3주차 세션

이은빈, 최예은

EWHA EURON

목차

#01 Node Embeddings

#02 Random Walk Approaches for Node

#03 Embedding entire graphs

#04 Node2Vec

#05 Embedding Entire Graphs

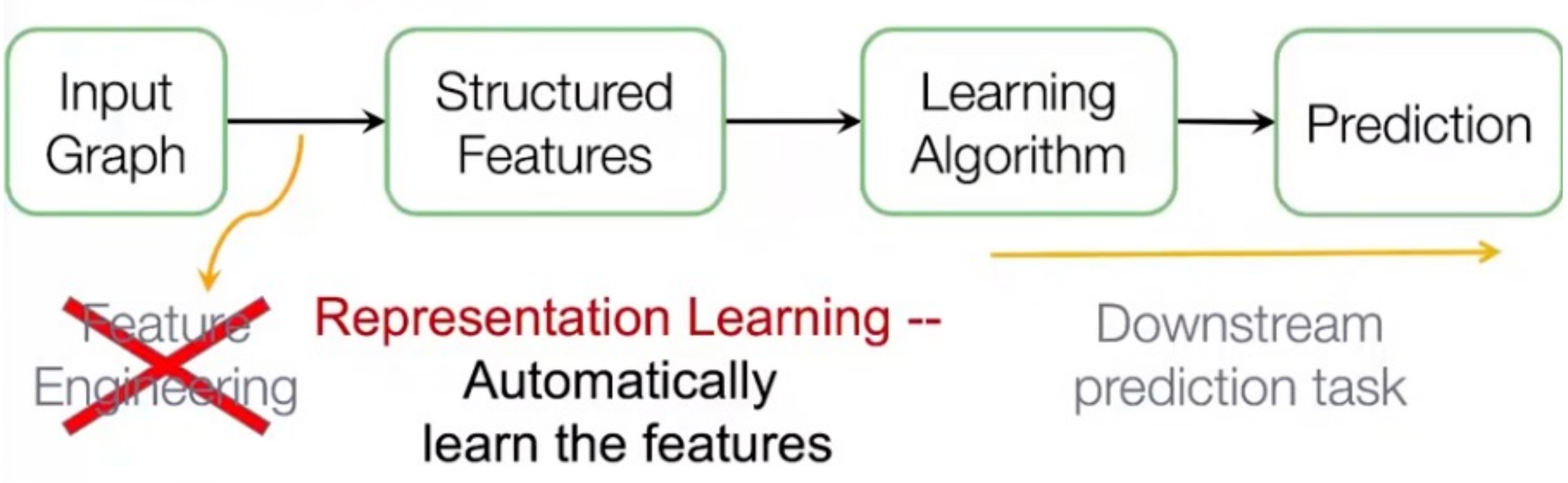
EWHA
EURON

Node Embeddings

Node Embeddings

#0. Traditional Machine Learning for Graphs

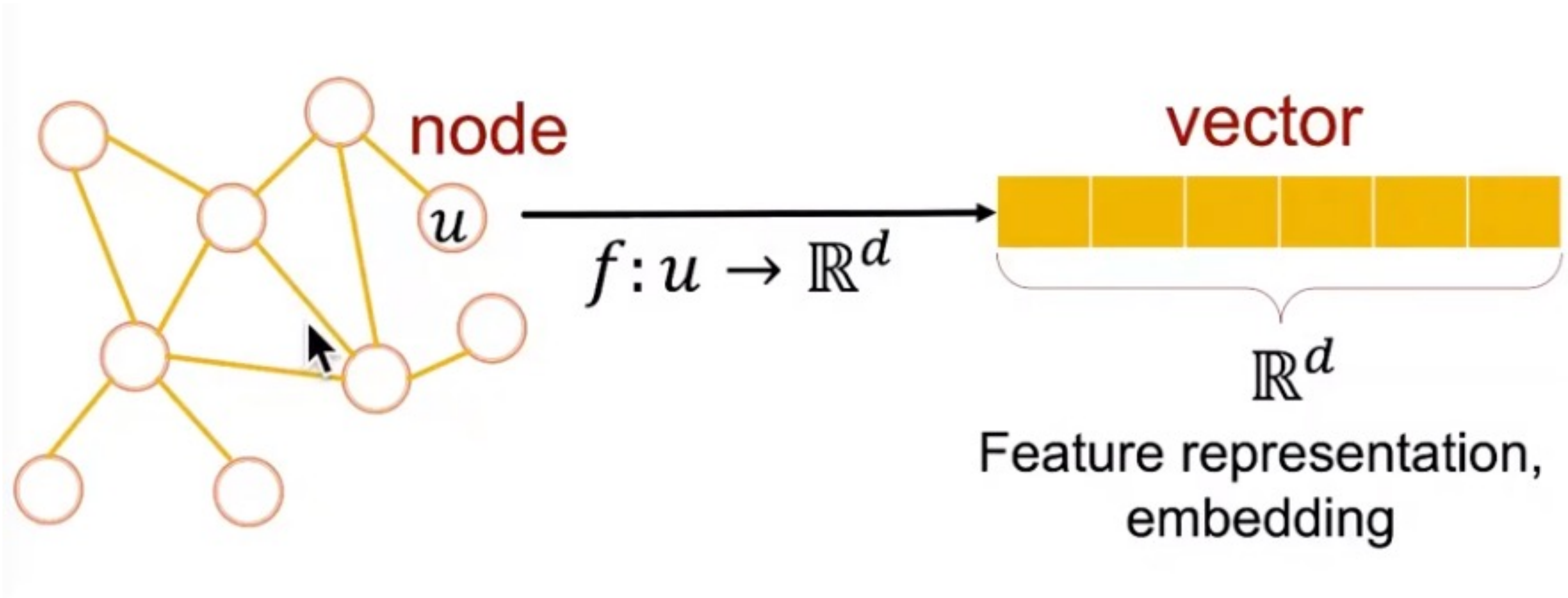
