



10주차 세션

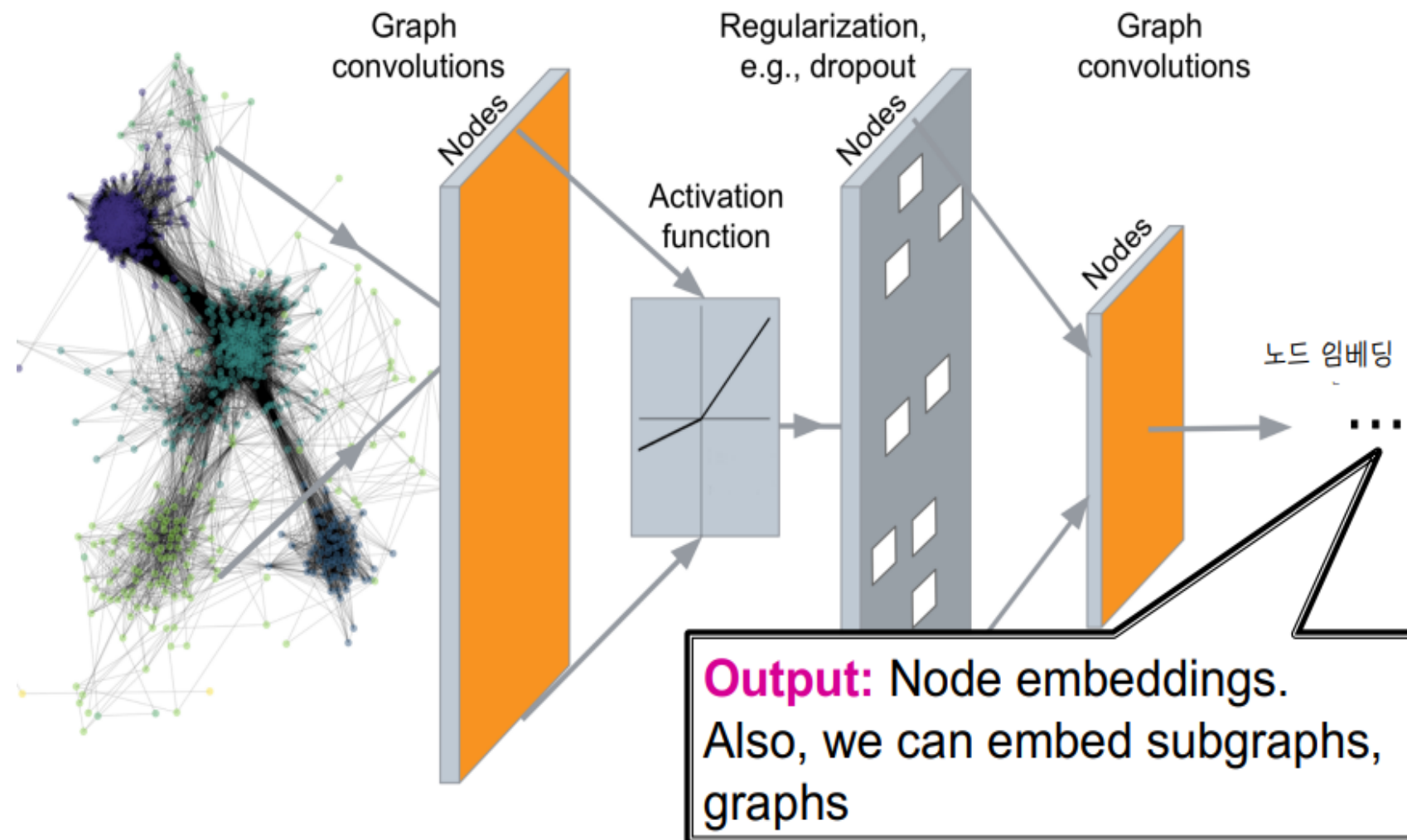
DL팀 이다현, 최예은

General Perspective

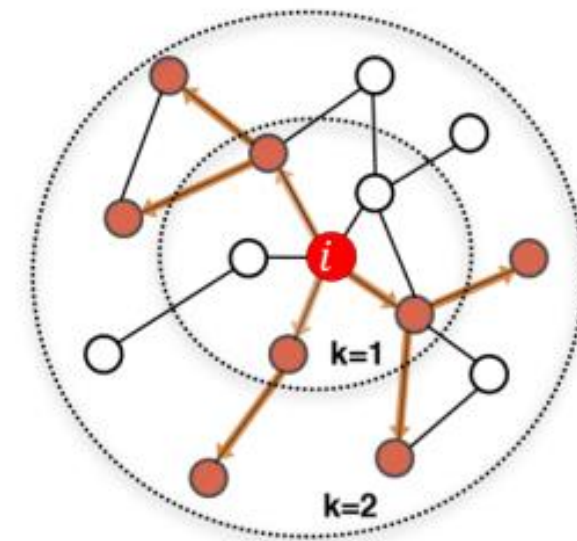


1.1 Recap

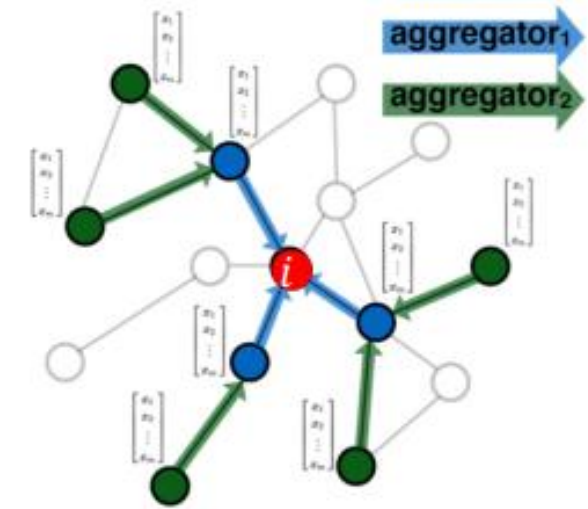
✓ Deep Graph Encoders



지난시간에 GNN 을 통한 임베딩에 관해 학습함

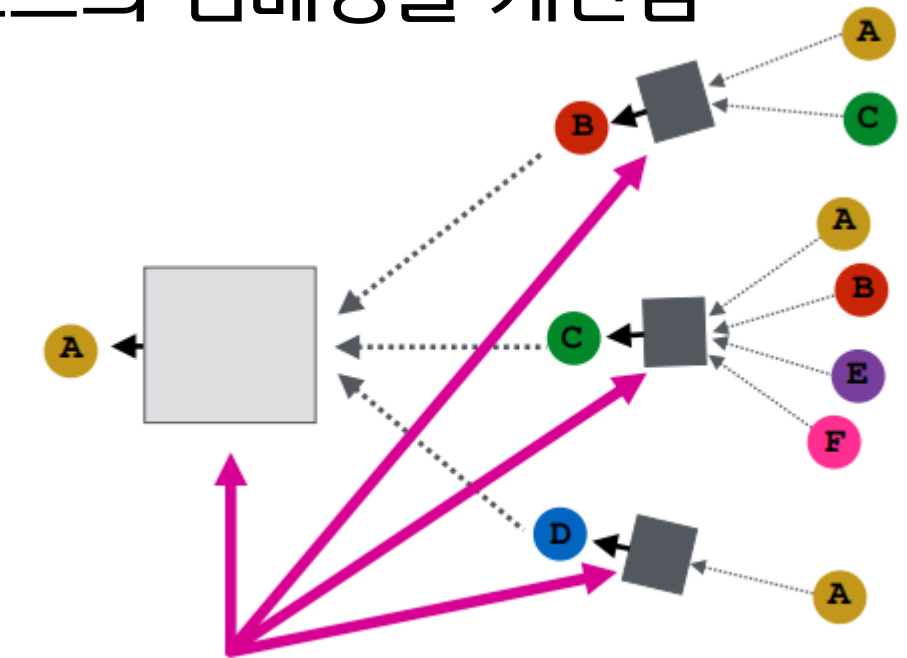
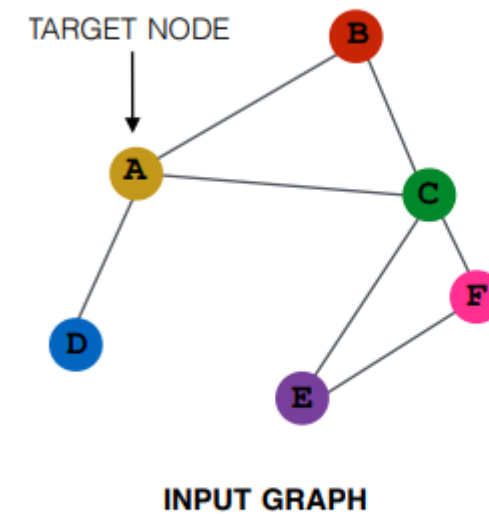


