Graph Neural Network

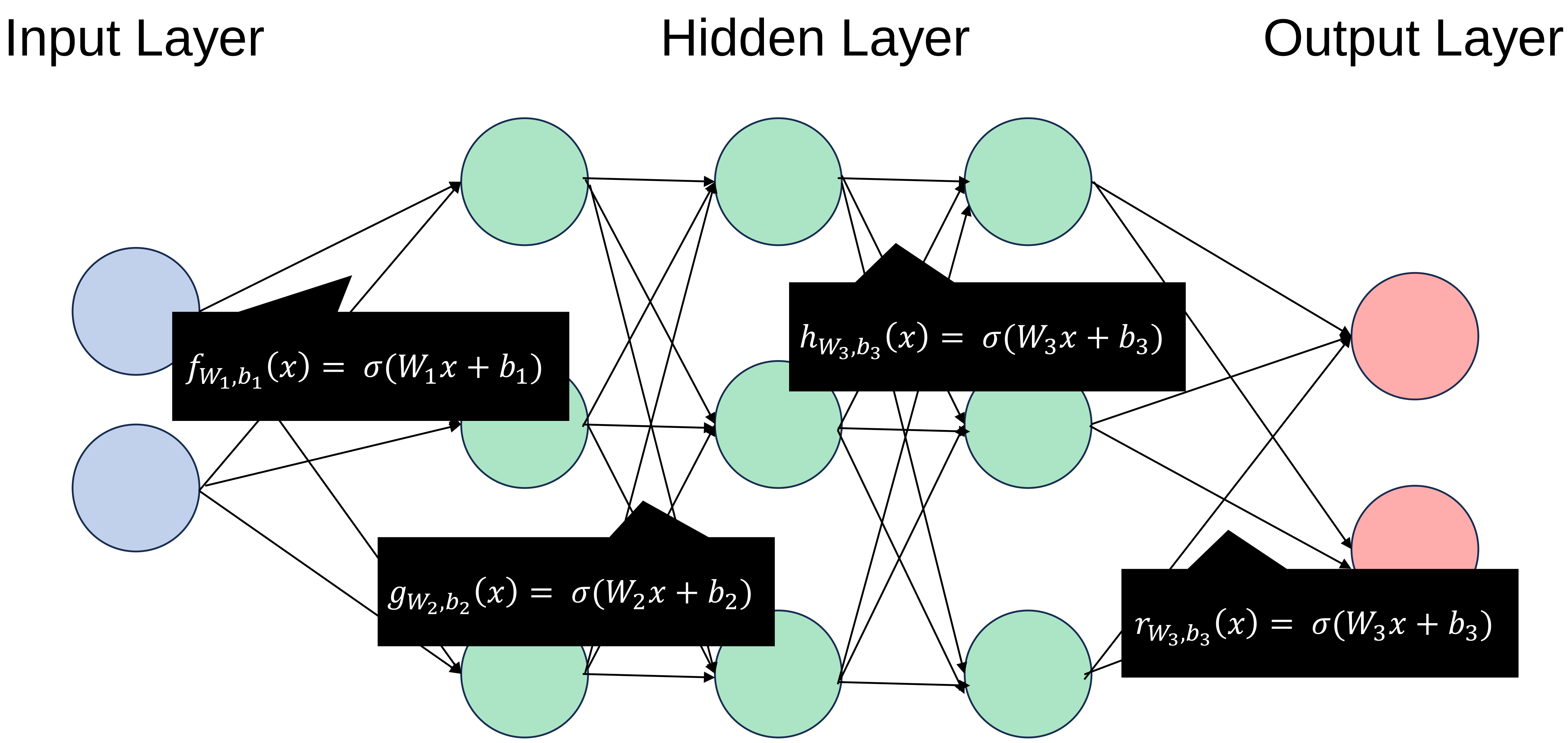
Hyelin Choi

Department of Mathematics
Sungkyunkwan University

February 15, 2024

Table of Contents

Deep Neural Network



Autoencoder

입력 데이터를 최대한 압축시킨 후, 이를 본래의 입력 형태로 복원시키는 신경망

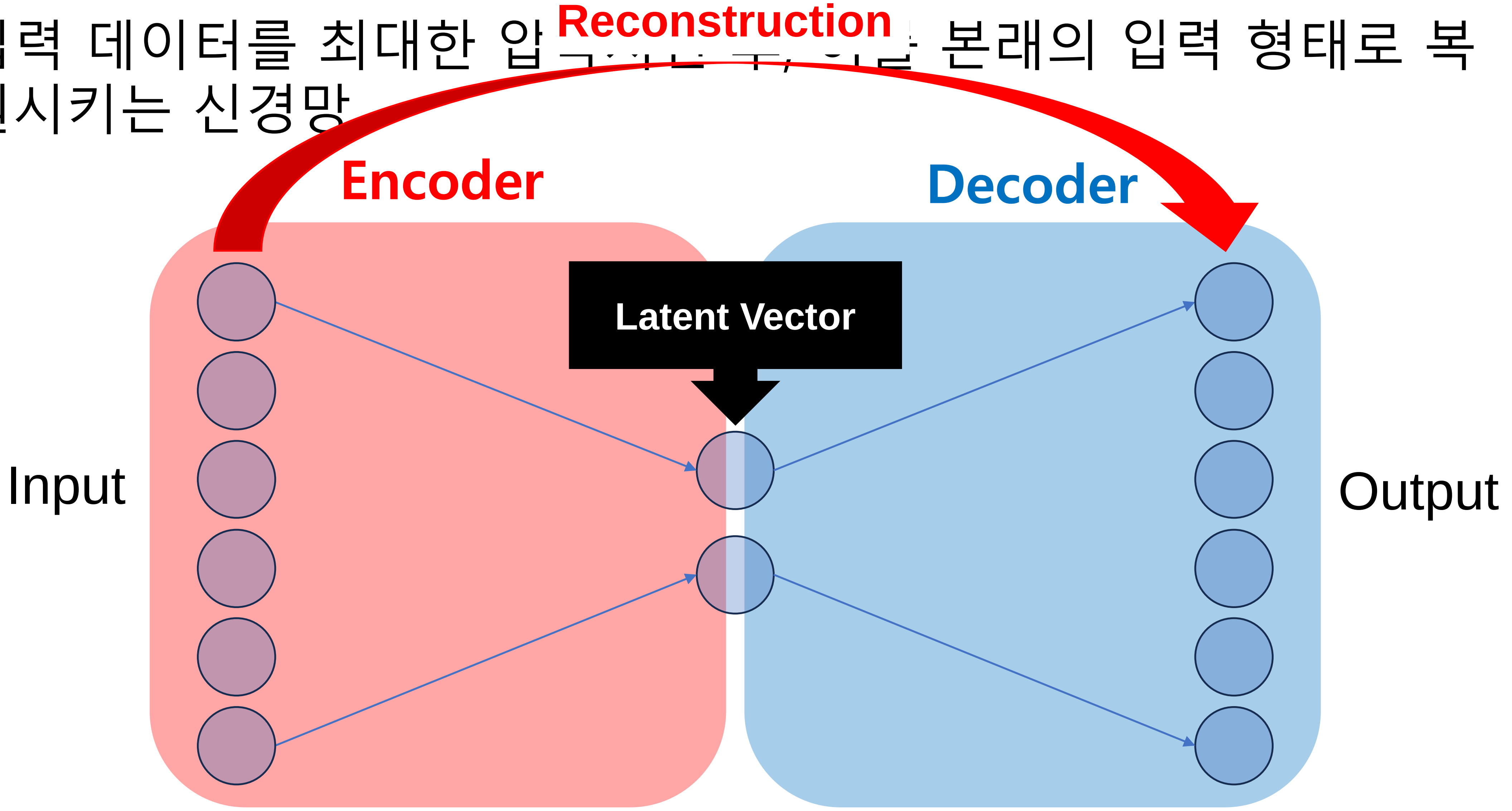


Table of Contents

I. Neighborhood autoencoder methods

  I. Deep Neural Graph Representations (DNGR)

  II. Structural Deep Network Embeddings (SDNE)

Neighborhood Autoencoder Methods

The autoencoder objective for DNGR and SDNE:

$$DEC\left(ENC(s_i)\right) = DEC(z_i) \approx s_i$$

Neighborhood Autoencoder Methods

Each node $\boldsymbol{v_i}$ is associated with a neighborhood vector, $\boldsymbol{s_i} \in \mathbb{R}^{|V|}$, which corresponds to v_i's row in the matrix S.

$$S_{ij} = s_G(v_i, v_j).$$

Random surfing
을 이용해서 정의

$$S = \begin{bmatrix} s_G(v_1, v_1) & s_G(v_1, v_2) & \cdots & s_G(v_1, v_n) \\ s_G(v_2, v_1) & s_G(v_2, v_2) & \cdots & s_G(v_2, v_n) \\ \vdots & \vdots & \vdots & \vdots \\ s_G(v_i, v_1) & s_G(v_i, v_2) & \cdots & s_G(v_i, v_n) \\ \vdots & \vdots & \ddots & \vdots \\ s_G(v_n, v_1) & s_G(v_n, v_2) & \cdots & s_G(v_n, v_n) \end{bmatrix}$$

$\boldsymbol{s_i}$

Original network에서 i번째 노드와 다른 노드간의 유사도

Deep Neural Graph Representations (DNGR)

1. We introduce random surfing model to capture graph structural information and generate a probabilistic co-occurrence matrix.
2. We calculate the PPMI matrix.
3. We use a stacked denoising autoencoder.

Random Surfing

- **Random Surfing**

그래프의 노드를 무작위로 탐색하면서 이동하는 것을 의미한다.

현재 노드에서 이웃노드로 이동이 일어날 수 있으며, 각각의 이동은 특정 확률로 결정된다.

State Transition Matrix

Random Surfing

- **State Transition Matrix**

The state transition probability is defined by

$$P_{ss'} = \Pr(s_{t+1} = s' | s_t = s)\,.$$

We assume there is a transition matrix A that captures the transition probability between different nodes.

$$A = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{bmatrix}$$

node 2 에서 node n 으로 이동할 확률

Random Surfing

1. 그래프의 노드들을 무작위로 정렬한다.
2. 현재 노드를 i번째 노드라고 가정한다.
3. row vector $\boldsymbol{p}_{\boldsymbol{k}}^{\boldsymbol{i}}$를 다음과 같이 정의한다 :

$$\boldsymbol{p}_{\boldsymbol{k}}^{\boldsymbol{i}} = [\, \tilde{p}_k^{i,1} \;\; \tilde{p}_k^{i,2} \;\; \cdots \;\; \tilde{p}_k^{i,j} \;\; \cdots \;\; \tilde{p}_k^{i,n} \,]$$

Node i 에서 k번 이동했을때
Node 2에 도착할 확률

Random Surfing

4. k번 이동한 후에 각각의 노드에 도착할 확률을 다음과 같이 정의할 수 있다:

$$p_k^i = p_{k-1}^i A = p_0^i A^k.$$

- p_0^i : i번째 성분이 1이고 나머지는 0인 one-hot vector.

$$p_k^i = p_{k-1}^i A = p_0^i A^k$$

ex) $p_2^i = p_1^i A$

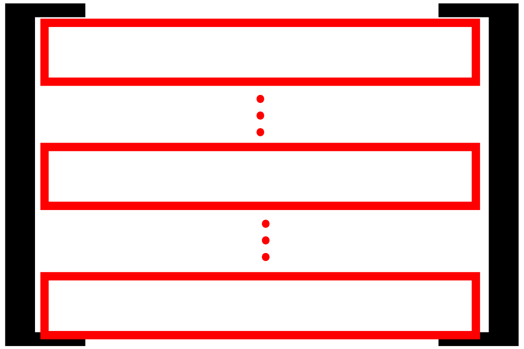
Random Surfing

5. 모든 노드에 대하여 p_k^i 를 구하고 이를 위에서 아래로 쌓아서 PCO matrix 를 만든다.

v_1 에서 k번 이동한 후, 각각의 노드에 도착할 확률 : $\boldsymbol{p_k^1}$

v_i 에서 k번 이동한 후, 각각의 노드에 도착할 확률 : $\boldsymbol{p_k^i}$

v_n 에서 k번 이동한 후, 각각의 노드에 도착할 확률 : $\boldsymbol{p_k^n}$



$\boldsymbol{PCO[i,j]}$

: v_i에서 k번 이동한 후 v_j 에 도착할 확률

Deep Neural Graph Representations (DNGR)

1. We introduce random surfing model to capture graph structural information and generate a probabilistic co-occurrence matrix.
2. We calculate the PPMI matrix.
3. We use a stacked denoising autoencoder.

PMI matrix

- Pointwise Mutual Information matrix (PMI matrix)

The PMI of $x \in X$ and $y \in Y$ quantifies the discrepancy between the probability of their co-occurrence given their joint distribution and their individual distributions, assuming independence.

We write <u>PMI matrix</u> as follows

x와 y가 동시에 발생할 확률

$$PMI(x;y) = \log\frac{p(x,y)}{p(x)p(y)}$$

생할 확률

y가 발생할 확률

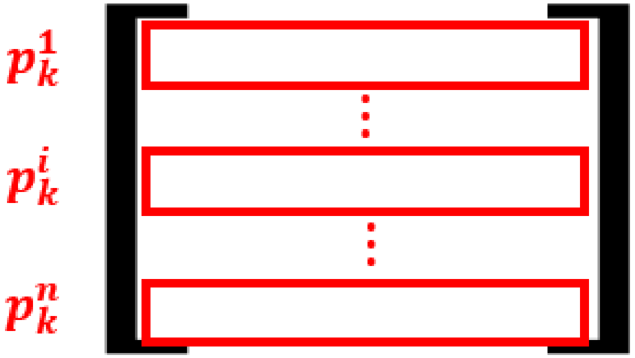
- **PMI 값이 크다 = 두 노드의 유사도가 크다**
- **PMI 값이 작다 = 두 노드의 유사도가 작다**

PMI matrix

- Pointwise Mutual Information matrix (PMI matrix)

