

그래프를 이용한 기계 학습

#7 그래프의 정점을 어떻게 벡터로 표현할까?

신기정

(KAIST AI대학원)

1. 정점 표현 학습
2. 인접성 기반 접근법
3. 거리/경로/중첩 기반 접근법
4. 임의보행 기반 접근법
5. 변환식 정점 표현 학습의 한계
6. 실습: Node2Vec을 사용한 군집 분석과 정점 분류

1. 정점 표현 학습

1.1 정점 표현 학습이란?

1.2 정점 표현 학습의 이유

1.3 정점 표현 학습의 목표

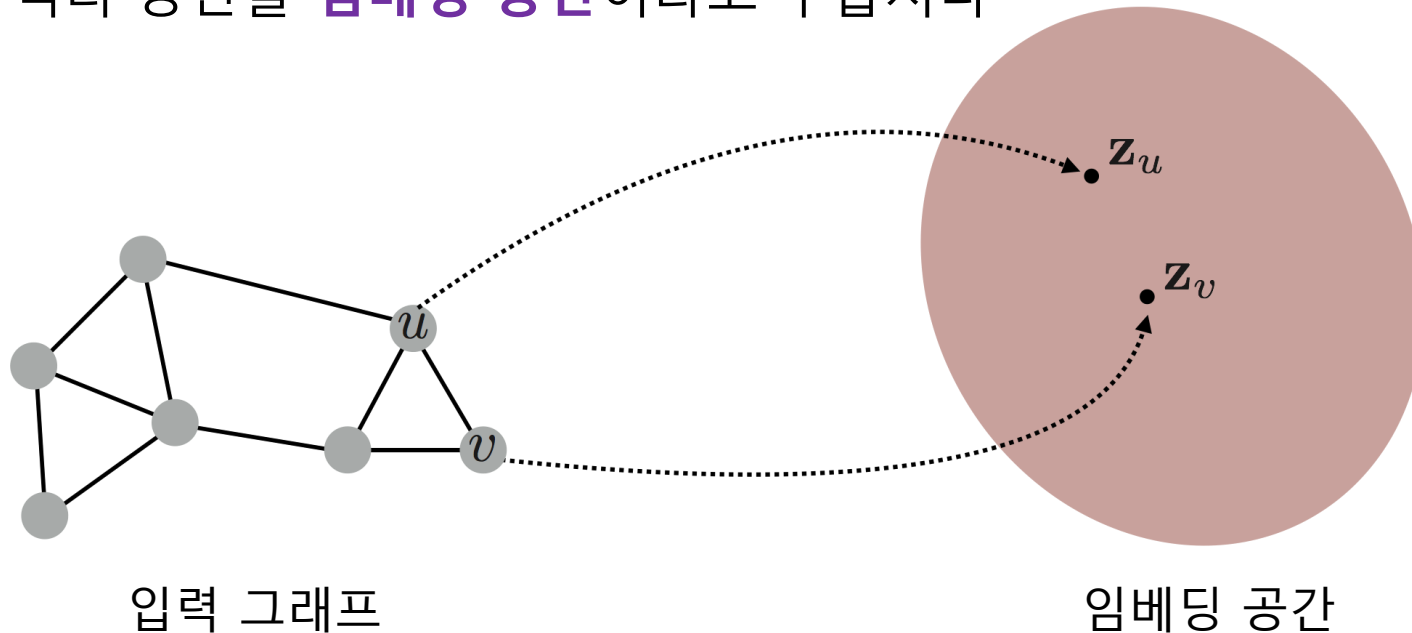
1.1 정점 표현 학습이란?

정점 표현 학습이란 **그래프의 정점들을 벡터의 형태로 표현하는 것**입니다

정점 표현 학습은 간단히 **정점 임베딩(Node Embedding)**이라고도 부릅니다

정점 임베딩은 벡터 형태의 표현 그 자체를 의미하기도 합니다

정점이 표현되는 벡터 공간을 **임베딩 공간**이라고 부릅니다

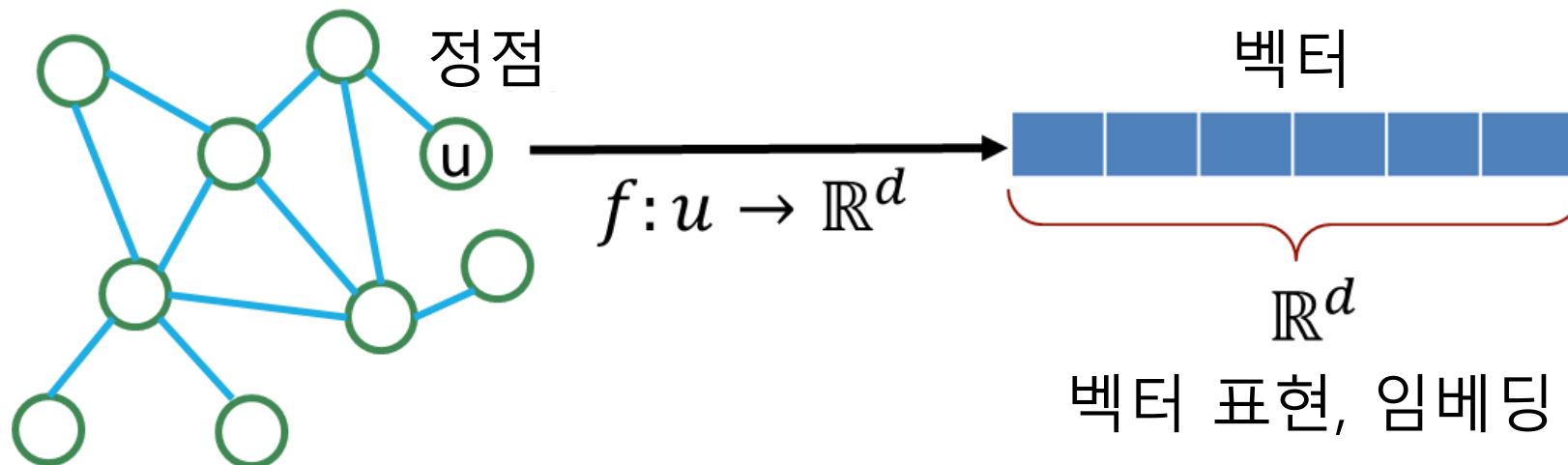


1.1 정점 표현 학습이란?

