Graph Neural Network

Hyelin Choi

Department of Mathematics
Sungkyunkwan University

February 22, 2024

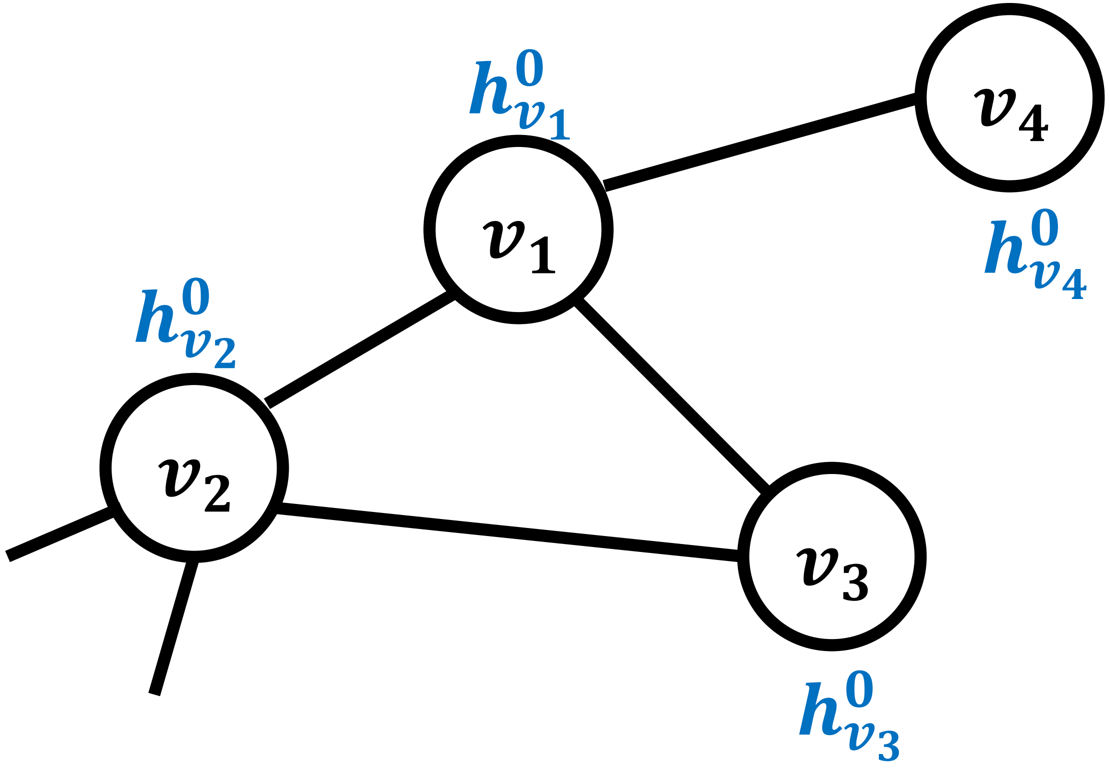
Table of Contents

Table of Contents

Convolutional Neighborhood Aggregation Method

Step 1

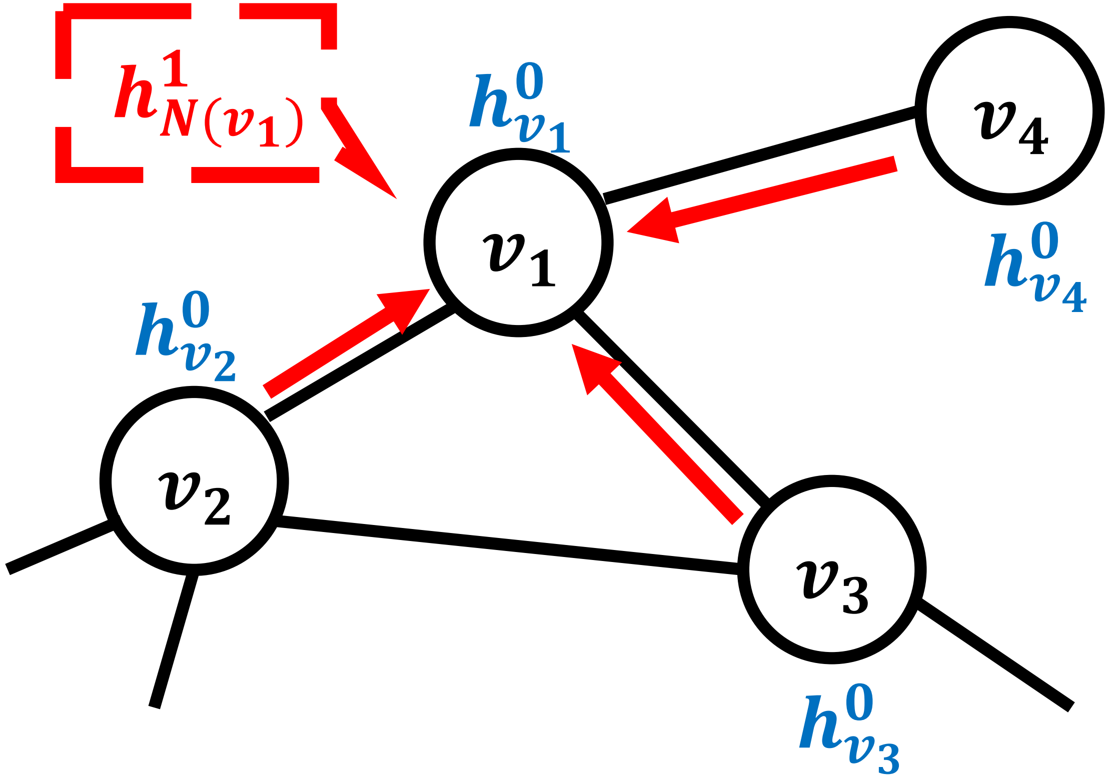
The node embeddings are initialized with node features.



Convolutional Neighborhood Aggregation Method

Step 2

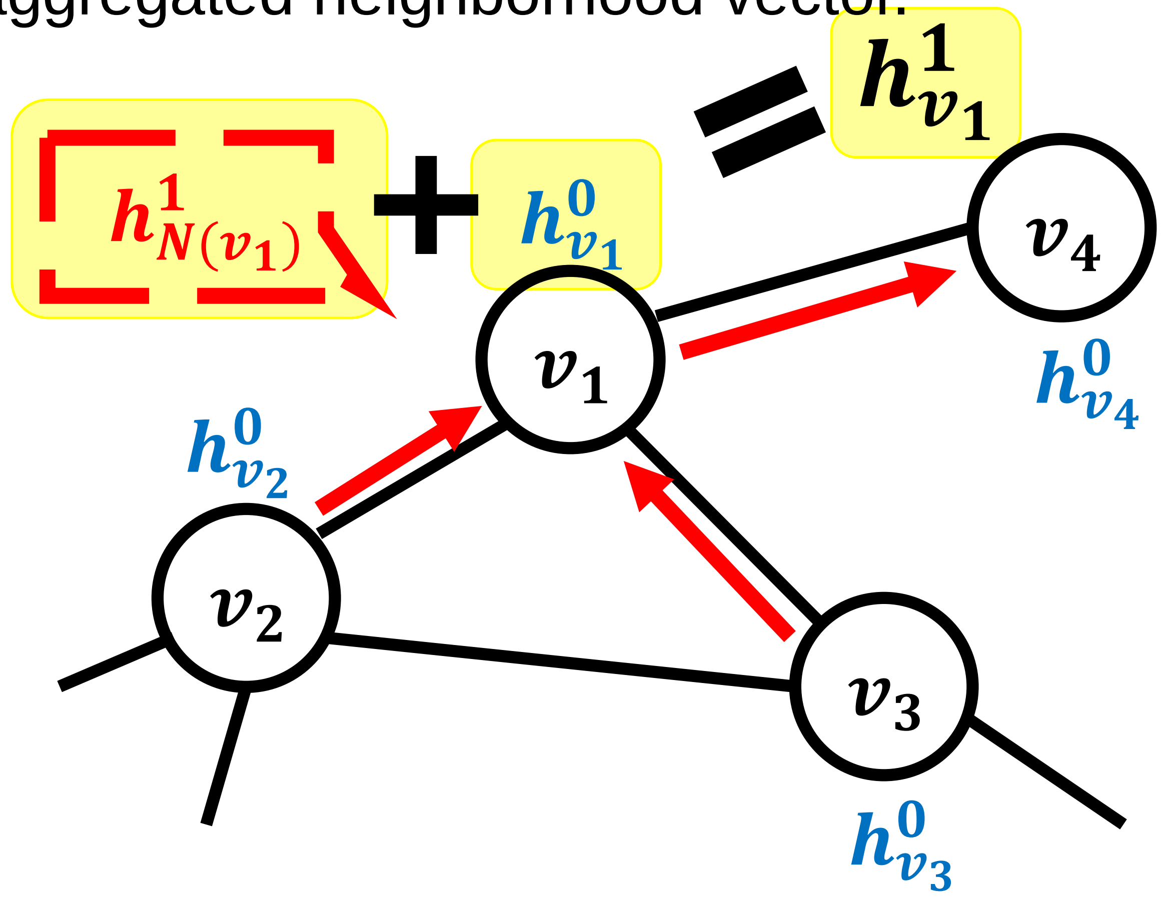
Nodes aggregate the embeddings of their neighbors, using an **aggregation function**.



Convolutional Neighborhood Aggregation Method

Step 3

Combine its previous embedding from the last iteration with its aggregated neighborhood vector.

