Graph Neural Network

Hyelin Choi

Department of Mathematics
Sungkyunkwan University

February 8, 2024

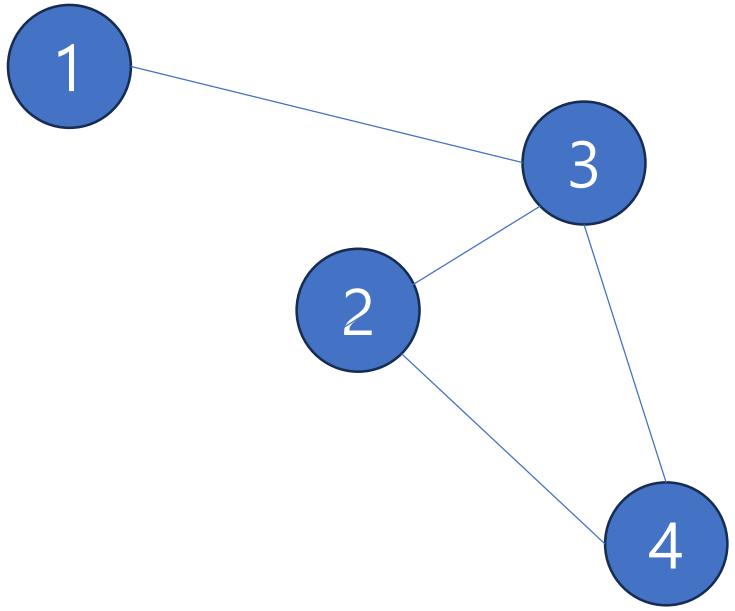
Table of Contents

Table of Contents

Graph

- Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$
- Set of Vertex(Node) $\mathbf{V}$
- Set of Edge $\mathbf{E}$
- Neighborhood
- Adjacency Matrix $\mathbf{A}$
- Feature Vector ← GNN에서 사용

Example



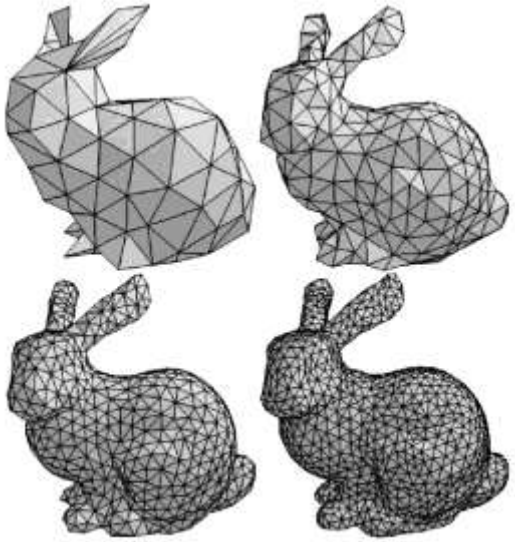
- Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$
- Set of Node **V={1, 2, 3, 4}**
- Set of Edge **E = {{1,3},{2,3},{2,4},{3,4}}**
- Neighborhood

