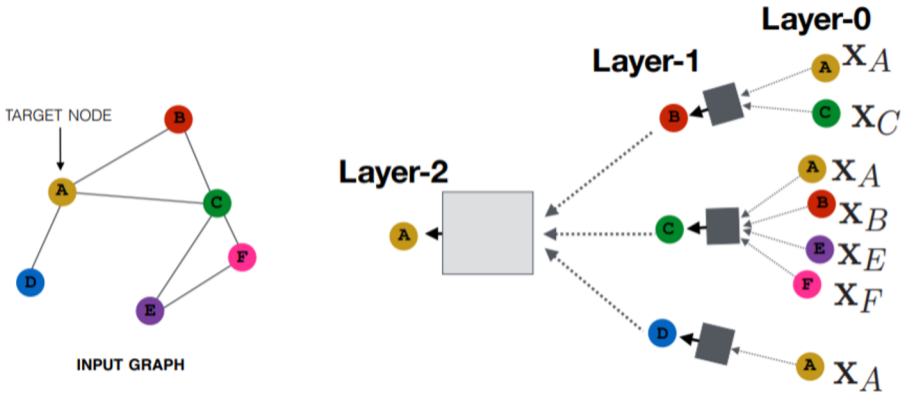
- GNN: ③ Readout
  - k 시점의 모든 Node 들의 hidden state를 결합하여 graph 의 hidden state 생성.
  - Graph level classification



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

- GNN Layer
  - CNN: Increase the receptive field
  - **GNN: Increase the hop of the graph**





- GNN Summary:
  ① Aggregate
  - 타겟 노드의 **이웃 노드들의 k-1 시점의 hidden state**를 결합

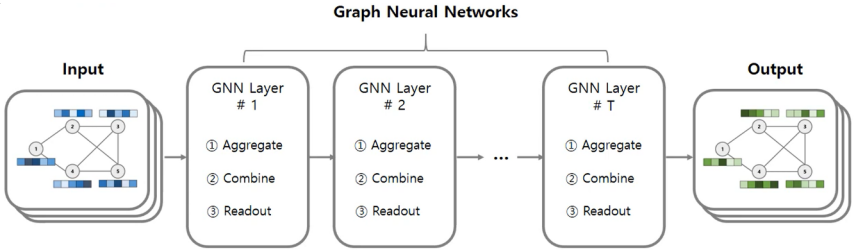
$$a_v^{(k-1)} = aggregate^k(\{h_u^{(k-1)}, \forall u \in N(v)\})$$

② Combine
- **k-1 시점 target node의 hidden state** 와 ① **aggregated information** 을 사용하여 k 시점의 target node의 hidden state를 update

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

③ Readout
- k 시점의 모든 Node 들의 hidden state를 결합하여 graph 의 hidden state 생성.
- Graph level classification

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



Graph Neural Network Overview

- GNN
  - **Aggregate, Combine function의 정의에 따라** 다양한 방식의 모델이 존재함.

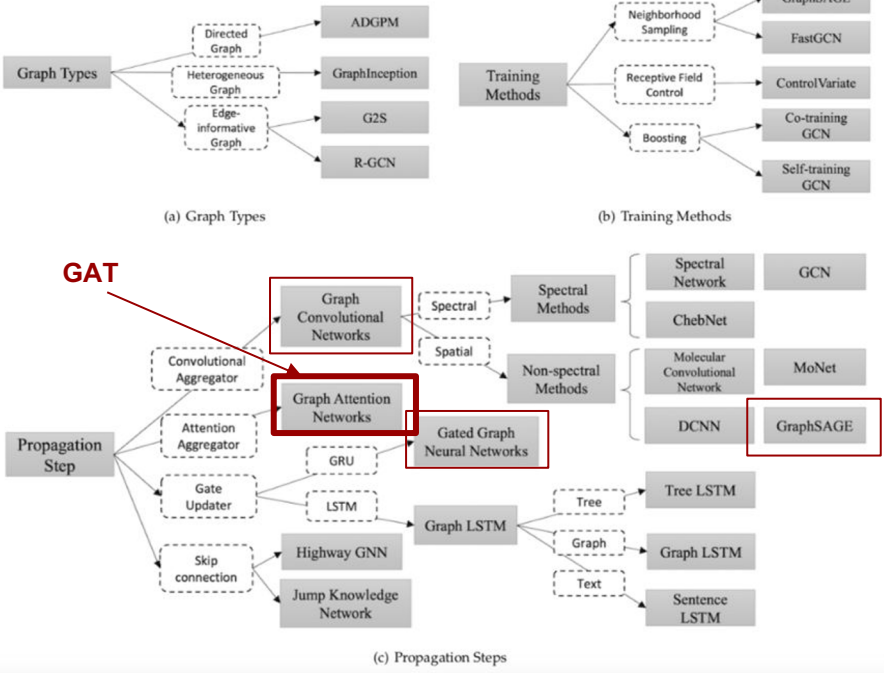


TABLE 2
Different variants of graph neural networks.