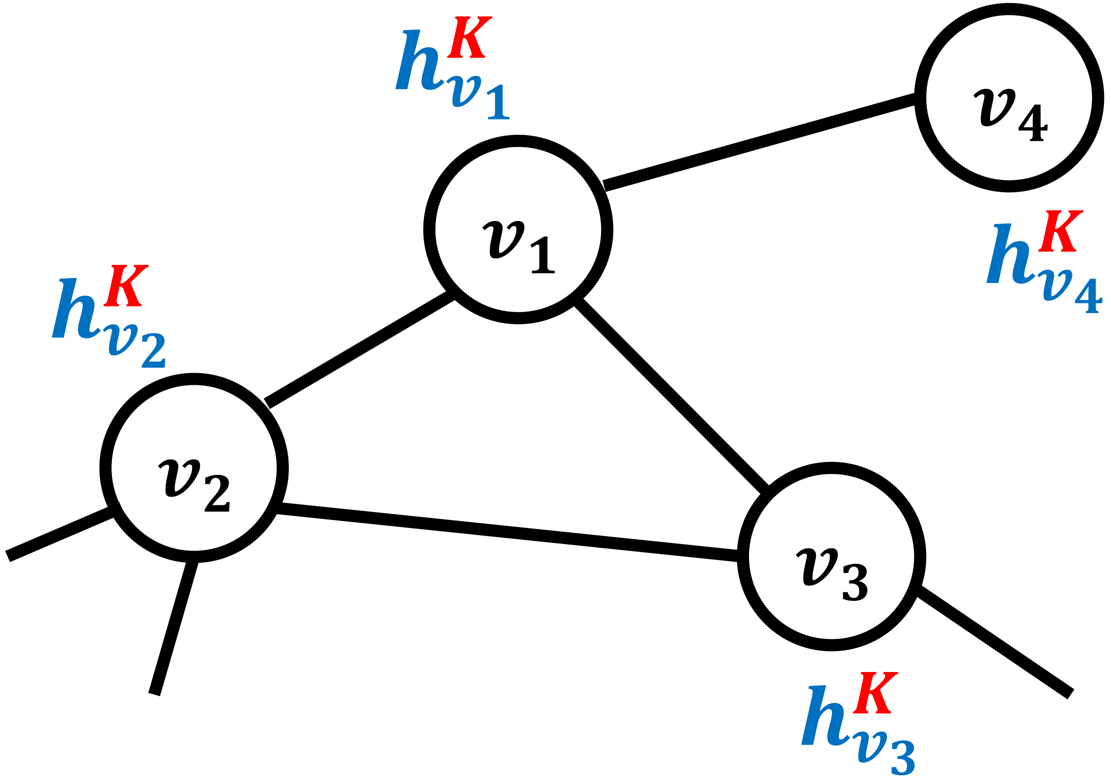


K iteration!

Convolutional Neighborhood Aggregation Method

Step 3

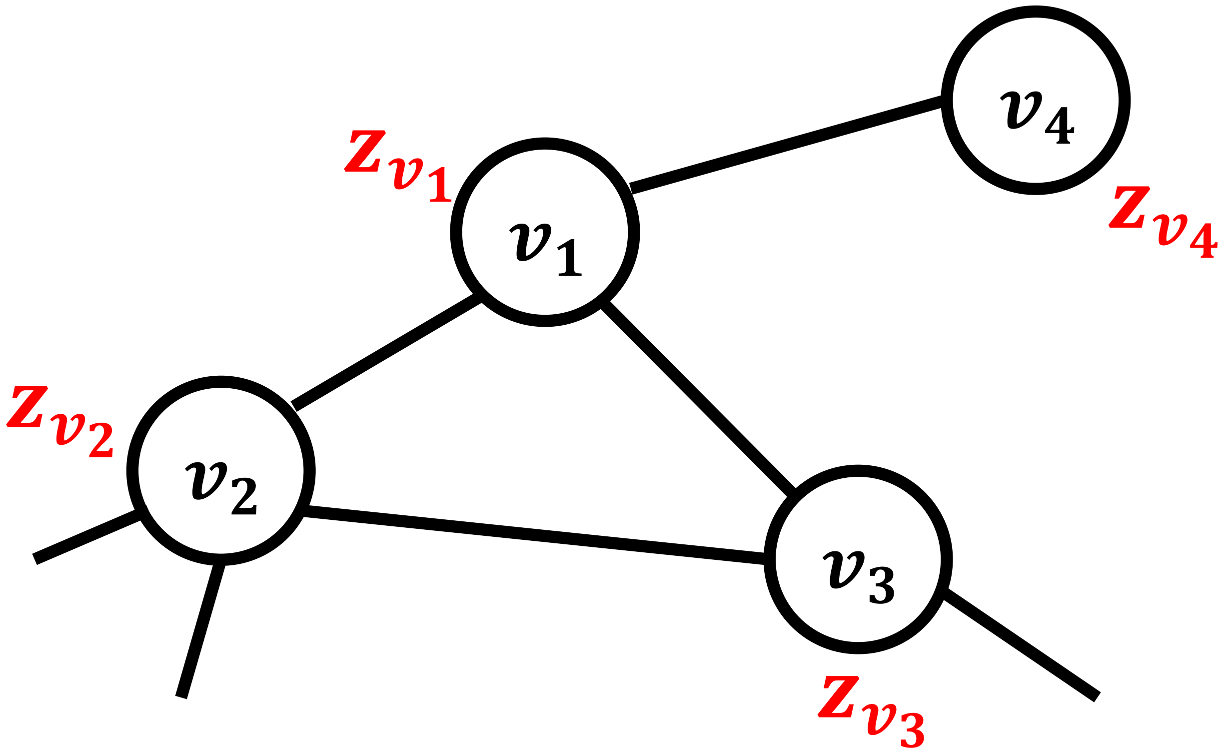
Combine its previous embedding from the last iteration with its aggregated neighborhood vector.



Convolutional Neighborhood Aggregation Method

Step 4

We finally get the vector representation for each node.



Convolutional Neighborhood Aggregation Method

Algorithm 1: Neighborhood-aggregation encoder algorithm. Adapted from [28].

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

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

1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;

for $k = 1...K$ **do**

  for $v \in \mathcal{V}$ **do**

    2 $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$; ← **Aggregation**

    3 $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$ ← **Concatenation**

  end

  $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$

end

4 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

---

trainable

Neighborhood Aggregation

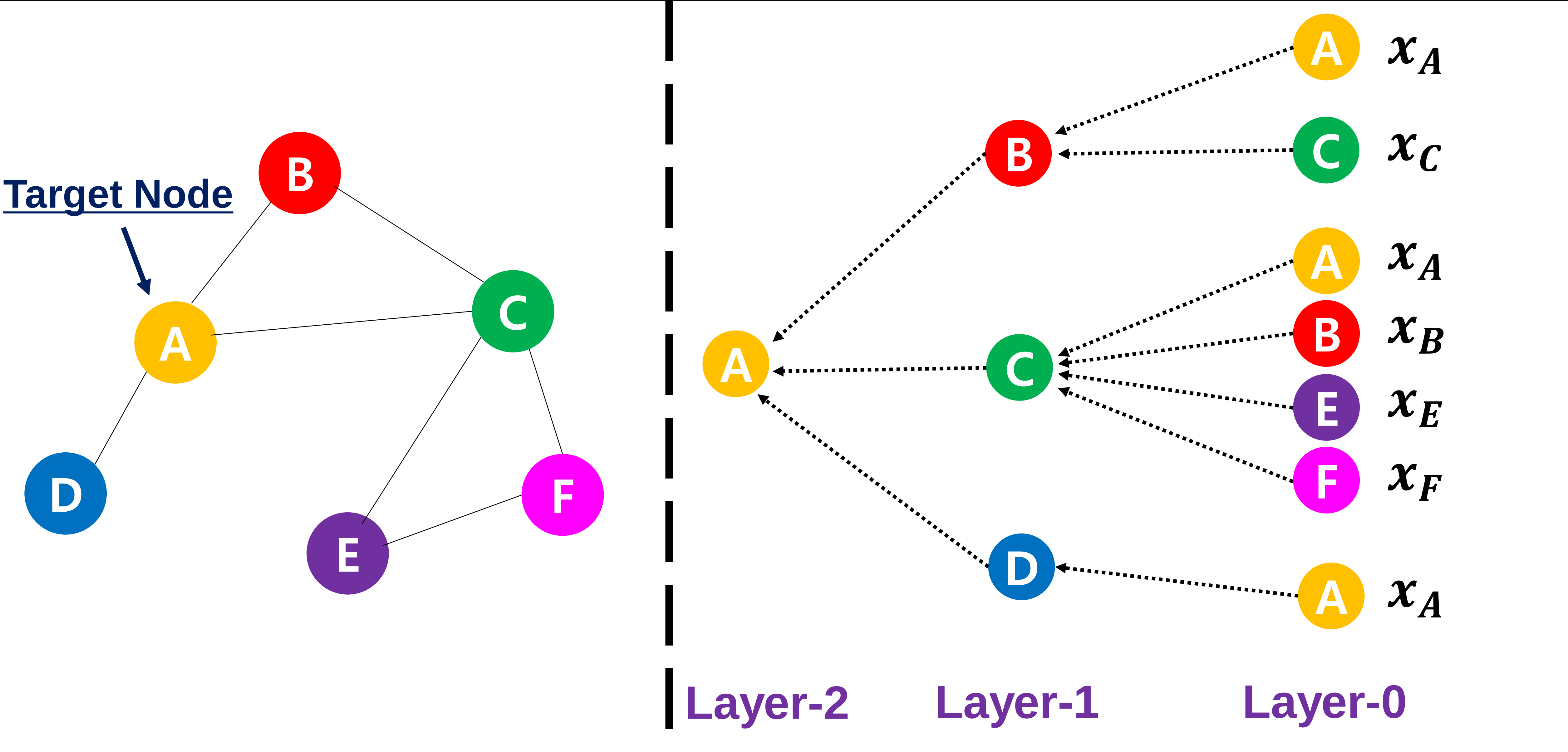


Table of Contents

Graph Neural Network

Initial embeddings are equal to node features.

$$h_v^0 = x_v$$

Concatenation

$$h_v^k = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall \mathrm{k} > 0$$

k-th layer embedding of v

Non-linearity

Aggregation
(ex. sum, mean, max)

Previous layer embedding of v (self-loop)