Determine node computation graph



Propagate and transform information

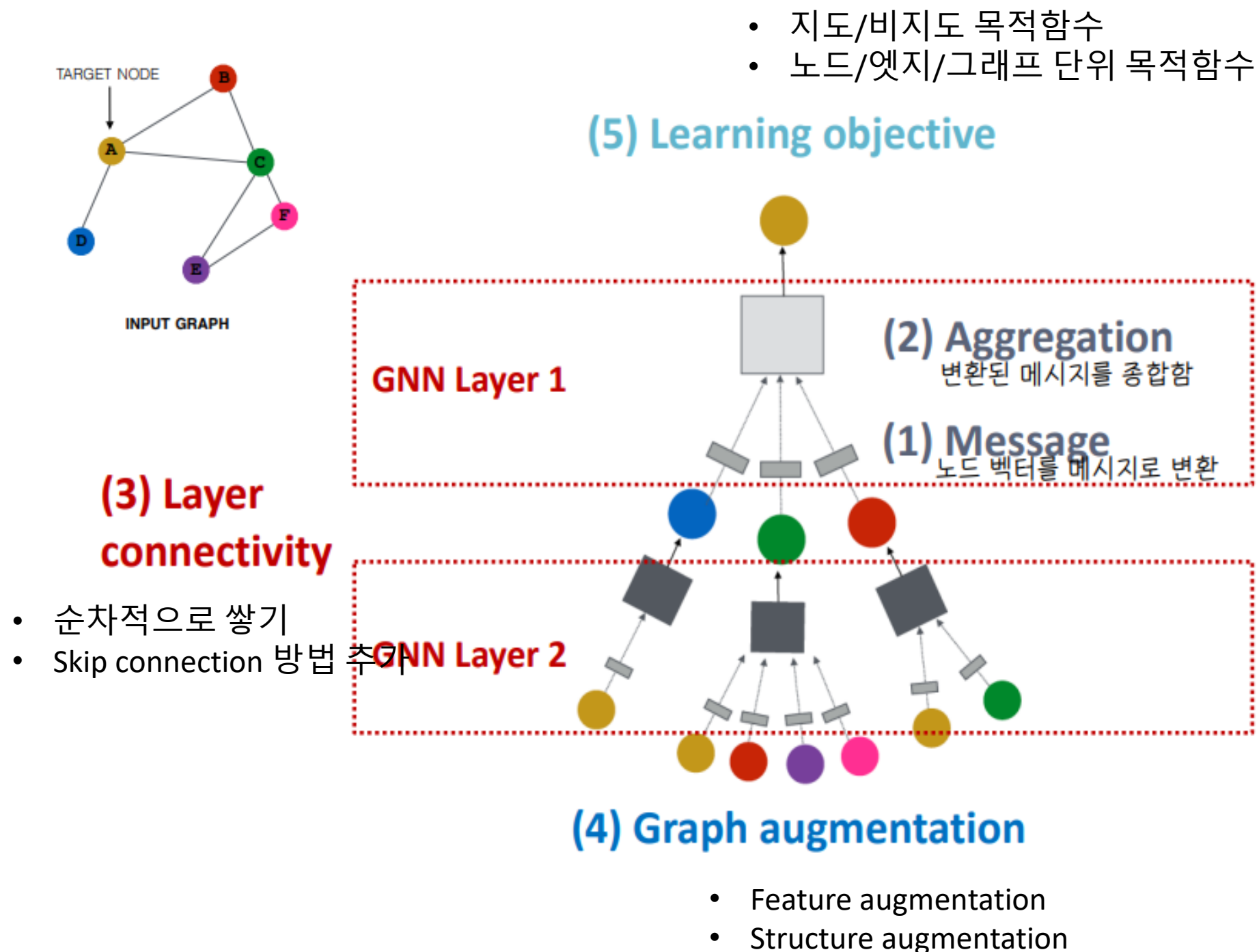
이웃노드로부터 특정 노드의 임베딩을 계산함



Neural networks

1.2 GNN Framework

✓ 5단계의 구조를 가짐



① Message + ② Aggregation = GNN

의 뼈대 이 과정에 대한 정의에 따라

신경망 종류가 세분화됨 : GCN,

GraphSAGE, GAT ...