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	Neural FPs	$\mathbf{h}_{\mathcal{N}_v}^t = \mathbf{h}_v^{t-1} + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{h}_v^t = \sigma(\mathbf{h}_{\mathcal{N}_v}^t \mathbf{W}_L^{\mathcal{N}_v})$
	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	$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\})$	$\mathbf{h}_v^t = \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \| \mathbf{h}_{\mathcal{N}_v}^t]\right)$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_k]))}{\sum_{j\in\mathcal{N}_v}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\sum_{k\in\mathcal{N}_v} \alpha_{vk}\mathbf{W}\mathbf{h}_k\right)$ Multi-head concatenation: $\mathbf{h}_{\mathcal{N}_v}^t = \Big\|_{m=1}^{M} \sigma\left(\sum_{k\in\mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k\right)$ 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	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k\in\mathcal{N}_v} \mathbf{h}_k^{t-1} + \mathbf{b}$	$\mathbf{z}_v^t = \sigma(\mathbf{W}^z\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^z\mathbf{h}_v^{t-1})$ $\mathbf{r}_v^t = \sigma(\mathbf{W}^r\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^r\mathbf{h}_v^{t-1})$ $\mathbf{h}_v^t = \tanh(\mathbf{W}\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}))$ $\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \mathbf{h}_v^t$
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 [44]	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^i \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^o \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^u \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k\in\mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$

The graph neural network model (2009)

- vanilla GNN
  - The graph neural network model (2009)
  - http://persagen.com/files/misc/scarselli2009graph.pdf

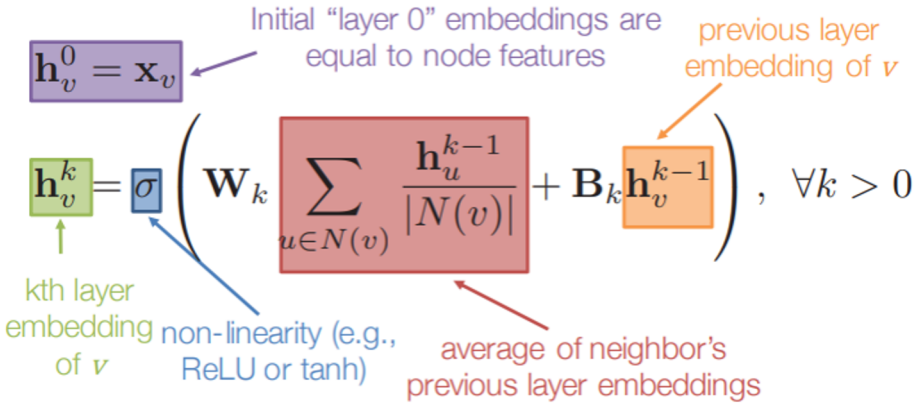
① Aggregate

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

② Combine

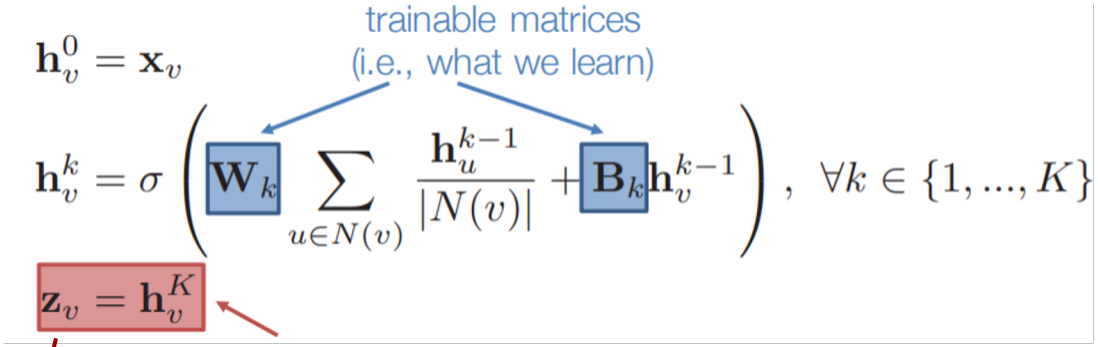
$$h_v^{(k)} = Relu\left(\mathrm{W}_{\mathrm{self}} h_v^{(k-1)} + \mathrm{W}_{\mathrm{neigh}} a_v^{(k-1)}\right)$$

- Basic approach: **Average neighbor messages and apply a neural network**



The graph neural network model (2009)

- Basic approach: **Average neighbor messages and apply a neural network**



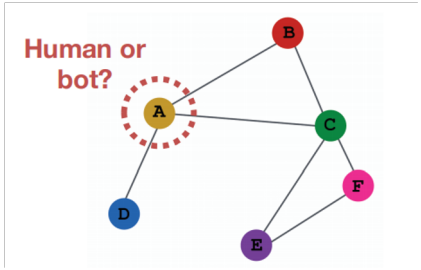
After K-layers of neighborhood aggregation, we get output embeddings for each node.

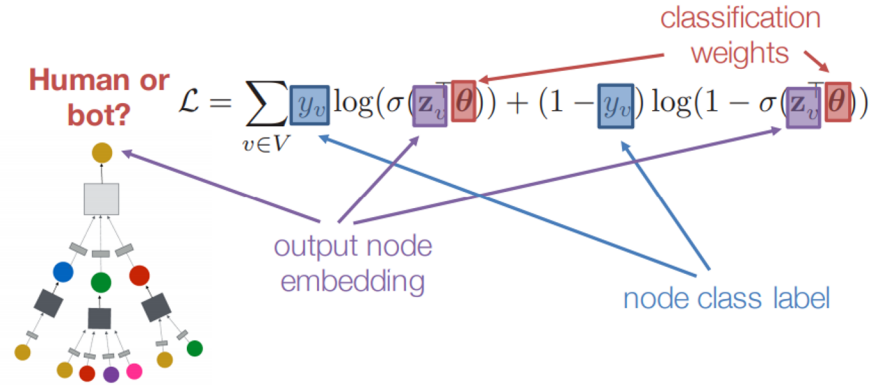
- Need to define a **loss function** on the embeddings, $\mathcal{L}(z_u)$.
- We can feed these embeddings into **any loss function** and run **stochastic gradient descent to train the aggregation parameters.**

The graph neural network model (2009)

- GNN: Training the Model
  - Unsupervised task
    - Graph structure 만 사용
    - Unsupervised loss function
      - a loss based on
        Random walks (node2vec, DeepWalk)
        Graph factorization
    - i.e. “similar” 노드가 similar embedding을 가지도록 model training