Graph Neural Network

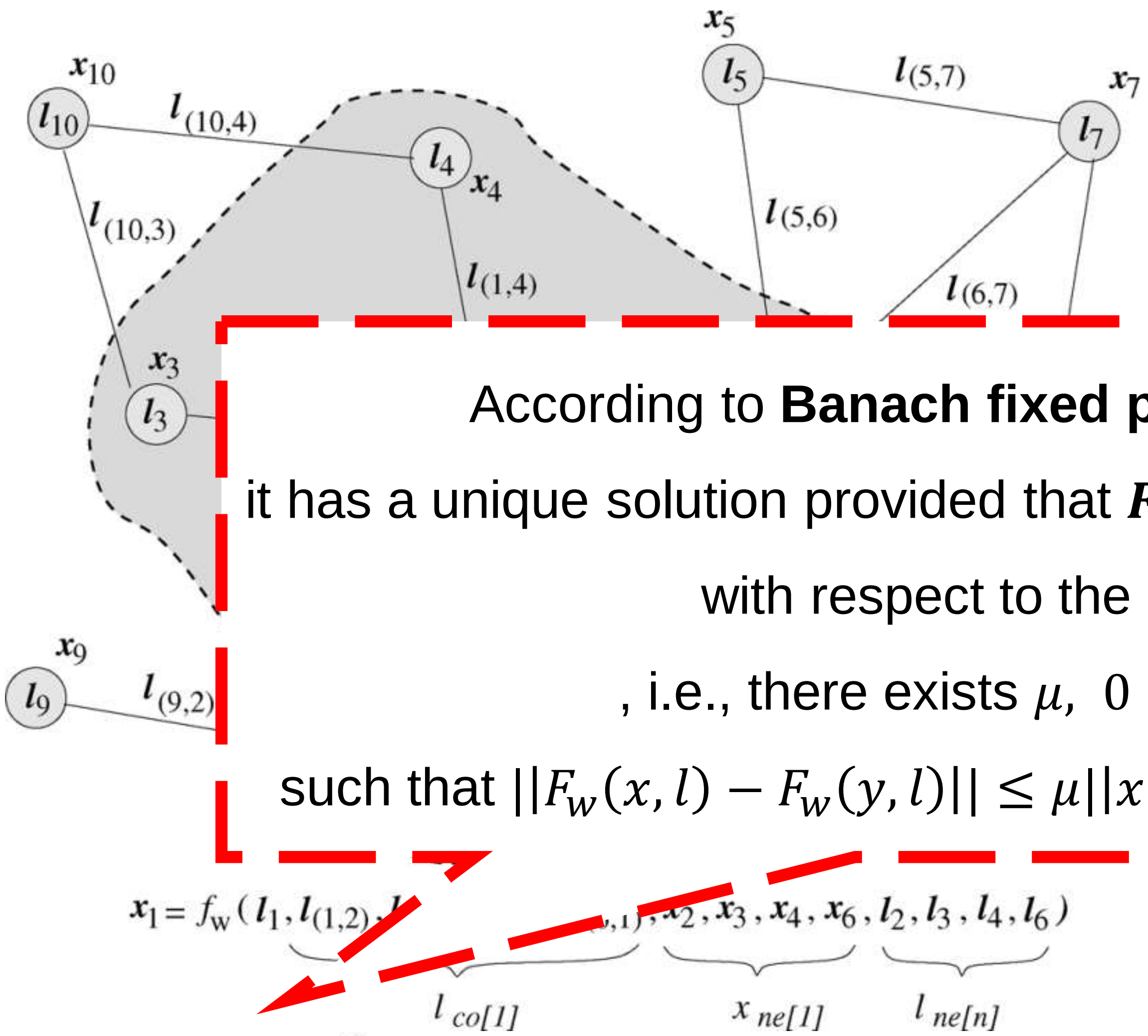
Trainable matrices
(i.e., what we learn)

$$h_v^0 = x_v$$

$$h_v^k = \sigma\left(\boxed{W_k} \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + \boxed{B_k} h_v^{k-1}\right), \quad \forall \mathrm{k} \in \{1, \cdots, K\}$$

$$z_v = h_v^K$$

- After **K**-layers of neighborhood aggregation, we get output embeddings for each node.
- We can feed these embeddings into any loss function and run stochastic gradient descent to train the **aggregation parameters**.



According to **Banach fixed point theorem**,

it has a unique solution provided that $\boldsymbol{F_w}$ is a **contraction map**

with respect to the state

, i.e., there exists $\mu,\ 0 \leq \mu < 1$,

such that $||F_w(x, l) - F_w(y, l)|| \leq \mu||x - y||$ holds for any x, y.

$$\mathbf{x}_n = f_{\mathbf{w}}\left(\mathbf{l}_n, \mathbf{l}_{co[n]}, \mathbf{x}_{ne[n]}, \mathbf{l}_{ne[n]}\right)$$

$$\mathbf{o}_n = g_{\mathbf{w}}\left(\mathbf{x}_n, \mathbf{l}_n\right)$$

local transition function (propagation function)

state of node n

node label of node n

edge label of neighborhood of node n

$$\mathbf{x}_n = f_{\mathbf{w}}\left(\mathbf{l}_n, \mathbf{l}_{co[n]}, \mathbf{x}_{ne[n]}, \mathbf{l}_{ne[n]}\right)$$

$$\mathbf{o}_n = g_{\mathbf{w}}\left(\mathbf{x}_n, \mathbf{l}_n\right)$$

output

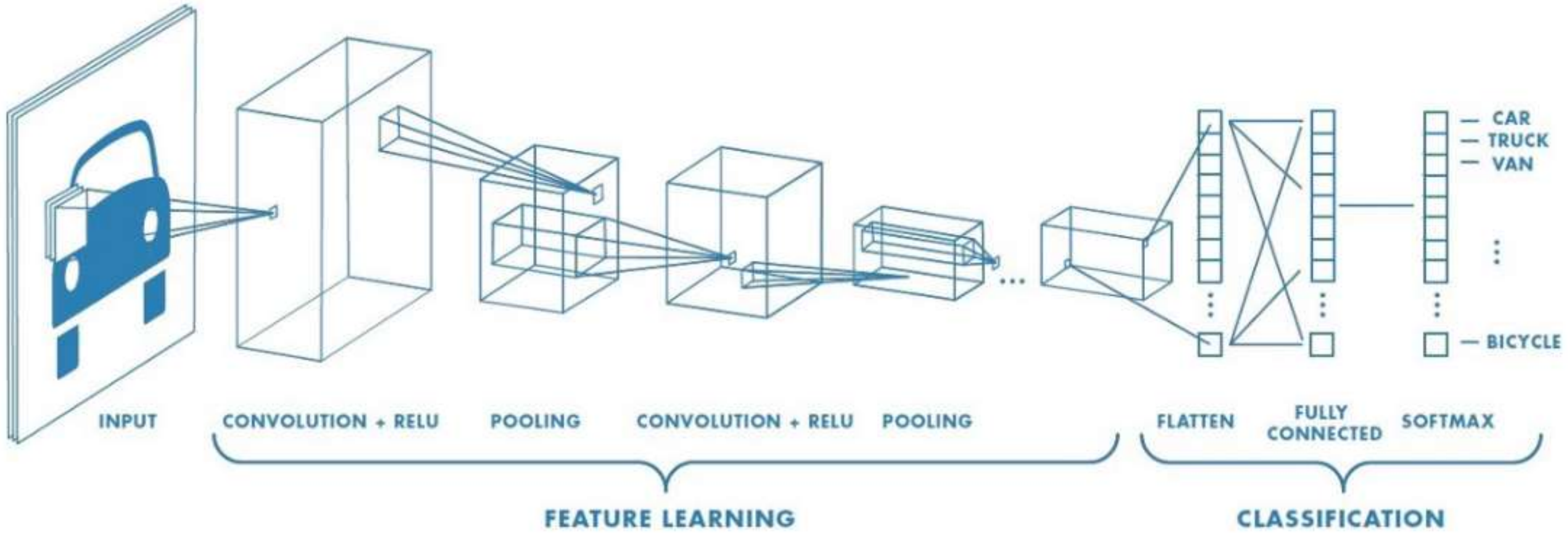
local output function

state of neighborhood of node n

node label of neighborhood of node n

Table of Contents

Graph Convolutional Network



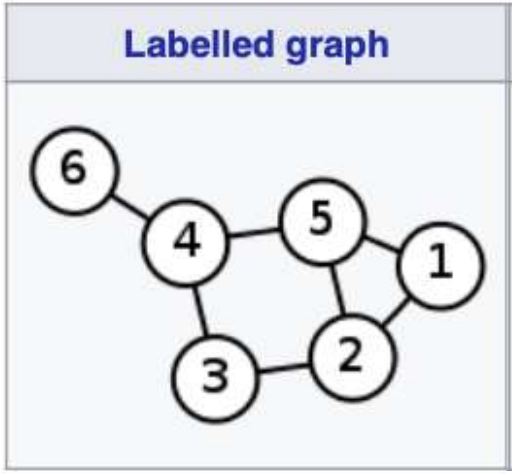
Source : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Propagation Rule

$$h_i^{(l+1)} = \sigma(W^{(l)} \sum_{j=1}^{N} h_j^{(l)} + b^{(l)})$$

$$H^{(l+1)} = \sigma(W^{(l)} H^{(l)} + b^{(l)})$$

$$H^{(l+1)} = \sigma(\textcolor{red}{\boldsymbol{A}} W^{(l)} H^{(l)} + b^{(l)})$$



Adjacency matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Graph Convolutional Network

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

$\underline{\tilde{A} = \mathrm{A} + \mathrm{I}}$ **(self-loop)**

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

normalization

$$\boldsymbol{H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}W^{(l)}H^{(l)} + b^{(l)})}$$

$\underline{\tilde{A} = \mathrm{A} + \mathrm{I}}$ **(self**

Table of Contents

GraphSAGE

- Simple Neighborhood Aggregation:

$$h_v^k = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right)$$

- GraphSAGE

$$h_v^k = \sigma\left(\left[W_k \cdot AGG\left(\{ h_u^{k-1}, \forall u \in N(v) \} \right), B_k h_v^{k-1} \right] \right)$$