③ layer connectivity : 다중 레이어를

쌓는 방법에 대한 논의

④ Graph augmentation : 효율적으로

계산 그래프를 수정하는 방법에 대한 논의

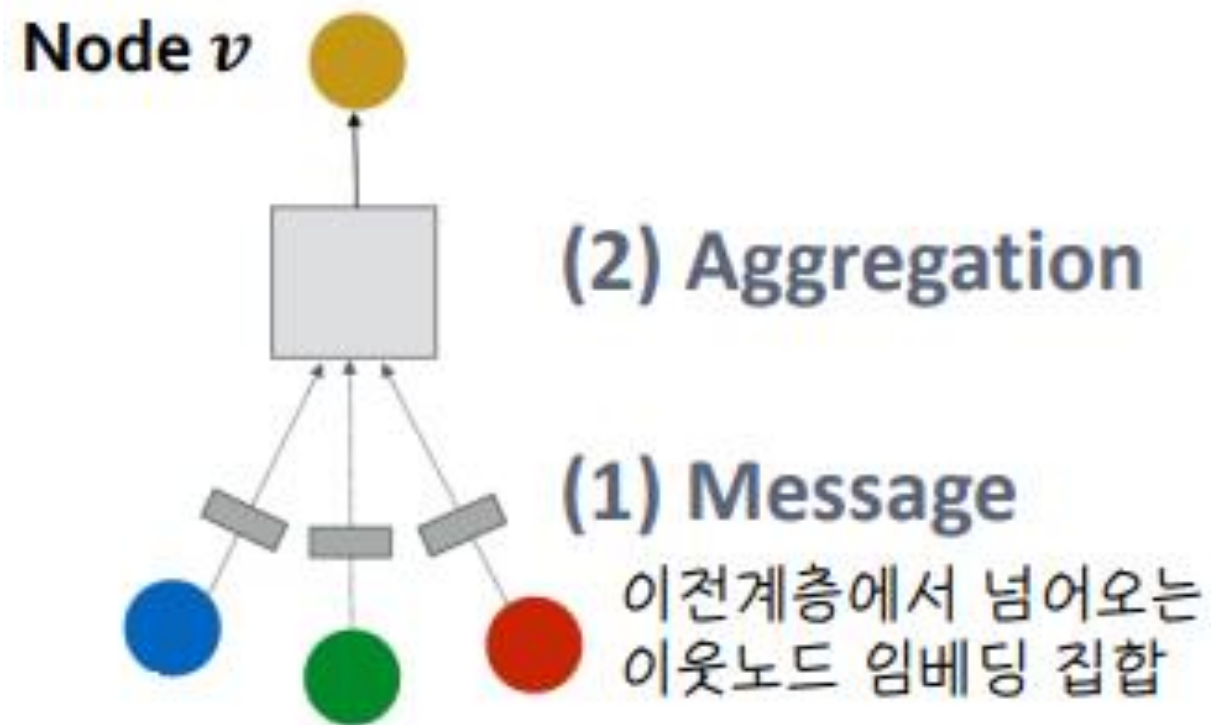
⑤ Learning objective : 목적함수에 대한

논의

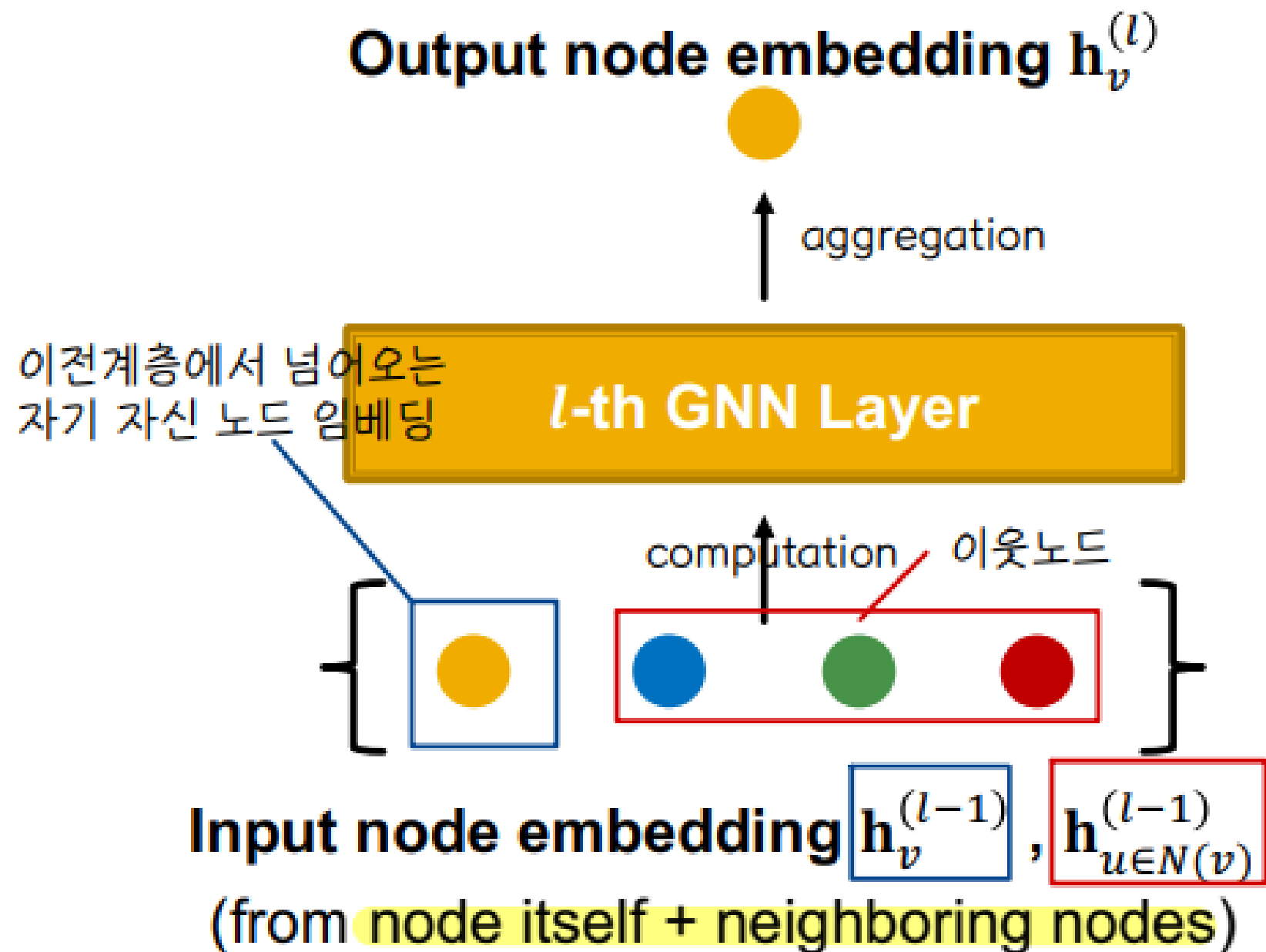
Single Layer of a GNN



2.1 Single GNN layer

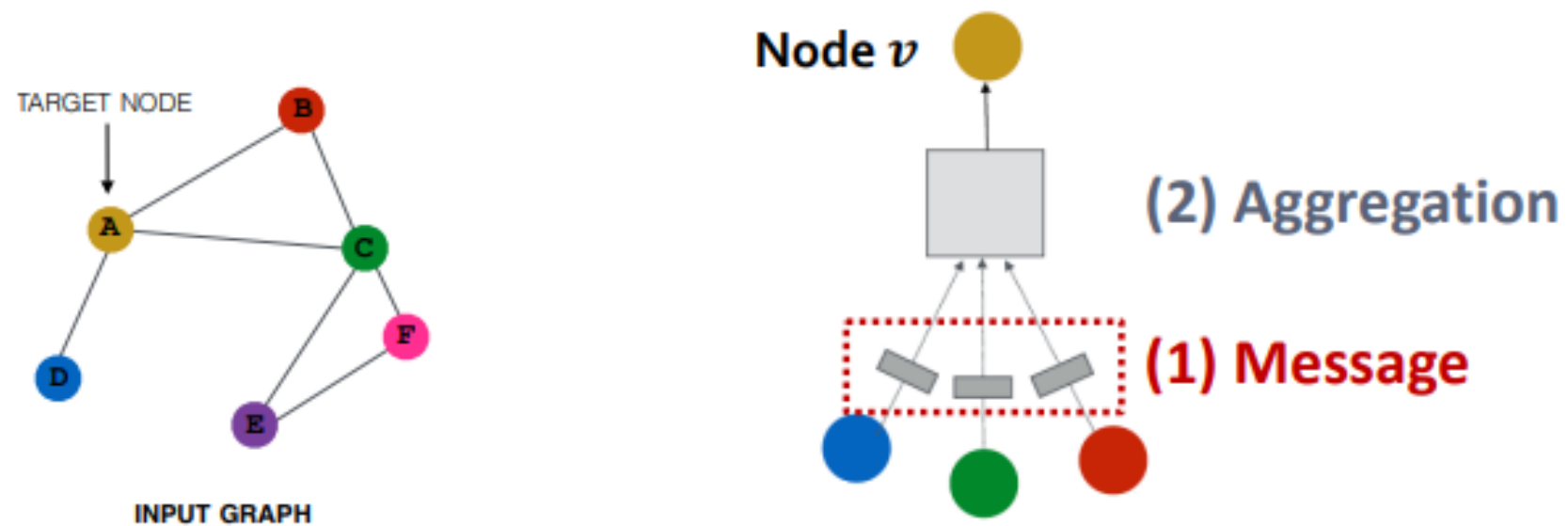


✓ vector 집합을 하나의 단일한 벡터로 압축하자



2.2 Message Computation

✓ MSG function



$$\mathbf{m}_u^{(l)} = \text{MSG}^{(l)} \left(\mathbf{h}_u^{(l-1)} \right)$$

- 이전노드에서 넘어온 각 노드의 정보 표현을 변형하여 다음 층으로 보낼 메시지를 생성함

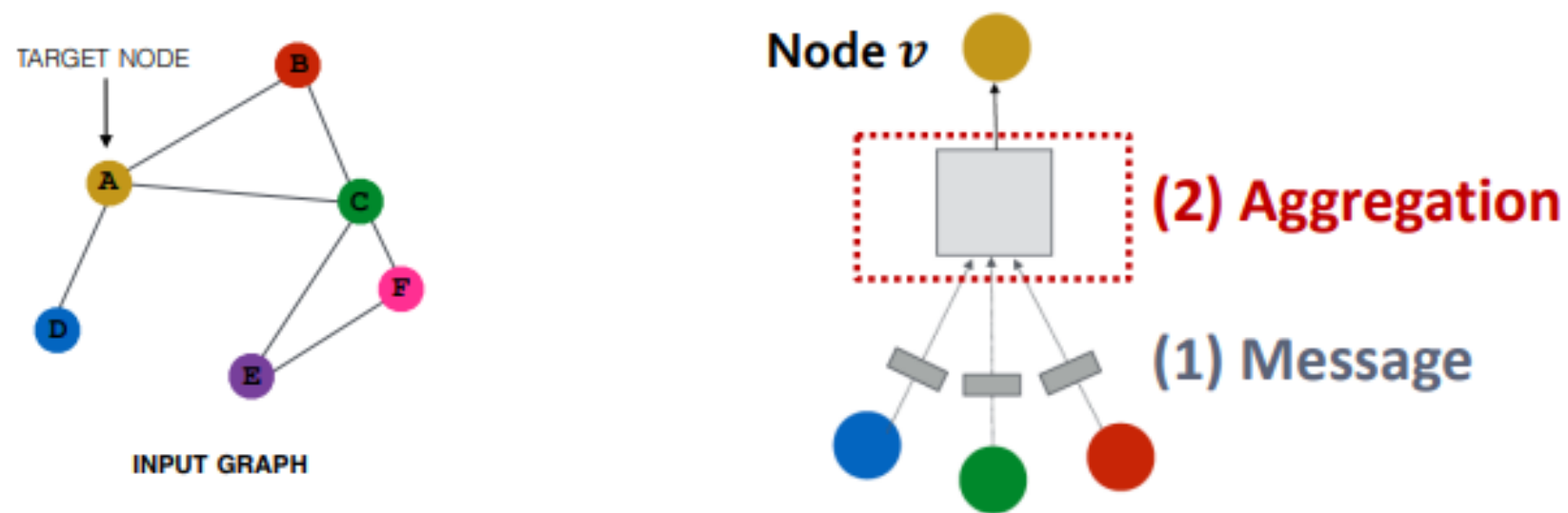
👁👁 MSG 예시

$$\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$$

- weight matrix 형태로 구성된 노드 feature 를 임베딩 벡터에 linear 하게 곱해서 변형하는 방법

2.2 Message Aggregation

✓ AGG function



$$\mathbf{h}_v^{(l)} = \underline{\text{AGG}}^{(l)} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right)$$

- 특정 노드 v 에 대해 이웃노드들인 u 의 메시지를 모두 집계해 하나의 벡터로 압축하는 과정

👁👁 AGG 예시

$$\mathbf{h}_v^{(l)} = \text{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$$



- sum, mean, max 함수와 같이
순서에 영향을 받지 않는
(permutation invariant) 함수라는
특징을 가져야 함

2.2 Message Aggregation

✓ 문제점 $\mathbf{h}_v^{(l)} = \text{AGG}^{(l)} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right)$

- 이웃노드에 대한 정보만 계산되어 자기 자신에 대한 정보가 소실될 수 있다.

👁👁 해결책 • 자기 자신에 대한 정보를 추가하자 : W, B

(1) Message  $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$ (이웃노드에 대한 가중치)  $\mathbf{m}_v^{(l)} = \mathbf{B}^{(l)} \mathbf{h}_v^{(l-1)}$ (자기 자신에 대한 가중치)

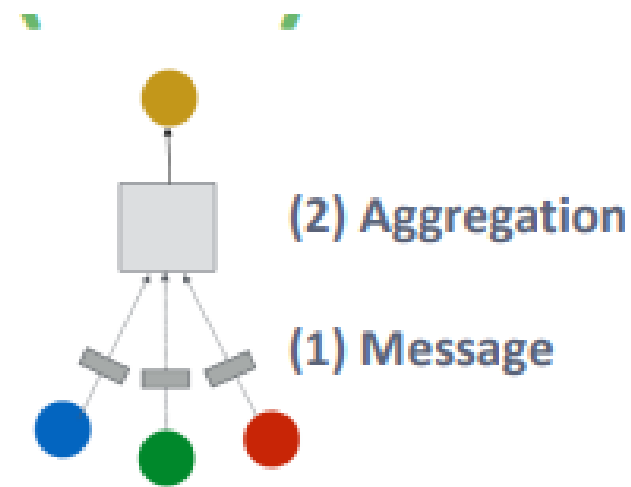
(2) Aggregation

$$\mathbf{h}_v^{(l)} = \text{CONCAT} \left(\underbrace{\text{AGG} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right)}_{\text{First aggregate from neighbors}}, \underbrace{\mathbf{m}_v^{(l)}}_{\text{Then aggregate from node itself}} \right)$$

(1), (2) 과정은 선형계산이기 때문에, 비선형성을 부여하기 위해 활성화 함수를 통과시키기도 함
: $\sigma(\cdot)$: ReLU(\cdot), Sigmoid(\cdot), ...

2.3 GNN layer : GCN

$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$



① Message

- 각 노드의 메시지는 node degree 로 정규화를 진행함

$$\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$$

② Aggregation

- 이웃노드 메시지를 합산하고 활성화함수를 통과 $\mathbf{h}_v^{(l)} = \sigma \left(\text{Sum} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right) \right)$

2.3 GNN layer : GraphSAGE

✓ GCN 을 기반으로 확장한 모델

$$\mathbf{h}_v^{(l)} = \sigma \left(\mathbf{W}^{(l)} \cdot \text{CONCAT} \left(\mathbf{h}_v^{(l-1)}, \text{AGG} \left(\left\{ \mathbf{h}_u^{(l-1)}, \forall u \in N(v) \right\} \right) \right) \right)$$

• 이웃 노드로부터 정보를 합계

$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \text{AGG} \left(\left\{ \mathbf{h}_u^{(l-1)}, \forall u \in N(v) \right\} \right)$$

• 자기 자신에 대한 정보를 concat

$$\mathbf{h}_v^{(l)} \leftarrow \sigma \left(\mathbf{W}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)}) \right)$$

👁👁 AGG 함수가 될 수 있는 것들

• Mean

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}$$

(GCN 과 동일한 방법)
이웃들의 가중 평균값을 구함
Aggregation **Message computation**

• Pool

$$\text{AGG} = \text{Mean}(\{\text{MLP}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

(2)mean 혹은 max 함수로 집계
(1)MLP 로 비선형적으로 만들고
Aggregation **Message computation**

• LSTM

$$\text{AGG} = \text{LSTM}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$$

이웃들로부터 오는 메시지 sequence model 에 LSTM 을 사용
(이때 sequential model 은 순서가 있으므로 이를 order invariant 하게 만들기 위해 이웃들의 순서를 섞어준다)
Aggregation

2.3 GNN layer : GraphSAGE

✓ | 2 normalization

$$\mathbf{h}_v^{(l)} \leftarrow \frac{\mathbf{h}_v^{(l)}}{\|\mathbf{h}_v^{(l)}\|_2} \quad \forall v \in V \text{ where } \|u\|_2 = \sqrt{\sum_i u_i^2} \text{ } (\ell_2\text{-norm})$$

- 모든 임베딩 벡터에 L2 규제를 적용할 수 있다.
- L2 규제를 적용하면 모든 벡터는 길이가 1인 벡터값을 가지게 된다.
- 기존 임베딩 벡터의 크기나 길이가 많이 다른 경우에는 정규화를 통해 성능을 향상시킬 수 있다.

(항상 성능이 향상하는 것은 아님)

복습과제

```
dataset = Planetoid("/tmp/Cora", name="Cora")
print(f'정규화 없이 행렬의 각 행의 값 합산 결과 : {dataset[0].x.sum(dim=-1)}')

dataset = Planetoid("/tmp/Cora", name="Cora", transform = T.NormalizeFeatures()) # 🍌
print(f'정규화를 적용해 행렬의 각 행의 값 합산 결과 : {dataset[0].x.sum(dim=-1)}') # dim = a
x18
```

☞ 정규화 없이 행렬의 각 행의 값 합산 결과 : tensor([9., 23., 19., ..., 18., 14., 13.])
정규화를 적용해 행렬의 각 행의 값 합산 결과 : tensor([1.0000, 1.0000, 1.0000, ..., 1.0000, 1.0000, 1.0000])

2.3 GNN layer : GAT

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights 이웃노드에 대한 중요도를 나타낸 값

- GCN/GraphSAGE에서는 중요도를 노드의 차수로 나눠서 표현한 것 $\nearrow \alpha_{vu} = \frac{1}{|N(v)|}$

☞ 노드의 차수로 중요도를 설정하면 그래프의 구조적인 특징만 반영하게 되고 노드 v 에 의존적이게 된다. 또한 노드 v 의 모든 이웃노드는 중요도 값이 동일하게 설정된다.