  ex) neighborhood of node 2 = node 3, node 4

- Adjacency Matrix $\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$

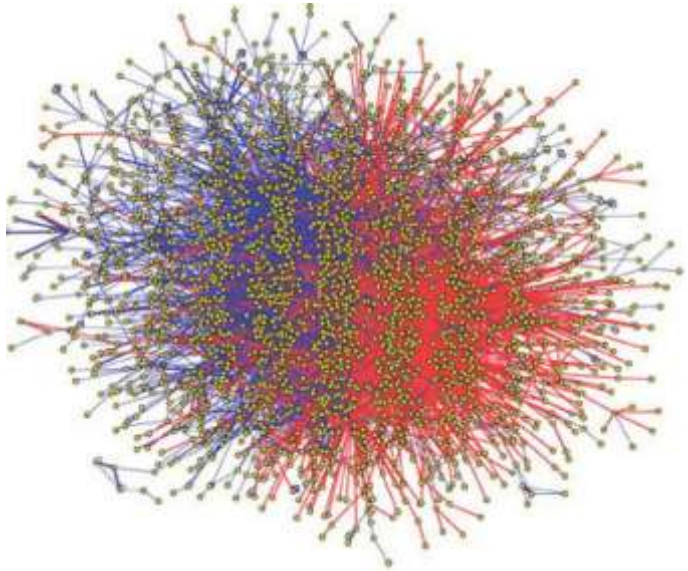
Table of Contents

Graph



3D Mesh

Social Network

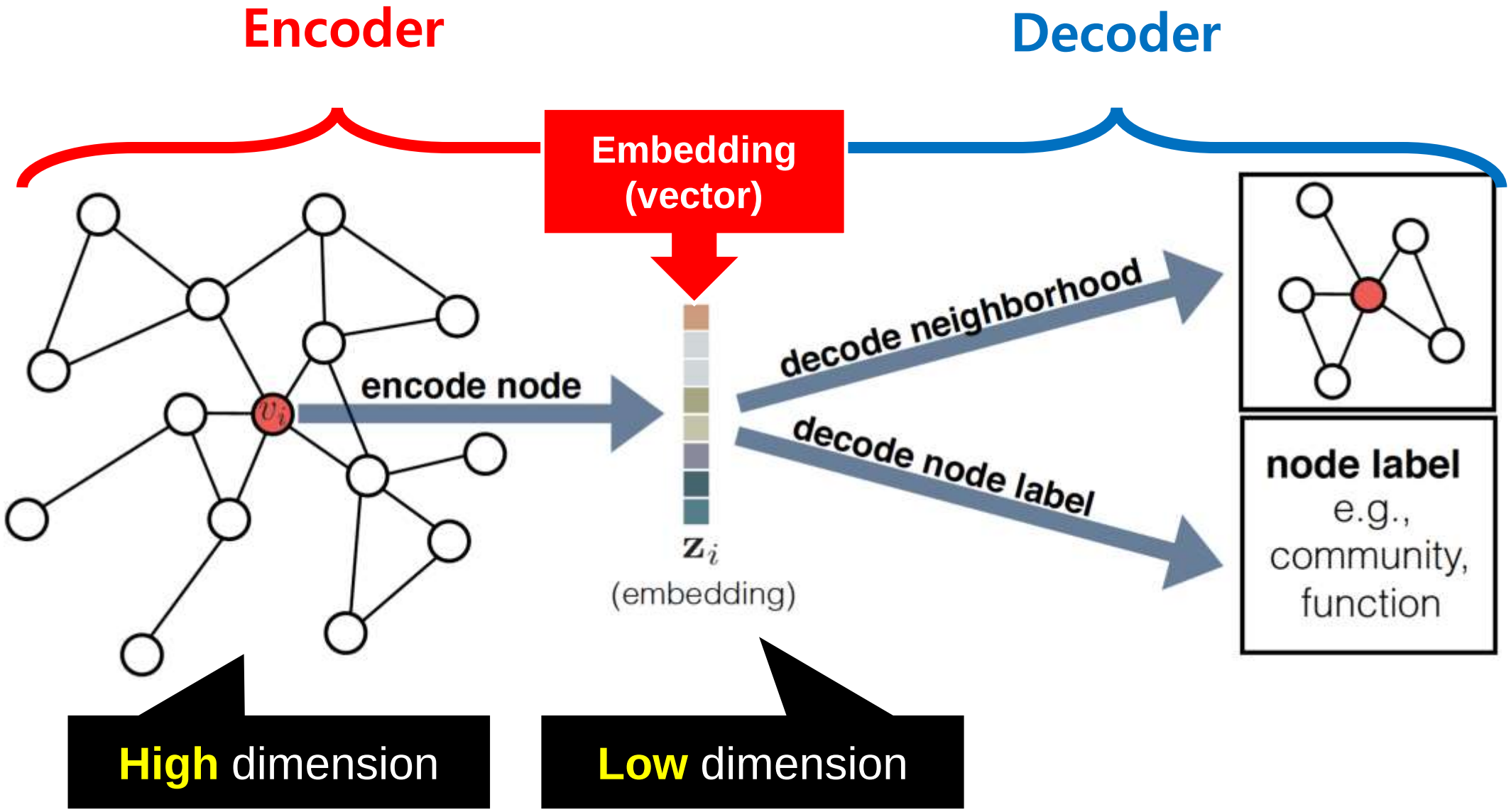


Biological Network

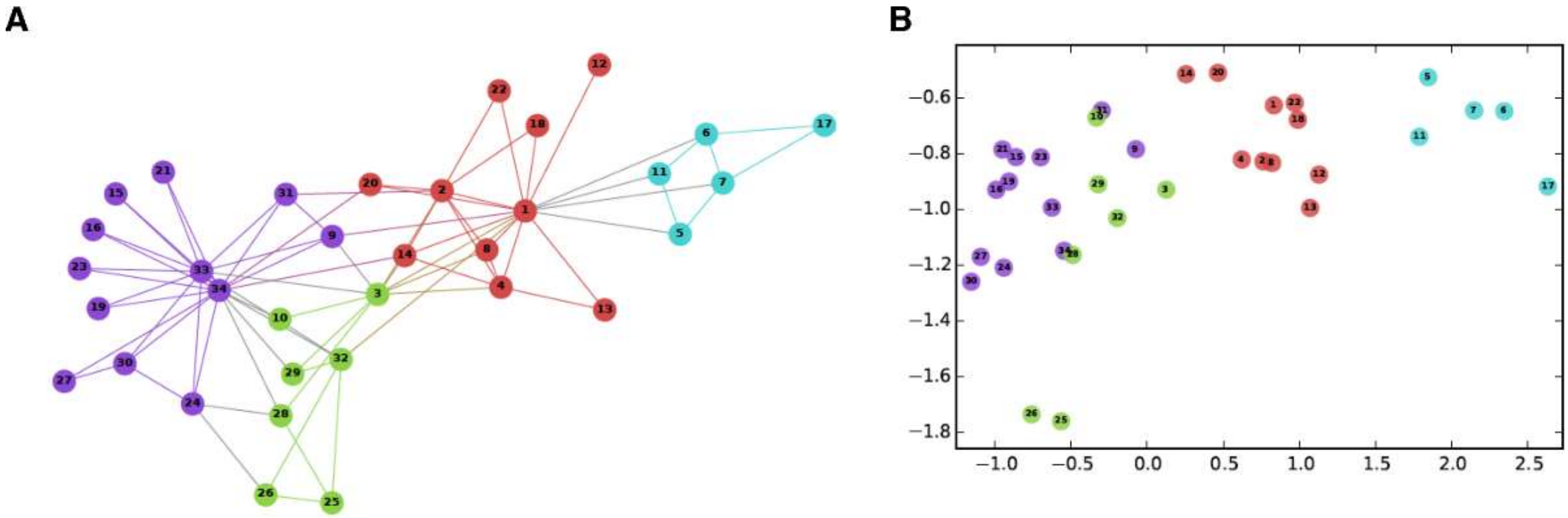
It is difficult to express these examples in grid structure.

Table of Contents

Encoder-Decoder



Encoder-Decoder



Encoder-Decoder

The **encoder** is a function,

contains **trainable** parameters

$$ENC : V \quad (4)$$

that maps nodes to vector embeddi

The **decoder** is a function,

No trainable parameters

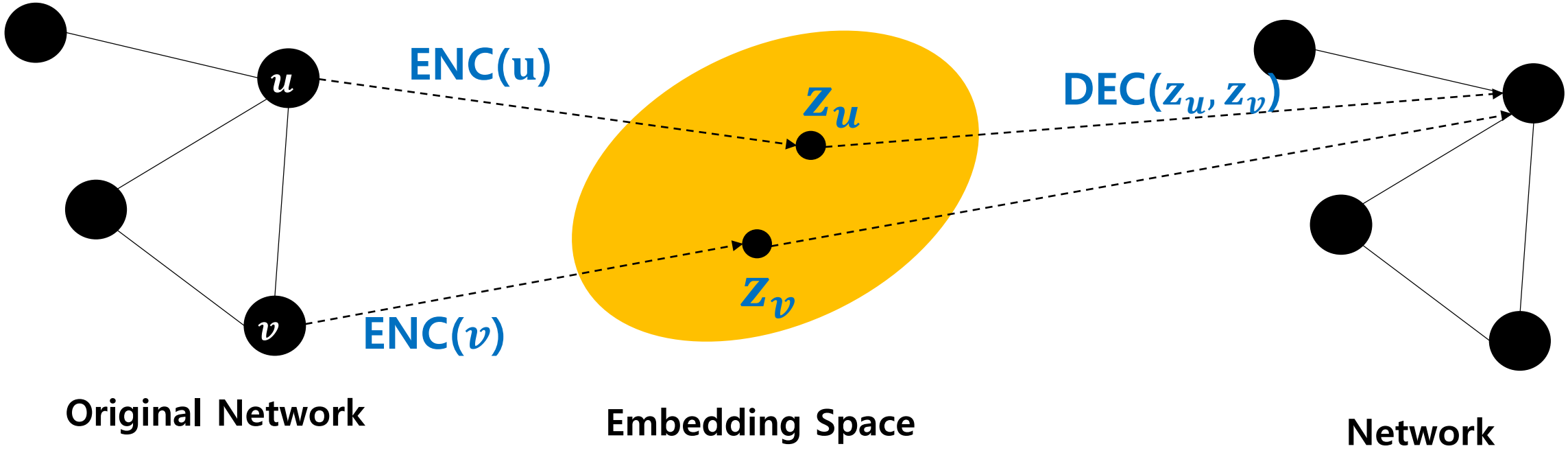
$$DEC : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+, \quad (5)$$

that accepts a set of node embeddings and decodes user-specified graph statistics from these embeddings.

ex)

1. $||z_i - z_j||_2^2$
2. $z_i^T z_j$
3. $\frac{e^{z_i^T z_j}}{\sum_{k \in V} e^{z_i^T z_k}}$

Encoder-Decoder



Goal: Optimize the encoder and decoder mappings to minimize the error

$$DEC\left(z_u, z_v\right) \approx s_G\left(u, v\right)$$

Original Network에서 두 노드 v_i, v_j의 유사도

Downstream Task

하나의 모델을 훈련하면

다음에 새로운 데이터가 들어왔을 때 모델을 새로 훈련하지 않고

이미 훈련된 모델을 이용해서 학습할 수 있다.