정점 표현 학습이란 **그래프의 정점들을 벡터의 형태로 표현하는 것**입니다

정점 표현 학습의 입력은 그래프입니다

주어진 그래프의 **각 정점 u 에 대한 임베딩, 즉 벡터 표현 z_u** 가 정점 임베딩의 **출력**입니다



1.2 정점 표현 학습의 이유

정점 임베딩의 결과로, 벡터 형태의 데이터를 위한 도구들을 그래프에도 적용할 수 있습니다

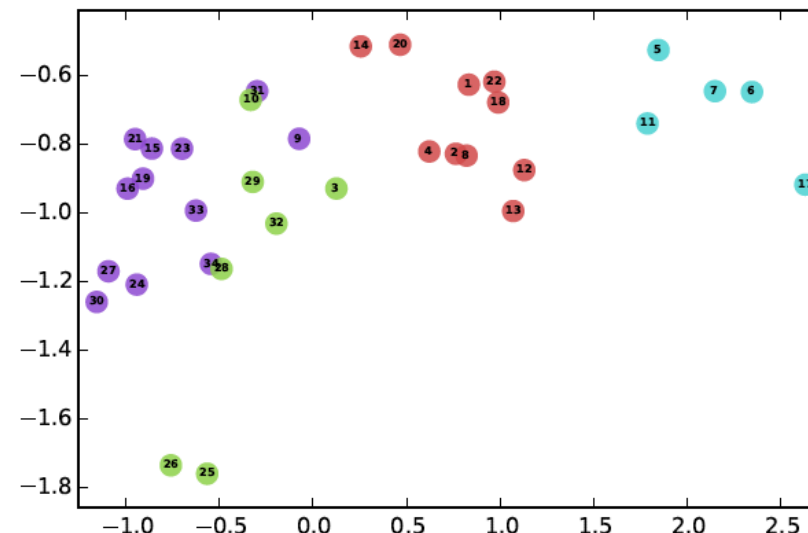
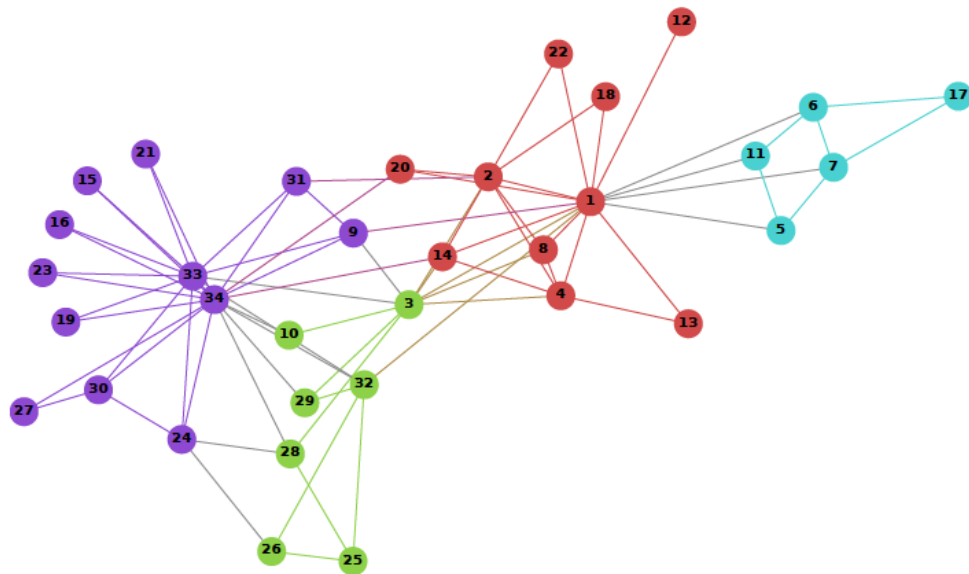
기계학습 도구들이 한가지 예시입니다
대부분 **분류기**(로지스틱 회귀분석, 다층 퍼셉트론 등) 그리고
군집 분석 알고리즘(K-Means, DBSCAN 등)은
벡터 형태로 표현된 사례(Instance)들을 입력으로 받습니다

그래프의 정점들을 벡터 형태로 표현할 수 있다면,
위의 예시와 같은 대표적인 도구들 뿐 아니라, 최신의 기계 학습도구들을
정점 분류(Node Classification), 군집 분석(Community Detection) 등에
활용할 수 있습니다

1.3 정점 표현 학습의 목표

어떤 기준으로 정점을 벡터로 변환해야 할까요?

그래프에서의 정점간 유사도를 임베딩 공간에서도 “보존”하는 것을 목표로 합니다

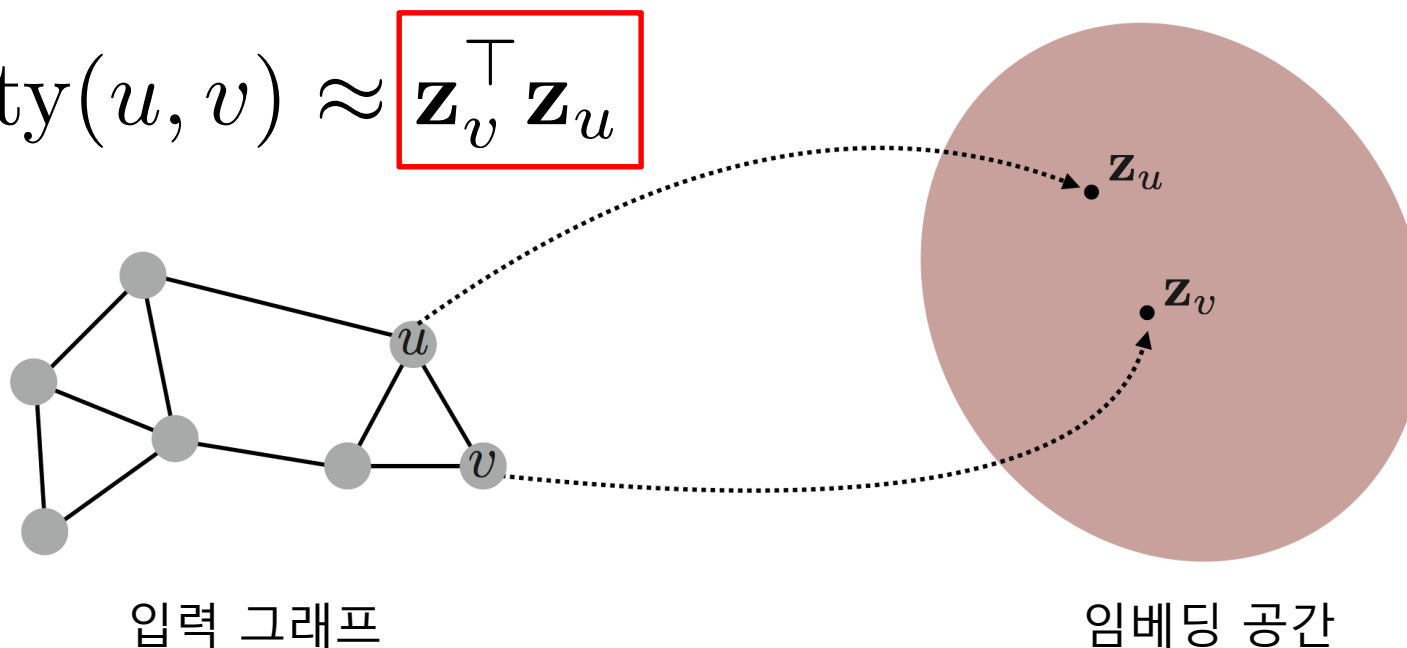


1.3 정점 표현 학습의 목표

임베딩 공간에서의 유사도로는 내적(Inner Product)를 사용합니다

임베딩 공간에서의 u 와 v 의 유사도는 둘의 임베딩의 내적 $\mathbf{z}_v^\top \mathbf{z}_u = \|\mathbf{z}_u\| \cdot \|\mathbf{z}_v\| \cdot \cos(\theta)$ 입니다
내적은 두 벡터가 클 수록, 그리고 같은 방향을 향할 수록 큰 값을 갖습니다