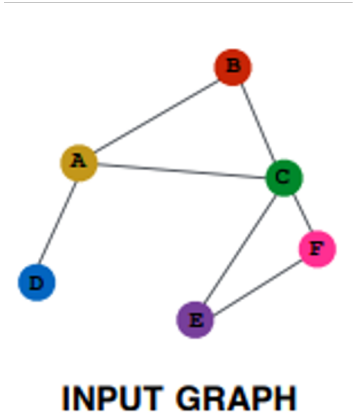
  - Supervised task
    - Directly train the model for a supervised task
    - e.g. node classification - an online social network
      - Cross-Entropy Loss



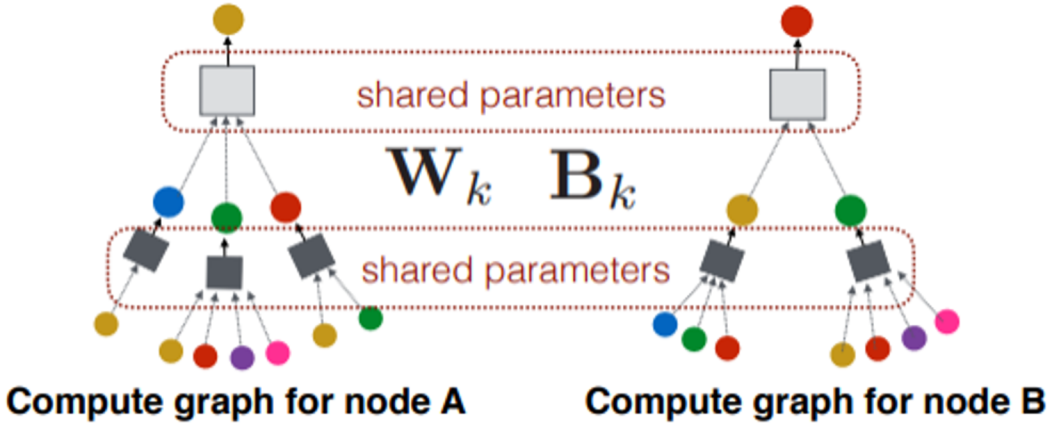


The graph neural network model (2009)

- Basic approach: Average neighbor messages and apply a neural network
- **The aggregation parameters are shared for all nodes.**
  - **So, we can generalize to unseen node**



INPUT GRAPH



Compute graph for node A

Compute graph for node B

Graph Attention Network (GAT)

Graph Neural Network Overview

- Graph Attention Network (GAT)
  - Graph Attention Networks (2018)
  - https://arxiv.org/pdf/1710.10903.pdf

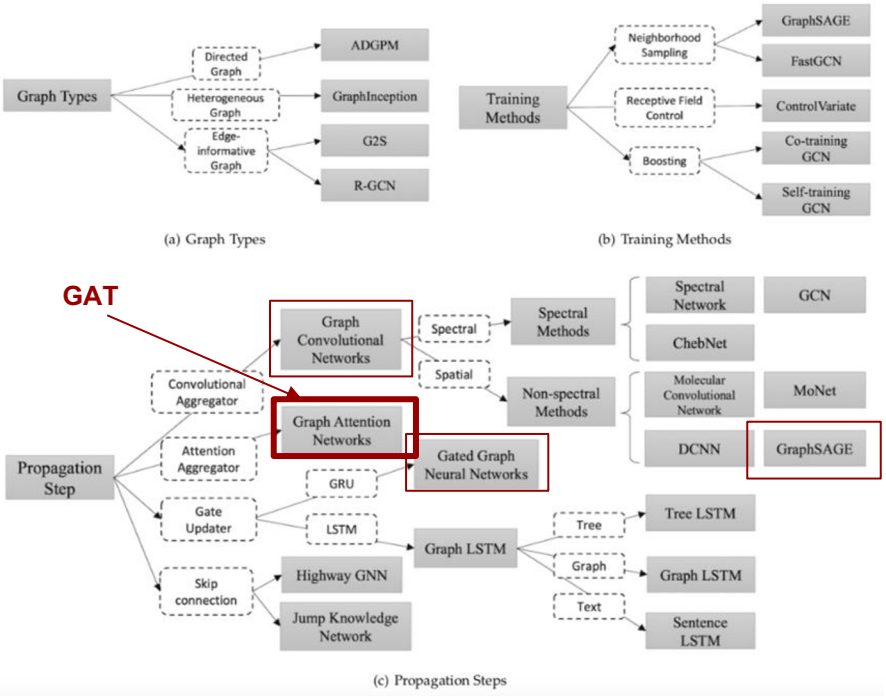


TABLE 2
Different variants of graph neural networks.

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	Neural FPs	$\mathbf{h}_{\mathcal{N}_v}^t = \mathbf{h}_v^{t-1} + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{h}_v^t = \sigma(\mathbf{h}_{\mathcal{N}_v}^t \mathbf{W}_L^{\mathcal{N}_v})$
	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	$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\})$	$\mathbf{h}_v^t = \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \| \mathbf{h}_{\mathcal{N}_v}^t]\right)$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_k]))}{\sum_{j\in\mathcal{N}_v}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\sum_{k\in\mathcal{N}_v} \alpha_{vk}\mathbf{W}\mathbf{h}_k\right)$ Multi-head concatenation: $\mathbf{h}_{\mathcal{N}_v}^t = \Big\|_{m=1}^{M} \sigma\left(\sum_{k\in\mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k\right)$ 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	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k\in\mathcal{N}_v} \mathbf{h}_k^{t-1} + \mathbf{b}$	$\mathbf{z}_v^t = \sigma(\mathbf{W}^z\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^z\mathbf{h}_v^{t-1})$ $\mathbf{r}_v^t = \sigma(\mathbf{W}^r\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^r\mathbf{h}_v^{t-1})$ $\mathbf{h}_v^t = \tanh(\mathbf{W}\mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}))$ $\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \mathbf{h}_v^t$
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 [44]	$\mathbf{h}_{\mathcal{N}_v}^{ti} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^i \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{to} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^o \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{k\in\mathcal{N}_v} \mathbf{U}_{m(v,k)}^u \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{ti} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{to} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k\in\mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$