GraphSAGE

Neighborhood의 정보를 모으는 함수

- Mean aggregator:

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

- LSTM aggregator:

$$AGG = LSTM([h_u^{k-1}, \forall u \in \pi(N(v))])$$

- Pooling aggregator:

$$AGG = \gamma(\{Qh_u^{k-1}, \forall u \in N(v)\})$$

Symmetric vector function
(element-wise mean/max)

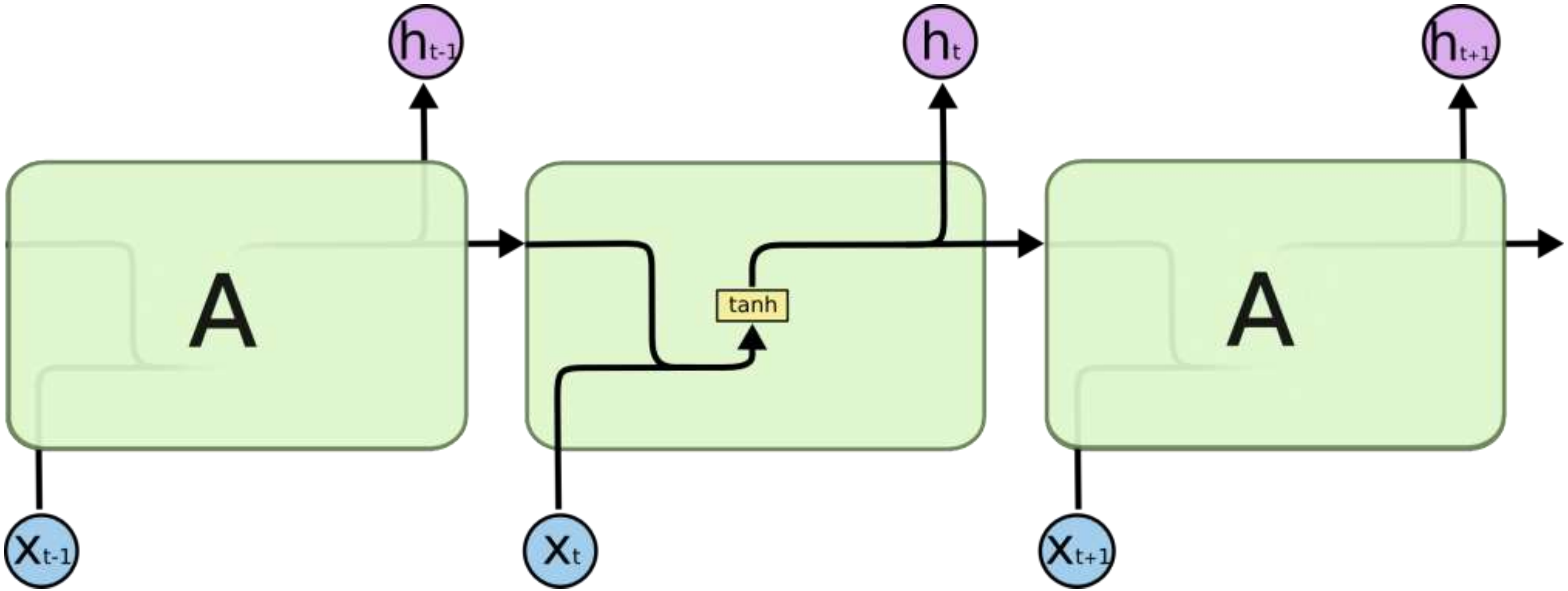
LSTM (Long Short-Term Memory)

- LSTM aggregator:

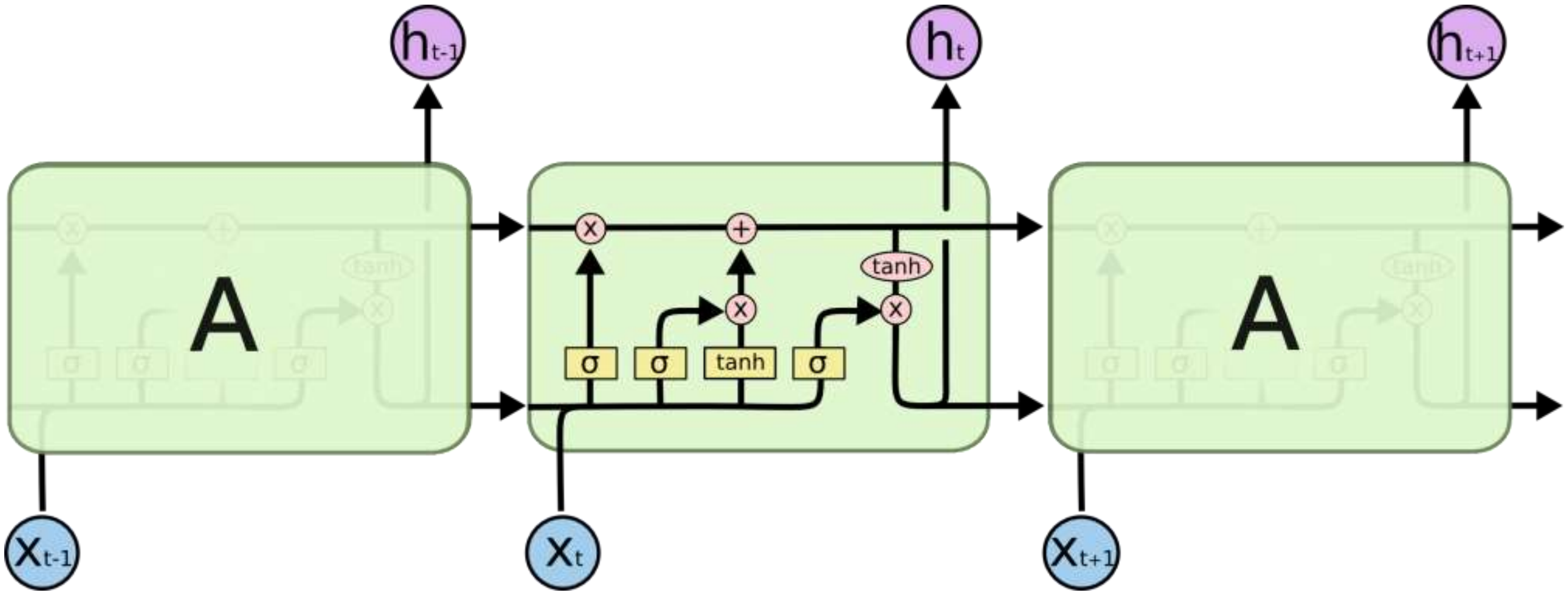
$$AGG = LSTM([h_u^{k-1}, \forall u \in \pi(N(v))])$$

https://wikidocs.net/152773

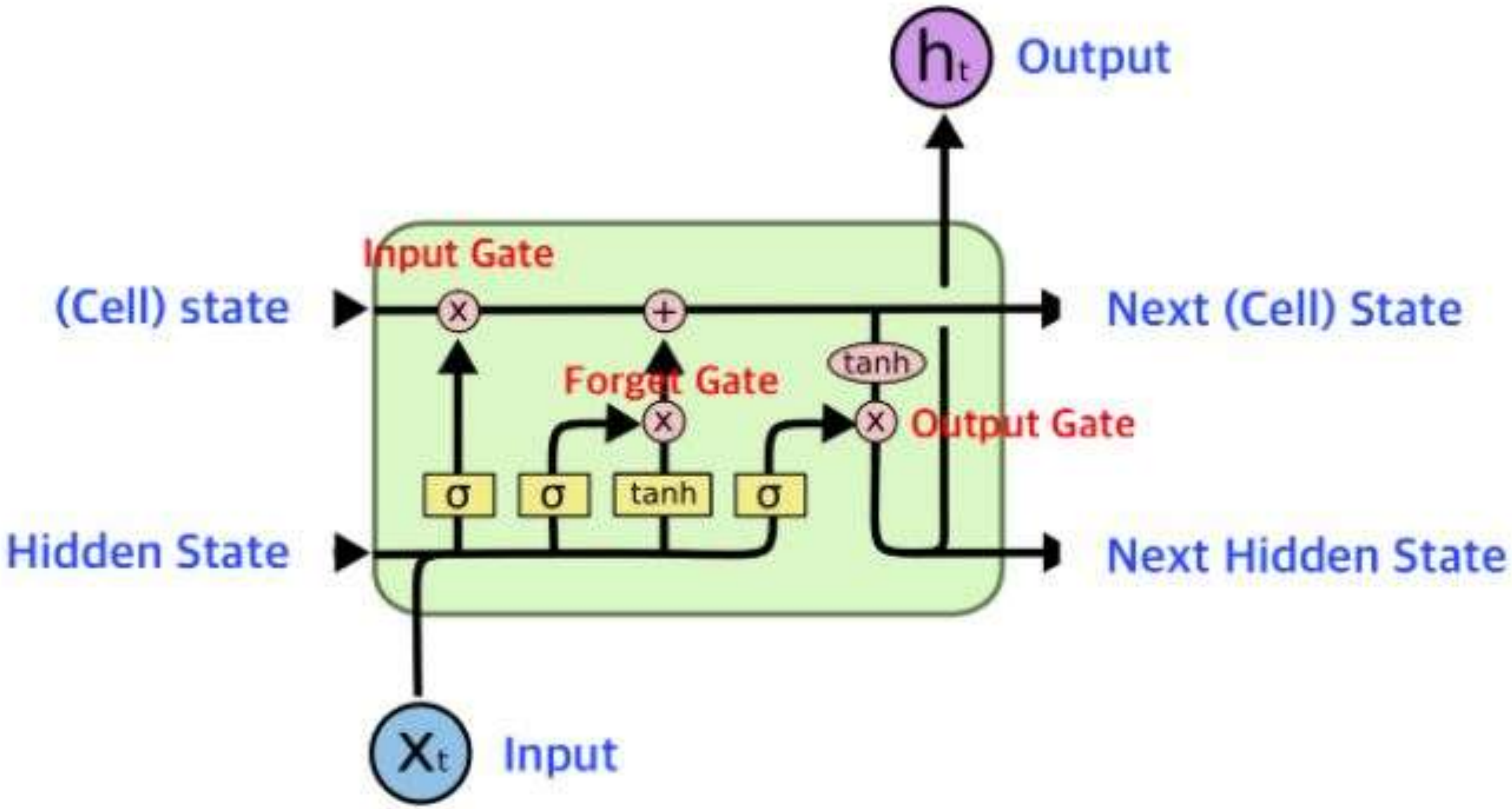
LSTM (Long Short-Term Memory)