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

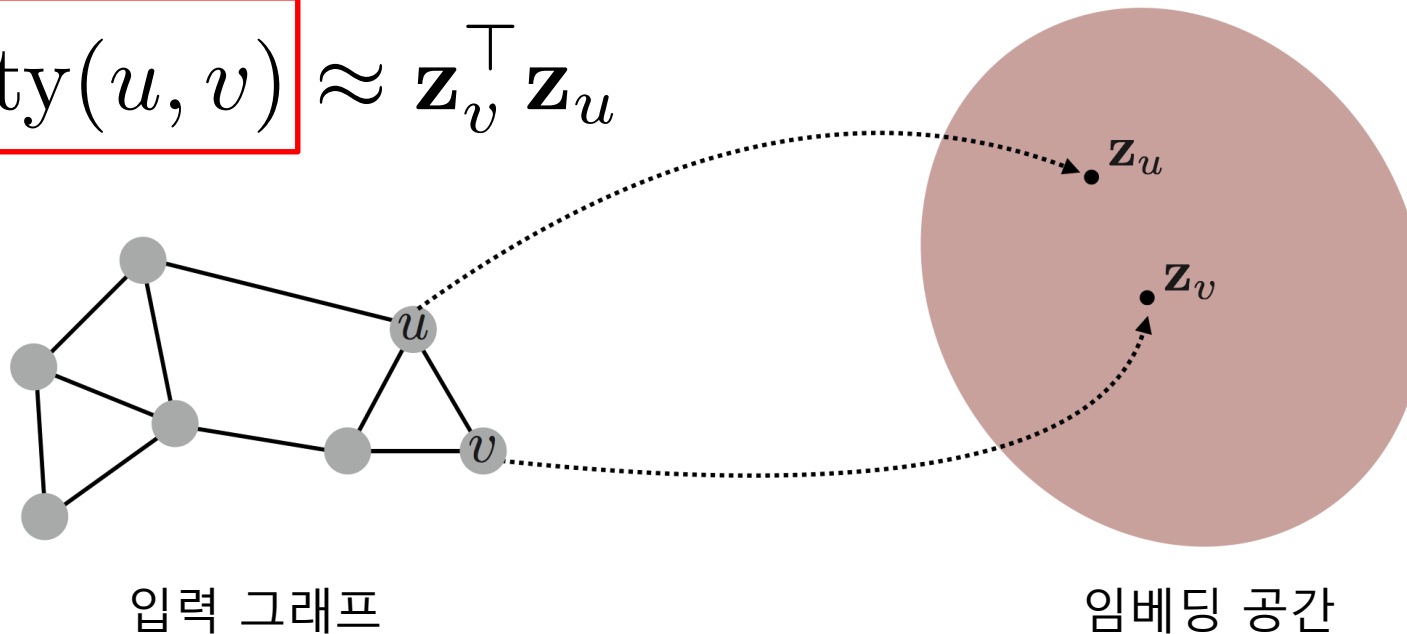


1.3 정점 표현 학습의 목표

그래프에서 두 정점의 유사도는 어떻게 정의할까요?

이 질문에는 여러가지 답이 있을 수 있습니다
본 강의에서는 대표적인 답을 몇 가지를 설명합니다

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

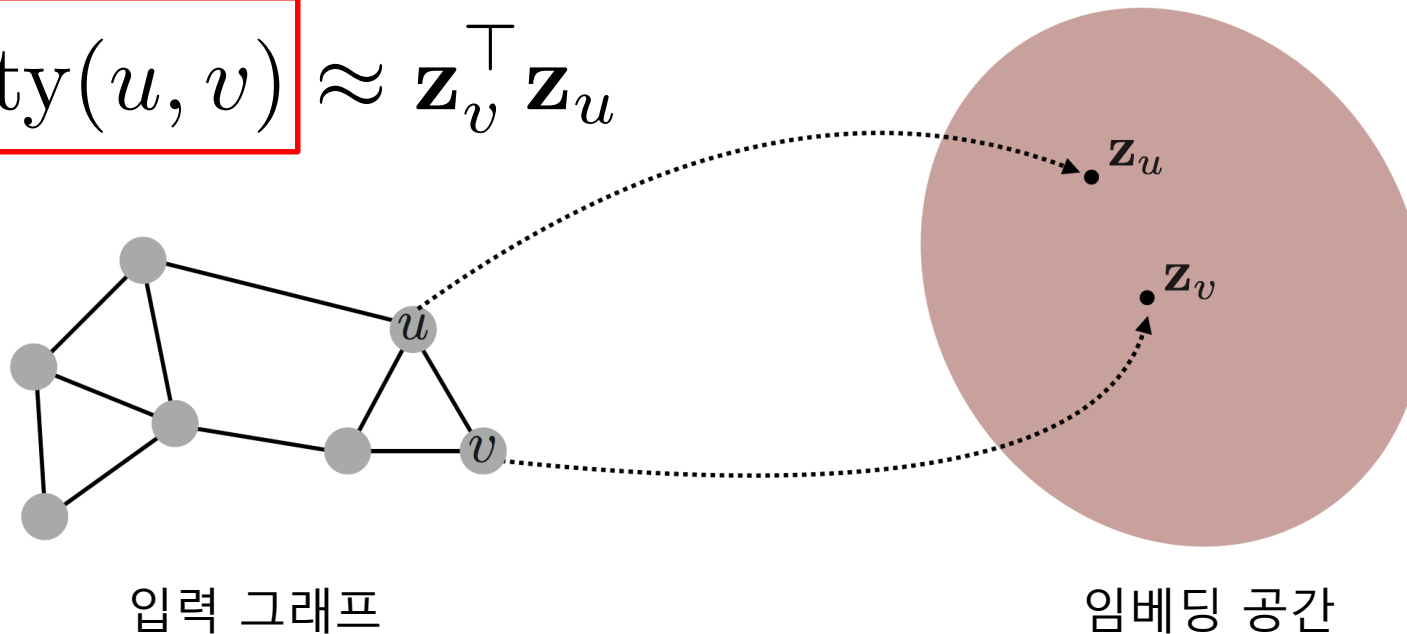


1.3 정점 표현 학습의 목표

정리하면, 정점 임베딩은 다음 두 단계로 이루어집니다

- 1) 그래프에서의 정점 유사도를 정의하는 단계
- 2) 정의한 유사도를 보존하도록 정점 임베딩을 학습하는 단계

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$



2. 인접성 기반 접근법

2.1 인접성 기반 접근법

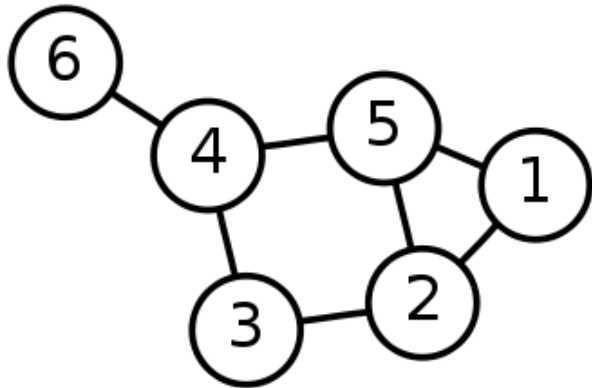
2.2 인접성 기반 접근법의 한계

2.1 인접성 기반 접근법

인접성(Adjacency) 기반 접근법에서는 **두 정점이 인접할 때 유사**하다고 간주합니다

두 정점 u 와 v 가 인접하다는 것은 둘을 직접 연결하는 간선 (u, v) 가 있음을 의미합니다

인접행렬(Adjacency Matrix) A 의 u 행 v 열 원소 $A_{u,v}$ 는 u 와 v 가 인접한 경우 1 아닌 경우 0입니다
인접행렬의 원소 $A_{u,v}$ 를 두 정점 u 와 v 의 유사도로 가정합니다



	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0

2.1 인접성 기반 접근법

인접성 기반 접근법의 **손실 함수(Loss Function)**는 아래와 같습니다

즉, 이 손실 함수가 최소가 되는 정점 임베딩을 찾는 것을 목표로 합니다
손실 함수 최소화를 위해서는 (확률적) 경사하강법 등이 사용됩니다

The diagram illustrates the loss function formula with color-coded boxes and arrows pointing to their respective parts:

- 손실 함수** (Loss Function): Points to the \mathcal{L} in a red box.
- 모든 정점 쌍에 대하여 합산** (Sum over all pairs of vertices): Points to the summation symbol \sum and the domain $(u, v) \in V \times V$ in a green box.
- 임베딩 공간에서의 유사도** (Similarity in the embedding space): Points to the dot product $\mathbf{z}_u^\top \mathbf{z}_v$ in a purple box.
- 그래프에서의 유사도** (Similarity in the graph): Points to the adjacency matrix element $\mathbf{A}_{u,v}$ in a blue box.