Pretrained Model

Downstream Task

Node Embedding

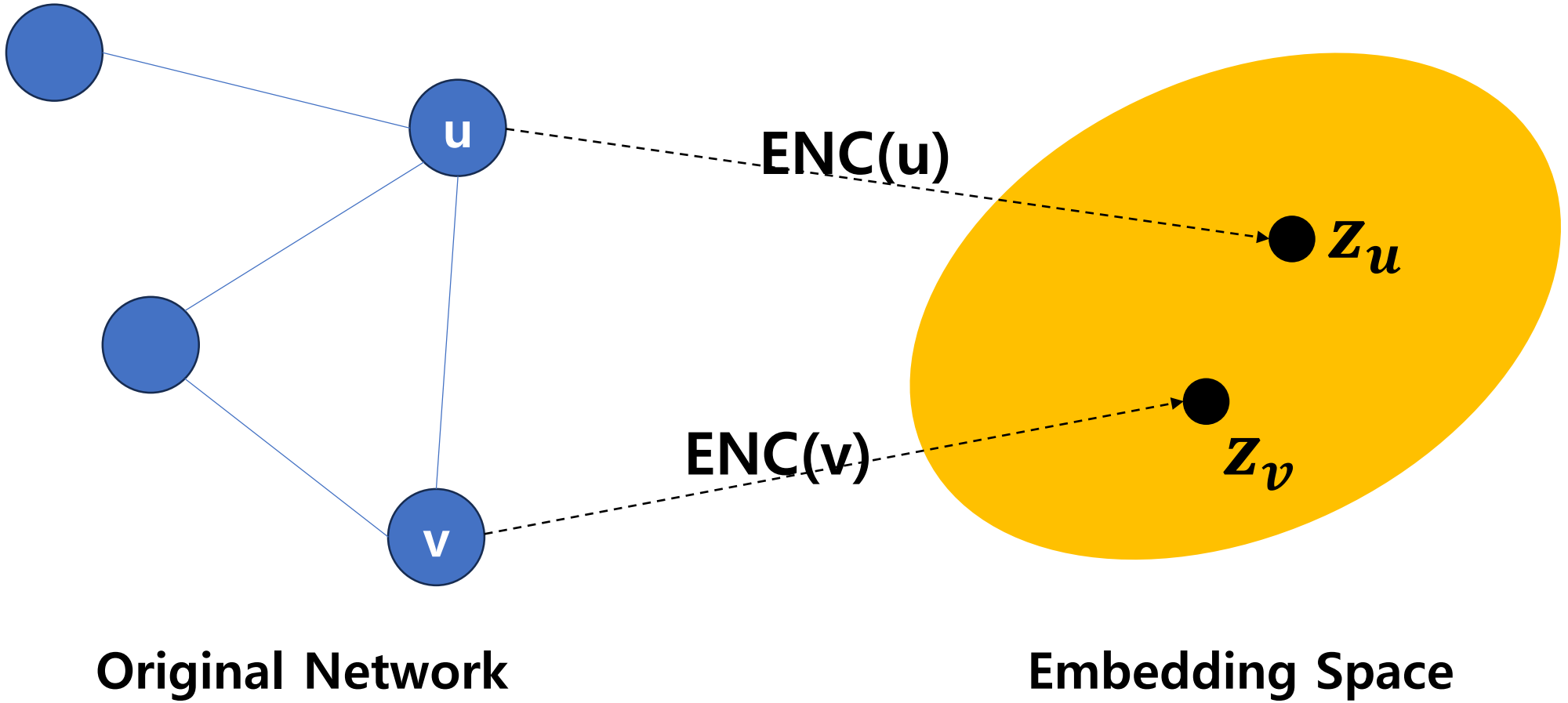


Table of Contents

Shallow Embedding

Type	Method	Decoder	Proximity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps [4]	$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2$	general	$\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization [1]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	GraRep [9]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}, \mathbf{A}_{i,j}^2, ..., \mathbf{A}_{i,j}^k$	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	HOPE [44]	$\mathbf{z}_i^\top \mathbf{z}_j$	general	$\|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
Random walk	DeepWalk [46]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \mid v_i)$	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$
	node2vec [27]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \mid v_i)$ (biased)	$-s_{\mathcal{G}}(v_i, v_j) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$

Table : A summary of some well-known **shallow embedding algorithm**.

Shallow Embedding

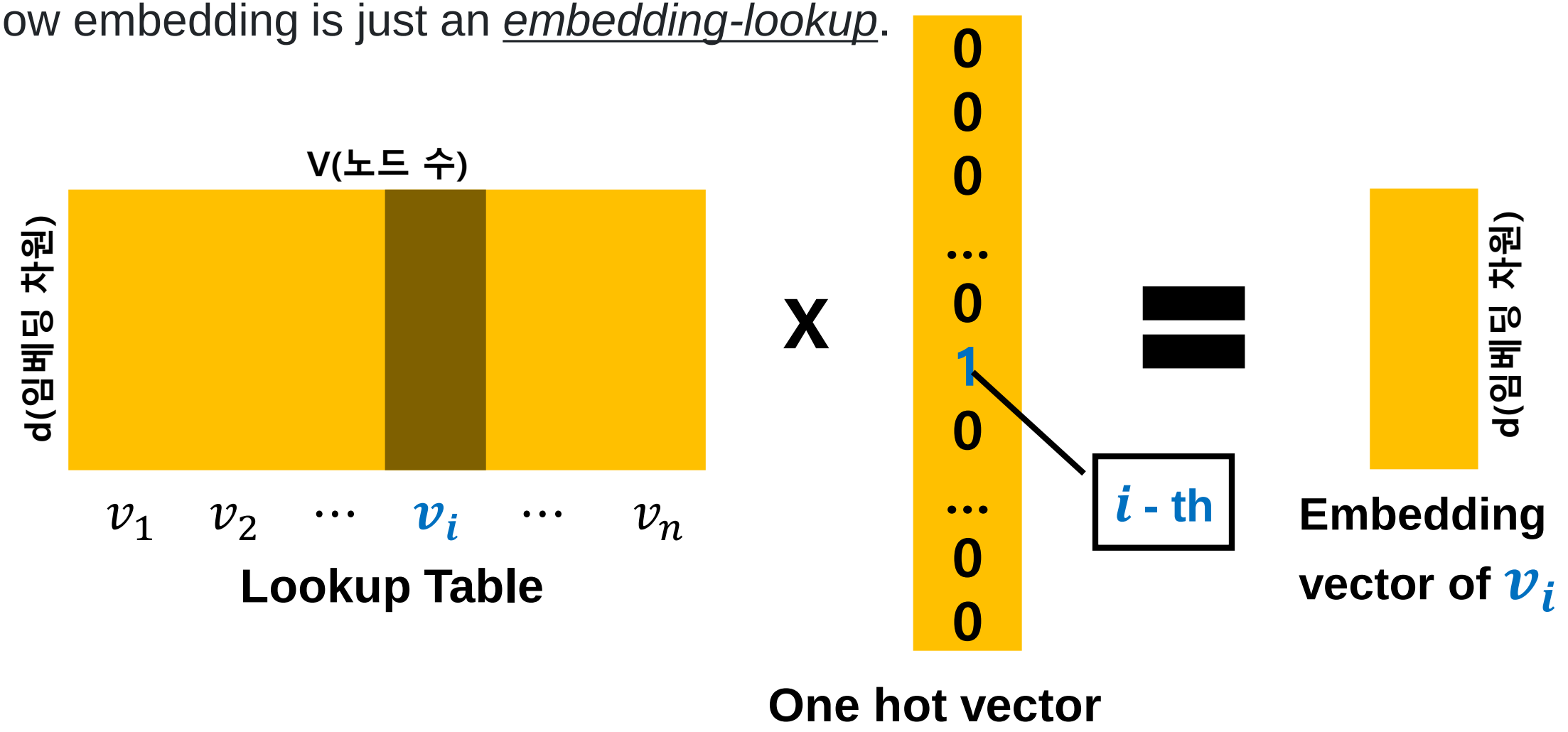
For shallow embedding approaches, the encoder function is

$$ENC(v) = \mathbf{Z}v,$$

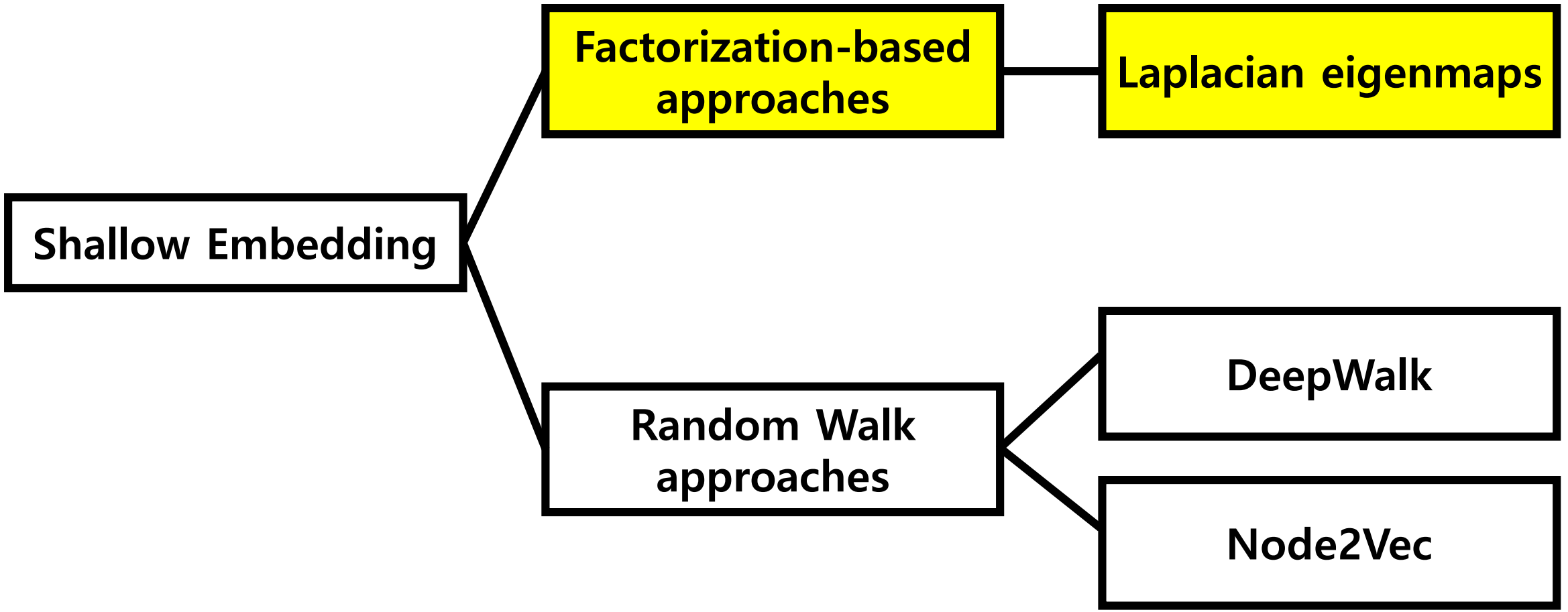
where **Z** is a matrix containing embedding vectors for all nodes and v is a one-hot indicator vector.