☹ 그러나 이웃노드들은 중요도를 동일하게 갖지 않는다 → Cognitive attention

↪ 데이터에서 중요한 부분에 집중하여, 신경망이 컴퓨팅 파워를 해당 부분에 집중시키도록 함

↪ 상황에 따라 그래프에서 중요한 부분에 대한 정의는 다르므로 훈련을 통해 학습하게 됨

2.3 GNN layer : GAT

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights 이웃노드에 대한 중요도를 나타낸 값

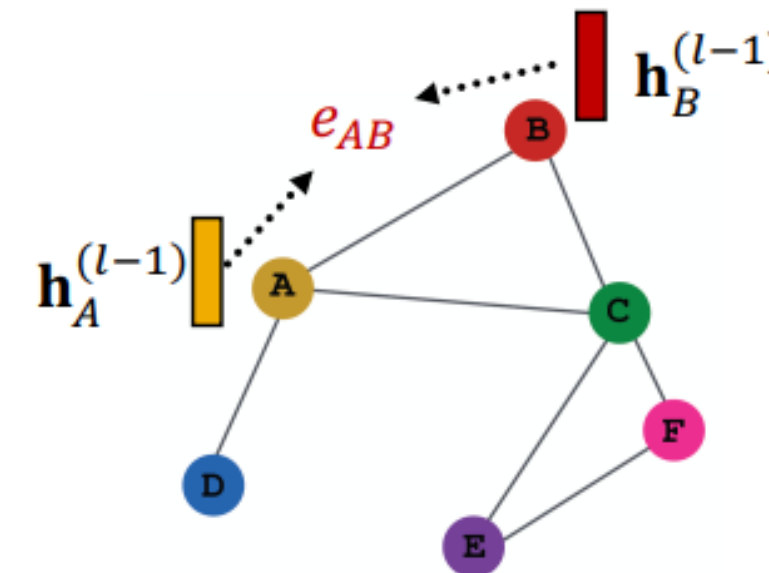
- Goal : 이웃노드마다 각각 다른 가중치를 부여해 중요성을 구체화 시킨다.
- How : Attention mechanism a ※ a 와 α 의 표기 주의

$$e_{vu} = a(\mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)} \mathbf{h}_v^{(l-1)}) \quad \text{노드 } u \text{ 와 } v \text{ 의 메시지를 기반으로 계산한다.}$$

↳ 노드 v 로 오는 노드 u 의 메시지에 대한 중요도

노드 A 로 오는 노드 B 의 중요도

$$e_{AB} = a(\mathbf{W}^{(l)} \mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)} \mathbf{h}_B^{(l-1)})$$



2.3 GNN layer : GAT

- 소프트맥스 함수를 적용해 e 를 정규화 시키어 최종 attention weights 를 계산

- Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:
노드 v 에 대한 모든 알파값은 1

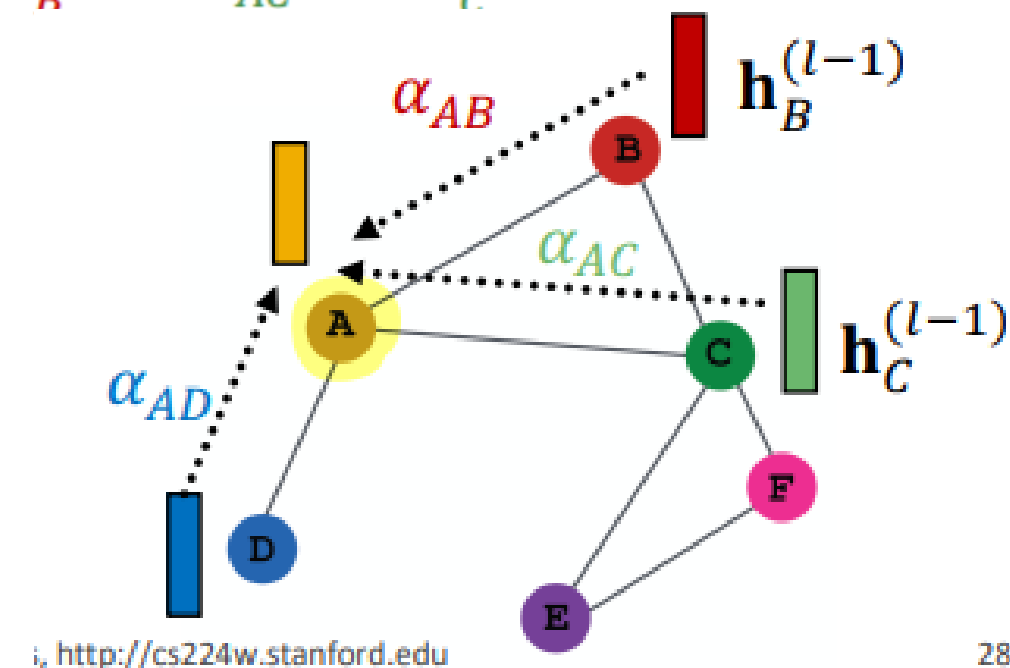
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

- 구한 알파값을 기반으로 가중합을 계산

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Weighted sum using α_{AB} , α_{AC} , α_{AD} :

$$\mathbf{h}_A^{(l)} = \sigma(\alpha_{AB} \mathbf{W}^{(l)} \mathbf{h}_B^{(l-1)} + \alpha_{AC} \mathbf{W}^{(l)} \mathbf{h}_C^{(l-1)} + \alpha_{AD} \mathbf{W}^{(l)} \mathbf{h}_D^{(l-1)})$$



2.3 GNN layer : GAT

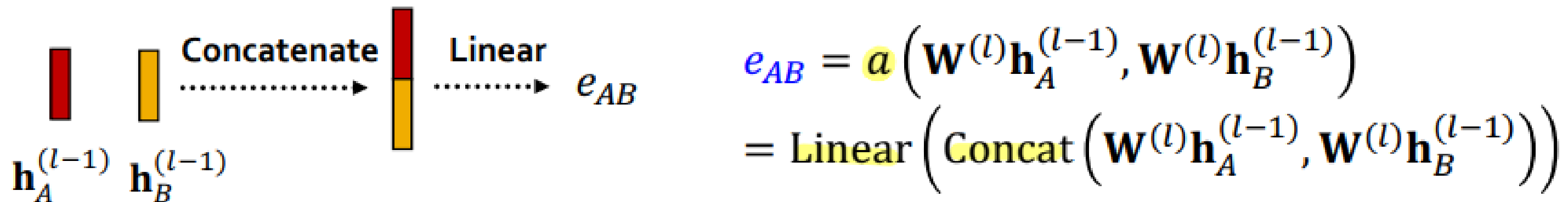
⦿⦿ Attention mechanism a 의 형태는 어떻게 구성되어 있는가? – 딱히 정해진 형태는 없음

* 예제 : 간단한 단일 레이어 신경망을 사용한 경우

■ E.g., use a simple single-layer neural network

■ a have trainable parameters (weights in the Linear layer)

a (attention mechanism) 는 훈련가능한 파라미터



Weight matrix 와 함께 end-to-end 로 학습될 수 있다.

2.3 GNN layer : GAT

- attention 계산의 또 다른 변형 : Multi-head attention

$$\mathbf{h}_v^{(l)}[1] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^1 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

$$\mathbf{h}_v^{(l)}[2] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^2 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

$$\mathbf{h}_v^{(l)}[3] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^3 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

Outputs are aggregated:

- By concatenation or summation

- $\mathbf{h}_v^{(l)} = \text{AGG}(\mathbf{h}_v^{(l)}[1], \mathbf{h}_v^{(l)}[2], \mathbf{h}_v^{(l)}[3])$

각각 다른 attention coefficient 로 여러 개의 \mathbf{h}_v 를 구해 이를 종합하여 최종 \mathbf{h}_v 로 사용한다.

Benefits of Attention Mechanism

Attention Mechanism은 잠재적으로 중심노드에 대한 각 주변노드의 다른 중요도를 잡아냄.

1. Computationally efficient

- 어텐션 매커니즘은 행렬 연산이 전부이고, 멀티 헤드 어텐션의 경우 병렬처리가 쉽게 진행될 수 있음.
- 엣지에 대해 병렬적으로 처리될 수 있기 때문에 계산이 매우 빠르고 효율적으로 가능함.

2. Storage efficient

- 그래프를 sparse matrix로 저장할 수 있어 램 관리 측면에서 아주 효율적임.
- 고정된 수의 파라미터를 사용하여 그래프의 크기에 영향을 받지 않음.