We write PMI matrix as follows

$$PMI\left(v_i; v_j\right) = \log \frac{p(v_i, v_j)}{p(v_i)p(v_j)}.$$



$\boldsymbol{PCO[i,j]}$

: v_i에서 k번 이동한 후 v_j 에 도착할 확률

PPMI matrix

- **Positive Pointwise Mutual Information matrix (PPMI matrix)**

We write PMI matrix as follows

$$\mathrm{PMI}(x;y) = \log \frac{p(x,y)}{p(x)p(y)}.$$

0

We assign each negative value to 0 to form the PPMI matrix

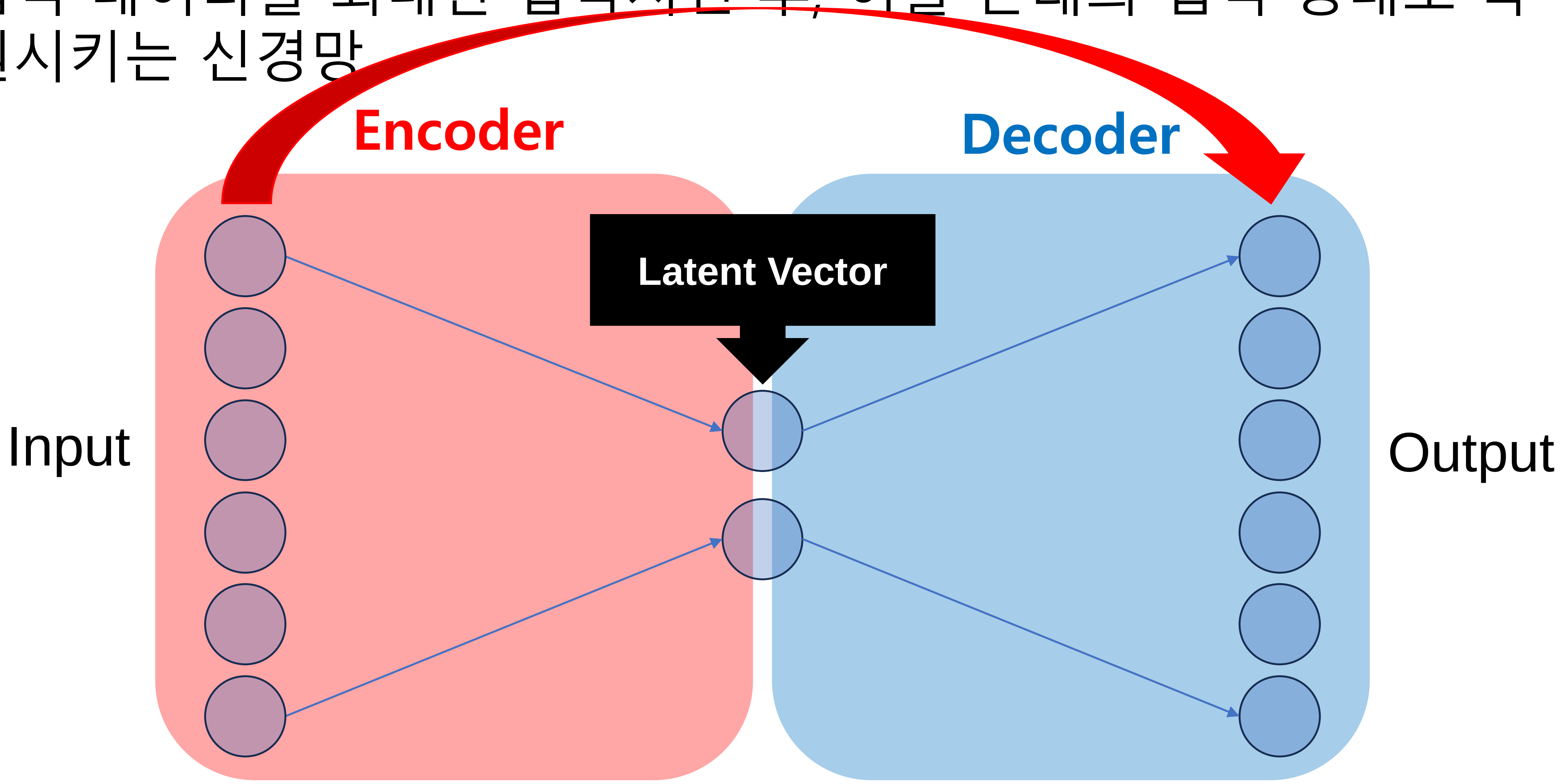
$$PPMI(x;y) = \max(0, PMI(x;y)).$$

Deep Neural Graph Representations (DNGR)

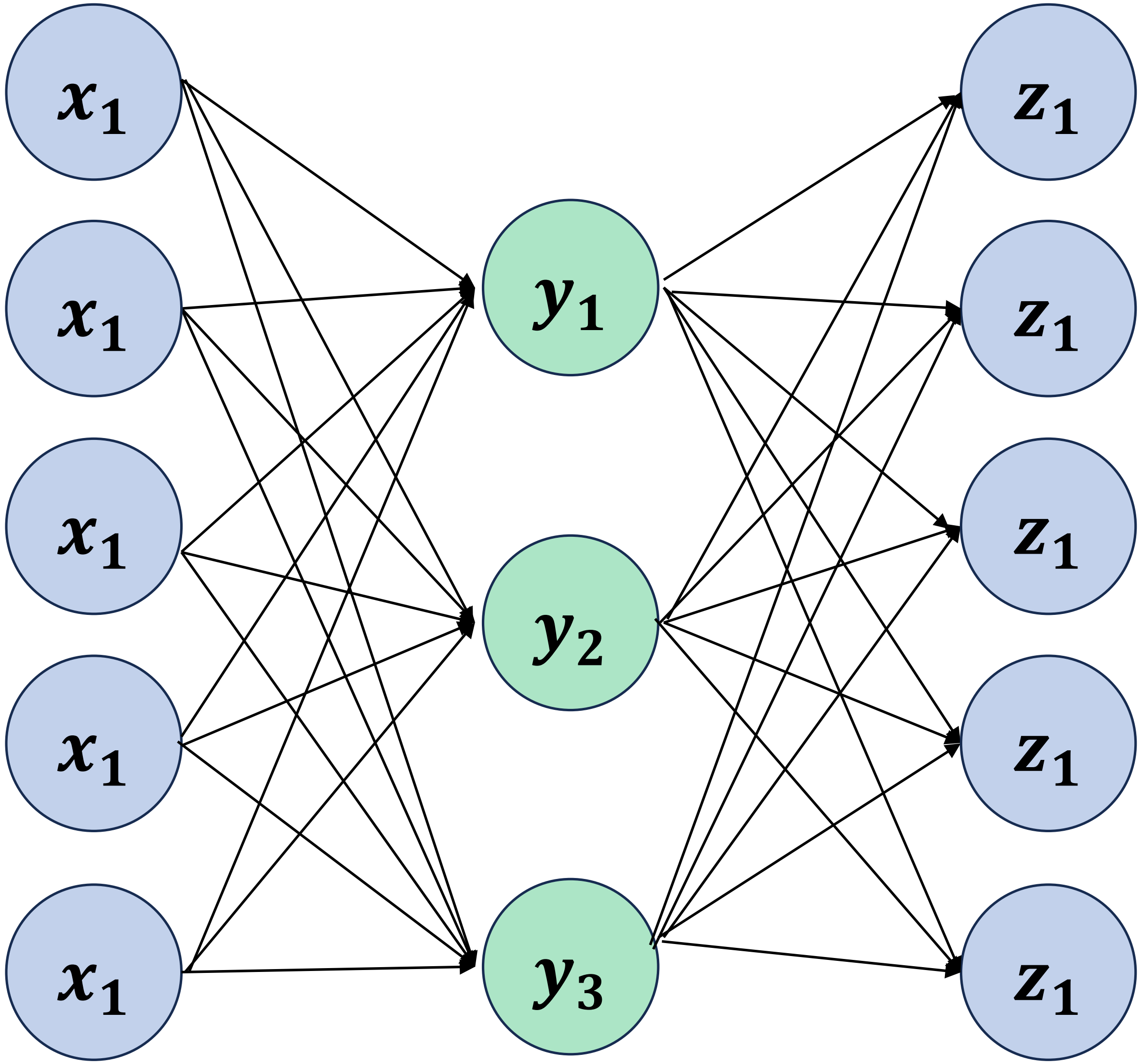
1. We introduce random surfing model.
2. We calculate the PPMI matrix.
3. We use a stacked denoising autoencoder.