Shallow Embedding

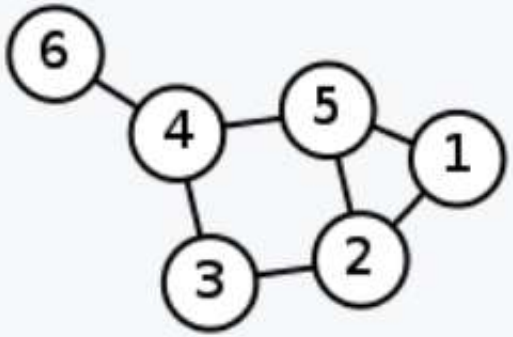
Shallow embedding is just an *embedding-lookup*.



Shallow Embedding



Laplacian Eigenmaps

Labelled graph	D Degree matrix	A Adjacency matrix	L Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

- Degree matrix D
- Adjacency matrix A
- Laplacian matrix L

- **L = D-A**
- $D_{ii} = \sum_j A_{ji}$

Laplacian Eigenmaps

- Graph $\rightarrow$ $\mathbb{R}$

We denote node $i's$ embedding as $\boldsymbol{y_i}$.

We wish to minimize

$$\sum_{i,j=1}^{n}(y_i - y_j)^2 A_{ij}$$

for $y_i \in \mathbb{R}$ and $1 \leq i \leq n$.

Laplacian Eigenmaps

We have

$$\frac{1}{2}\sum_{i,j=1}^{n}(y_i - y_j)^2 A_{ij} = \boldsymbol{y}^T L\boldsymbol{y}$$

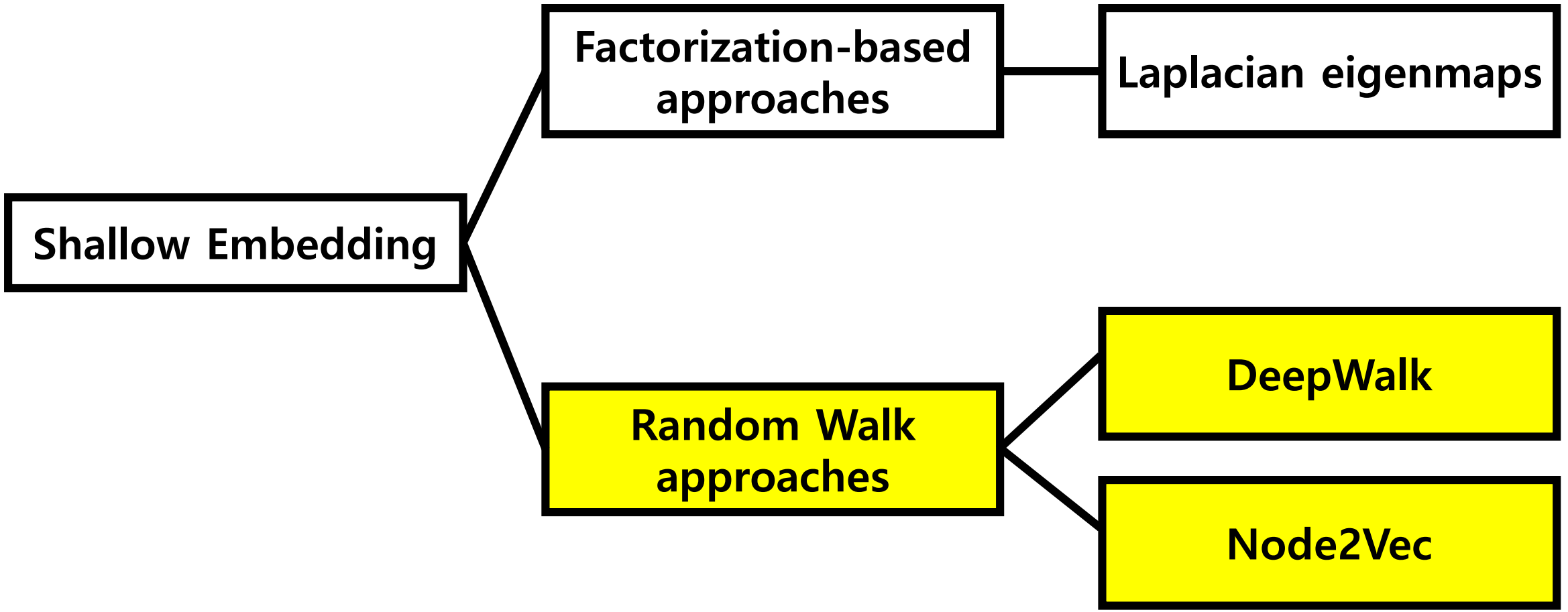
where $L = D - A$ and $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)^T$.

Therefore, the minimization problem reduces to finding

$$\underset{y}{argmin}\, y^T L y.$$

인접한 node끼리 유사해지도록 각 노드의 feature vector을 update하고 싶은 경우, 이 식을 최소화하는 y를 구하면 된다.

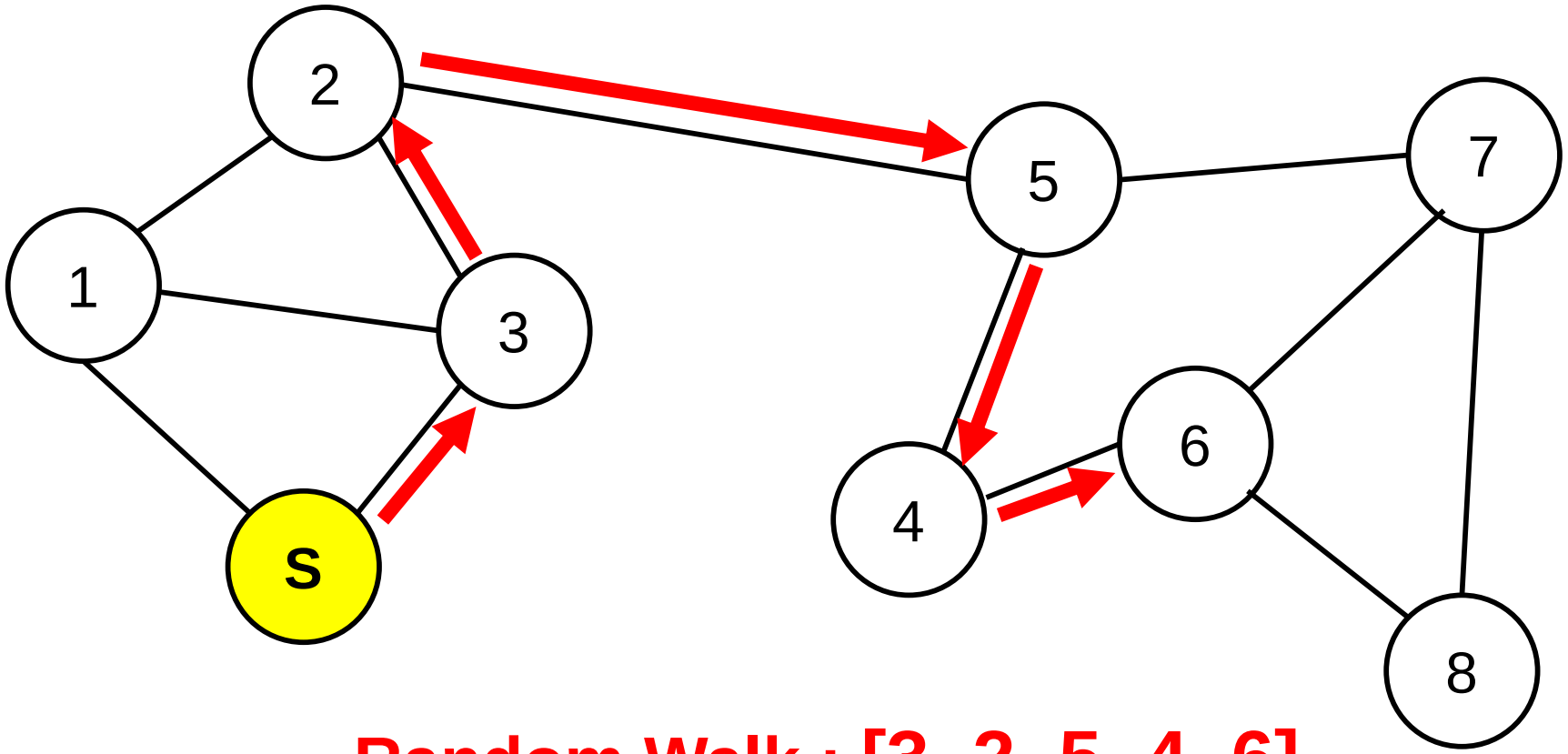
Shallow Embedding



Random Walk

Parameter: number of random walk (32~64 per node)
number of step size (40)