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \|^2$$

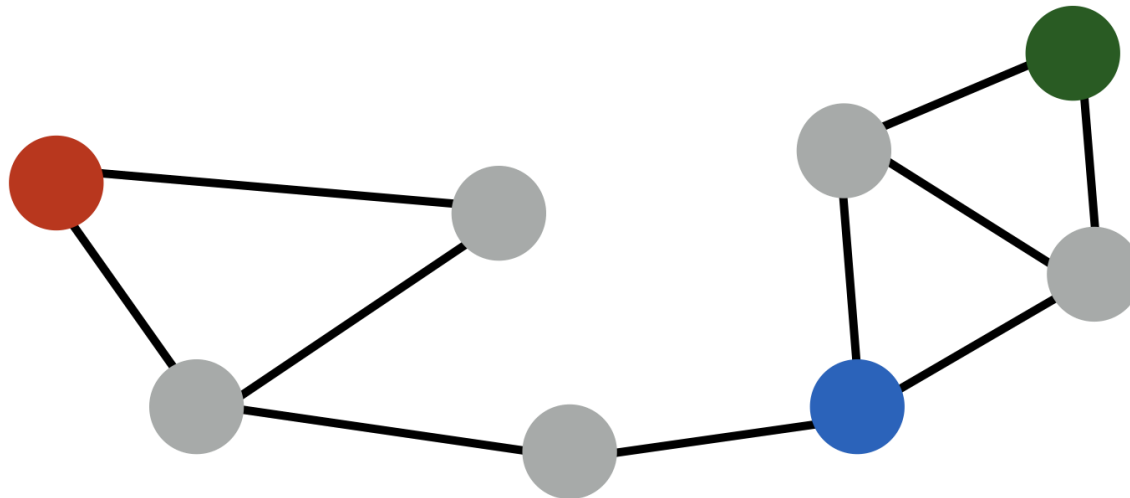
2.2 인접성 기반 접근법의 한계

인접성만으로 유사도를 판단하는 것은 한계가 있습니다

빨간색 정점과 파란색 정점은 거리가 3인 반면

초록색 정점과 파란색 정점은 거리가 2입니다

인접성만을 고려할 경우 이러한 사실에 대한 고려 없이, 두 경우의 유사도는 0으로 같습니다

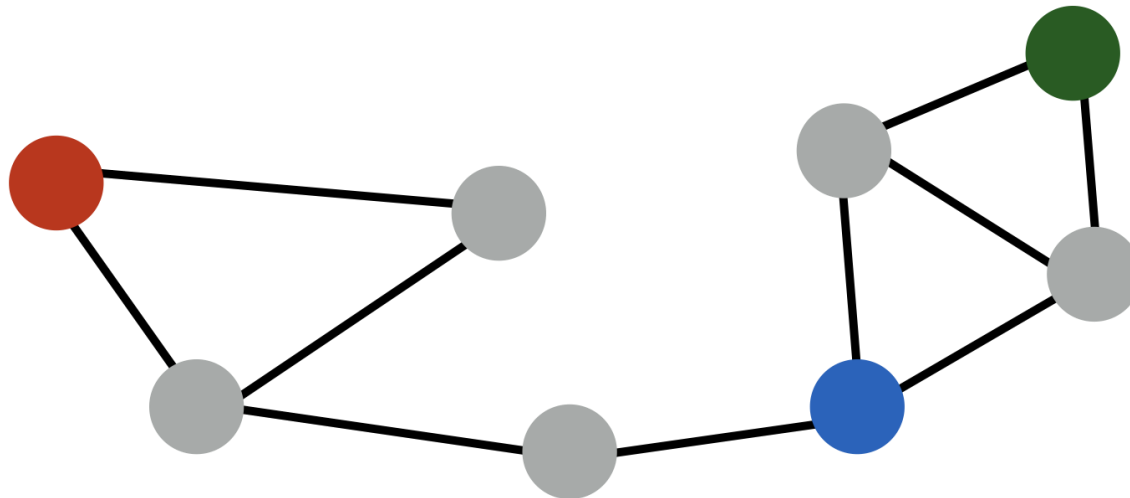


2.2 인접성 기반 접근법의 한계

인접성만으로 유사도를 판단하는 것은 한계가 있습니다

군집 관점에서는 빨간색 정점과 파란색 정점은 다른 군집에 속하는 반면
초록색 정점과 파란색 정점은 다른 군집에 속합니다

인접성만을 고려할 경우 이러한 사실에 대한 고려 없이, 두 경우의 유사도는 0으로 같습니다



3. 거리/경로/중첩 기반 접근법

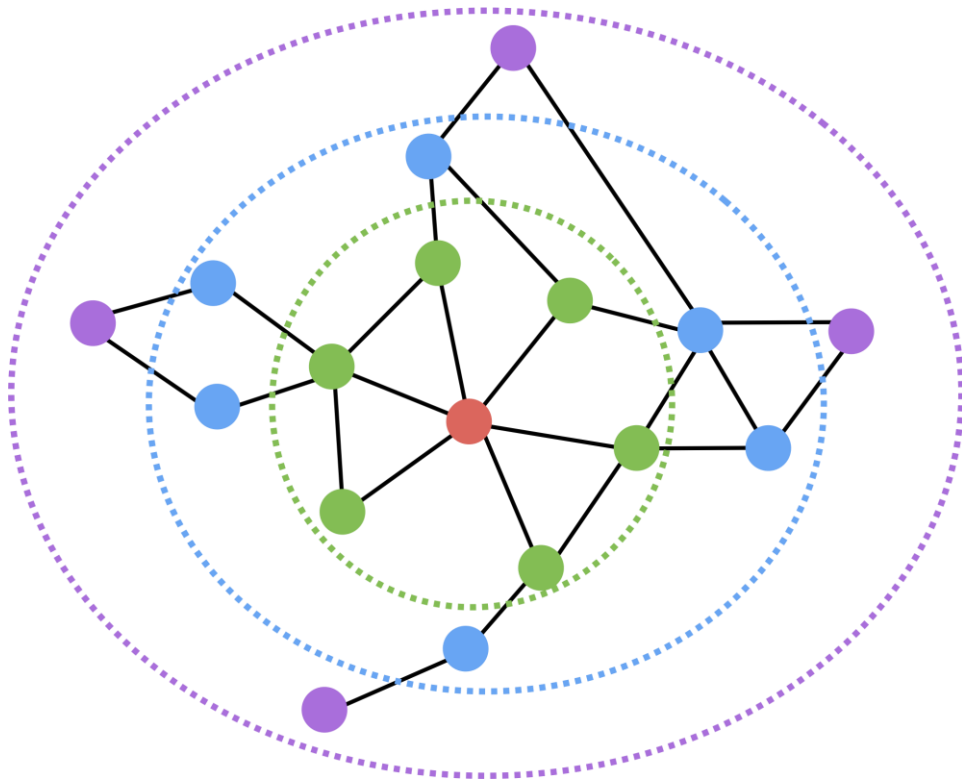
3.1 거리 기반 접근법

3.2 경로 기반 접근법

3.3 중첩 기반 접근법

3.1 거리 기반 접근법

거리 기반 접근법에서는 두 정점 사이의 거리가 충분히 가까운 경우 유사하다고 간주합니다



예를 들어, "충분히"의 기준을 2로 가정합니다

빨간색 정점은 초록색 그리고 파란색 정점들과 유사합니다. 즉 유사도가 1입니다

반면, 빨간색 정점은 보라색 정점과는 유사하지 않습니다. 즉 유사도가 0입니다

3.2 경로 기반 접근법

경로 기반 접근법에서는 두 정점 사이의 경로가 많을 수록 유사하다고 간주합니다

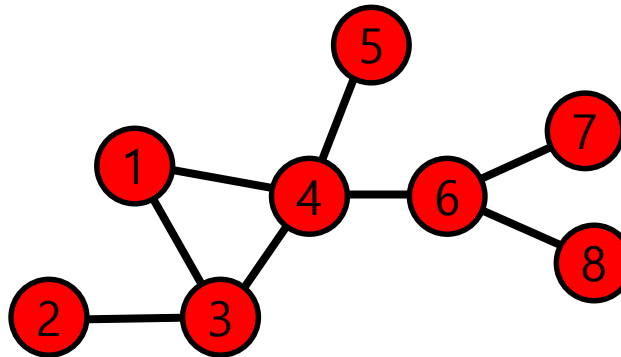
정점 u 와 v 의 사이의 경로(Path)는 아래 조건을 만족하는 정점들의 순열(Sequence)입니다

- (1) u 에서 시작해서 v 에서 끝나야 합니다
- (2) 순열에서 연속된 정점은 간선으로 연결되어 있어야 합니다

경로가 아닌 순열의 예시:

1, 6, 8

1, 3, 4, 5, 6, 8



3.2 경로 기반 접근법

경로 기반 접근법에서는 두 정점 사이의 경로가 많을 수록 유사하다고 간주합니다

두 정점 u 와 v 의 사이의 경로 중 거리가 k 인 것은 수는 $\mathbf{A}_{u,v}^k$ 와 같습니다
즉, 인접 행렬 \mathbf{A} 의 k 제곱의 u 행 v 열 원소와 같습니다

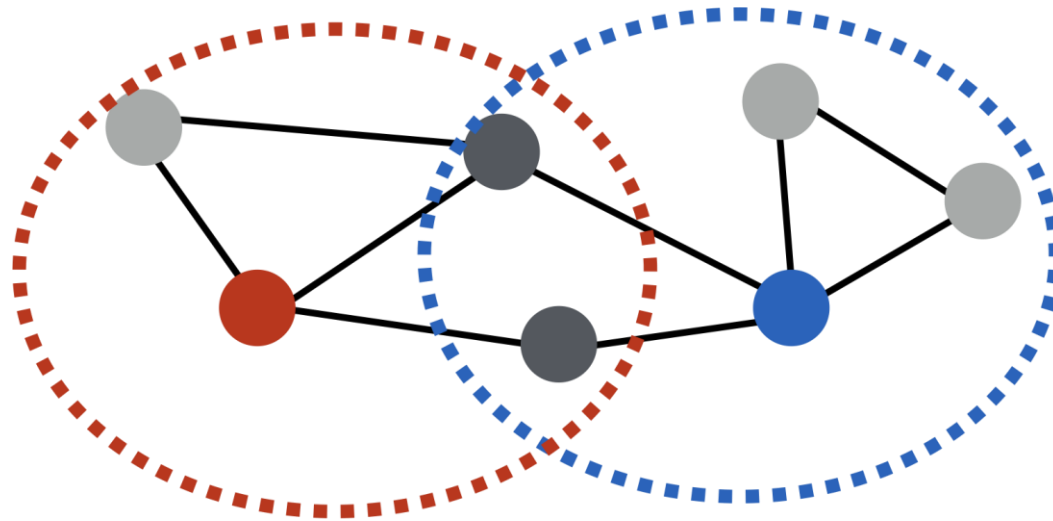
경로 기반 접근법의 손실 함수는 다음과 같습니다

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}^k\|^2$$

3.3 중첩 기반 접근법

중첩 기반 접근법에서는 **두 정점이 많은 이웃을 공유**할 수록 유사하다고 간주합니다