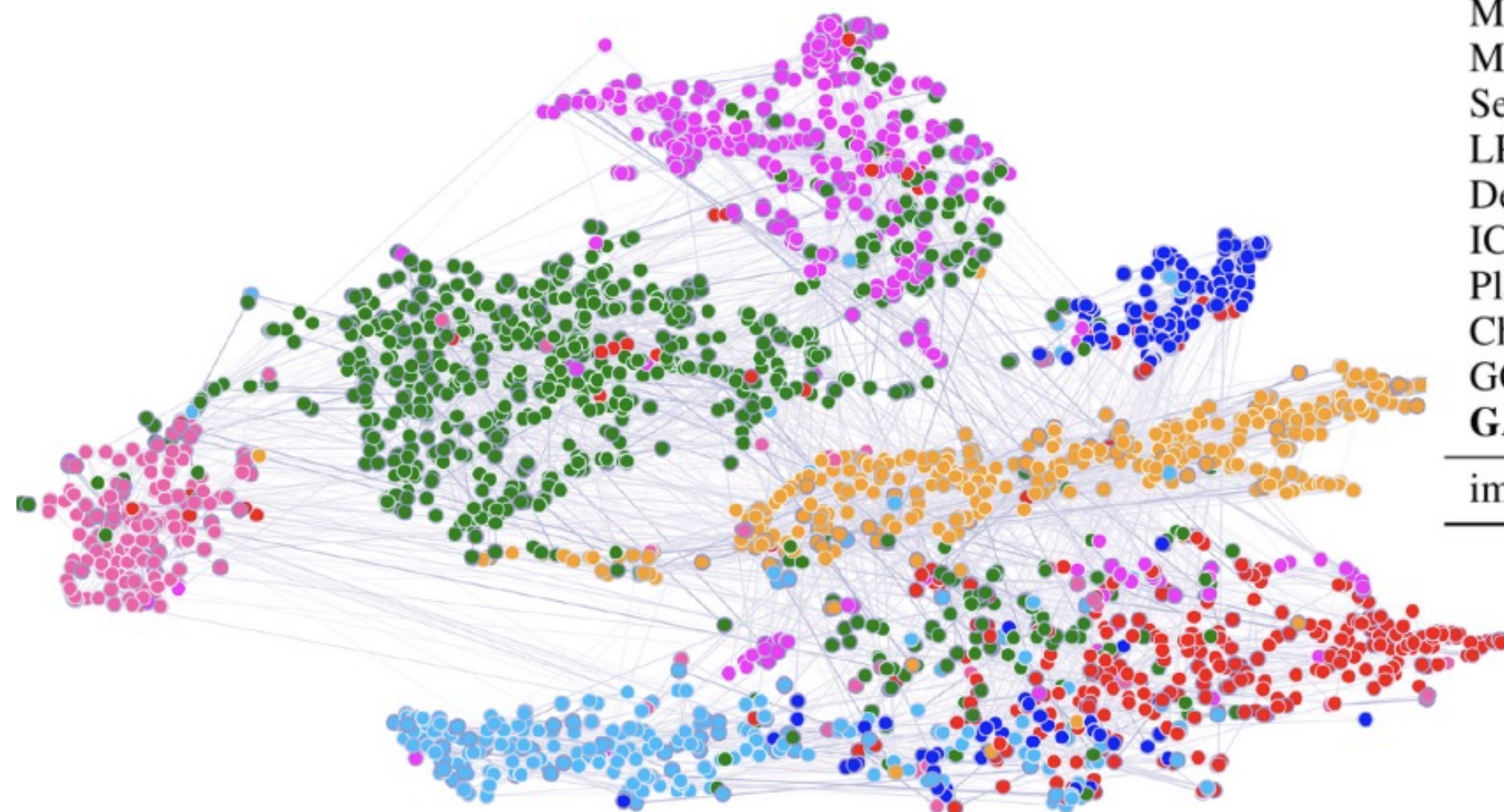
3. Localized

- 이웃노드에만 집중하도록 함 -> 전체 그래프 구조를 파악하지 않음.

4. Inductive capability

- shared edge-wise mechanism으로, edge가 몇 개이든, 그래프의 크기가 어느정도이든 상관없이 적용가능함.

GAT example : Cora Citation Net



Method	Cora
MLP	55.1%
ManiReg (Belkin et al., 2006)	59.5%
SemiEmb (Weston et al., 2012)	59.0%
LP (Zhu et al., 2003)	68.0%
DeepWalk (Perozzi et al., 2014)	67.2%
ICA (Lu & Getoor, 2003)	75.1%
Planetoid (Yang et al., 2016)	75.7%
Chebyshev (Defferrard et al., 2016)	81.2%
GCN (Kipf & Welling, 2017)	81.5%
GAT	83.3%
improvement w.r.t GCN	1.8%

Attention mechanism can be used with many different graph neural network models

In many cases, attention leads to performance gains

MLP << Random Walk << GCN << **GAT**

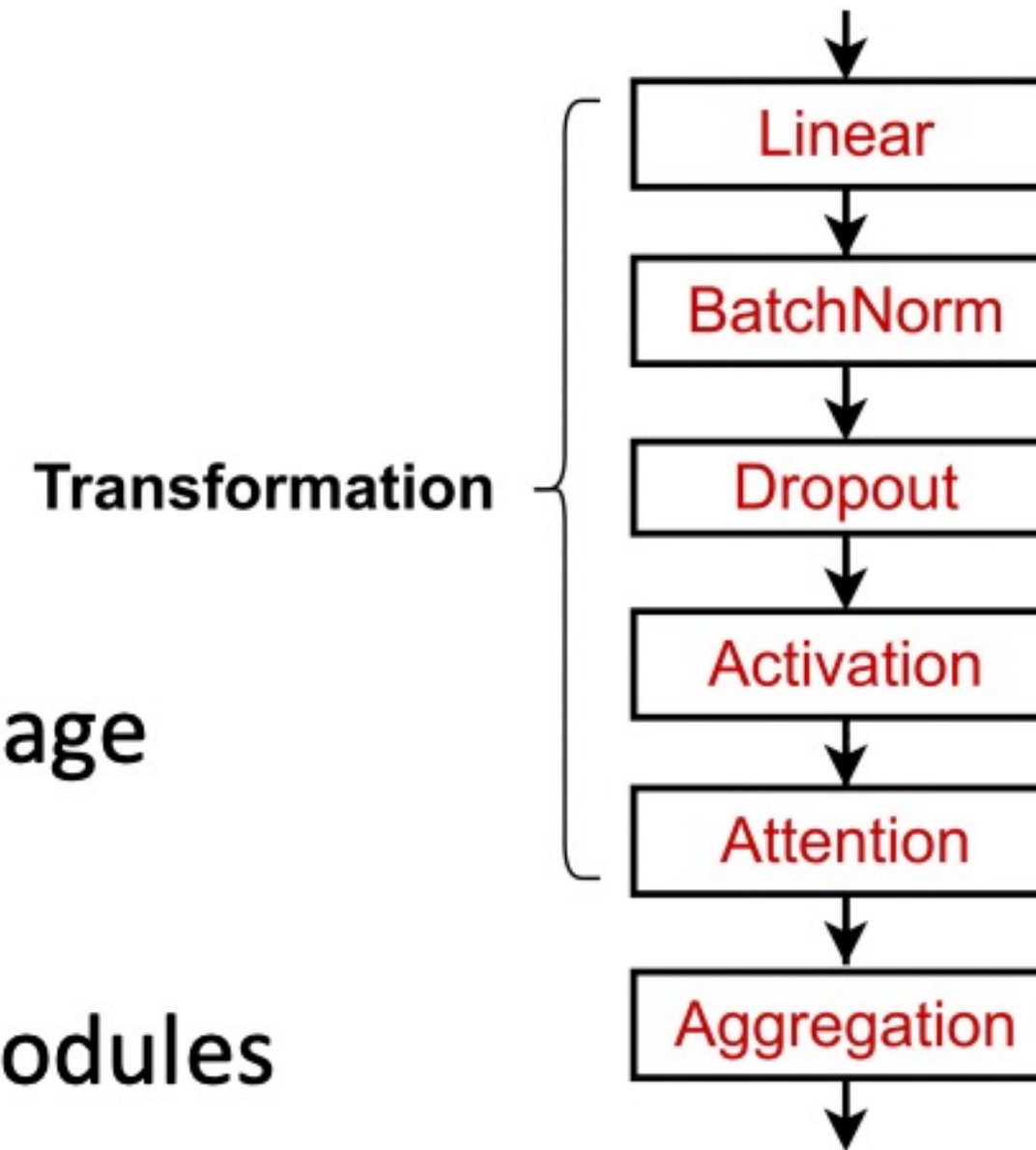
GNN Layers in Practice



Modern deep learning modules in GNN

- **Batch Normalization:**
 - Stabilize neural network training
- **Dropout:**
 - Prevent overfitting
- **Attention/Gating:**
 - Control the importance of a message
- **More:**
 - Any other useful deep learning modules

A suggested GNN Layer



Modern deep learning modules in GNN

Batch Normalization

Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$
 N node embeddings

Trainable Parameters:
 $\gamma, \beta \in \mathbb{R}^D$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D}$
Normalized node embeddings

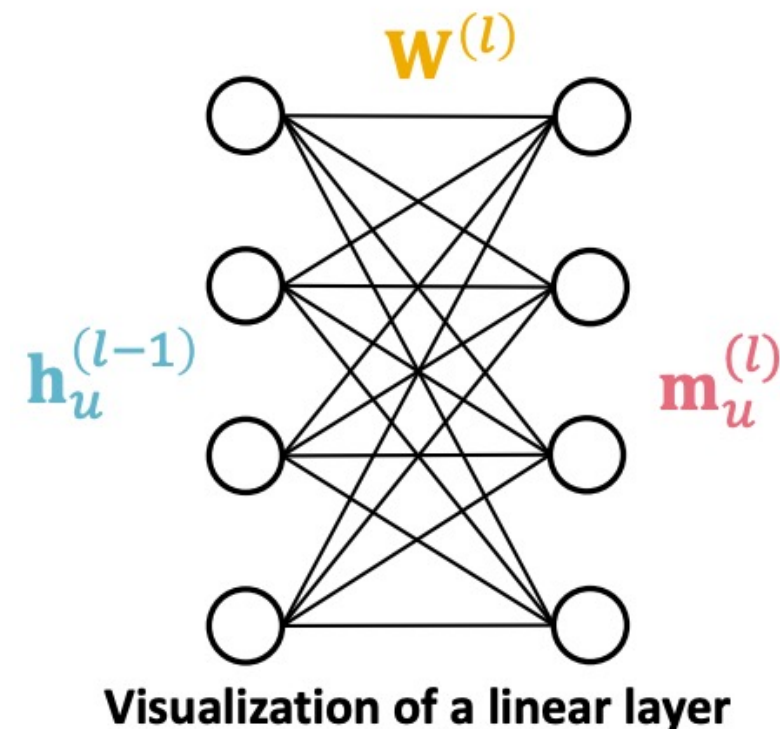
Step 1:
Compute the mean and variance over N embeddings

$$\mu_j = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_{i,j}$$
$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_{i,j} - \mu_j)^2$$

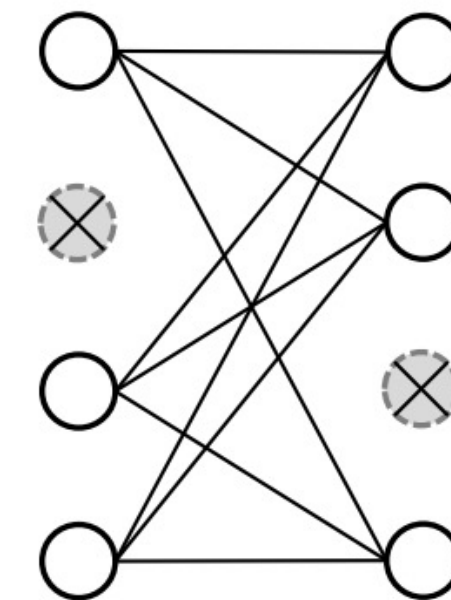
Step 2:
Normalize the feature using computed mean and variance

$$\hat{\mathbf{x}}_{i,j} = \frac{\mathbf{x}_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$
$$\mathbf{y}_{i,j} = \gamma_j \hat{\mathbf{x}}_{i,j} + \beta_j$$