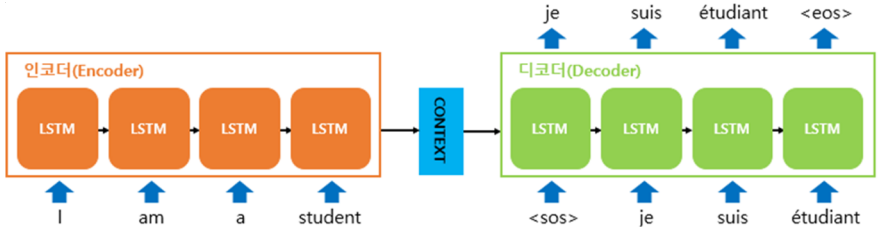
Attention

Attention Overview

- Attention Overview
  - RNN (seq2seq)
  - Attention mechanism
    - Neural Machine Translation by Jointly Learning to Align and Translate (2014)
    - https://arxiv.org/pdf/1409.0473.pdf
  - Transformer
    - Attention is all you need (2017)
    - https://arxiv.org/pdf/1706.03762.pdf

Attention Overview: RNN(seq2seq)

- Attention Overview : RNN(seq2seq)
  - seq2seq는 번역기에서 대표적으로 사용되는 모델.
  - seq2seq는 크게 2개의 아키텍처로 구성
    - 인코더 (Encoder)
      - 입력 문장의 모든 단어들을 순차적으로 입력받음.
      - 해당 단어 정보들을 압축해서 하나의 벡터로 만듦. -> 하나의 벡터 = 컨텍스트 벡터 (Context vector)
      - 입력 문장의 정보가 하나의 컨텍스트 벡터로 모두 압축되면, 인코더는 컨텍스트 벡터를 디코더로 전송.
    - 디코더 (Decoder)
      - 컨텍스트 벡터를 받음.
      - 번역된 단어를 한 개씩 순차적으로 출력.



- Context vector
  - Fixed size (고정된 사이즈)의 벡터
    - 문장이 길어지는 경우 => Context vector가 모든 내용을 충분히 함축하기에는 사이즈가 작다는 문제가 있음.
  - 인코더에서 나왔던 모든 state들을 활용하지 않음.
    - 단순히 마지막에 나온 state를 context vector 라고 하며, 하나의 context vector에 의해서 translation 이 이뤄짐.

- **Attention Mechanism**
  - Neural Machine Translation by Jointly Learning to Align and Translate (2014)
  - 모든 RNN cell의 state를 활용
  - 디코더에서 dynamic 하게 context vector를 활용하여 번역을 하면, Fixed size의 벡터 문제를 해결할 수 있음.

- **Attention**
  - **Key, Query, Value**
  - Dictionary 자료형의 결과 리턴 과정
    - ① Similarity(key, Query) : key와 Query의 유사도 계산
    - ② Sim * Value (sim, value) : 유사도와 value를 곱함
    - ③ Result(outputs) : 유사도와 value를 곱한 값의 합을 리턴

$$result = \sum_{i} similarity(key, query) * value$$

$$A(q, K, V) = \sum_{i} softmax(f(K, q))\, V$$

```
① def Similarity(Query, Key)
        if Query = Key:
            return 1
        else:
            return 0
```

```
② def SimXValue(Sim,Value)
        output = Sim X int(value)
        return output
```

```
③ def Result(outputs)
        return sum(outputs)
```

- **Key, Query, Value in Attention**
  - Attention: Query와 Key의 유사도를 계산한 후, value의 가중합을 계산하는 과정
  - Attention score: Value에 곱해지는 가중치

$$A(q,K,V) = \sum_i softmax(f(K,q))V$$

- Key, Value = Hidden states of encoder h{i} (각 time-step 별 인코더의 출력)
- Query = Hidden state of decoder s{2} (현재 time-step의 디코더의 출력)
- feature = Context vector at time-step 2