Autoencoder

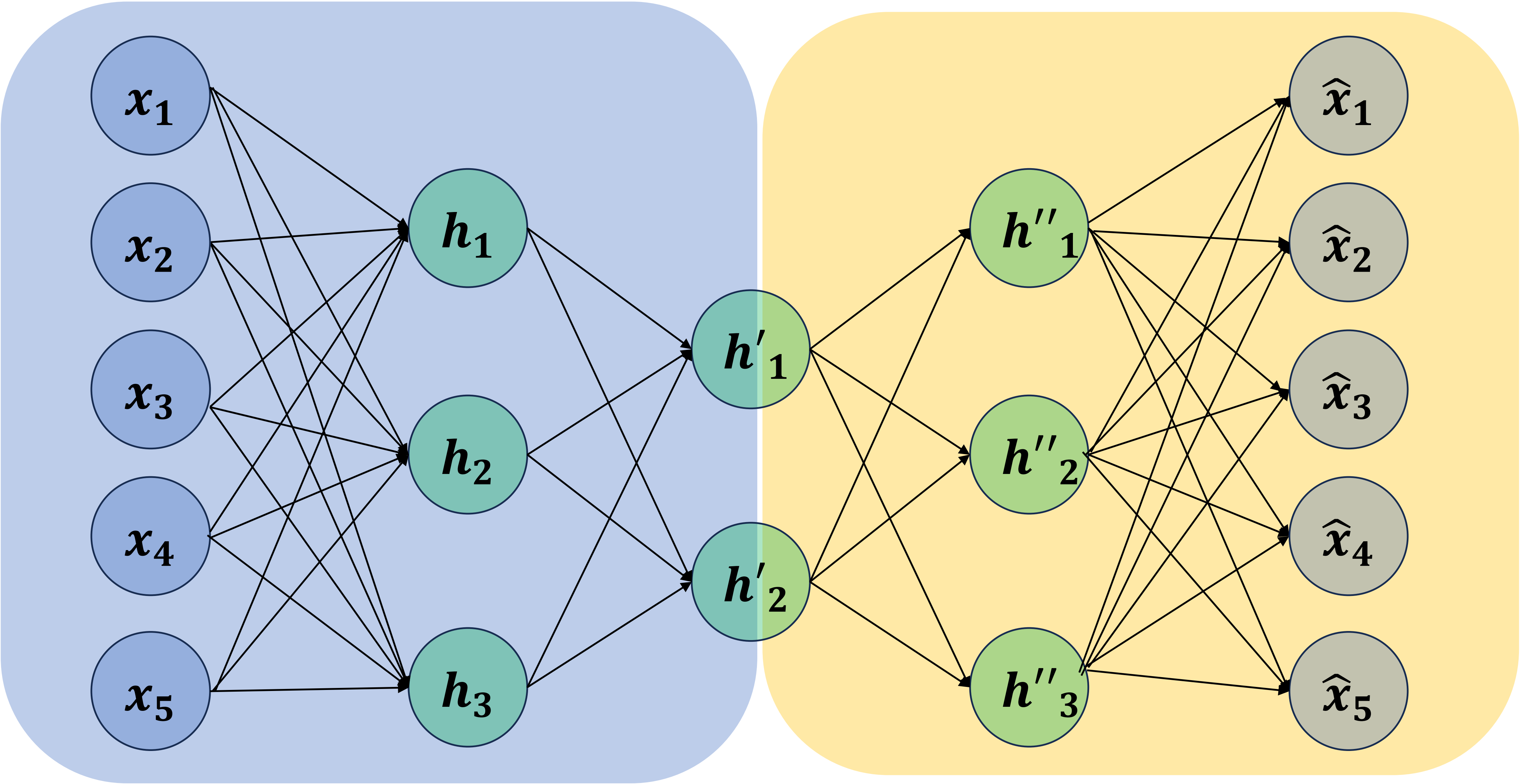
입력 데이터를 최대한 압축시킨 후, 이를 본래의 입력 형태로 복원시키는 신경망