- 각각의 domain과 task에 맞게 feature engineering
- 노드, 링크, 그래프 레벨 변수 생성 후 다시 task에 적용
- Automatic하게 feature를 학습하는 방법은 없을까?



Node Embeddings

#1. Graph Representation Learning

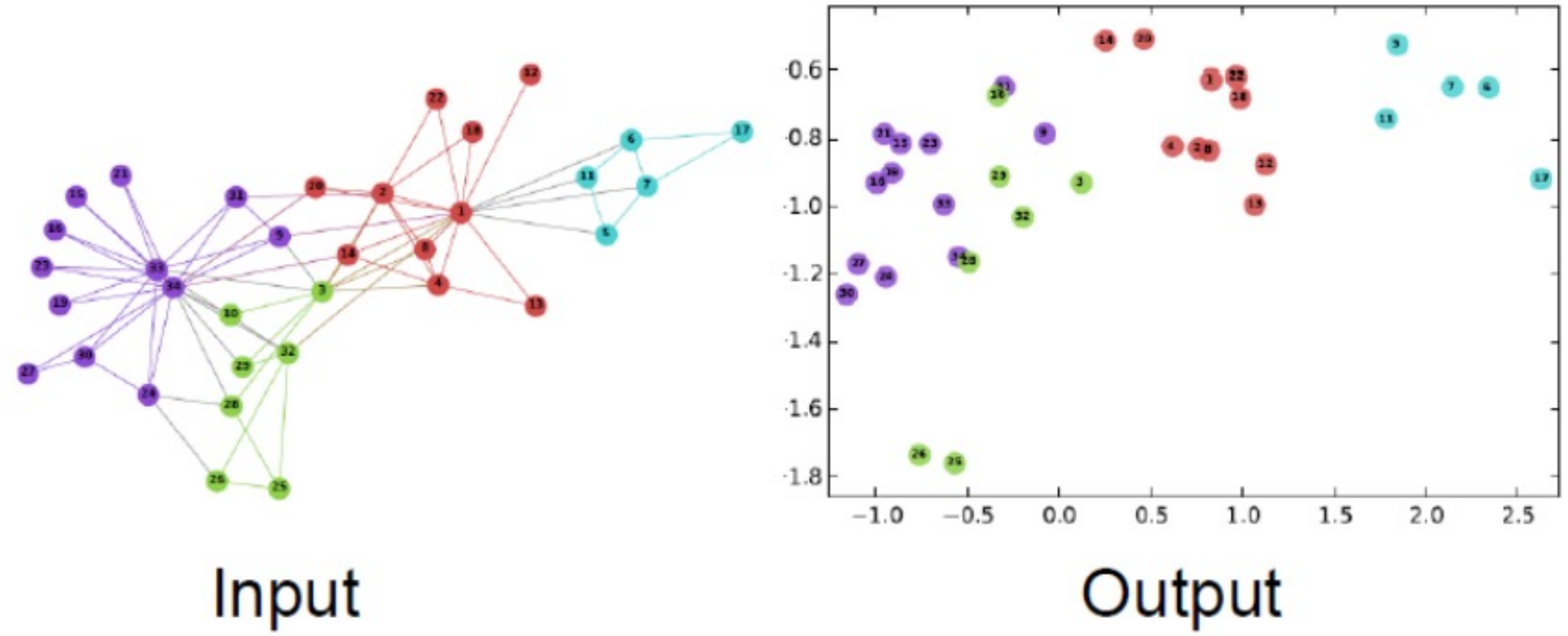
- Automatic하게 feature를 학습하는 방법
- 그래프의 node, link, graph를 임의의 d 차원 벡터로 변환시키는 과정
- 노드 간의 embedding 유사도은 network의 유사도을 나타낸다