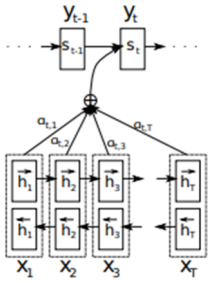


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

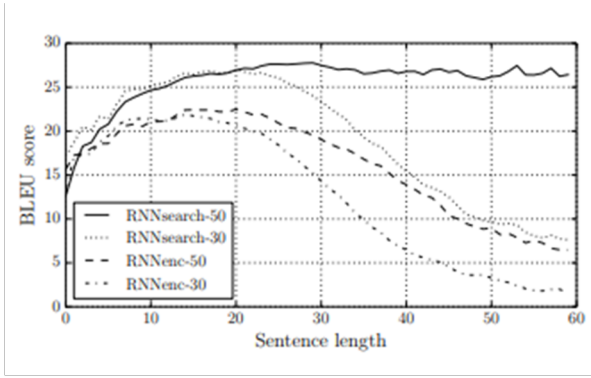
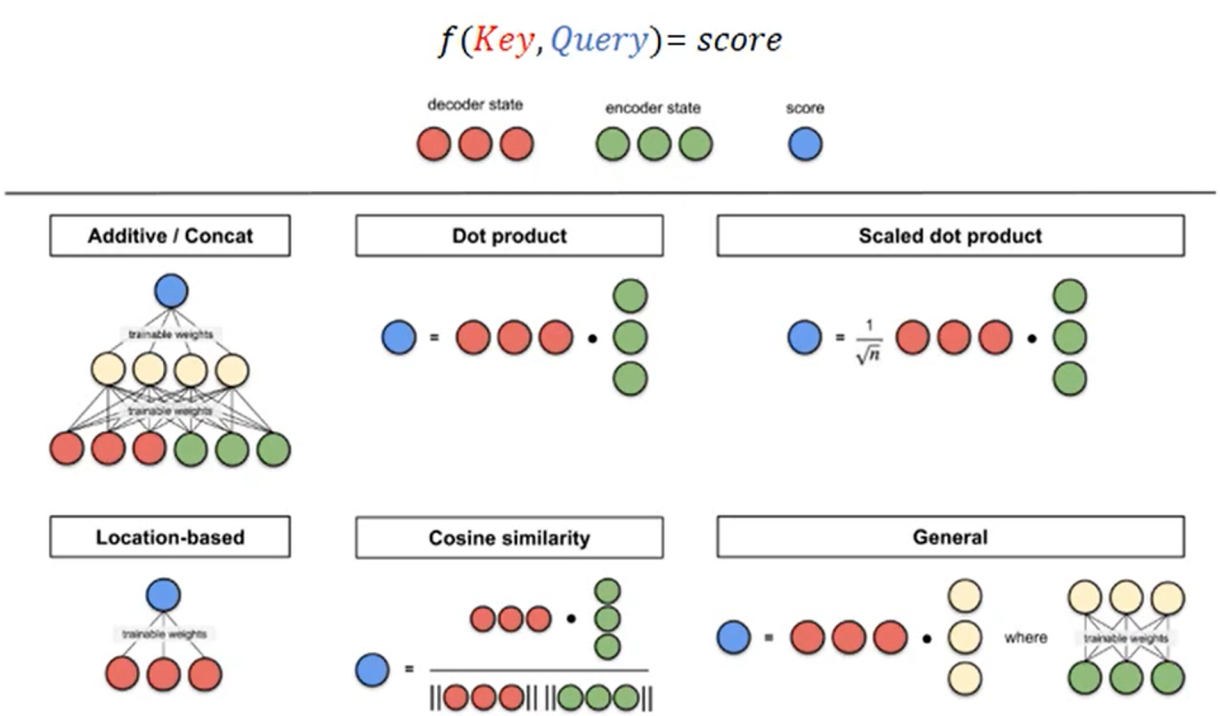


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

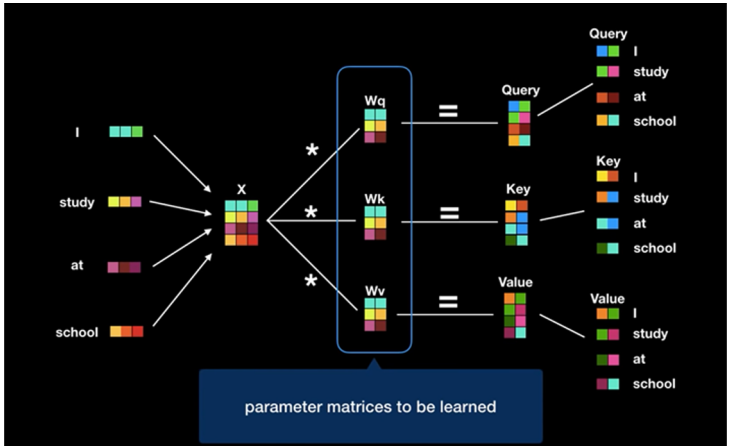
- **하지만, 여전히 RNN cell을 순차적으로 계산해서 느림. (병렬처리 불가능)**
- 성능도 더 높일 수 있는 가능성이 있을 듯.

- **Similarity Function (Alignment model)**
  - Vector 간 유사도를 계산하는 다양한 방법을 similarity function 으로 사용 가능.
  - Bahdanau Attention (2014), Graph Attention Network (2018) -> Addictive/Concat
  - Transformer (2017) -> Scaled-dot product



- **Transformer**
  - Attention is all you need (2017)
  - Attention 만으로도, 입력 데이터에서 중요한 정보들을 찾아내 단어를 encoding -> RNN 제거
  - RNN의 순차적인 계산은 transformer에서 행렬 곱으로 한번에 처리됨

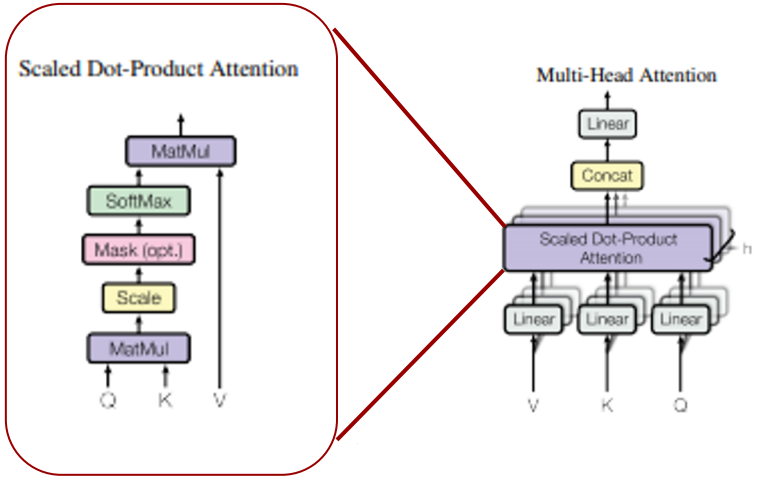
- Key, Value, Query = Hidden state of word embedding vector
- Similarity function = Scaled-dot product



$$A(q, K, V) = \sum_i softmax(f(K, q))\, V$$

$$f(K, Q) = QK^{\mathrm{T}} \quad (K = XW^K, Q = XW^Q, V = XW^V)$$

- **Transformer**



$$A(q, K, V) = \sum_i softmax(f(K, q))\, V$$

1. MatMul: Key와 Query의 similarity 구함 (dot-product attention)
2. Scale: key의 차원수에 루트 값을 취한 값으로 나눠줌.
3. Softmax: 계산한 유사도에 softmax 값을 취해 확률값
4. Softmax 확률값과 value를 곱해 최종 feature를 구함.