Dropout



Dropout
→



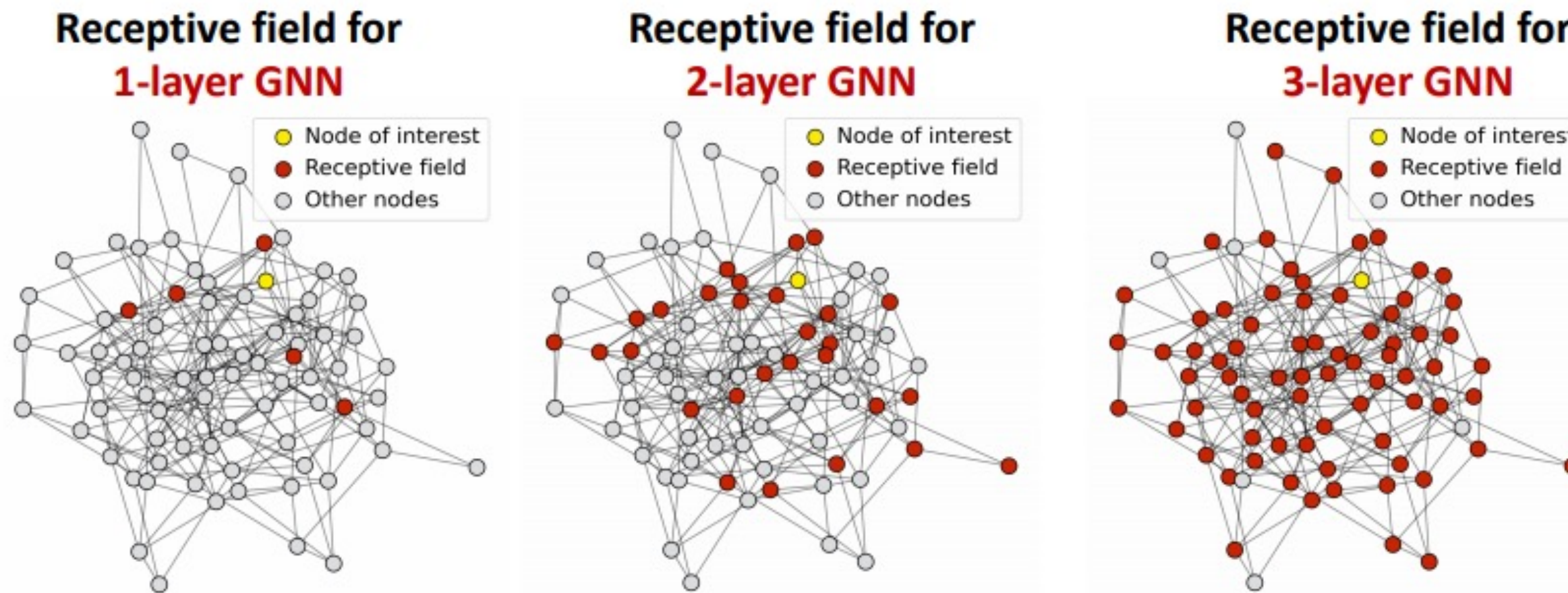
message transformation 시 선형변환이 이루어지는데 이때 drop out

Stacking Layers of a GNN



Stacking Layers of a GNN

GNN 레이어를 CNN처럼 깊게 쌓아보자! -> 안됨. Over Smoothing Problem이 생김 -> WHY?

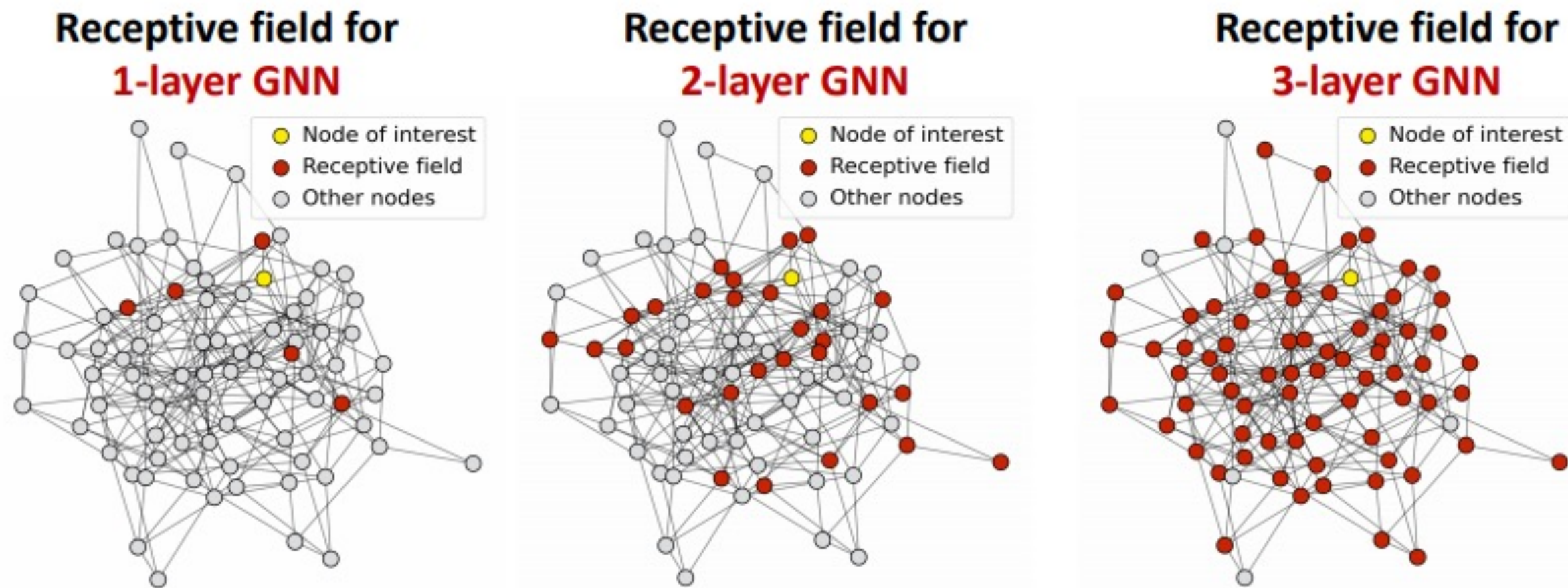


Over Smoothing Problem : 모든 노드 임베딩이 비슷한 값으로 수렴하는 현상

원인 – 한 hop을 지날수록 중심노드의 임베딩을 결정할 때 관여하는 노드 집합(Receptive field)가 급격하게 증가함.

Stacking Layers of a GNN

GNN 레이어를 CNN처럼 깊게 쌓아보자! -> 안됨. Over Smoothing Problem이 생김 -> WHY?



GNN 레이어를 많이 쌓는다 -> 노드들의 receptive field가 매우 비슷해진다 -> 노드 임베딩이 비슷해진다. -> over smoothing problem 발생!

How to Solve?

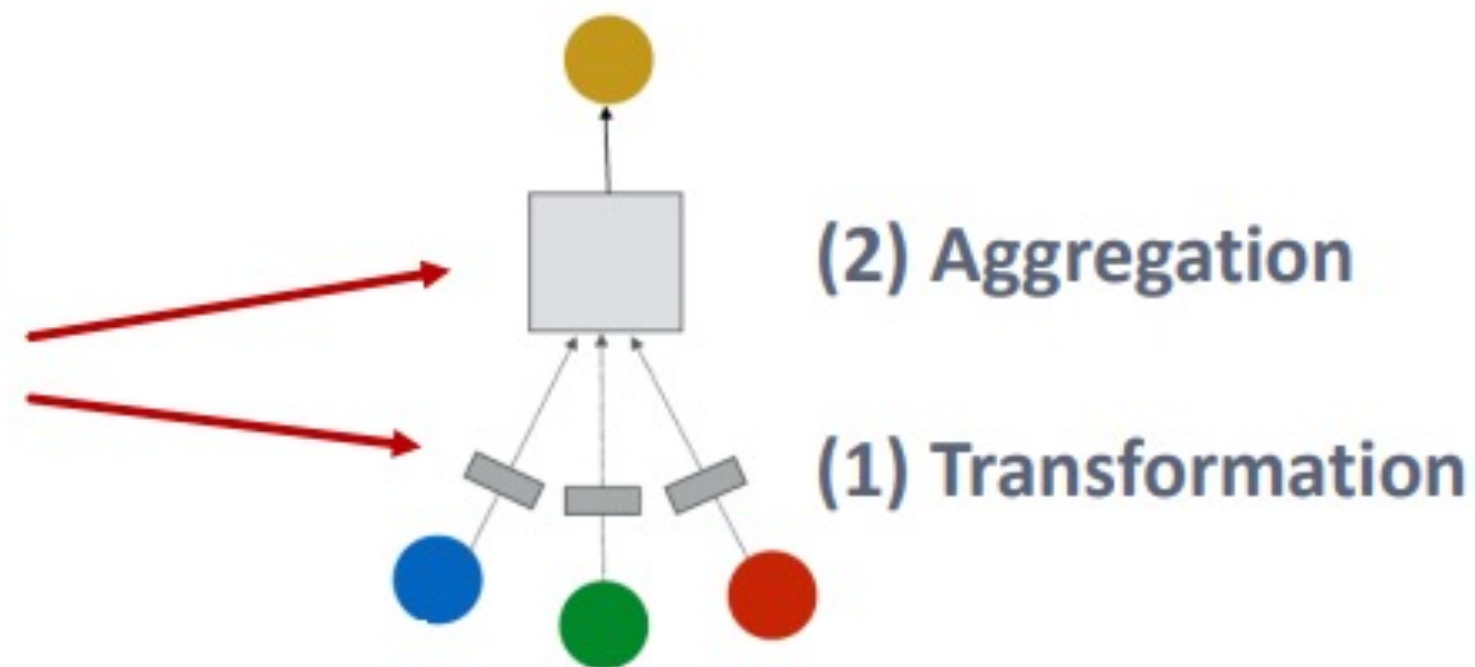
1. Do not Stack !