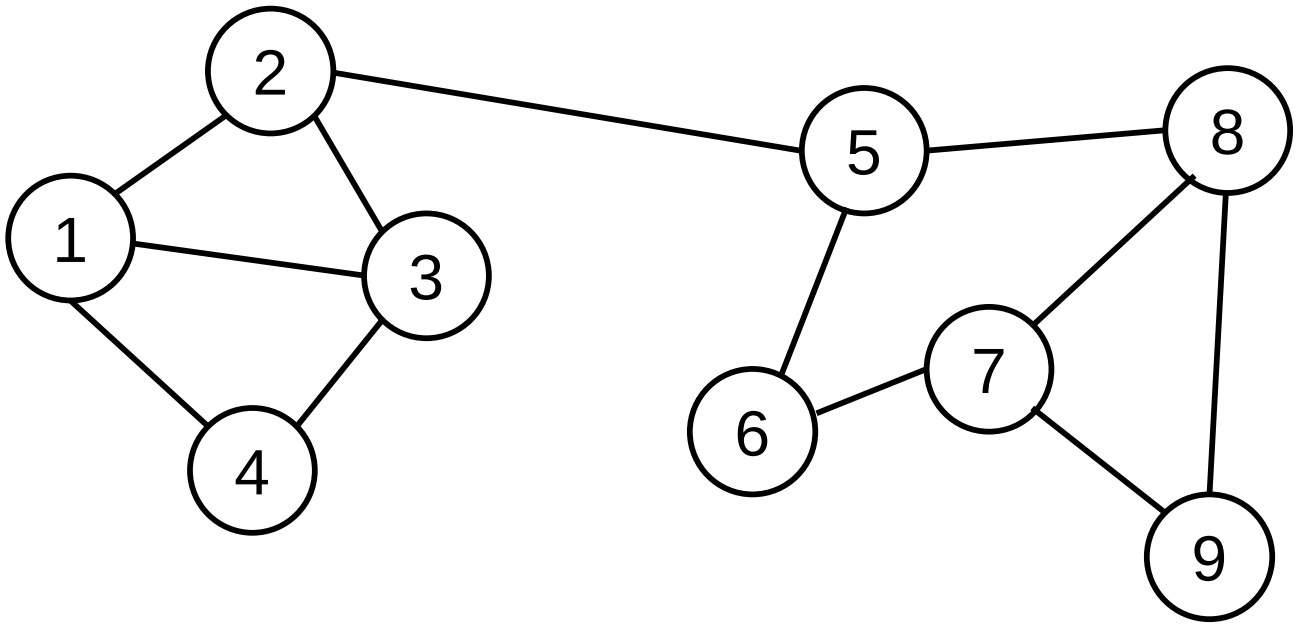
Step size : 5



Random Walk : [3, 2, 5, 4, 6]

Random Walk

- number of random walk :2
- number of step size : 3



- Node 1

[2,5,8]
[3,2,5]

- Node 2

[5,8,9]
[5,6,7]

- Node 3

[2,5,8]
…

Random Walk Approaches

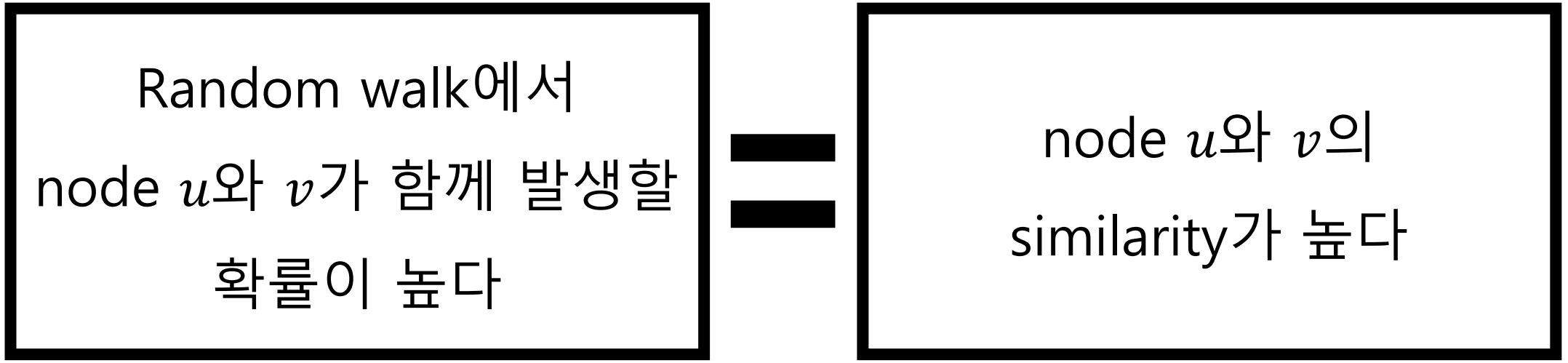


Table of Contents

Skip-gram

목표: 중심단어로부터 주변단어를 유추

Window size: 1

ex) I don’t like studying math.

중심단어: studying

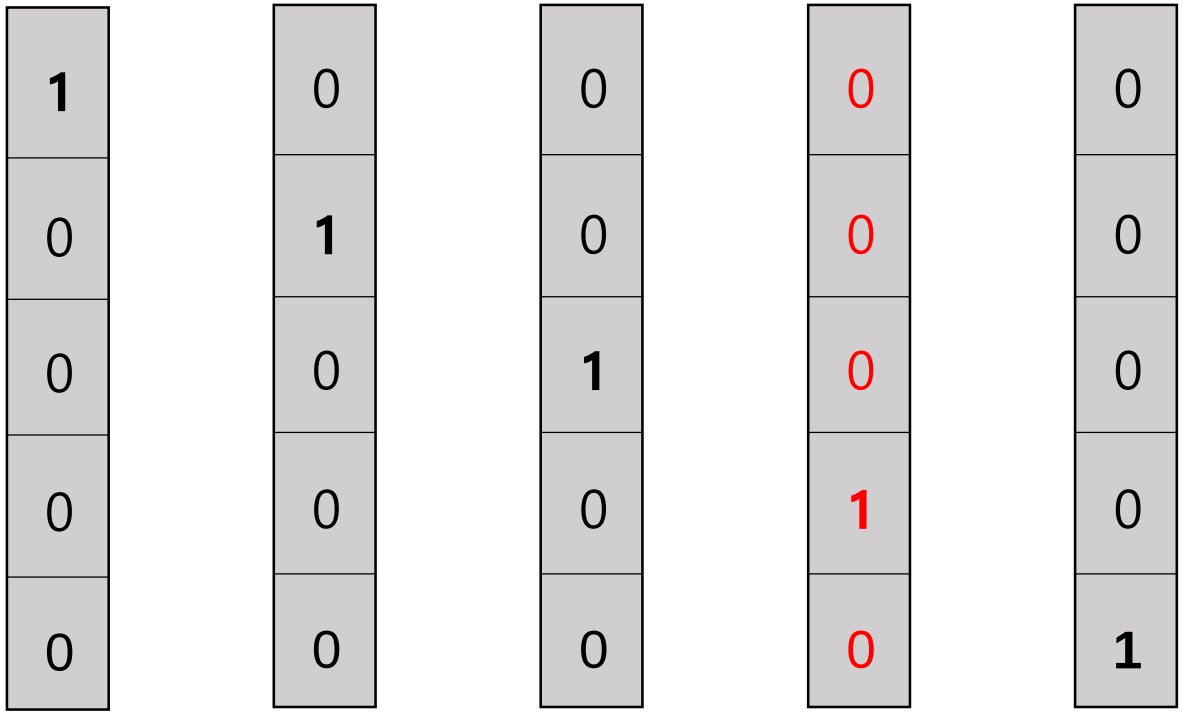
주변단어: “like”, “math”

Skip-gram

Step 1

- 단어를 one-hot vector으로 표현한다.
- 중심 단어와 window size를 설정한다.