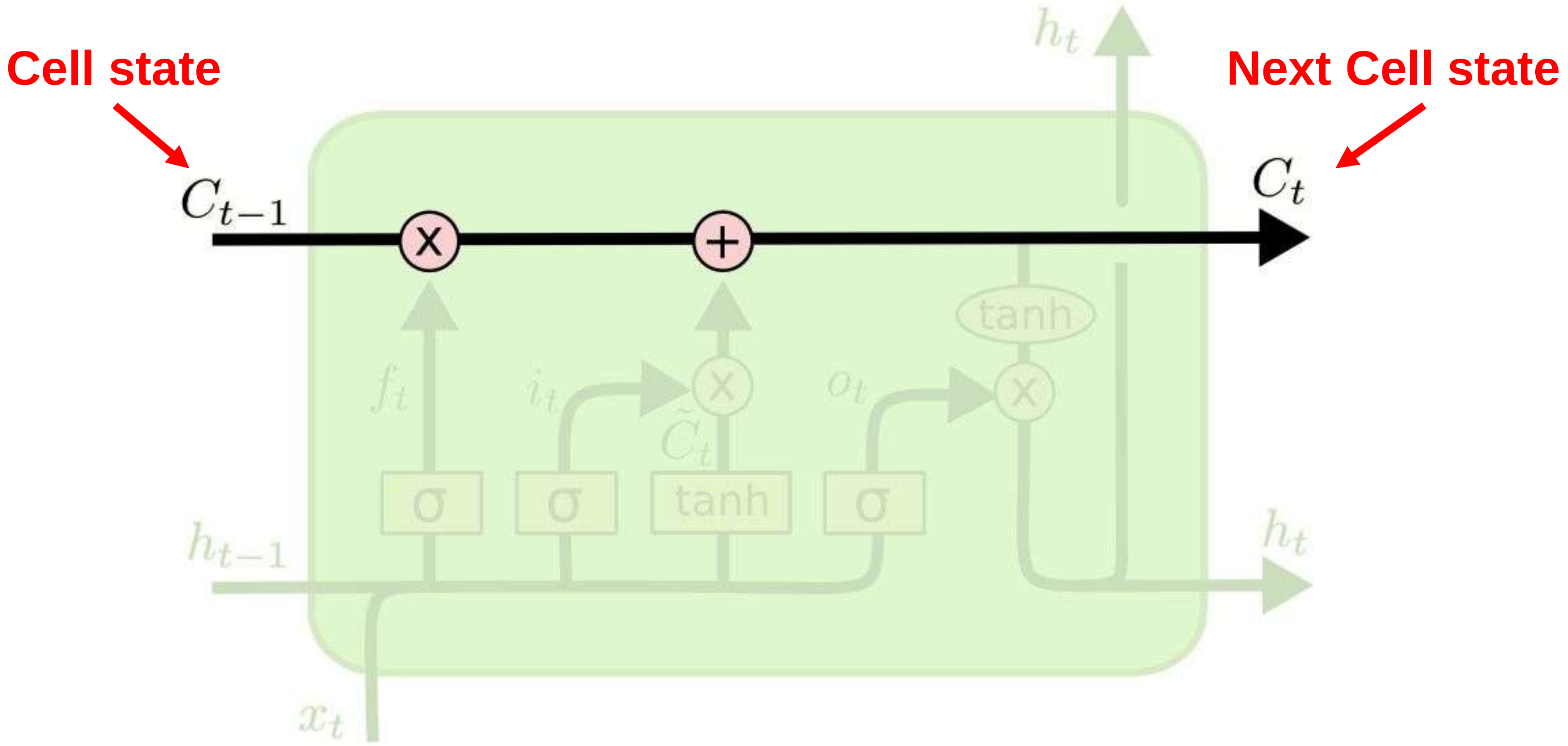
LSTM (Long Short-Term Memory)



LSTM (Long Short-Term Memory)

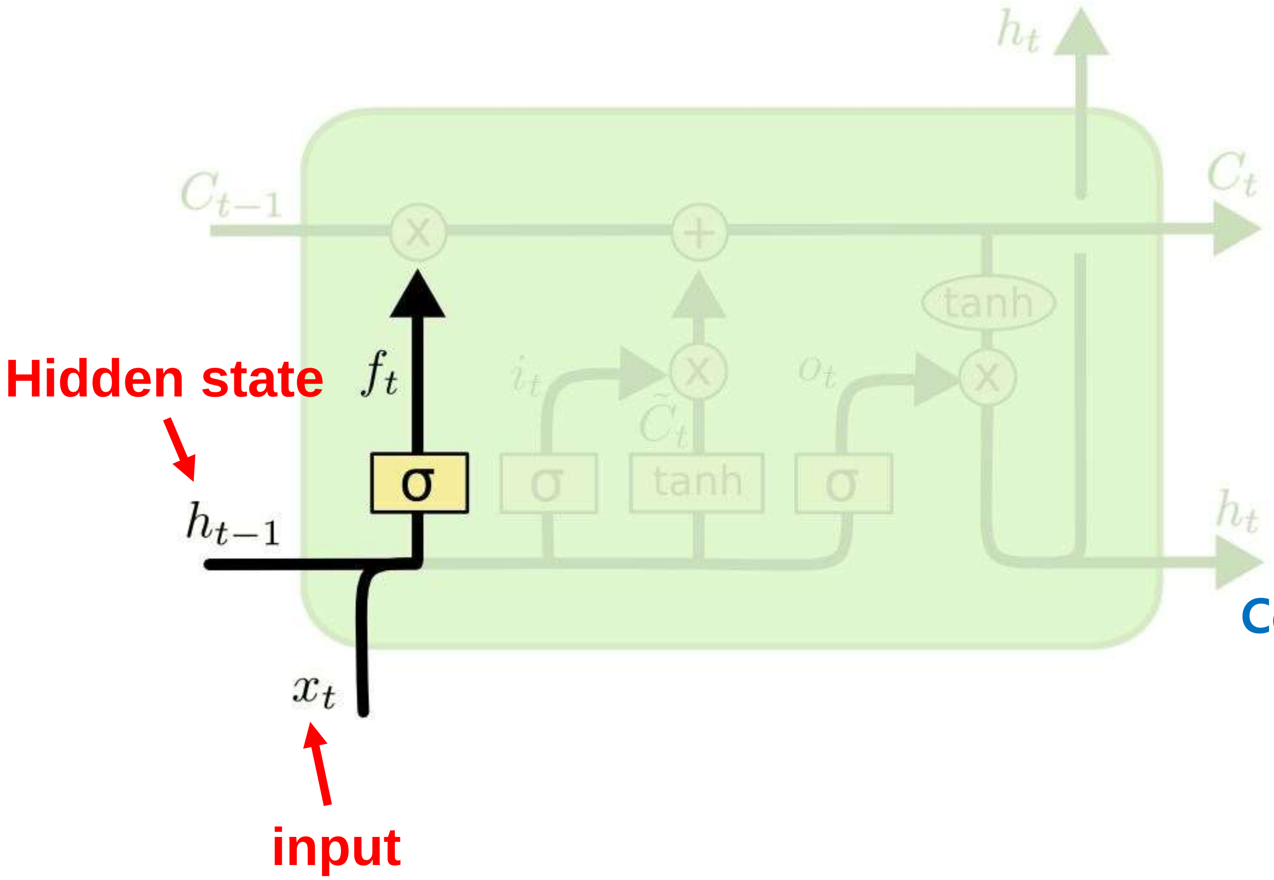


Cell State



Cell state
Next Cell state
C_{t-1}
C_t
h_t
f_t
i_t
$\tilde{C}_t$
o_t
σ
σ
tanh
σ
tanh
h_{t-1}
h_t
x_t

Forget Gate



- Sigmoid function :