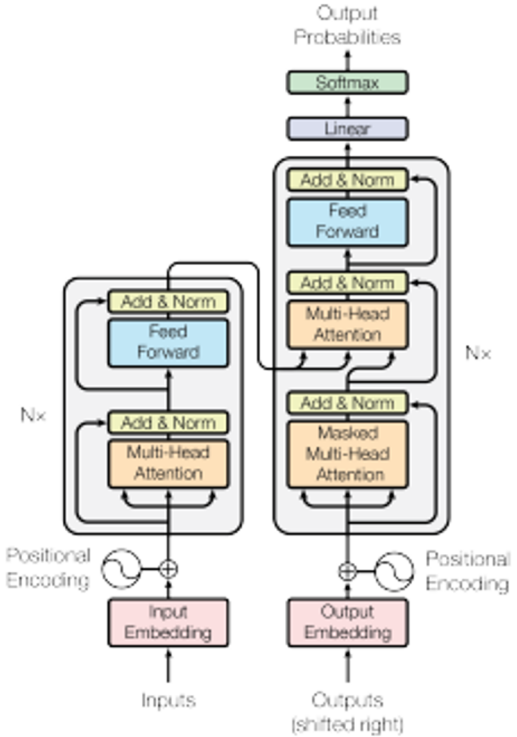
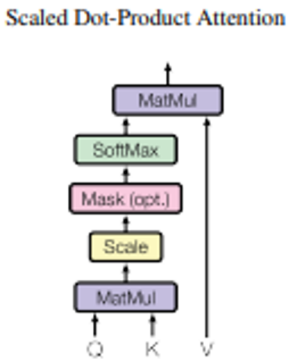
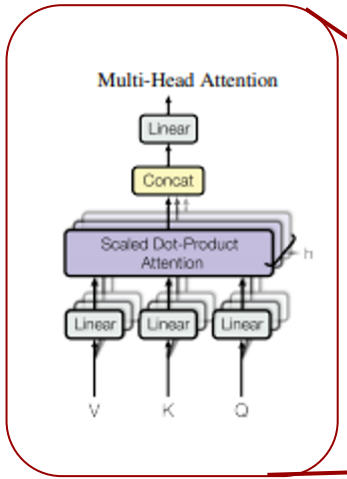


Figure 1: The Transformer - model architecture.

- **Transformer**





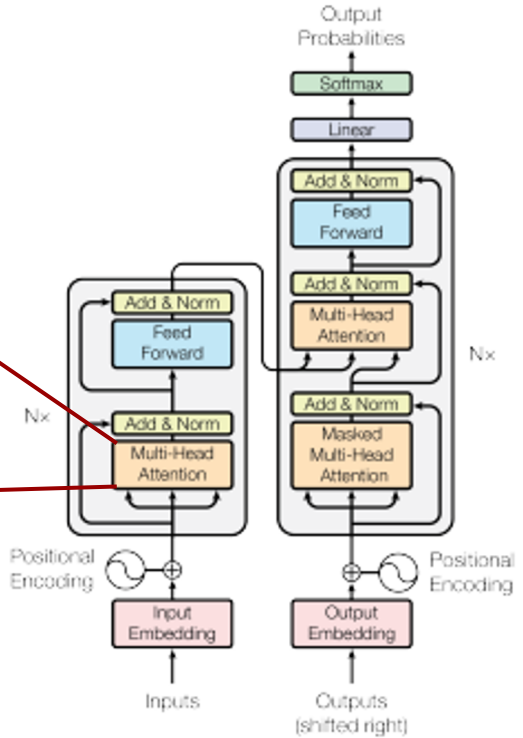


Figure 1: The Transformer - model architecture.

$$Q` = QW_i^Q \quad K` = KW_i^K \quad V` = VW_i^V \quad (i = 1 \dots h)$$

Multi-Head Attention

- Scaled Dot-product attention을 여러 번 수행

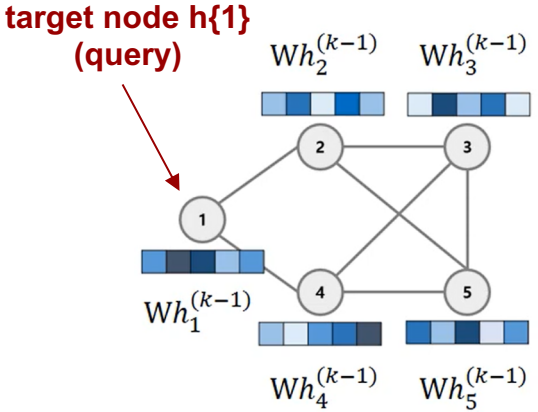
- Key, Query, Value를 linear transformation 해서 계산.
- weight matrix를 여러개를 만듦. (i.e. X 라는 값을 더 다양하게 표현.)

Graph Attention Networks (2018)

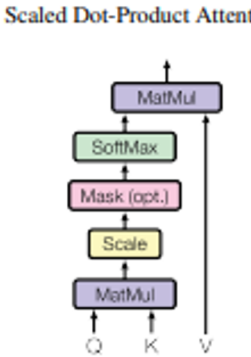
- **GAT architecture**
  ① Aggregate
  - 타겟 노드의 **이웃 노드들의 k-1 시점의 hidden state**를 결합