하지만 레이어를 적게 쌓으면 파라미터 수가 적고 모델의 표현력이 떨어짐.

이에 대한 해결방법 2가지

1) Make Transformation/Aggregation become a Deep Neural Network

If needed, each box could include a **3-layer MLP**



Message Aggregation과 Message Transformation 모두 어떠한 딥러닝 레이어 혹은 affine 변환 레이어를 가지고 있기 때문에 이를 보다 깊게 하면 모델의 표현력은 향상될 수 있다.

How to Solve?

2) Add layer that do not Pass Message

Over-smoothing problem은 메시지가 너무 많은 receptive field를 지나왔기 때문에 발생함.

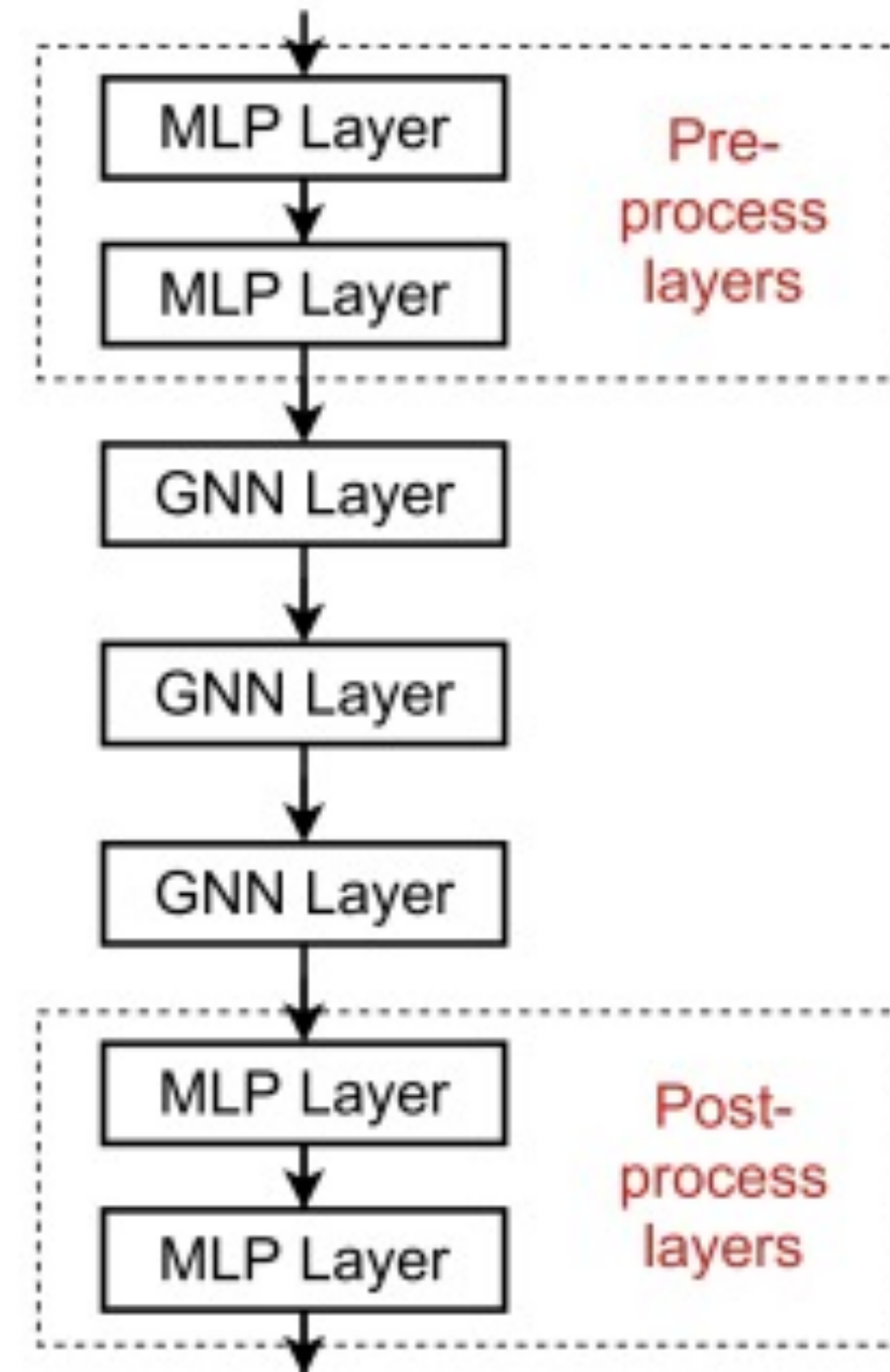
-> GNN layer 전과 후에 MLP layer을 추가할 수 있음!

Pre-processing layers: Important when encoding node features is necessary.

E.g., when nodes represent images/text

Post-processing layers: Important when reasoning / transformation over node embeddings are needed

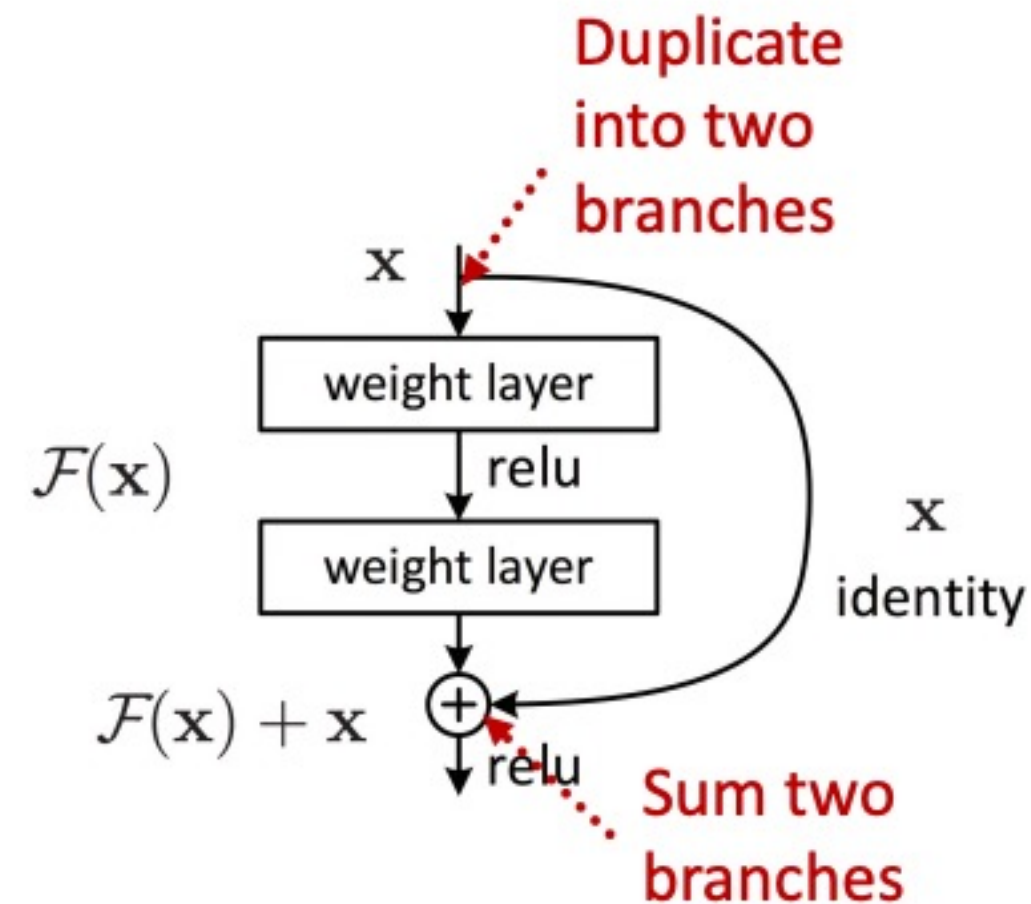
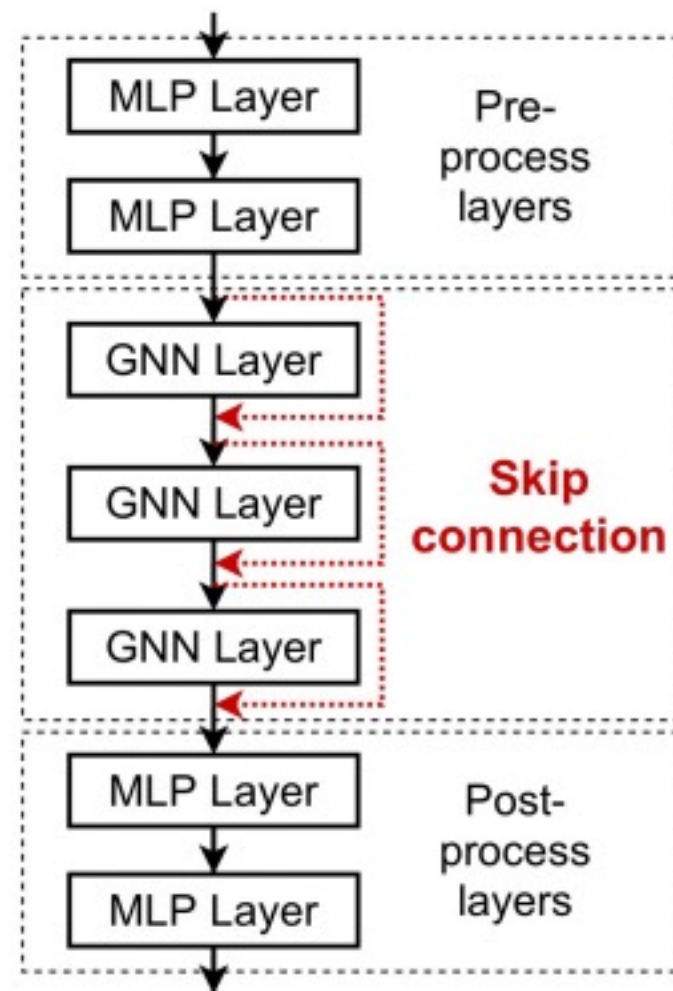
E.g., graph classification, knowledge graphs



How to Solve?

그럼에도 GNN layer를 많이 필요로 한다면?