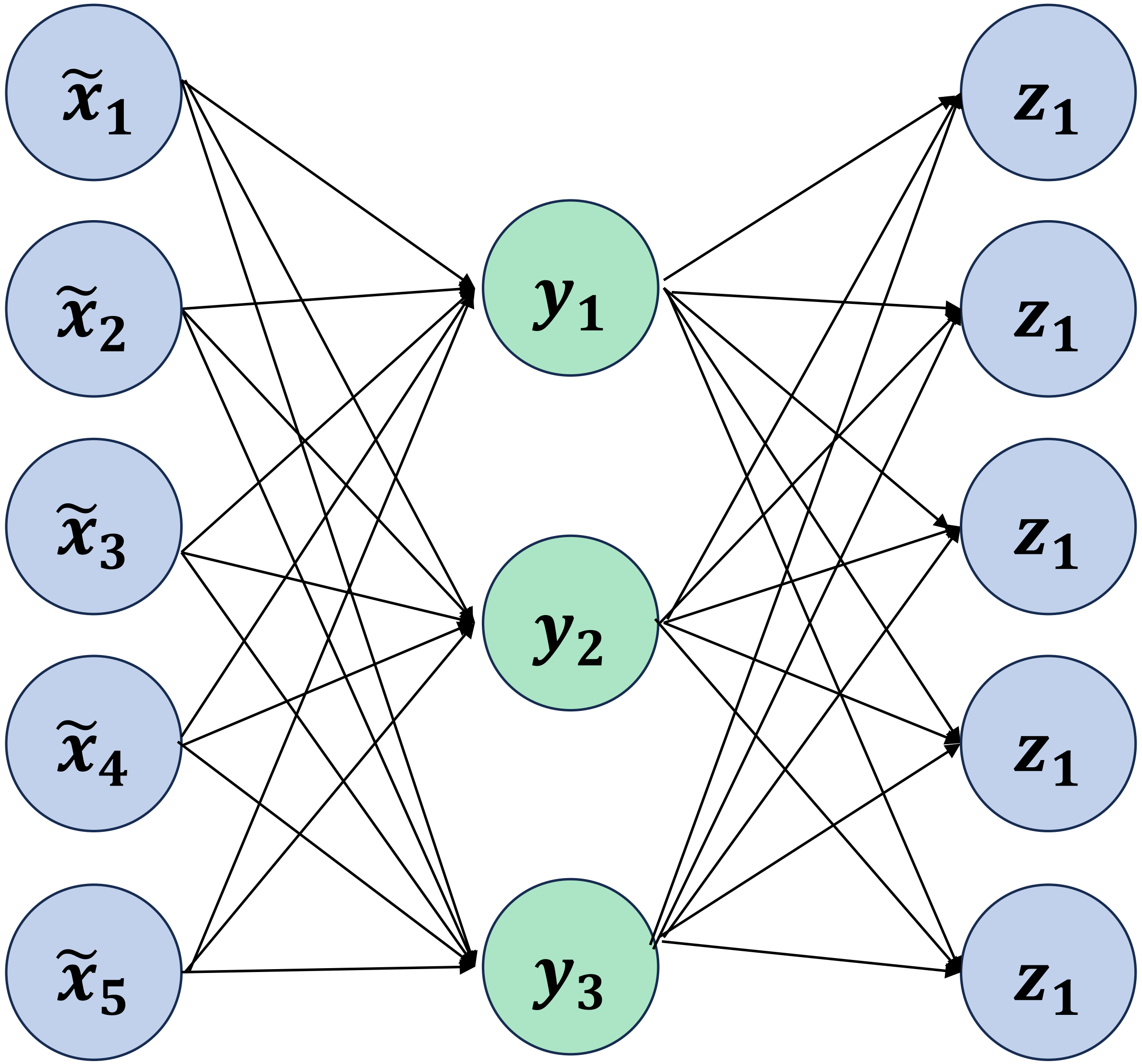
Single Layer Autoencoder



Stacked Autoencoder



Denoising Autoencoder



$$\tilde{x} = x + \boldsymbol{noise}$$

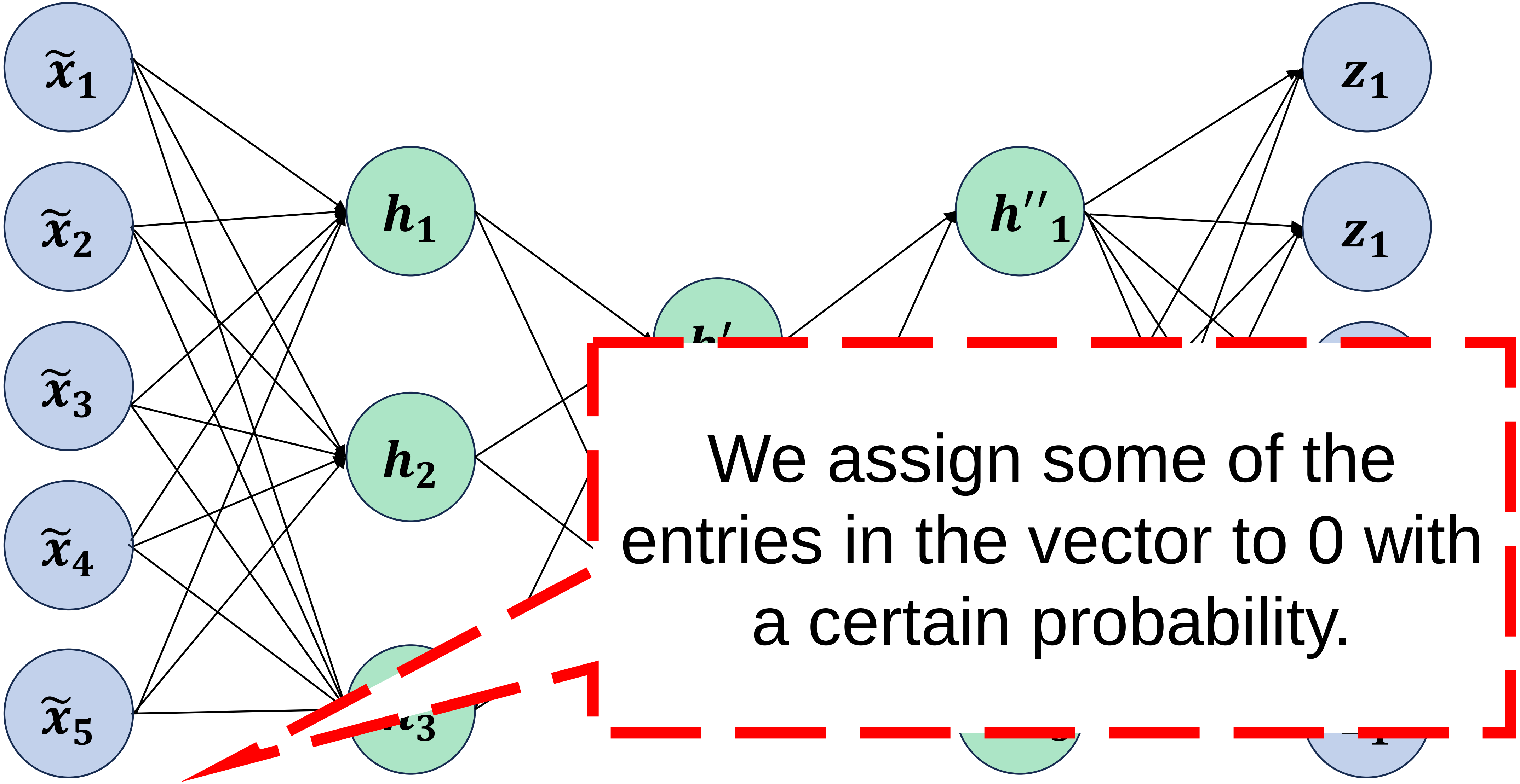
Denoising Autoencoder

- **Reason for $\tilde{x} = x + \boldsymbol{noise}$**

High dimensional input data often contains redundant information and noise.

It is believed that the denoising strategy can effectively reduce noise and enhance robustness.

Stacked Denoising Autoencoder



We assign some of the entries in the vector to 0 with a certain probability.

$\tilde{x} = x + noise$

Stacked Denoising Autoencoder

We are interested in:

$$\min_{\theta_1,\cdots,\theta_N} \sum_{i=1}^{n} L(x^{(i)}, f_{\theta_N} \circ \cdots \circ f_{\theta_1}(\tilde{x}^{(i)}))$$

where $\tilde{x}^{(i)}$ is corrupted input data of $x^{(i)}$, N is the number of layer and n is the number of input data.