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

  - **Attention**
    - 대상: 타겟 노드와 이웃 노드를 모두 포함한 hidden embedding
    - Key, Query, Value = k-1 시점의 hidden state (**self-attention**)
    - Similarity Function(f) = 1 - layer Feed - Forward NN
    - Activation Function = LeakyRelu



e_{11}	e_{12}		e_{14}	

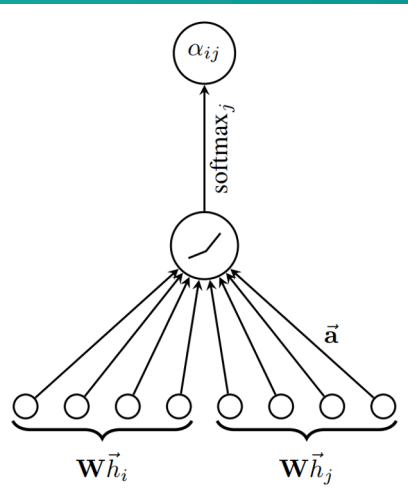


$$A(q,K,V) = \sum_i softmax(f(K,q))\,V$$

- **GAT architecture**
  ① Aggregate
  - 타겟 노드의 **이웃 노드들의 k-1 시점의 hidden state**를 결합

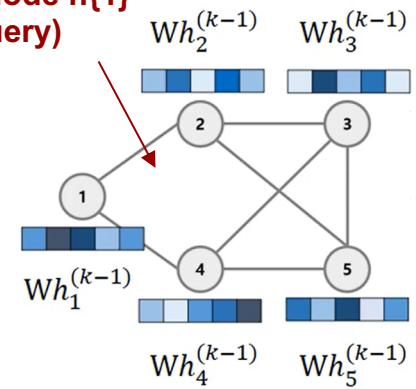
$$a_v^{(k-1)} = Attention(\{h_u^{(k-1)}, u \in N(v) \cup \{v\}\})$$

  - **Attention**
    - 대상: 타겟 노드와 이웃 노드를 모두 포함한 hidden embedding
    - Key, Query, Value = k-1 시점의 hidden state (**self-attention**)
    - Similarity Function(f) = 1 - layer Feed - Forward NN
    - Activation Function = LeakyRelu



$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

target node h{1}
(query)



e_{11}	e_{12}		e_{14}	

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

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

- **GAT architecture**

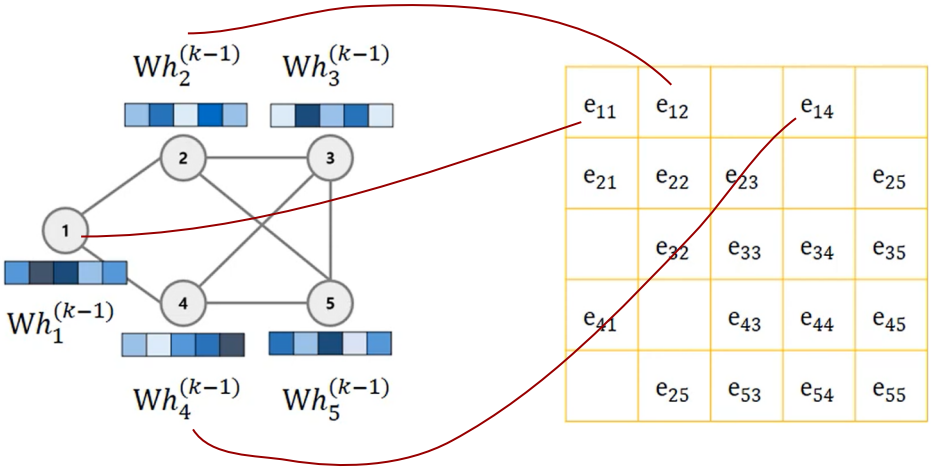
  ② Combine

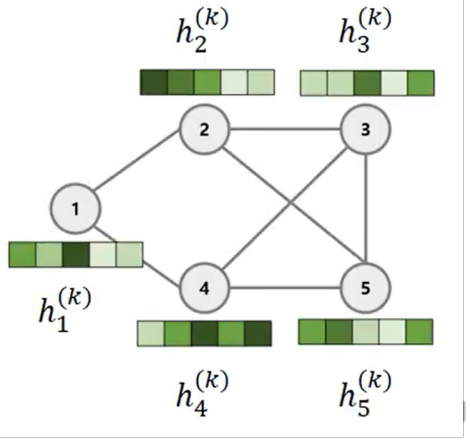
  - k-1 시점 target node의 hidden state 와 ① aggregated information 을 사용하여 k 시점의 target node의 hidden state를 update

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

  - ① aggregated information & **k-1 hidden state (=> Value)** 의 weight-sum

$$h_v^{(k)} = \sum_{u \in N(v) \cup \{v\}} a_u^{(k-1)} h_u^{(k-1)}$$



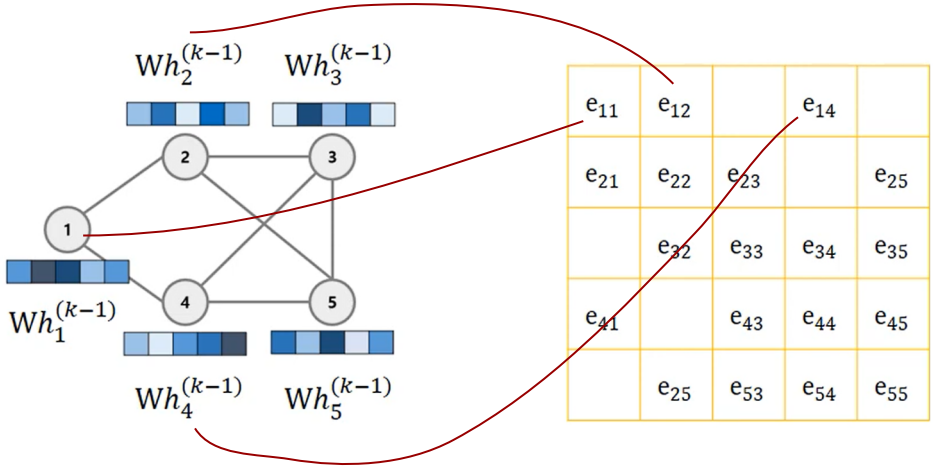


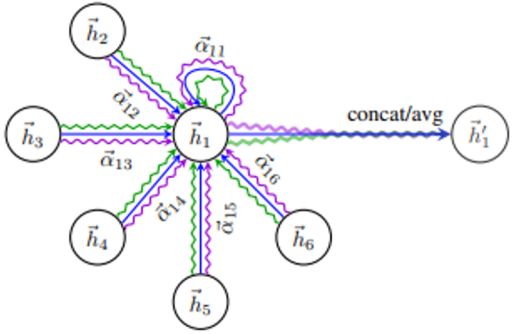
- **GAT architecture**
  ② Combine
  - k-1 시점 target node의 hidden state 와 ① aggregated information 을 사용 k 시점의 target node의 hidden state를 update