=> Skip Connection



Idea of skip connections:

Before adding shortcuts:

$$F(\mathbf{x})$$

After adding shortcuts:

$$F(\mathbf{x}) + \mathbf{x}$$

Jure Leskovec, Stanford CS224W: Machine Learning with Graphs. <http://cs224w.stanford.edu>

67

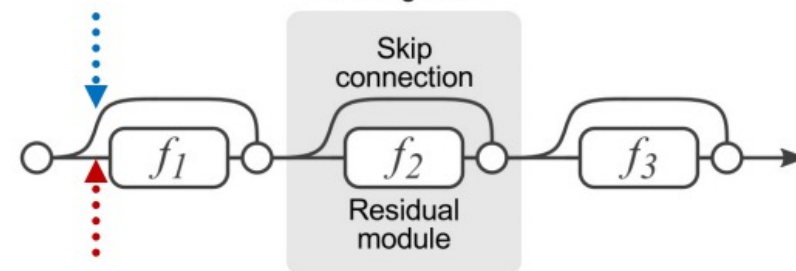
skip connection은 어떠한 함수 $f(x)$ 와 입력값 x 를 더하여 해당 레이어의 비선형 함수에 통과시키겠다는 아이디어

Skip Connection

- Why do skip connections work?

- **Intuition:** Skip connections create **a mixture of models**
- N skip connections $\rightarrow 2^N$ possible paths
- Each path could have up to N modules
- We automatically get **a mixture of shallow GNNs and deep GNNs**

Path 2: skip this module
Building block

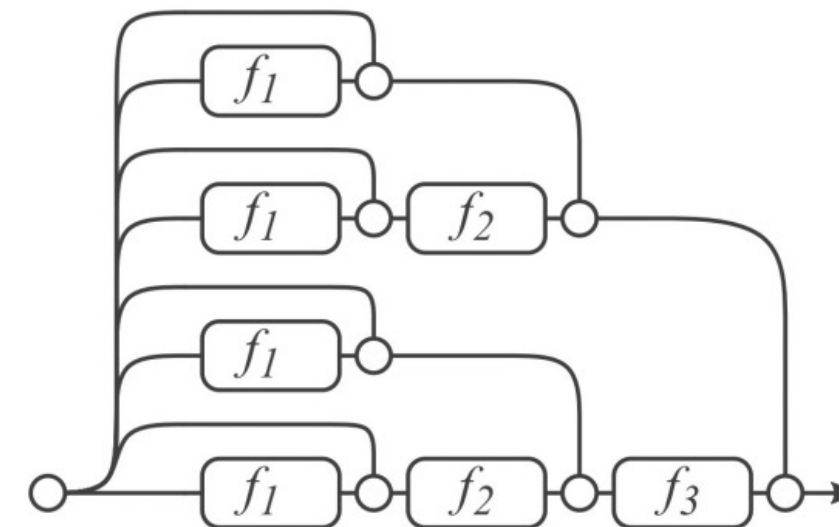


Path 1: include this module

(a) Conventional 3-block residual network

All the possible paths:

$$2 * 2 * 2 = 2^3 = 8$$



(b) Unraveled view of (a)

n개의 레이어에 대해 skip connection을 적용하면 입력값이 출력에 이르기까지 총 2^n 개의 경로의 경우를 가진다는 것

Skip Connection 예시

- A standard GCN layer

- $$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$

This is our $F(\mathbf{x})$

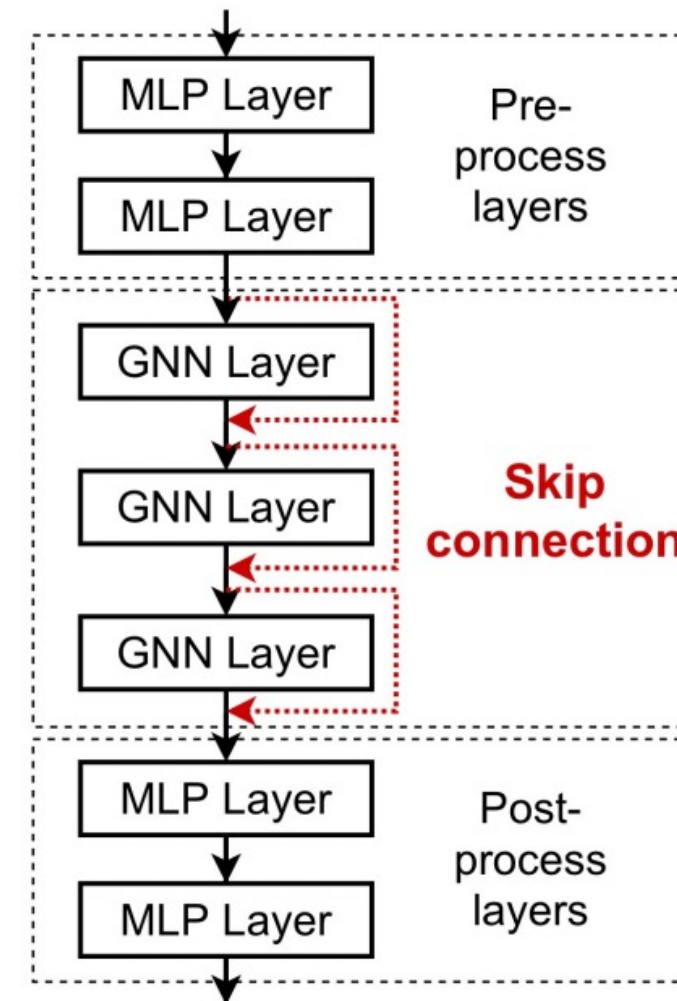
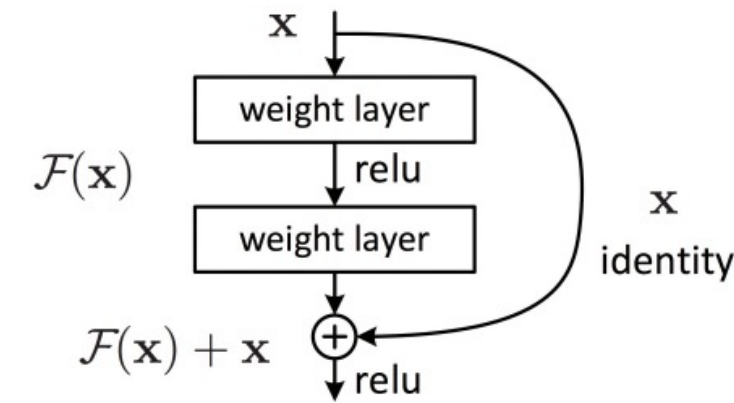
- A GCN layer with skip connection

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

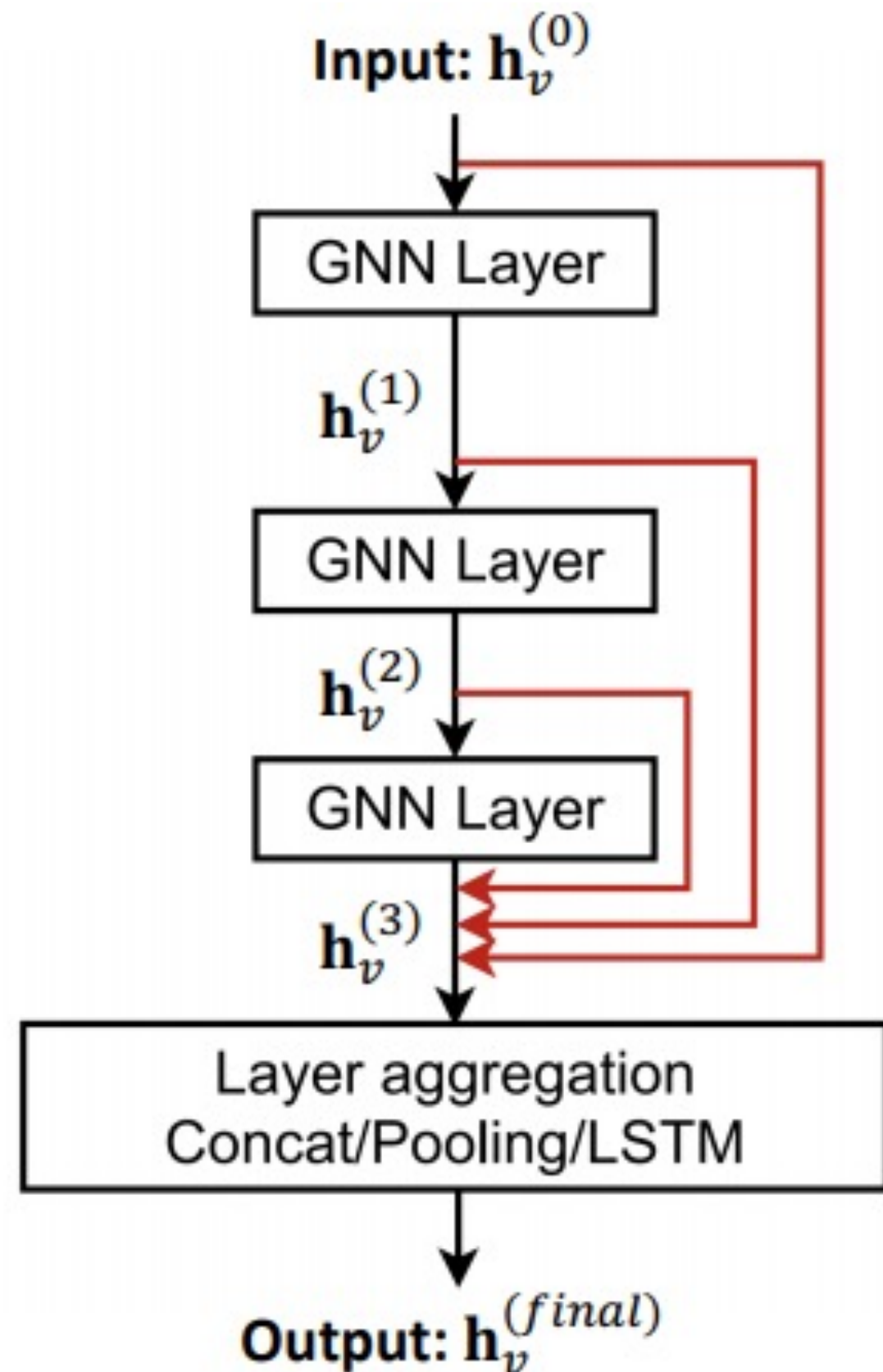
$F(\mathbf{x})$

+ \mathbf{x}

Message from 이웃



Skip Connection 예시



Other options:

- 모든 레이어가 자신의 레이어를 skip하지 않고, 곧바로 출력으로 skip하도록 설정하는 것.
- 이는 최종 출력은 모든 레이어에서의 임베딩 벡터를 종합(aggregate)하여 최종적인 임베딩 벡터를 만드는 것임.
- Adding하면서 무슨 정보가 더 중요한지 파악할 수 있음.

THANK YOU