아래 그림에서 **빨간색 정점**은 **파란색 정점**과 **두 명의 이웃을 공유**하기 때문에 유사도는 2가 됩니다



3.3 중첩 기반 접근법

중첩 기반 접근법에서는 두 정점이 많은 이웃을 공유할 수록 유사하다고 간주합니다

정점 u 의 이웃 집합을 $N(u)$ 그리고 정점 v 의 이웃 집합을 $N(v)$ 라고 하면
두 정점의 공통 이웃 수 $S_{u,v}$ 는 다음과 같이 정의 됩니다

$$S_{u,v} = |N(u) \cap N(v)| = \sum_{w \in N(u) \cap N(v)} 1$$

중첩 기반 접근법의 손실 함수는 다음과 같습니다

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \left\| \mathbf{z}_u^\top \mathbf{z}_v - S_{u,v} \right\|^2$$

3.3 중첩 기반 접근법

공통 이웃 수를 대신 **자카드 유사도** 혹은 **Adamic Adar 점수**를 사용할 수도 있습니다

자카드 유사도(Jaccard Similarity)는 공통 이웃의 수 대신 비율을 계산하는 방식입니다

$$\frac{|N_u \cap N_v|}{|N_u \cup N_v|}$$

Adamic Adar 점수는 공통 이웃 각각에 가중치를 부여하여 가중합을 계산하는 방식입니다

$$\sum_{w \in N_u \cap N_v} \frac{1}{d_w}$$

4. 임의보행 기반 접근법

4.1 임의보행 기반 접근법

4.2 DeepWalk와 Node2Vec

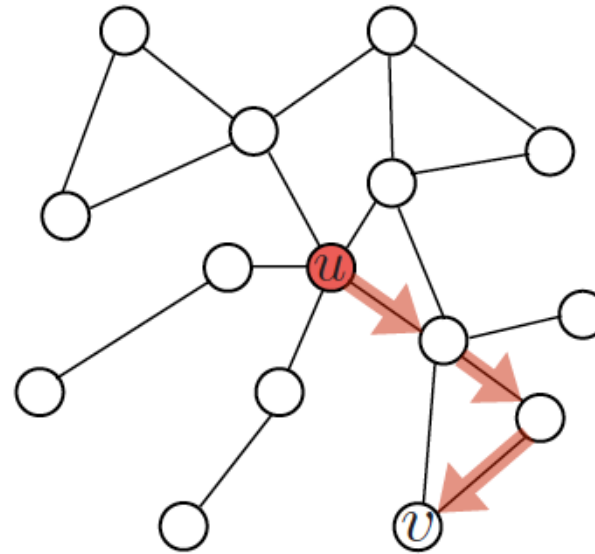
4.3 손실 함수 근사

4.1 임의보행 기반 접근법

임의보행 기반 접근법에서는 **한 정점에서 시작하여 임의보행을 할 때 다른 정점에 도달할 확률**을 유사도로 간주합니다

임의보행이란 현재 정점의 이웃 중 하나를 균일한 확률로 선택하는 이동하는 과정을 반복하는 것을 의미합니다

임의보행을 사용할 경우 시작 정점 주변의 **지역적 정보**와 **그래프 전역 정보**를 모두 고려한다는 장점이 있습니다



4.1 임의보행 기반 접근법

임의보행 기반 접근법은 세 단계를 거칩니다

- 1) 각 정점에서 시작하여 임의보행을 반복 수행합니다
- 2) 각 정점에서 시작한 임의보행 중 도달한 정점들의 리스트를 구성합니다
이 때, 정점 u 에서 시작한 임의보행 중 도달한 정점들의 리스트를 $N_R(u)$ 라고 합니다
한 정점을 여러 번 도달한 경우, 해당 정점은 $N_R(u)$ 에 여러 번 포함될 수 있습니다
- 3) 다음 손실함수를 최소화하는 임베딩을 학습합니다

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

u 에서 시작한 임의보행이 v 에 도달할 확률을 임베딩으로부터 추정한 결과를 의미합니다

4.1 임의보행 기반 접근법

어떻게 임베딩으로부터 **도달 확률**을 추정할까요?