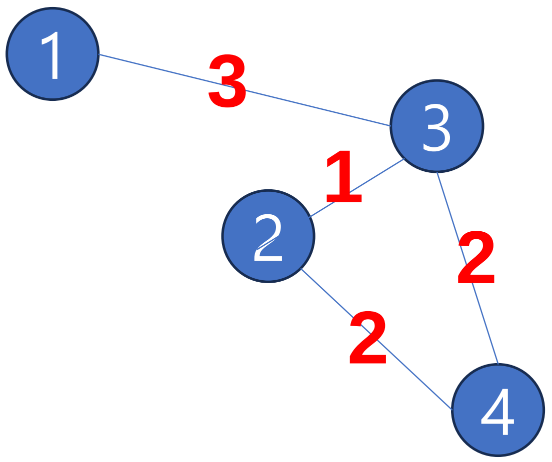
Table of Contents

Graph

Definition 1. (Graph) A graph is denoted as $G = (V, E)$, where $V = \{v_1, \cdots, v_n\}$ represents n nodes and $E = \{e_{i,j}\}_{i,j=1}^{n}$ represents the edges. Each edge $e_{i,j}$ is associated with a weight $s_{i,j} \geq 0$. For v_i and v_j not linked by an edge, $s_{i,j} = 0$. Otherwise, for unweighted graph $s_{i,j} = 1$ and for weighted graph, $s_{i,j} > 0$

Graph

Each edge $e_{i,j}$ is associated with a weight $s_{i,j} \geq 0$. For v_i and v_j not linked by an edge, $s_{i,j} = 0$. Otherwise, for unweighted graph $s_{i,j} = 1$ and for weighted graph, $s_{i,j} > 0$.



Adjacency Matrix of **Unweighted** Graph

$$S = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix of **Weighted** Graph

$$S = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 2 & 2 \\ 3 & 2 & 0 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix}$$

First-Order Proximity

Definition 2. (First-Order Proximity) The first-order proximity describes the pairwise proximity between nodes. For any pair of nodes, if $s_{i,j} > 0$, there exists positive first-order proximity between v_i and v_j. Otherwise, the first-order proximity between v_i and v_j is 0.

The first-order proximity captures the local network structure.

Second-Order Proximity

Definition 3. (Second-Order Proximity) The second-order proximity between a pair of nodes describes the proximity of their pair's neighborhood structure. Let $N_u = \{s_{u,1}, \cdots, s_{u,|V|}\}$ denote the first-order proximity between v_u and other nodes. Then, second-order proximity is determined by the similarity of N_u and N_v.

The second-order proximity captures the global network structure.

Network Embedding

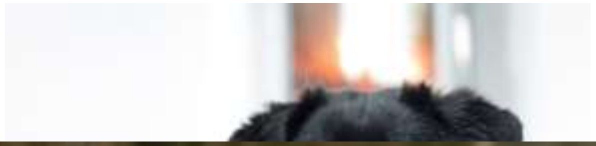
Definition 4. (Network Embedding) Given a graph denoted as $G = (V, E)$, network embedding aims to learn a mapping function $f : v_i \rightarrow \boldsymbol{y_i} \in \mathbb{R}^d$, where $d \ll |V|$. The objective of the function is to make the similarity between $\boldsymbol{y_i}$ and $\boldsymbol{y_j}$ explicitly preserve the first-order and second-order proximity of v_i and v_j.

Structural Deep Network Embedding (SDNE)

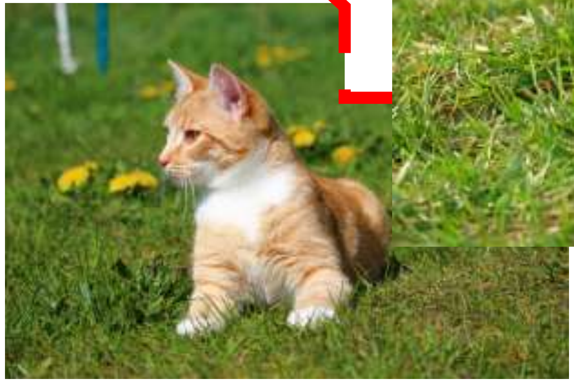
We propose a semi-supervised deep model, which simultaneously optimizes the first-order and second-order proximity.

Supervised Learning

Dog



Dog



Cat

Cat

Cat

Unsupervised Learning

Unlabeled Data

Semi-Supervised Learning

Supervised Learning

Second-order proximity

Small number of labeled data ⇠ Supervised Learning

Large number of unlabeled data ⇠ Unsupervised Learning

First-order proximity

소량의 labeled data를 통한 약간의 가이드로 성능을 끌어올릴 수 있다.

Loss Function of Semi-Supervised Learning

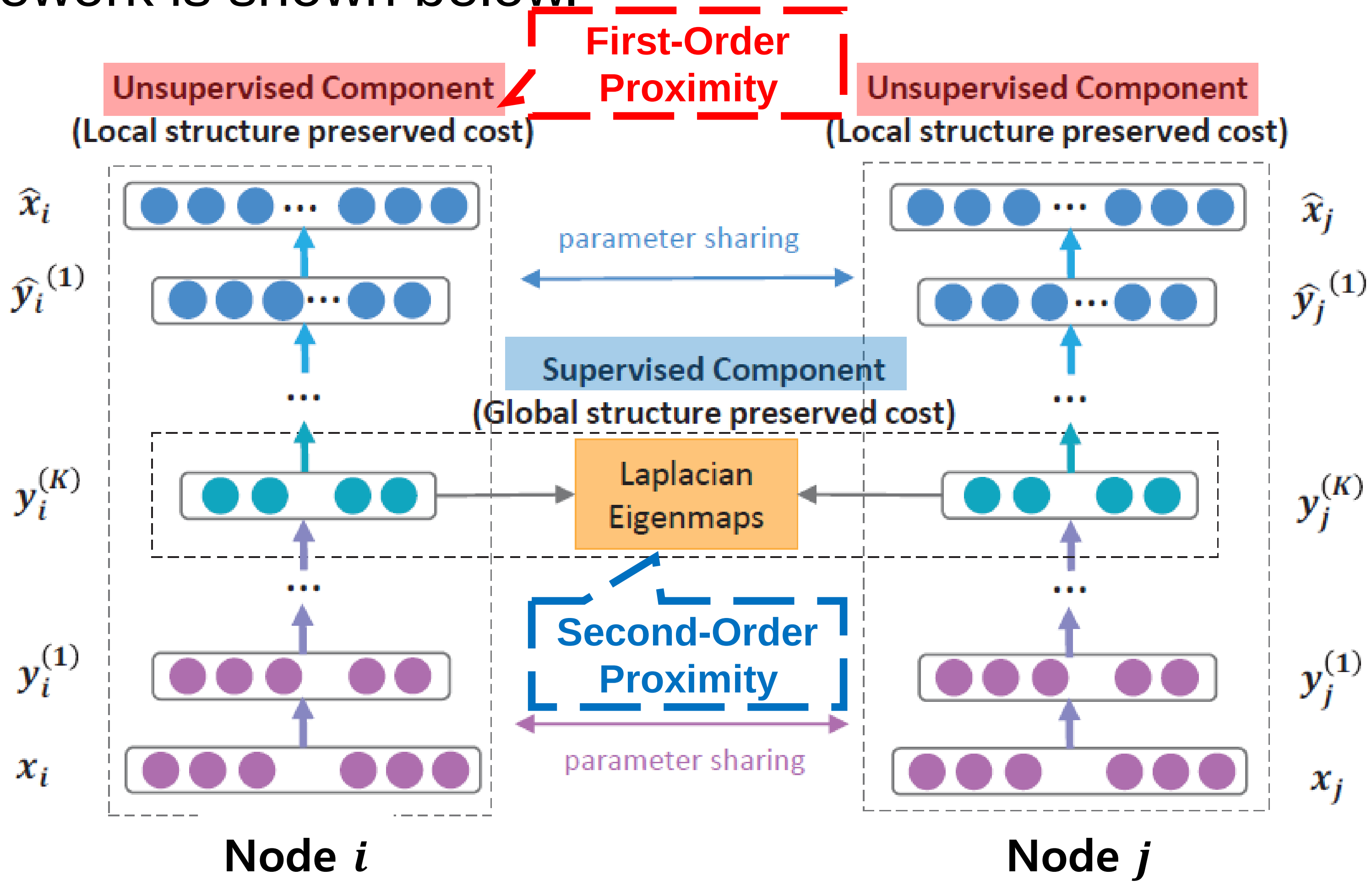
We write the loss function of semi-supervised learning as follows:

$$Loss = L_s + L_u,$$

where L_s is the loss function of supervised learning and L_u is the loss function of unsupervised learning.