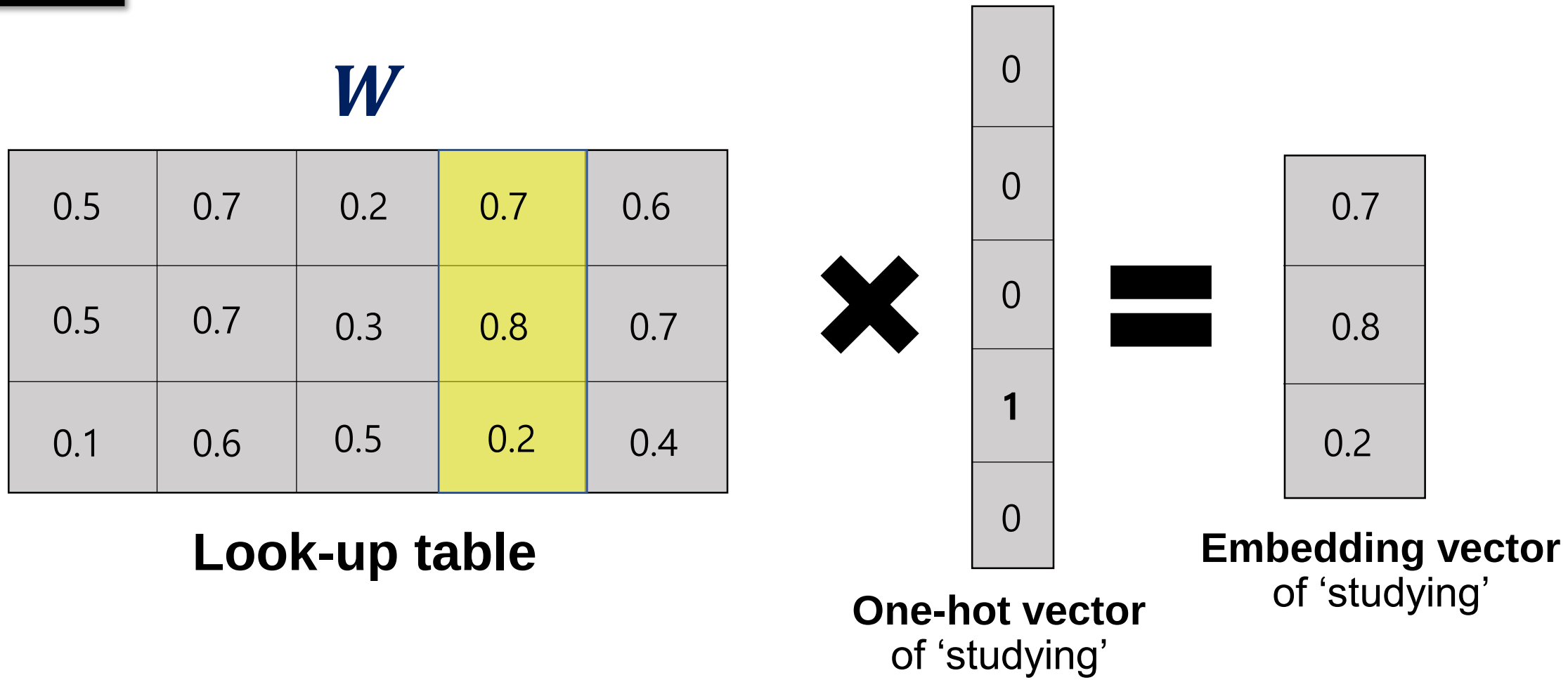
I don’t like studying math.