정점 u 에서 시작한 임의보행이 정점 v 에 도달할 확률 $P(v|z_u)$ 을 다음과 같이 추정합니다

$$P(v|z_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

즉 유사도 $\mathbf{z}_v^\top \mathbf{z}_u$ 가 높을 수록 도달 확률이 높습니다

4.1 임의보행 기반 접근법

추정한 도달 확률을 사용하여 손실함수를 완성하고 이를 최소화하는 임베딩을 학습합니다

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log \left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \right)$$

모든 시작점에 대하여 합산

임의보행 중 마주친 모든 정점에 대하여 합산

임베딩으로부터 추정한 도달 확률

4.2 DeepWalk와 Node2Vec

임의보행의 방법에 따라 **DeepWalk**와 **Node2Vec**이 구분됩니다

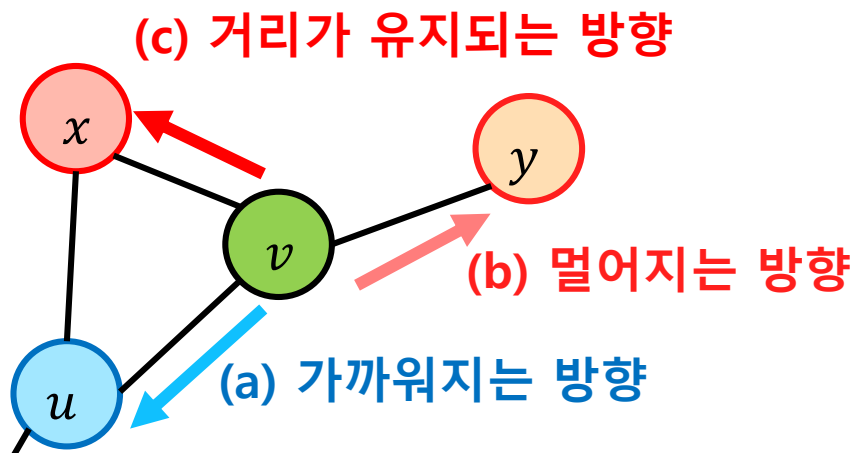
DeepWalk는 앞서 설명한 **기본적인 임의보행**을 사용합니다
즉, 현재 정점의 이웃 중 하나를 **균일한 확률**로 선택하는 이동하는 과정을 반복합니다

4.2 DeepWalk와 Node2Vec

Node2Vec은 2차 치우친 임의보행(Second-order Biased Random Walk)을 사용합니다

현재 정점(예시에서 v)과 직전에 머물렀던 정점(예시에서 u)을 모두 고려하여 다음 정점을 선택합니다

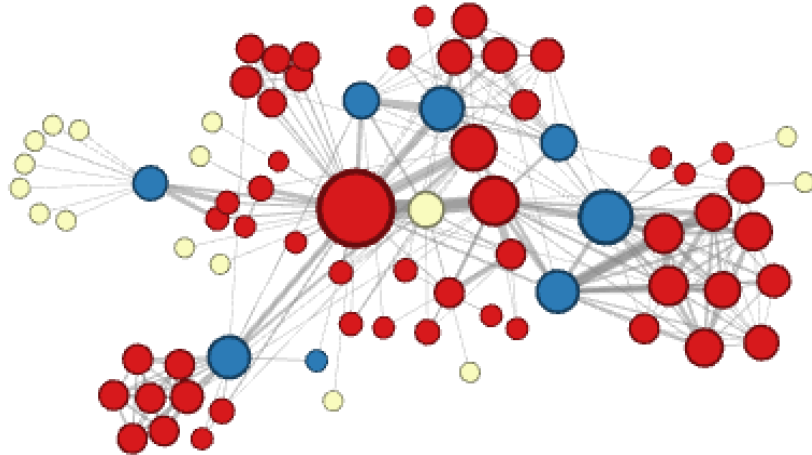
직전 정점의 거리를 기준으로 경우를 구분하여 차등적인 확률을 부여합니다



4.2 DeepWalk와 Node2Vec

Node2Vec에서는 부여하는 확률에 따라서 다른 종류의 임베딩을 얻습니다

아래 그림은 Node2Vec으로 임베딩을 수행한 뒤, K-means 군집 분석을 수행한 결과입니다



멀어지는 방향에 높은 확률을 부여한 경우

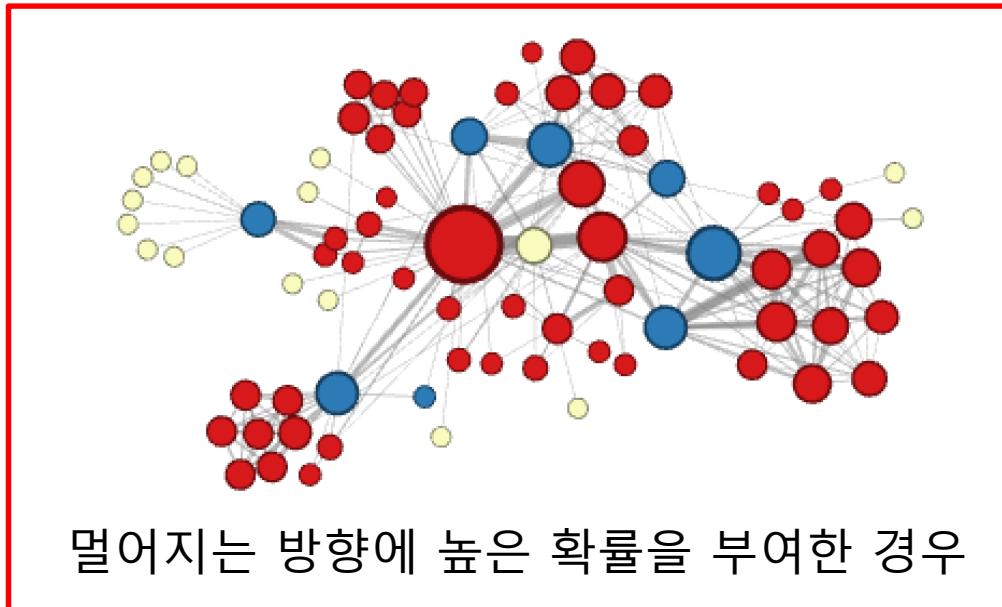


가까워지는 방향에 높은 확률을 부여한 경우

4.2 DeepWalk와 Node2Vec

Node2Vec에서는 부여하는 확률에 따라서 다른 종류의 임베딩을 얻습니다

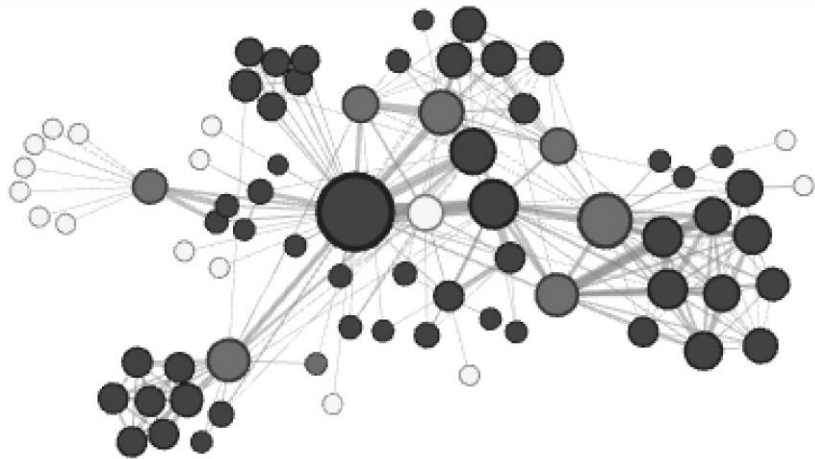
멀어지는 방향에 높은 확률을 부여한 경우,
정점의 역할(다리 역할, 변두리 정점 등)이 같은 경우 임베딩이 유사합니다