Framework

We propose a semi-supervised deep model to perform network embedding, whose framework is shown below.



Framework

We design the supervised component to exploit the first-order proximity.

And we design the unsupervised component to exploit second-order proximity.

Loss Function

To preserve the first-order and second-order proximity simultaneously, we minimize the following loss function:

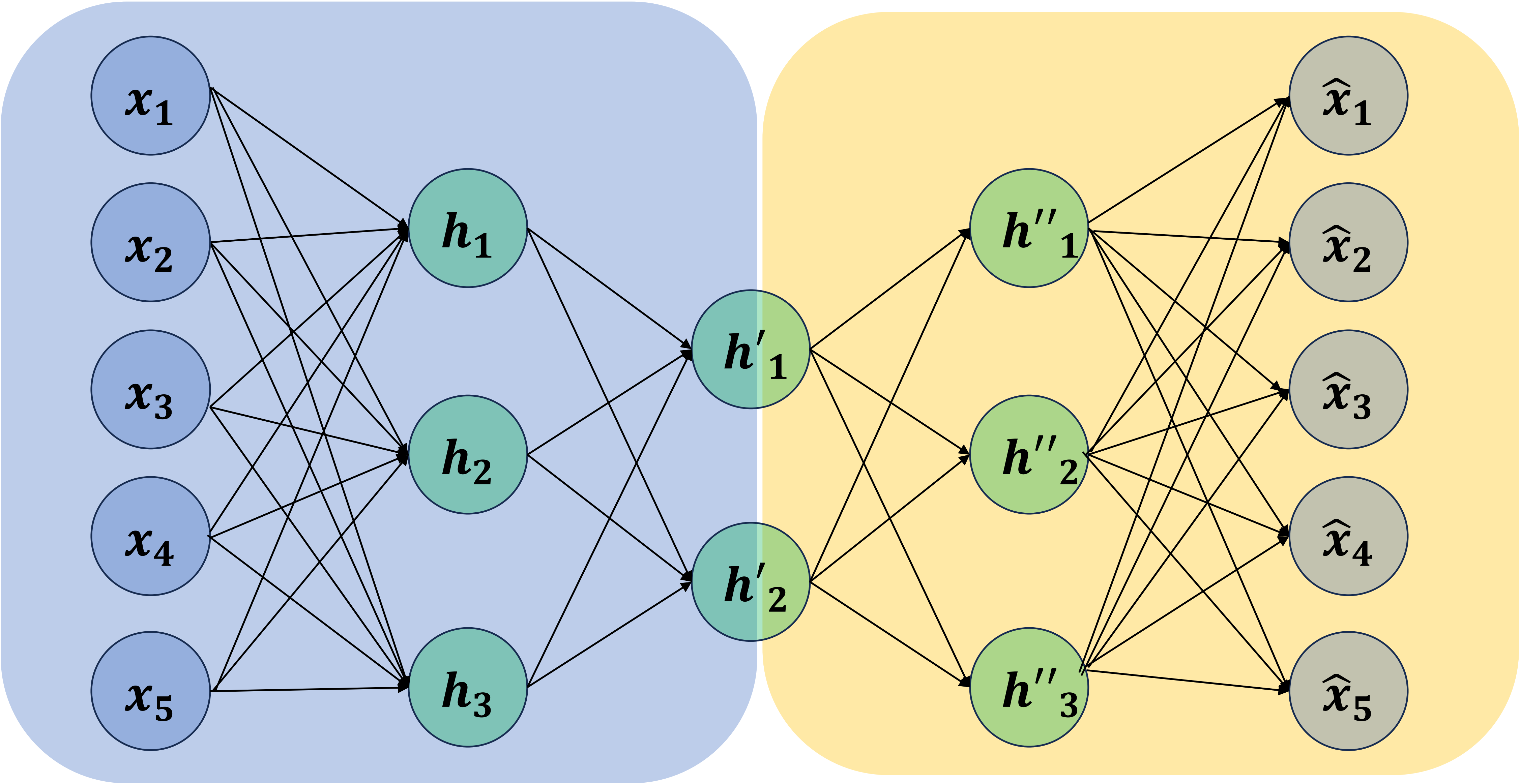
$$\mathcal{L}_{mix} = \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \nu\mathcal{L}_{reg}$$

$$= \|(\hat{X} - X) \odot B\|_F^2 + \alpha \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 + \nu\mathcal{L}_{reg}$$

where L_{reg} is an $L2$-norm regularizer term to prevent overfitting, which is defined as follows:

$$L_{reg} = \frac{1}{2} \sum_{k=1}^{n} (||W^{(k)}||_F^2 + ||\widehat{W}^{(k)}||_F^2)$$

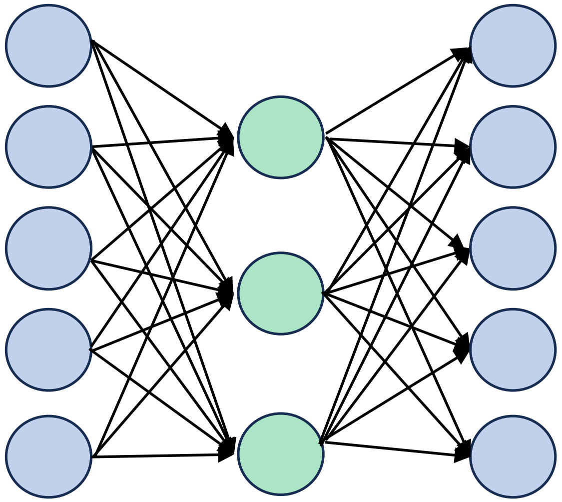
Deep Autoencoder



Notation

Table 1: Terms and Notations

Symbol	Definition
n	number of vertexes
K	number of layers
$S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$	the adjacency matrix for the network
$X = \{\mathbf{x}_i\}_{i=1}^n, \hat{X} = \{\hat{\mathbf{x}}_i\}_{i=1}^n$	the input data and reconstructed data
$Y^{(k)} = \{\mathbf{y}_i^{(k)}\}_{i=1}^n$	the k-th layer hidden representations
$W^{(k)}, \hat{W}^{(k)}$	the k-th layer weight matrix
$\mathbf{b}^{(\mathbf{k})}, \hat{\mathbf{b}}^{(\mathbf{k})}$	the k-th layer biases
$\theta = \{W^{(k)}, \hat{W}^{(k)}, \mathbf{b}^{(k)}, \hat{\mathbf{b}}^{(k)}\}$	the overall parameters



Loss Function (First-Order Proximity)

The loss function is defined as follows:

$$\mathcal{L}_{1st} = \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i^{(K)} - \mathbf{y}_j^{(K)}\|_2^2 = \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2$$

The loss function above borrows the idea of Laplacian Eigenmaps, which incurs a penalty when similar nodes are mapped far away in the embedding space.