window size: 1

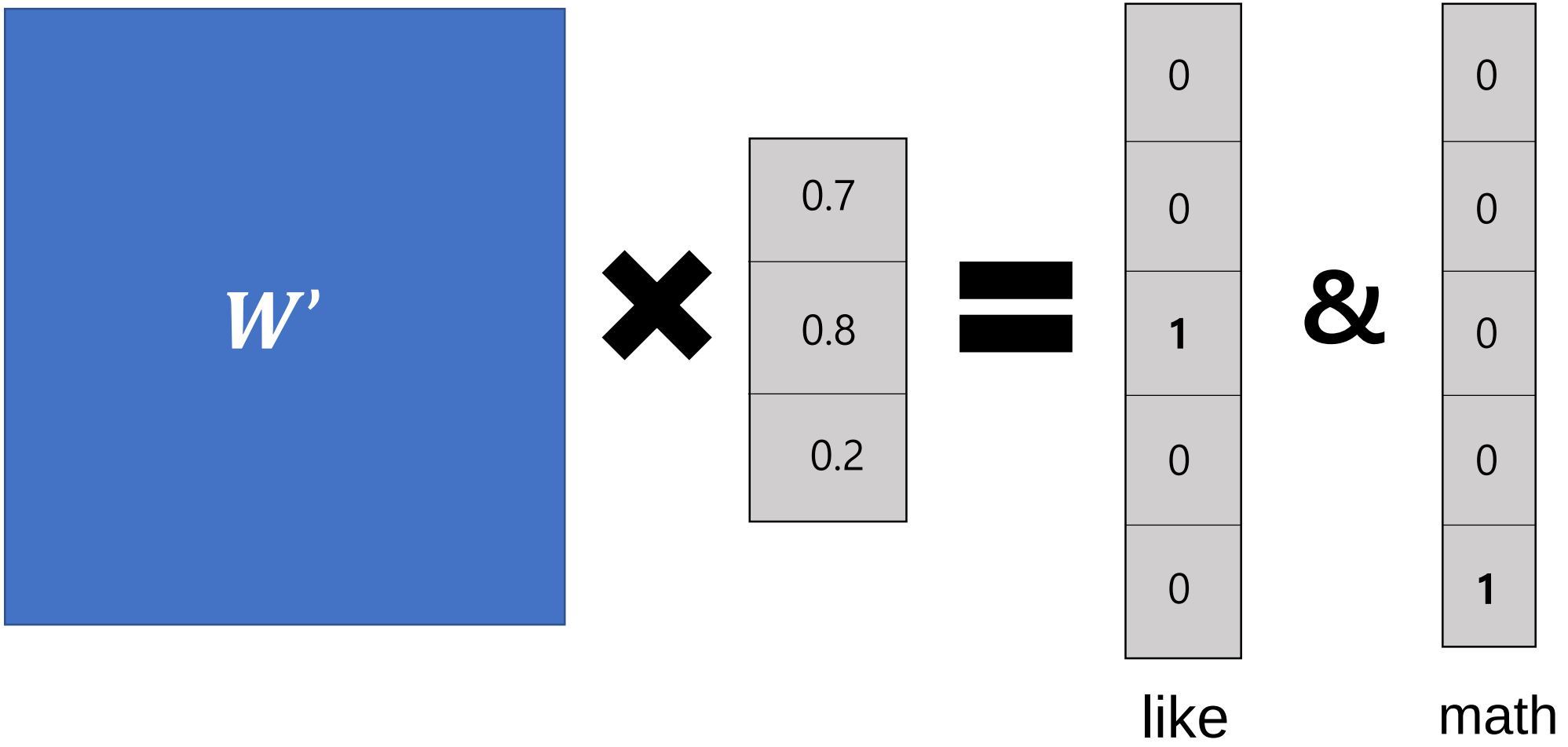
Skip-gram

Step 2 중심단어를 원하는 차원으로 임베딩, ex) 3-dim

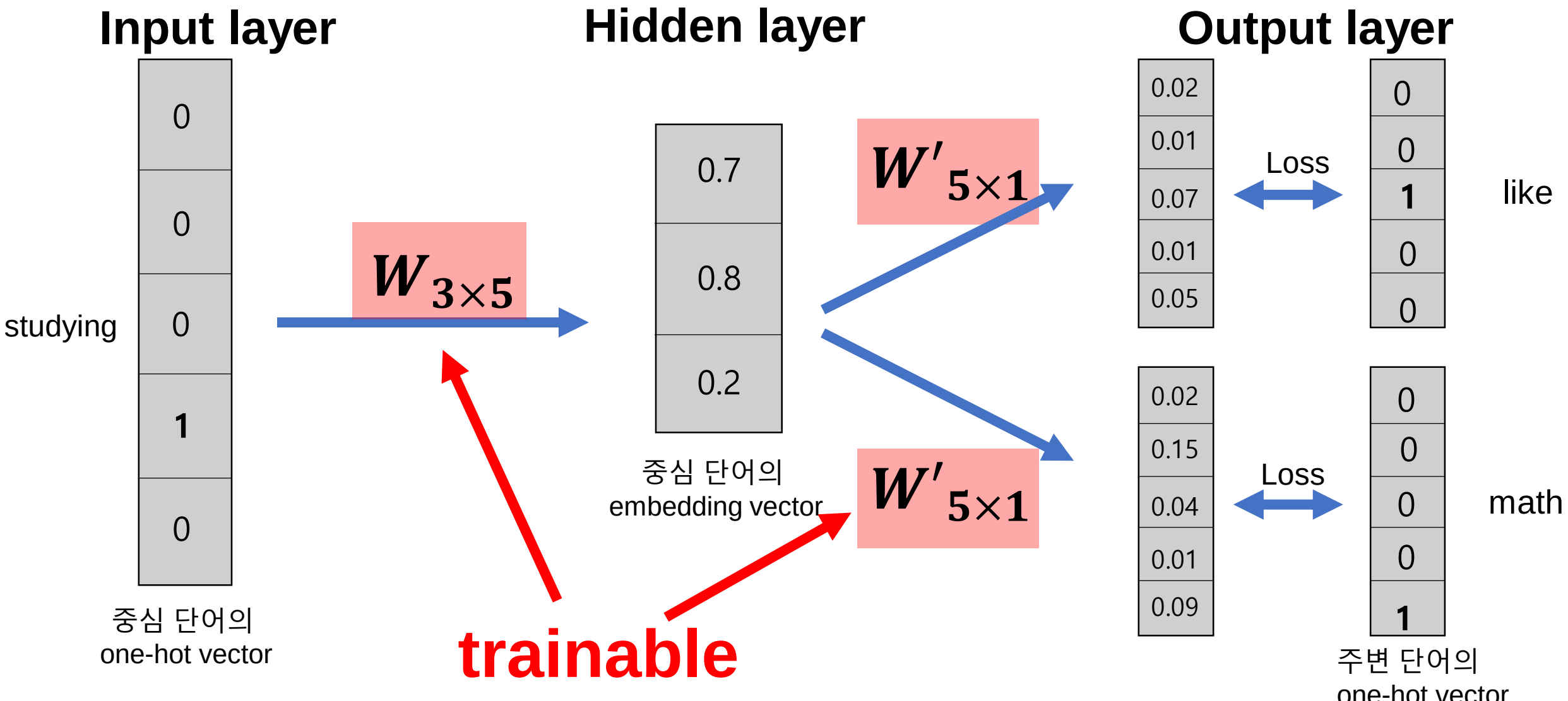


Skip-gram

Step 3 Loss function이 최소가 되도록, 가중치 행렬 W, W'을 학습

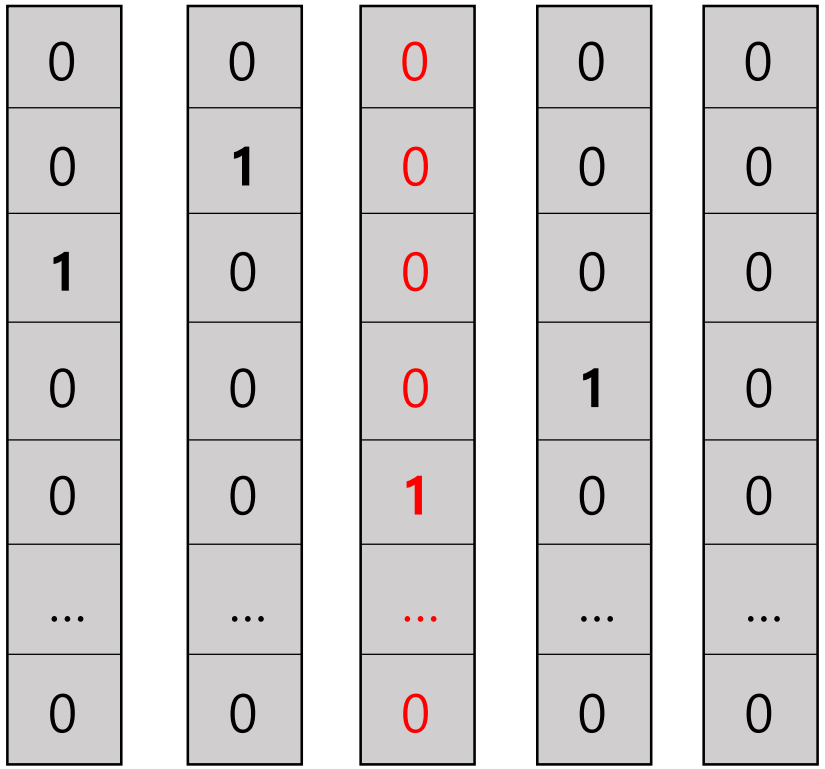


Skip-gram


Input layer
Hidden layer
Output layer
0
0
0
1
0
studying
중심 단어의
one-hot vector
$W_{3\times5}$
0.7
0.8
0.2
중심 단어의
embedding vector
$W'_{5\times1}$
$W'_{5\times1}$
trainable
0.02
0.01
0.07
0.01
0.05
Loss
0
0
1
0
0
like
0.02
0.15
0.04
0.01
0.09
Loss
0
0
0
0
1
math
주변 단어의
one-hot vector

DeepWalk

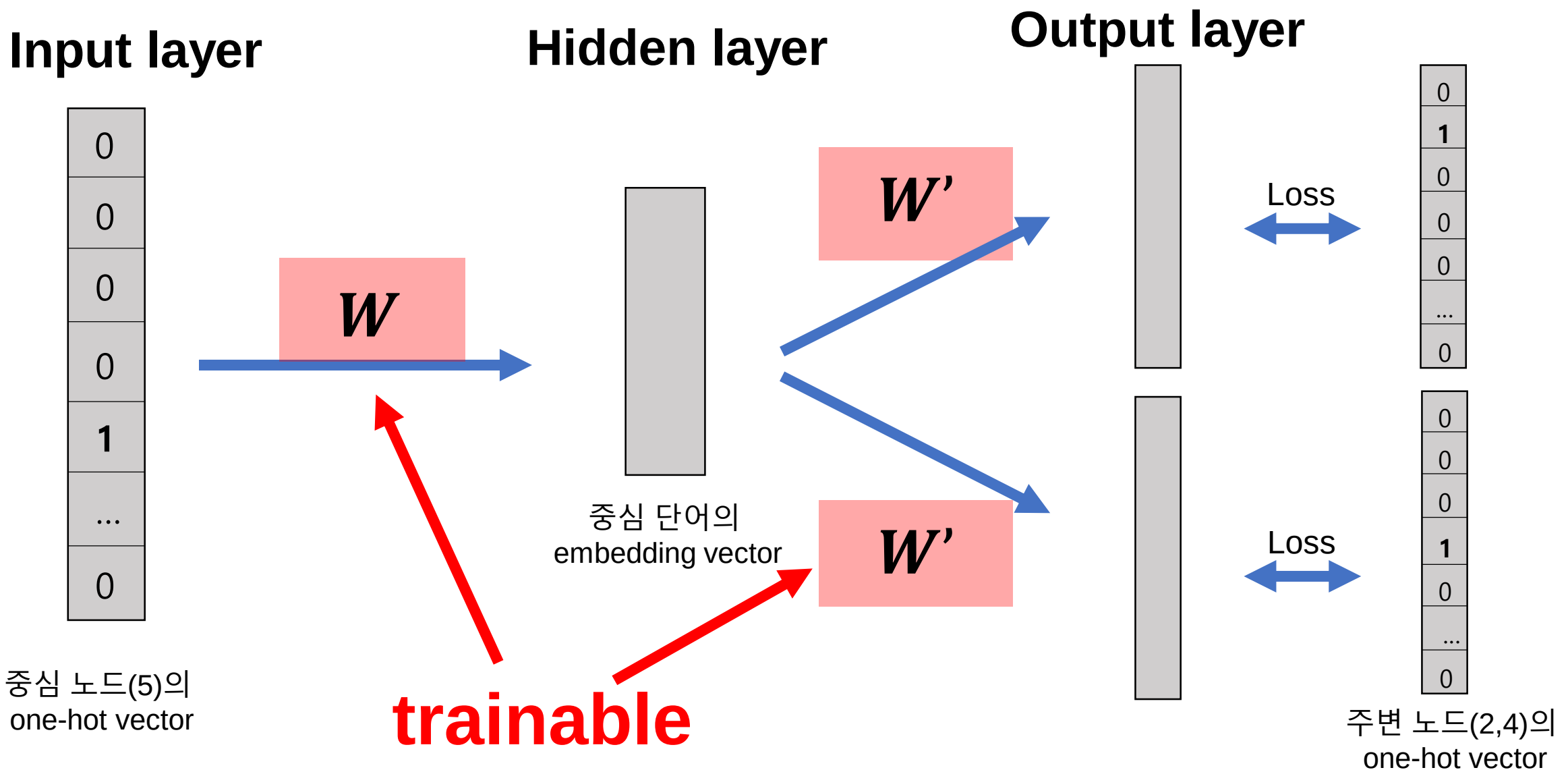
- **Node = {1, 2, 3, 4, 5, 6, 7, 8}**
- **Random Walk : [3, 2, 5, 4, 6]**