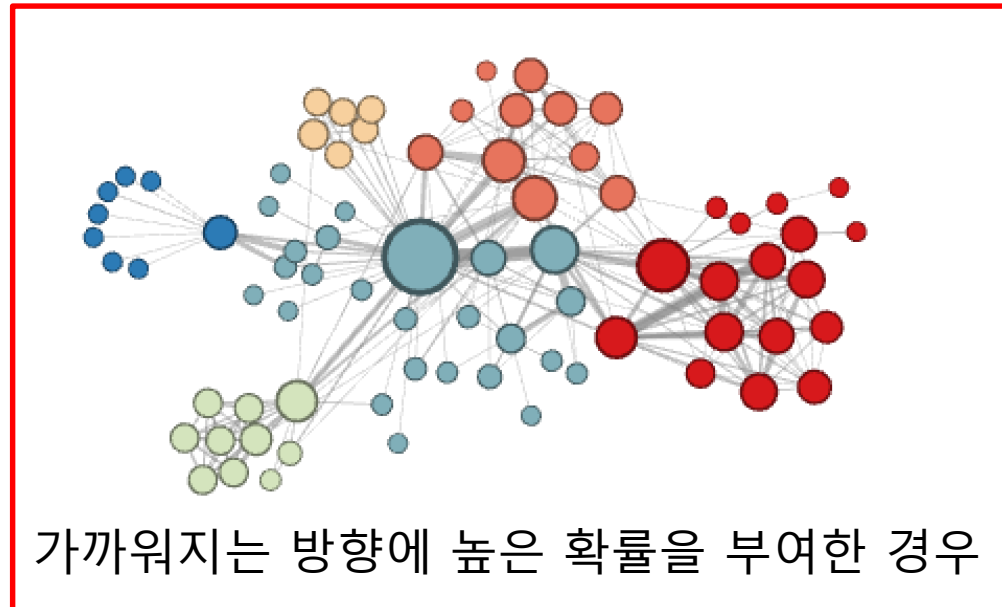
4.2 DeepWalk와 Node2Vec

Node2Vec에서는 부여하는 확률에 따라서 다른 종류의 임베딩을 얻습니다

가까워지는 방향에 높은 확률을 부여한 경우,
같은 군집(Community)에 속한 경우 임베딩이 유사합니다



멀어지는 방향에 높은 확률을 부여한 경우



가까워지는 방향에 높은 확률을 부여한 경우

4.3 손실 함수 근사

임의보행 기법의 손실함수는 계산에 정점의 수의 제곱에 비례하는 시간이 소요됩니다

중첩된 합 때문입니다

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log \left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \right)$$

정점이 많은 경우, 제곱은 매우 큰 숫자입니다


예시로 1억의 제곱은 1경 = 10,000조 입니다


4.3 손실 함수 근사

따라서 많은 경우 근사식을 사용합니다

모든 정점에 대해서 정규화하는 대신 몇 개의 정점을 뽑아서 비교하는 형태입니다
이 때 뽑힌 정점들을 **네거티브 샘플**이라고 부릅니다

$$\log \left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \right)$$
$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^k \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

 **sigmoid 함수**

 **확률 분포**

4.3 손실 함수 근사

따라서 많은 경우 근사식을 사용합니다

연결성에 비례하는 확률로 **네거티브 샘플**을 뽑으며,
네거티브 샘플이 많을 수록 학습이 더욱 안정적입니다

$$\log \left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \right)$$
$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^k \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

sigmoid 함수 **확률 분포**

5. 변환식 정점 표현 학습의 한계

5.1 변환식 정점 표현 학습과 귀납식 정점 표현 학습

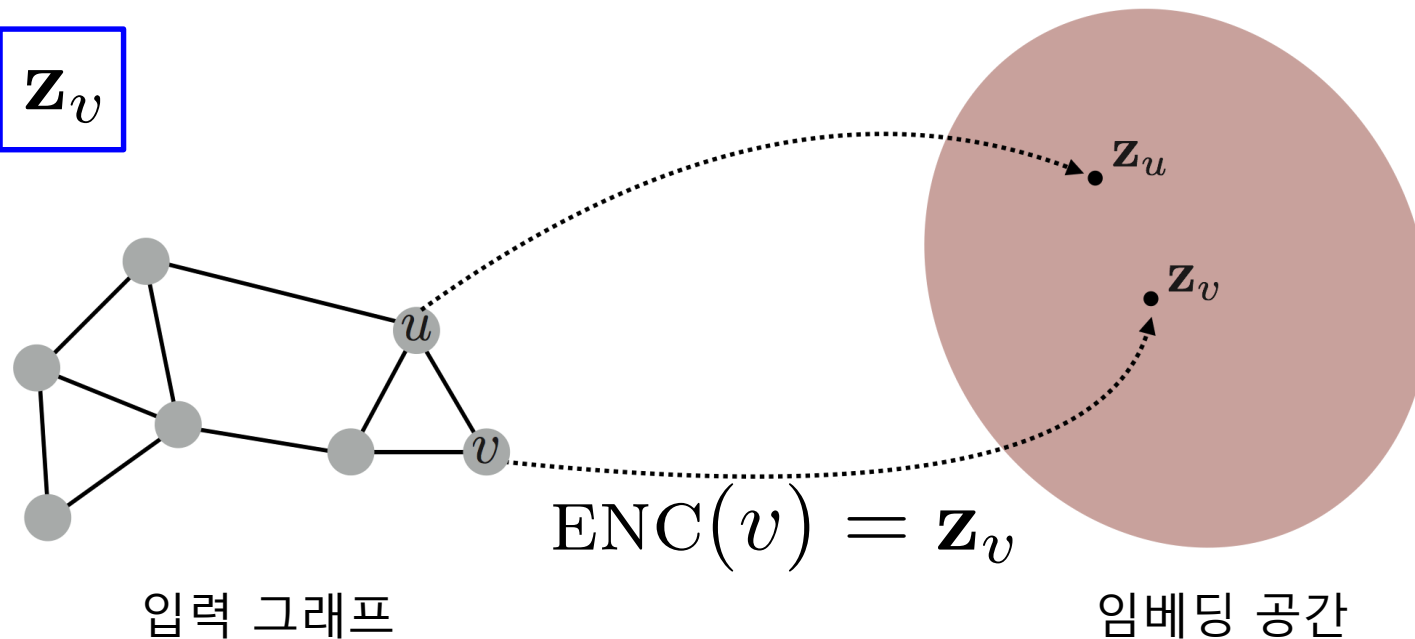
5.2 변환식 정점 표현 학습의 한계

5.1 변환식 정점 표현 학습과 귀납식 정점 표현 학습

지금까지 소개한 정점 임베딩 방법들을 **변환식(Transductive)** 방법입니다

변환식(Transductive) 방법은 학습의 결과로 정점의 임베딩 자체를 얻는다는 특성이 있습니다
정점을 임베딩으로 변화시키는 함수, 즉 인코더를 얻는 귀납식(Inductive) 방법과 대조됩니다

$$\boxed{\text{ENC}}(v) = \boxed{\mathbf{z}_v}$$



5.2 변환식 정점 표현 학습의 한계

변환식 임베딩 방법은 여러 한계를 갖습니다

- 1) 학습이 진행된 이후에 추가된 정점에 대해서는 임베딩을 얻을 수 없습니다
- 2) 모든 정점에 대한 임베딩을 미리 계산하여 저장해두어야 합니다
- 3) 정점이 속성(Attribute) 정보를 가진 경우에 이를 활용할 수 없습니다