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 $1 \leq i \leq n$.

Loss Function (Second-Order Proximity)

Reconstructed data

Input data

$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2$$

Neighborhood structure

$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|\hat{\mathbf{s}}_i - \mathbf{s}_i\|_2^2$$

Since each s_i characterizes the neighborhood structure of the node v_i

★ $$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|(\hat{\mathbf{s}}_i - \mathbf{s}_i) \odot \mathbf{b}_i\|_2^2$$

Penalty

We impose more penalty to the reconstruction error of the non-zero elements than that of zero elements.

Loss Function (Second-Order Proximity)

We impose more penalty to the reconstruction error of the non-zero elements than that of zero elements. The revised loss function is shown as follows:

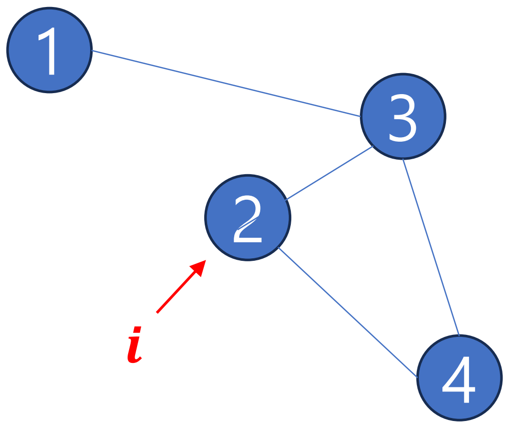
$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|(\hat{\mathbf{s}}_i - \mathbf{s}_i) \odot \mathbf{b}_i\|_2^2$$

where $\odot$ means the Hadamard product, $b_i = \{b_{i,j}\}_{j=1}^{n}$. If $s_{i,j} = 0$, $b_{i,j} = 1$, else $b_{i,j} = \beta > 1$.

$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|(\hat{\mathbf{s}}_i - \mathbf{s}_i) \odot \mathbf{b}_i\|_2^2$$

If $s_{i,j} = 0$, $b_{i,j} = 1$

If $s_{i,j} > 0$, $b_{i,j} > 1$



Loss Function

To preserve the first-order and second-order proximity simultaneously, we minimize the following loss function:

$$\begin{aligned}\mathcal{L}_{mix} &= \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \nu\mathcal{L}_{reg} \\ &= \|(\hat{X} - X) \odot B\|_F^2 + \alpha \sum_{i,j=1}^{n} s_{i,j}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2 + \nu\mathcal{L}_{reg}\end{aligned}$$

where L_{reg} is an $L2$-norm regularization term to prevent overfitting, which is defined as follows:

$$L_{reg} = \frac{1}{2}\sum_{k=1}^{n}(||W^{(k)}||_F^2 + ||\widehat{W}^{(k)}||_F^2)$$

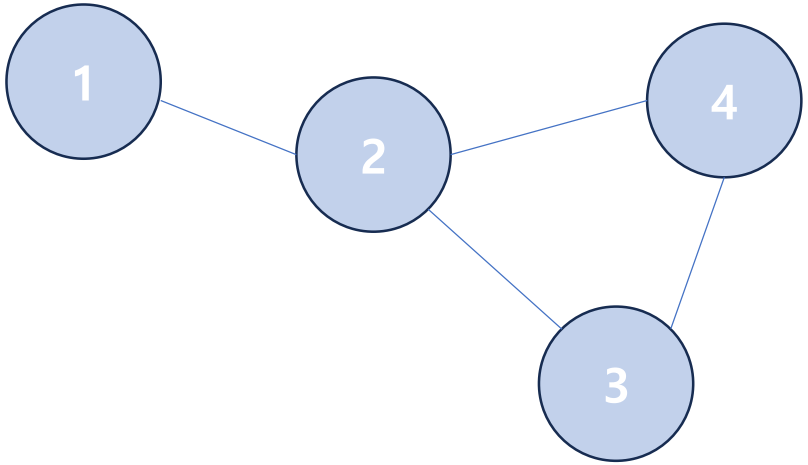
Thank you for listening.

- 이전 연구 -> PPMI matrix & SVD 이용해서 matrix factorizaiton

Co-occurrence Matrix

- **Co-occurrence matrix**

ex)



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

Window size : 1

Stacked Autoencoder

심층 신경망(deep neural network)의 한 유형인데, 입력 데이터를 저차원의 표현으로 압축한 다음 다시 원래의 차원으로 재구성하는 방법을 사용합니다. 이 과정은 입력 데이터에 대한 잠재적인 특징을 추출하는 데 도움이 됩니다.

PMI matrix

- PMI 값이 크다 = 두 노드의 유사도가 크다
- PMI 값이 작다 = 두 노드의 유사도가 작다

p(x,y) 값이 크다 = x와 y가 자주 동시에 발생한다
= 두 노드의 유사도가 크다 = PMI값이 크다

matrix (PMI matrix)

antifies the discrepancy between
rence given their joint distribution
s, assuming independence.

X와 y가 동시에 발생할 확률

$$pmi(x;y) = \log \frac{p(x,y)}{p(x)p(y)}$$

X가 발생할 확률

Y가 발생할 확률

PMI matrix

- Pointwise Mutual Information matrix (PMI matrix)

We write PMI matrix as follows

$$PMI(x;y) = \log \frac{p(x,y)}{p(x)p(y)}.$$

We use co-occurrence matrix to rewrite PMI matrix

$$\mathrm{PMI}(x;y) = \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{\frac{C(x,y)}{N}}{\frac{C(x)}{N}\frac{C(y)}{N}}$$

where $C(x,y)$ is the number of co-occurrence of node x and node y, $C(x)$ is the occurrence of node x and N is the number of nodes.

PPMI matrix

- Positive Pointwise Mutual Information matrix (PPMI matrix)

We write PMI matrix as follows

$$\mathrm{PMI}(x;y) = \log\frac{p(x,y)}{p(x)p(y)}.$$

We assign each negative value to 0 to form the PPMI matrix

$$PPMI(x;y) = \max(0, PMI(x;y)).$$

Singular Value Decomposition (SVD)

We perform dimension reduction using SVD.

We assume that the PPMI matrix X can be decomposed into three matrices $X = U\Sigma V^T$ where U and V are orthonormal matrices and Σ is a diagonal matrix.

In other words,

$$X \approx X_d = U_d \Sigma_d V_d^T$$

Here U_d and V_d are the left d columns of U and V corresponding to the top-d singular values (in Σ_d). Then the word representation matrix R can be:

$$R = U_d (\Sigma_d)^{1/2} \text{ or } R = U_d.$$

The PPMI matrix X is the product of the word representation matrix and the context matrix. The SVD procedure provides us a way of finding the matrix R from the matrix X.