1. 중심 노드 및 window size 설정
2. Look up table(W) X one-hot vector
3. 그 결과에 X W'
4. 주변 노드의 one-hot vector 와의 차이를 줄이는 방향으로 W, W′ 학습

이 과정을 모든 node, 모든 random walk에 대하여 반복

DeepWalk


Input layer
Hidden layer
Output layer
0
0
0
0
1
...
0
W
W'
W'
중심 단어의
embedding vector
Loss
Loss
0
1
0
0
0
...
0
0
0
0
1
0
...
0
중심 노드(5)의
one-hot vector
trainable
주변 노드(2,4)의
one-hot vector

DeepWalk

Algorithm 1 DeepWalk(G, w, d, γ, t)

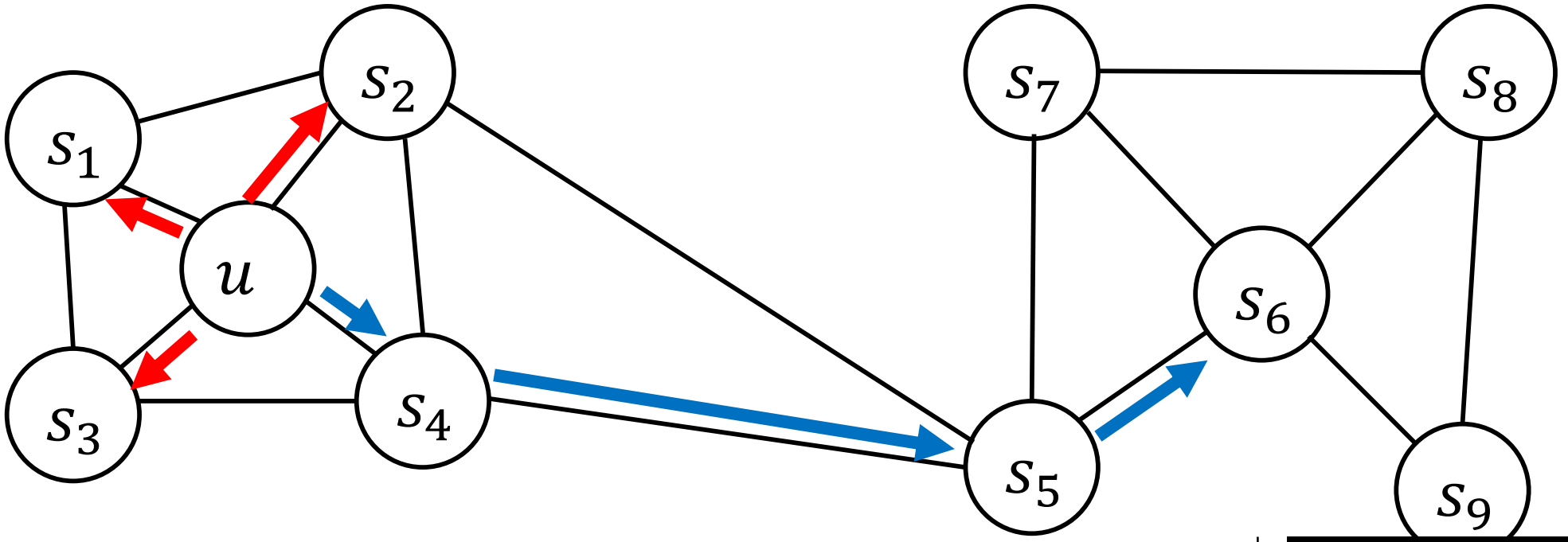
Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

for each $v_j \in \mathcal{W}_{v_i}$ **do**
  for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for

Node2Vec



→ BFS (Breath-first Sampling) — Local View

→ DFS (Depth-first Sampling) — Global View

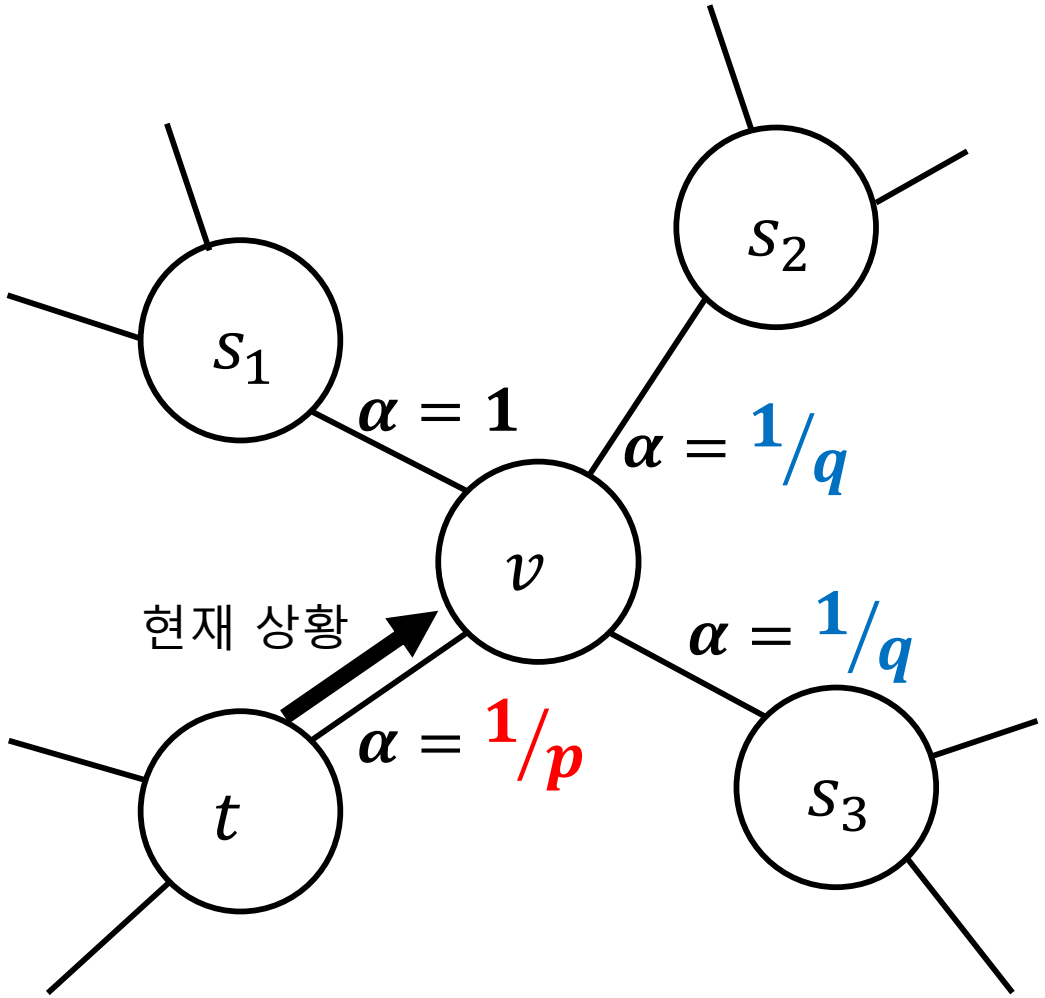
Node2Vec

- **Return parameter, p**
- **In-out parameter, q**
- Search bias, α

$p \downarrow \approx$ BFS (너비우선탐색)

$q \downarrow \approx$ DFS (깊이우선탐색)

$q \uparrow \approx$ BFS (너비우선탐색)



Node2Vec

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
  π = PreprocessModifiedWeights(G, p, q)
  $G' = (V, E, \pi)$
  Initialize $walks$ to Empty
  for $iter = 1$ **to** r **do**
    for all nodes $u \in V$ **do**
      $walk$ = node2vecWalk(G', u, l)
      Append $walk$ to $walks$
  f = StochasticGradientDescent(k, d, $walks$)
  return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
  Inititalize $walk$ to $[u]$
  for $walk\_iter = 1$ **to** l **do**
    $curr = walk[-1]$
    V_{curr} = GetNeighbors($curr$, G')
    s = AliasSample(V_{curr}, π)
    Append s to $walk$
  return $walk$

Loss Function

Random walk approaches attempt to minimize the following cross-entropy loss:

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)}\right)$$

time complexity ⇑

Loss Function

We use **negative sampling** to reduce time complexity.

$$\log\left(\frac{\exp(z_u^T z_v)}{\sum_{n\in V}\exp(z_u^T z_n)}\right)$$

$$\approx \log(\exp(z_u^T z_v)) - \sum_{i=1}^{k}\log\left(\exp\left(z_u^T z_{n_i}\right)\right), n_i \sim P_V$$

- Higher k gives more robust estimates.

Thank you for listening.