$$\sigma(\mathrm{x}) = \frac{1}{1+e^{-x}}$$

$$\boxed{f_t} = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

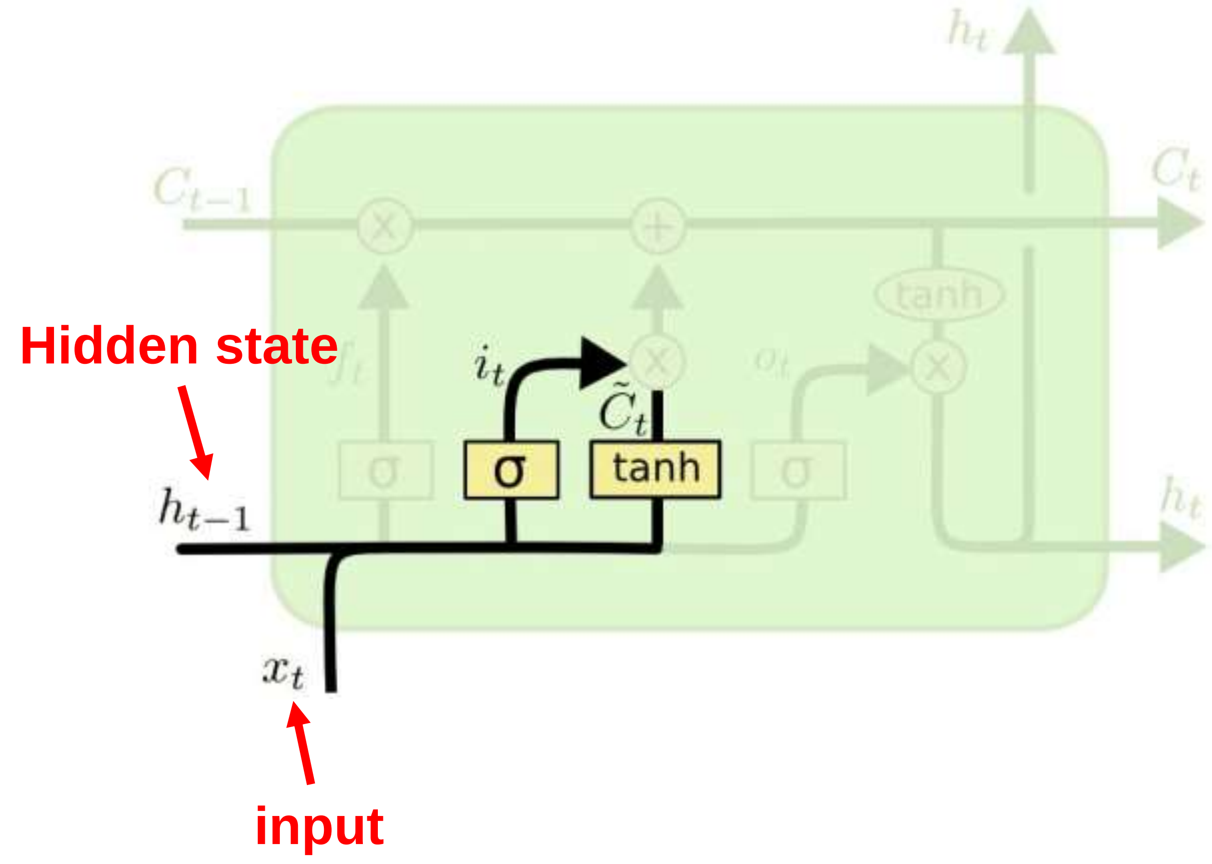
Cell state C_{t-1}을 잊는다

0

1

Cell state C_{t-1}를 기억한다

Input Gate



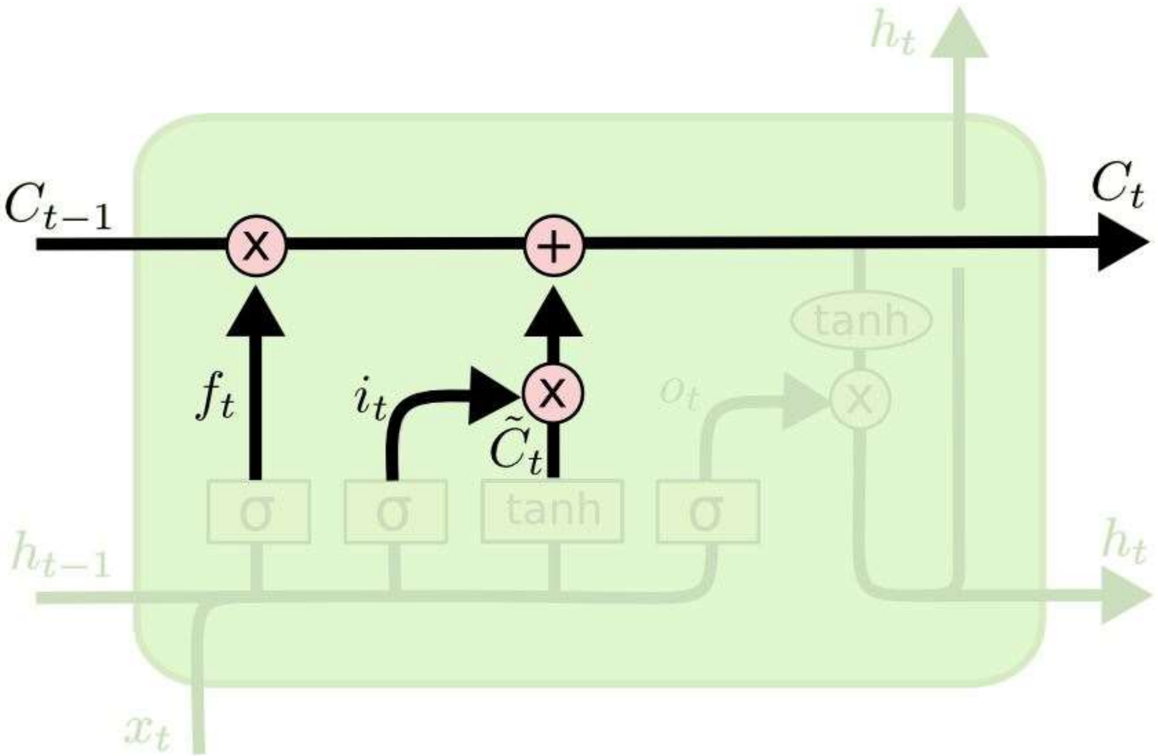
$\tilde{C}_t$의 가중치

$$\boxed{i_t} = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\boxed{\tilde{C}_t} = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Cell State에 추가될 수 있는 새로운 후보 값

Update



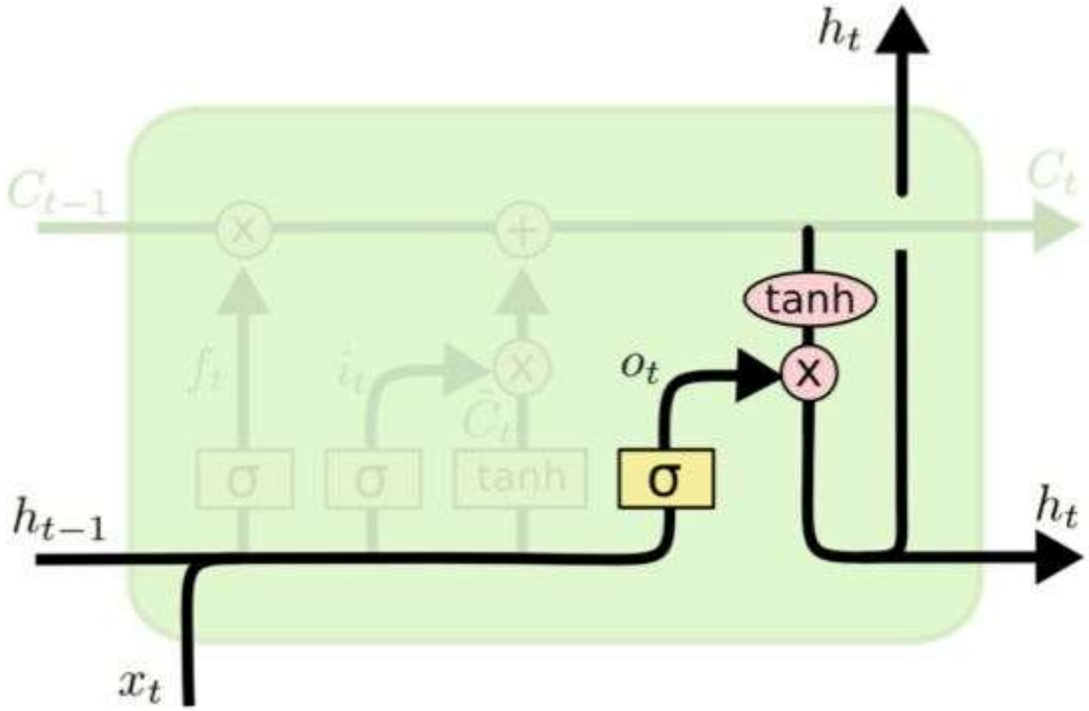
Cell state

$$C_t = f_t * C_{t-1} + \underline{i_t * \tilde{C}_t}$$

Forget Gate

Input Gate

Output Gate



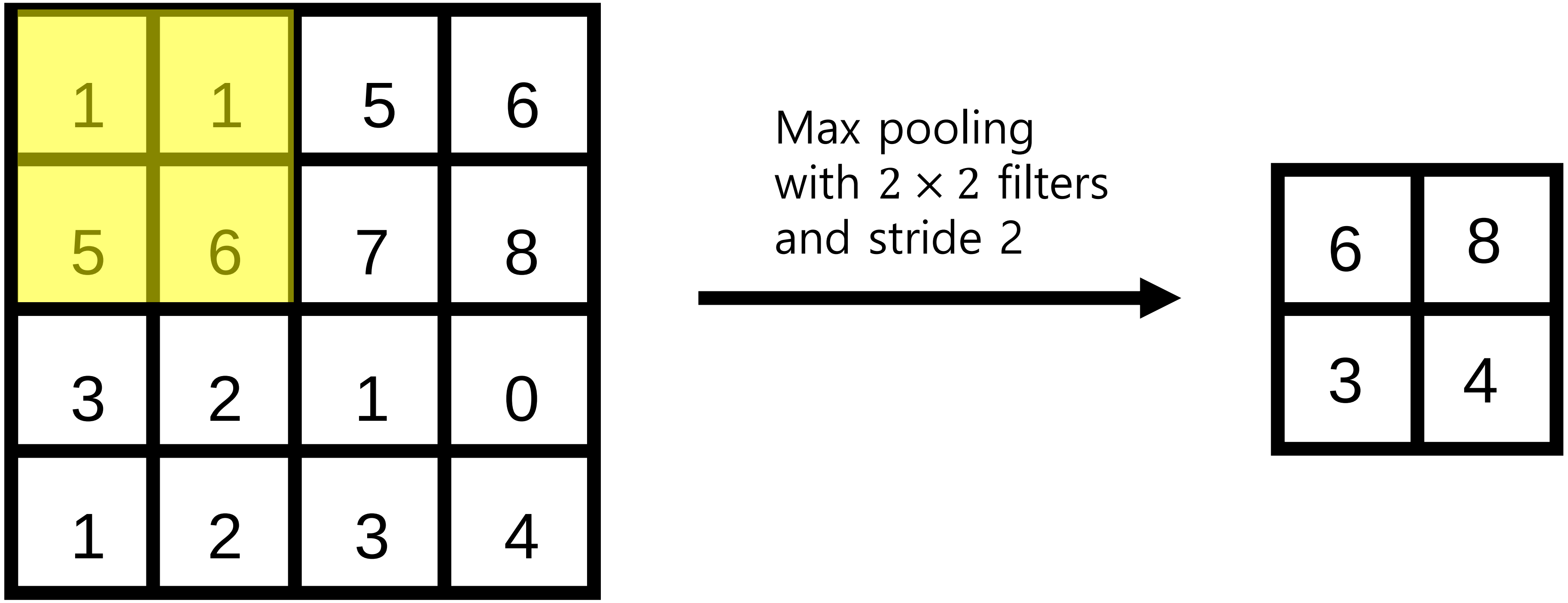
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Pooling