Node Embeddings

#1. Graph Representation Learning

- Tasks
  - Node classification
  - Link prediction
  - Graph classification
  - Anomalous node detection
  - Clustering
  - …

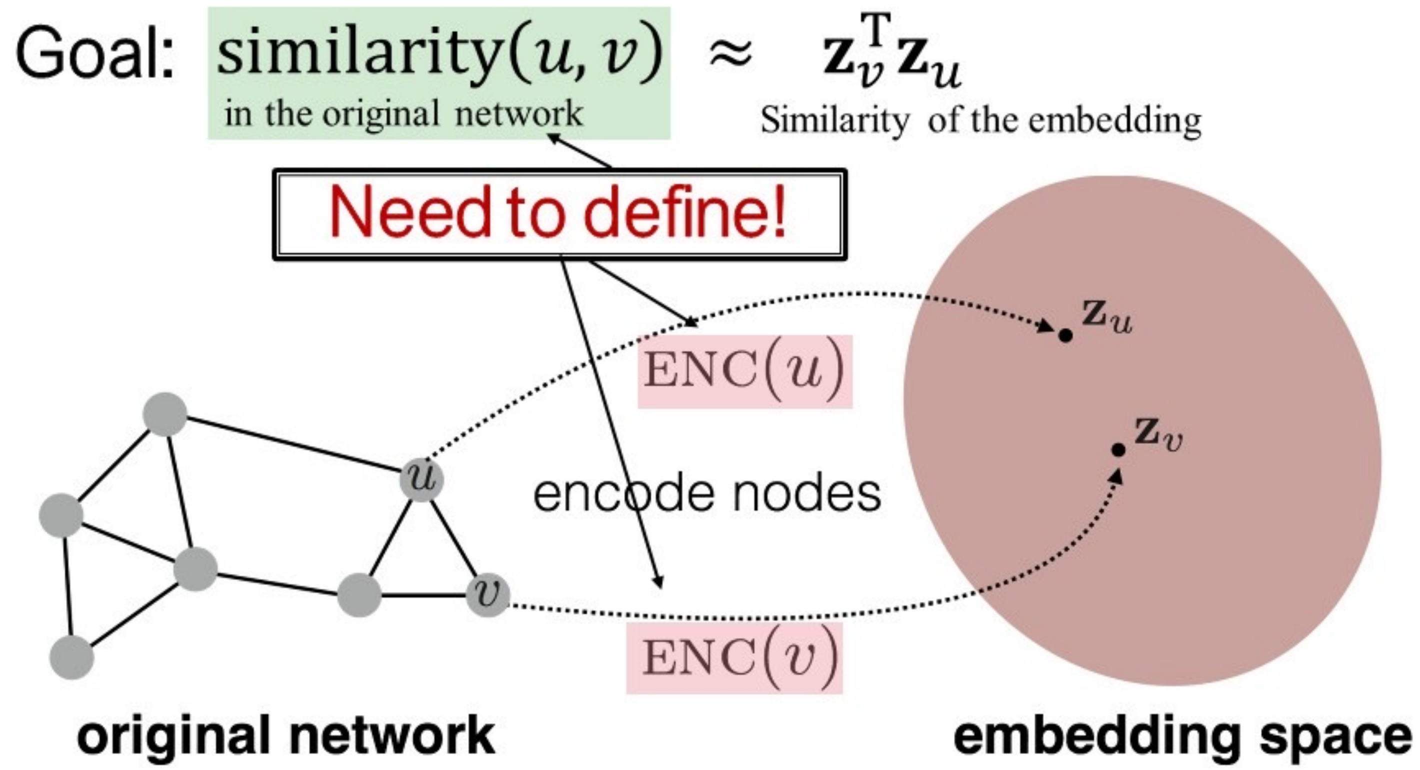


Node Embeddings

#2. Node Embedding

- 목표 : 임베딩 공간의 유사도가 그래프의 유사도에 근접하도록 노드를 Encoding
- 기본 요소 : V (노드 집합), A (인접행렬)
- 두 노드 u, v의 유사도를 임베딩 공간에서의 두 벡터 z_u , z_v 의 유사도로 근사