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

  - ① aggregated information & **k-1 hidden state (=> Value)** 의 weight-sum

$$h_v^{(k)} = \sum_{u \in N(v) \cup \{v\}} a_u^{(k-1)} h_u^{(k-1)}$$





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

$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

$$\vec{h}'_i = \sigma\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

- GAT Evaluation
  - Transductive learning
    - semi-supervised learning 방식과 유사.
    - 테스트 데이터의 레이블은 없지만 그 안에서 유의미한 패턴 추출이 가능.
    - 모델을 구축하지 않아서, 새로운 데이터가 입력을 들어왔을 때, 알고리즘을 처음부터 재시작할 필요성이 존재하여 계산 효율이 낮음 (high computational cost)
    - **Graph learning의 관점에서**
      - **그래프 상의 일부 노드와 일부 에지의 ground truth를 아는 상태에서 나머지 노드와 에지의 값을 추정하는 것.** 즉, known 노드와 에지를 가지고 supervised learning을 해서, 연결된 unknown 노드와 에지를 prediction 하는 방식
  - Inductive learning
    - supervised learning 방식과 유사
    - 이미 레이블이 된 훈련 데이터셋을 활용하여 모델 구축
    - **Graph learning의 관점에서**
      - **ground truth를 알고 있는 graph 들에 대해서 모델을 학습한 후, 전혀 새로운 graph 에 대해 노드와 에지의 값을 추정하는 것.**

- GAT Evaluation
  - Datasets

Table 1: Summary of the datasets used in our experiments.

	Cora	Citeseer	Pubmed	PPI
Task	Transductive	Transductive	Transductive	Inductive
# Nodes	2708 (1 graph)	3327 (1 graph)	19717 (1 graph)	56944 (24 graphs)
# Edges	5429	4732	44338	818716
# Features/Node	1433	3703	500	50
# Classes	7	6	3	121 (multilabel)
# Training Nodes	140	120	60	44906 (20 graphs)
# Validation Nodes	500	500	500	6514 (2 graphs)
# Test Nodes	1000	1000	1000	5524 (2 graphs)

- GAT Experimental setup

Transductive learning For the transductive learning tasks, we apply a two-layer GAT model. Its architectural hyperparameters have been optimized on the Cora dataset and are then reused for Citeseer. The first layer consists of $K = 8$ attention heads computing $F' = 8$ features each (for a total of 64 features), followed by an exponential linear unit (ELU) (Clevert et al., 2016) nonlinearity. The second layer is used for classification: a single attention head that computes C features (where C is the number of classes), followed by a softmax activation. For coping with the small training set sizes, regularization is liberally applied within the model. During training, we apply L_2 regularization with $\lambda = 0.0005$. Furthermore, dropout (Srivastava et al., 2014) with $p = 0.6$ is applied to both layers' inputs, as well as *to the normalized attention coefficients* (critically, this means that at each training iteration, each node is exposed to a stochastically sampled neighborhood). Similarly as observed by Monti et al. (2016), we found that Pubmed's training set size (60 examples) required slight changes to the GAT architecture: we have applied $K = 8$ output attention heads (instead of one), and strengthened the L_2 regularization to $\lambda = 0.001$. Otherwise, the architecture matches the one used for Cora and Citeseer.

Inductive learning For the inductive learning task, we apply a three-layer GAT model. Both of the first two layers consist of $K = 4$ attention heads computing $F' = 256$ features (for a total of 1024 features), followed by an ELU nonlinearity. The final layer is used for (multi-label) classification: $K = 6$ attention heads computing 121 features each, that are averaged and followed by a logistic sigmoid activation. The training sets for this task are sufficiently large and we found no need to apply L_2 regularization or dropout—we have, however, successfully employed skip connections (He et al., 2016) across the intermediate attentional layer. We utilize a batch size of 2 graphs during training. To strictly evaluate the benefits of applying an attention mechanism in this setting (i.e. comparing with a near GCN-equivalent model), we also provide the results when a *constant attention mechanism*, $a(x, y) = 1$, is used, with the same architecture—this will assign the same weight to every neighbor.

Both models are initialized using Glorot initialization (Glorot & Bengio, 2010) and trained to minimize cross-entropy on the training nodes using the Adam SGD optimizer (Kingma & Ba, 2014) with an initial learning rate of 0.01 for Pubmed, and 0.005 for all other datasets. In both cases we use an early stopping strategy on both the cross-entropy loss and accuracy (transductive) or micro-F_1 (inductive) score on the validation nodes, with a patience of 100 epochs[1].

- GAT Evaluation
  - Results

Table 2: Summary of results in terms of classification accuracies, for Cora, Citeseer and Pubmed. GCN-64* corresponds to the best GCN result computing 64 hidden features (using ReLU or ELU).

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%

Table 3: Summary of results in terms of micro-averaged F_1 scores, for the PPI dataset. GraphSAGE* corresponds to the best GraphSAGE result we were able to obtain by just modifying its architecture. Const-GAT corresponds to a model with the same architecture as GAT, but with a constant attention mechanism (assigning same importance to each neighbor; GCN-like inductive operator).

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

End of the Document