- Pooling aggregator:

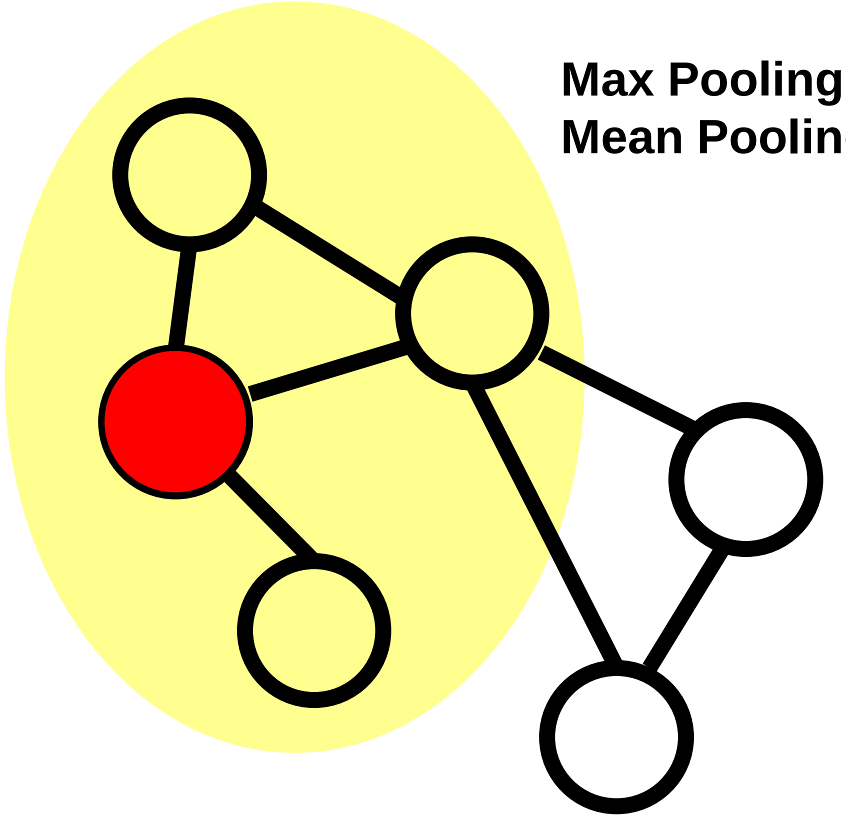
$$AGG = \gamma(\{Qh_u^{k-1}, \forall u \in N(v)\})$$



Pooling

- Pooling aggregator:

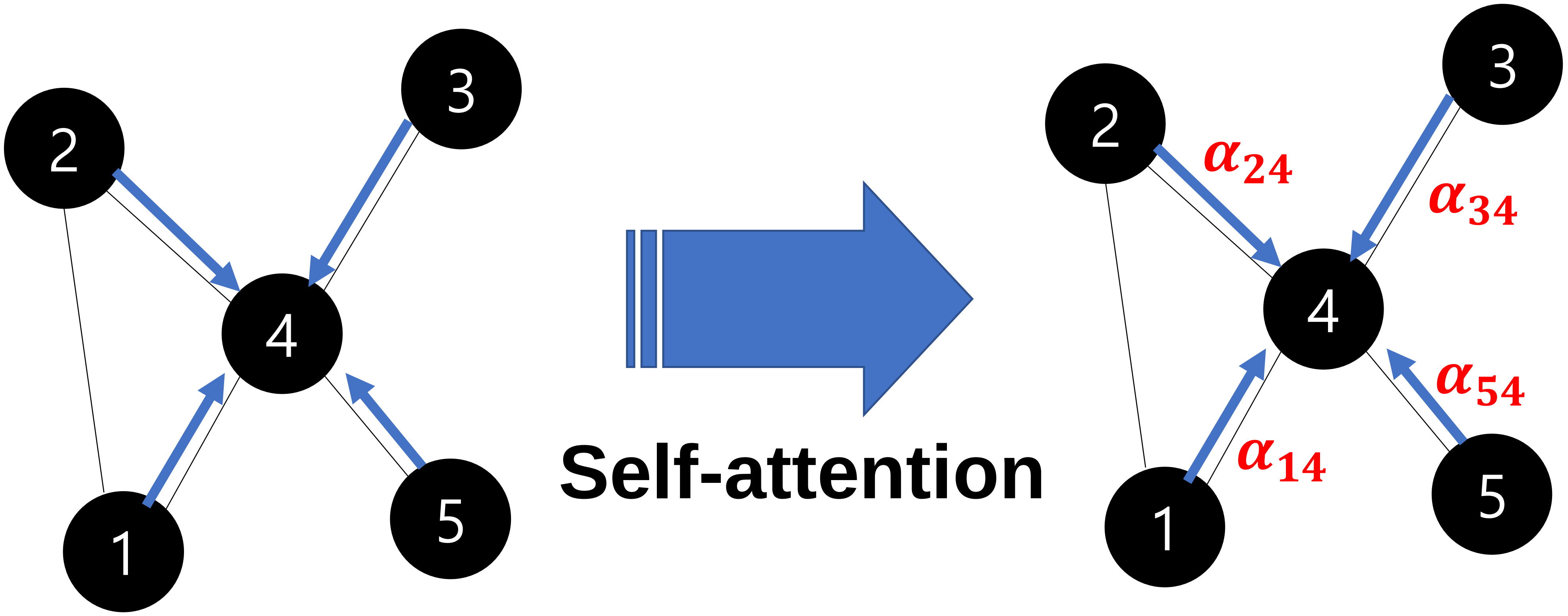
$$AGG = \gamma(\{Qh_u^{k-1}, \forall u \in N(v)\})$$



Max Pooling
Mean Pooling

Table of Contents

Graph Attention Network



Graph Attention Network

- We augment basic graph neural network model with attention.

$$h_v^0 = x_v$$

$$h_v^k = \sigma\left(\sum_{u \in N(v)} \boldsymbol{\alpha_{v,u}} W_k \frac{h_u^{k-1}}{|N(v)|} + \boldsymbol{\alpha_{v,v}} B_k h_v^{k-1}\right), \qquad \forall \mathrm{k} \in \{1, \cdots, K\}$$

$$z_v = h_v^K$$

Attention Score

1

Attention coefficients **self-attention** **Weight matrix**

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

1. Apply a shared linear transformation(W) to every node.
2. Perform ***self-attention***.

Attention coefficients indicate the importance of node j's features to node i.

2

Attention score **Attention coefficients**

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

Masked attention

1. **Normalize** them using the softmax function.

Self-attention

- **Scaled Dot-Product Attention**

The input consists of queries, keys and values.

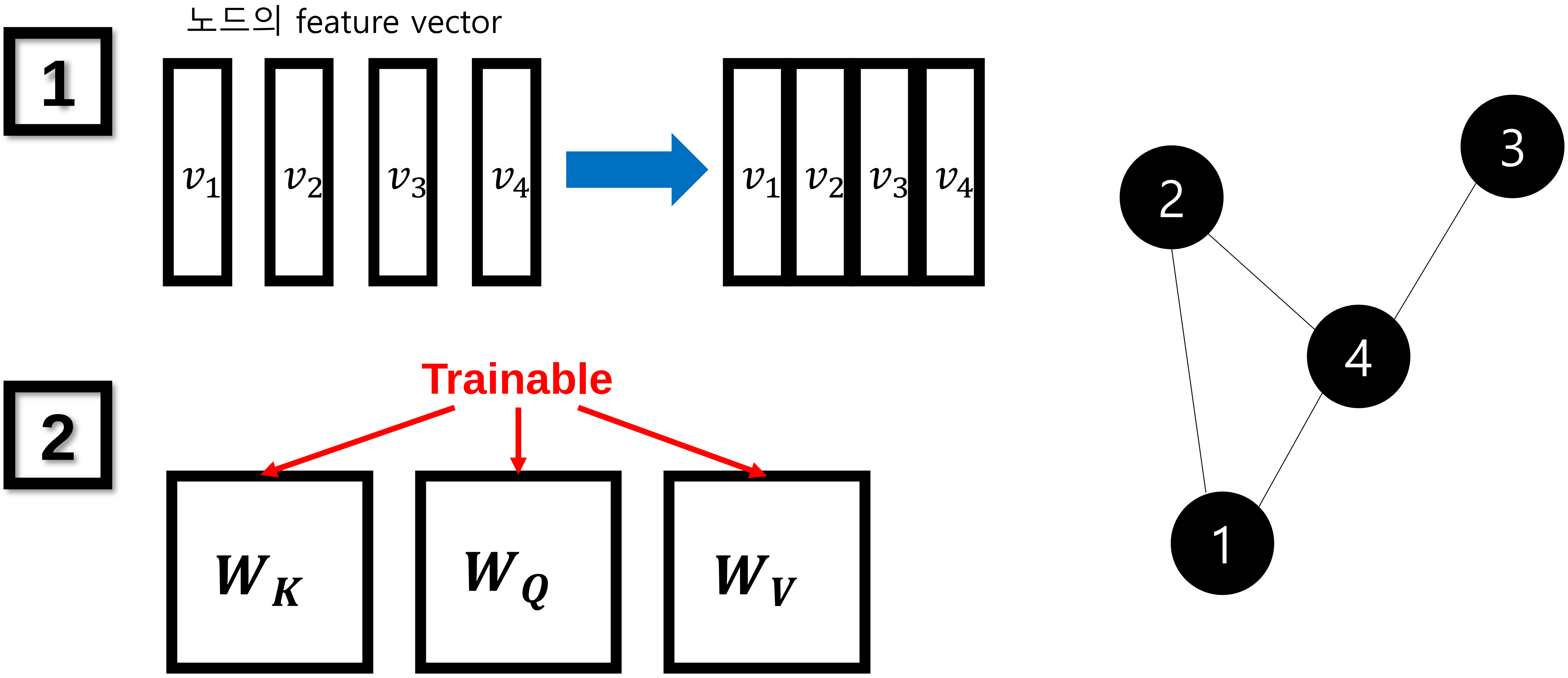
We compute the matrix of outputs as:

Attention Score

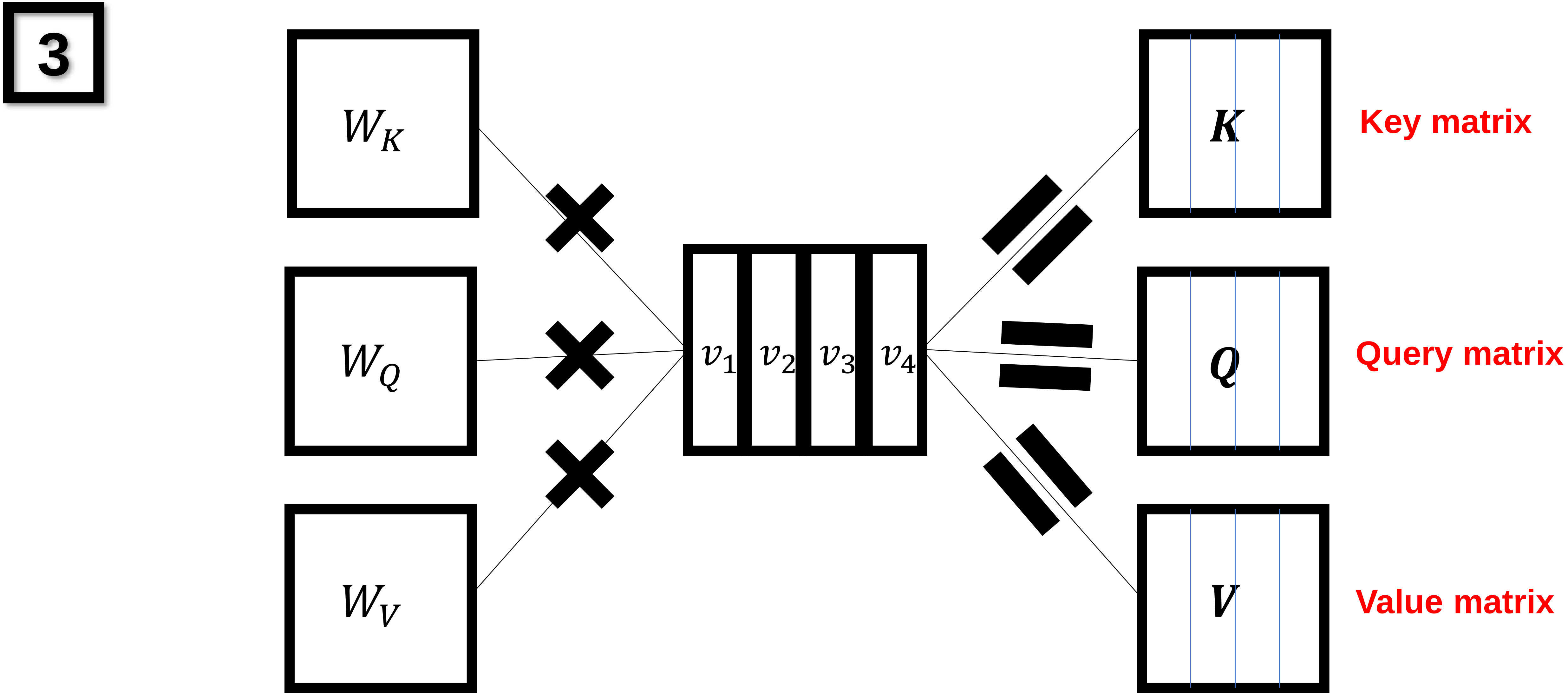
$$Att(K, Q, V) = softmax\left(\boxed{\frac{1}{\sqrt{d_k}} QK^T}\right) V$$

where $K = \begin{pmatrix} k_1^T \\ \vdots \\ k_n^T \end{pmatrix} \in \mathbb{R}^{n \times d_k}$, $Q = \begin{pmatrix} q_1^T \\ \vdots \\ q_m^T \end{pmatrix} \in \mathbb{R}^{m \times d_k}$ and $V = \begin{pmatrix} v_1^T \\ \vdots \\ v_n^T \end{pmatrix} \in \mathbb{R}^{n \times d_v}$.

Self-attention

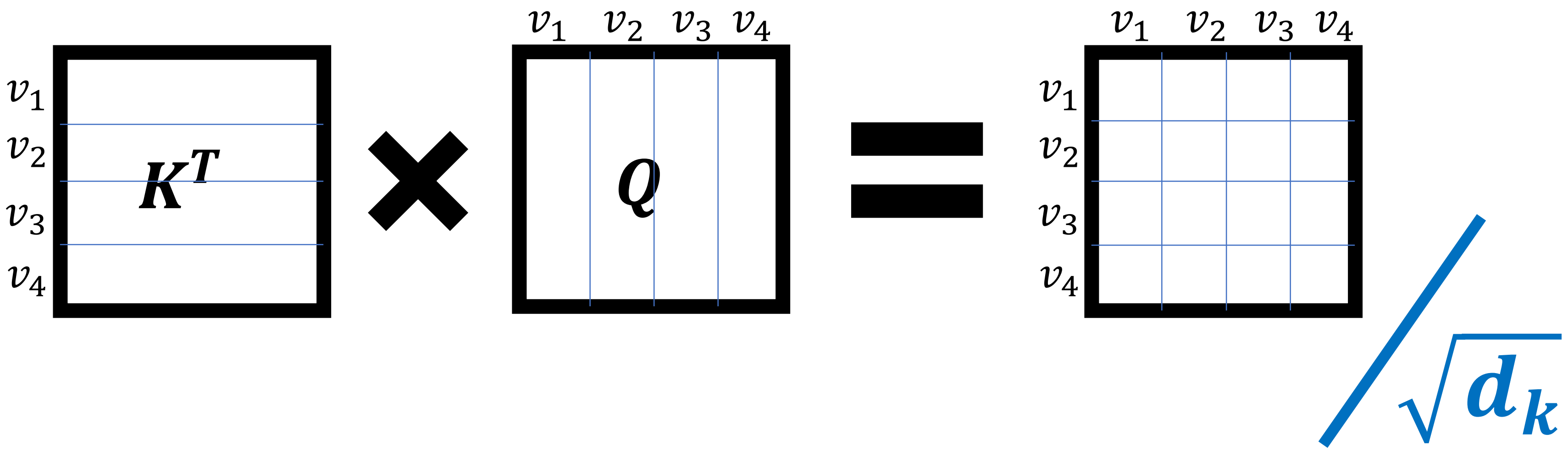


Self-attention



Self-attention

4



→ **Attention coefficients**

Softmax → **Attention score**

3

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

1. Compute a linear combination of the features.
2. Apply a *nonlinear* function σ.

expand to

multi-head attention

4

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

K <u>independent</u> attention mechanism

- || : concatenation
- α_{ij}^k: normalized attention coefficients computed by the **k-th** attention mechanism(a^k)
- W^k: weight matrix of k-th attentional head

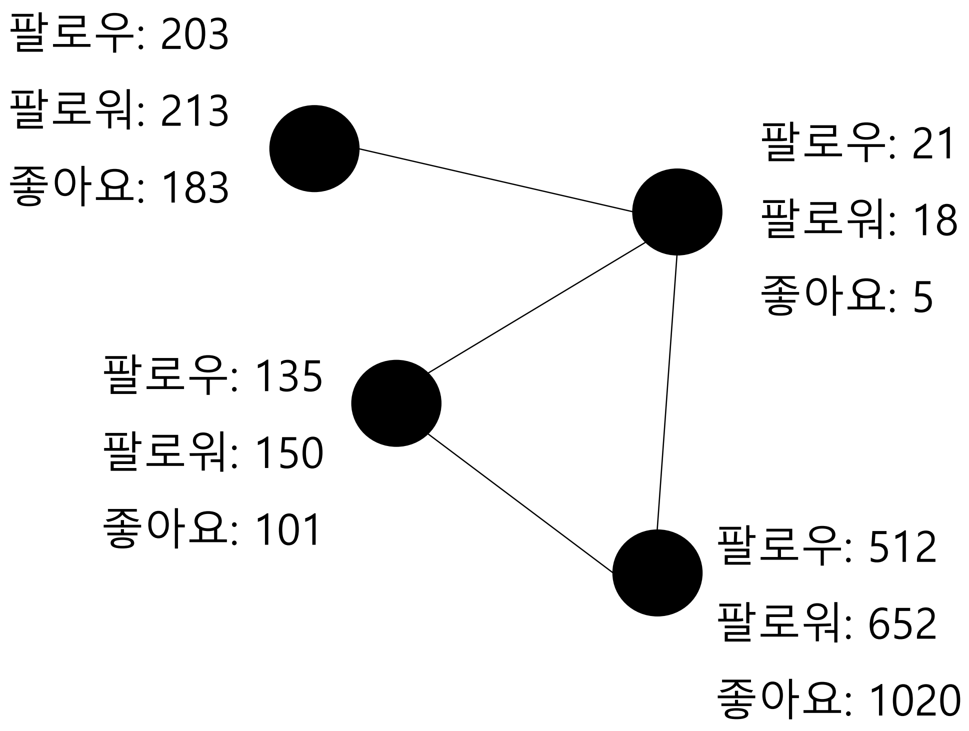
Multi-head Attention

ex) Instagram

팔로워 $\leftarrow head_1$

팔로우 $\leftarrow head_2$

좋아요 $\leftarrow head_3$



Multi-head Attention

$$MultiHead(Q, K, V) = Concat(head_1, \cdots, head_h)W^O$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

and the projections $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

Thank you for listening.