9강에서는 이런 단점을 극복한 귀납식 임베딩 방법을 소개할 예정입니다
대표적인 귀납식 임베딩 방법이 바로 그래프 신경망(Graph Neural Network)입니다

6. 실습: Node2Vec을 사용한 군집 분석 및 정점 분류

6.1 Node2Vec 수행

6.2 군집 분석

6.3 정점 분류

6.1 Node2Vec 수행

필요한 라이브러리를 불러옵니다

```
import networkx as nx
from node2vec import Node2Vec
from matplotlib import pyplot as plt
```


6.1 Node2Vec 수행

실습에 사용할 그래프를 파일에서 읽어옵니다
소설 레미자레블 등장인물 간의 공동 등장 그래프입니다

```
weighted_edgelist=[]
with open('./lesmis.mtx', 'r') as f:
    for line in f:
        l = line.strip().split()
        if l[0].isdigit() == False:
            continue
        weighted_edgelist.append((str(int(l[0])-1), str(int(l[1])-1), float(l[2])))

G = nx.Graph()
G.add_weighted_edges_from(weighted_edgelist)
```

6.1 Node2Vec 수행

Node2Vec을 수행합니다

```
node2vec = Node2Vec(G, dimensions=16, walk_length=4, num_walks=200, workers=4)
model = node2vec.fit(window=2, min_count=1, batch_words=4)
```

6.1 Node2Vec 수행

Nod2Vec의 출력으로 얻은 정점들의 임베딩을 확인해봅시다

```
print(model.wv['2'])
```

```
[-0.2945959  0.5058547  0.36930382 -0.52473325  0.23561095 -1.3964843  
 0.28874806 -0.4072109  0.5824377  1.3277762  0.2994783 -0.44092816  
 0.53834355  1.1411396 -0.8561847  0.6159473 ]
```

6.2 군집 분석

필요한 라이브러리를 불러옵니다

```
from sklearn.cluster import KMeans
import numpy as np
from matplotlib import pyplot as plt
```

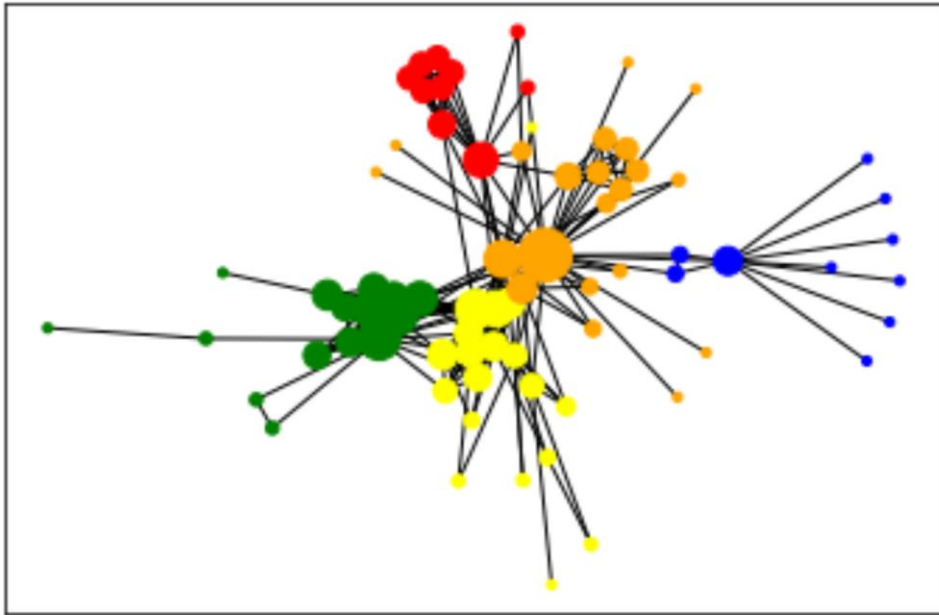
6.2 군집 분석

정점 임베딩을 입력으로 K-Means를 수행하여 정점의 군집을 찾습니다

```
vectors_array = np.zeros((len(G.nodes), 16))
for node in G.nodes:
    vectors_array[int(node)] = model.wv[node]
kmeans = KMeans(n_clusters=5, random_state=0).fit(vectors_array)
```

6.2 군집 분석

군집에 따라 정점의 색을 구분하여 시각화한 결과입니다



6.3 정점 분류

필요한 라이브러리를 불러옵니다

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

6.3 정점 분류

실습에 사용할 그래프를 파일에서 읽어옵니다

논문 간의 인용 네트워크이고, 논문의 주제가 정점의 유형으로 주어집니다

```
node_class = dict()
edgelist = list()
class_num = 1
class_name_to_num = dict()
with open('./cora.content', 'r') as f, open('./cora.cites', 'r') as f2:
    for line in f:
        l = line.strip().split()
        class_name = l[-1]
        if class_name not in class_name_to_num:
            class_name_to_num[class_name] = class_num
            class_num += 1
        node_class[l[0]] = class_name_to_num[class_name]
    for line in f2:
        l = line.strip().split()
        edgelist.append((l[1], l[0]))
```

```
G = nx.DiGraph()
G.add_edges_from(edgelist)
```


6.3 정점 분류

Node2Vec을 수행합니다

```
node2vec = Node2Vec(G, dimensions=16, walk_length=4, num_walks=200, workers=4)
model = node2vec.fit(window=2)
```

6.3 정점 분류

Node2Vec의 출력인 정점 임베딩을 학습 데이터와 평가 데이터로 분리합니다

```
X = list()
y = list()
node_name_to_idx = dict()
for i, (v, class_) in enumerate(node_class.items()):
    node_name_to_idx[v] = i
    X.append(model.wv[v])
    y.append(class_)
X = np.array(X)
y = np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle= True)
```

6.3 정점 분류

학습 데이터를 사용하여 분류기인 다층퍼셉트론을 학습합니다

```
clf = MLPClassifier(max_iter=500).fit(X_train, y_train)
y_predict = clf.predict(X_test)
```

6.3 정점 분류

평가 데이터를 사용하여 분류기의 성능을 평가합니다

```
print("Accuracy : {0:05.2f}% ".format(accuracy_score(y_test, y_predict)*100))
```

Accuracy : 77.70%

7강 정리

1. 정점 표현 학습

- 그래프의 정점들을 벡터로 표현하는 것
- 그래프에서 정점 사이의 유사성을 계산하는 방법에 따라 여러 접근법이 구분됨

2. 인접성 기반 접근법

3. 거리/경로/중첩 기반 접근법

4. 임의보행 기반 접근법

- 임의보행 방법에 따라 DeepWalk와 Node2Vec가 구분됨

5. 변환식 정점 표현 학습의 한계

6. 실습: Node2Vec을 사용한 군집 분석 및 정점 분류