- 과정
  a) Encoder ENC는 노드에서 임베딩으로 매핑
  b) Node similarity Function 정의
  c) Decoder DEC는 임베딩에서 유사도 점수로 매핑
  d) 유사도가 비슷해지도록 encoder의 parameter 최적화



Node Embeddings

#2. Two Key Components

1. Encoder

   각각의 노드를 low-dimensional vector로 매핑

2. Similarity Function

   벡터 공간의 relationship을 original network relationship에 매핑

   Similarity Function가 정의되면 Decoder가 similarity score로 임베딩을 다시 매핑

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

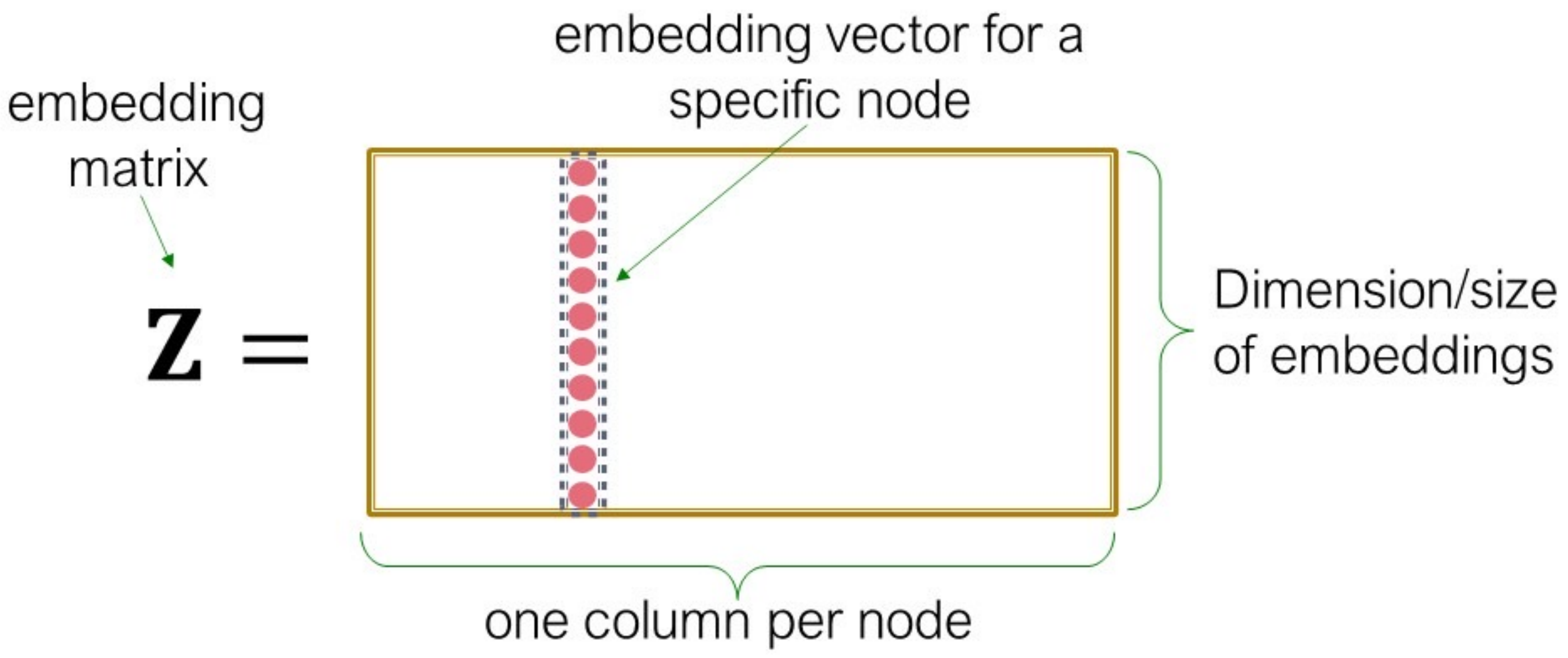
in the original network   Similarity of the embedding

Node Embeddings

#3. ‘Shallow’ Encoding

- 가장 간단한 형식의 Encoding
- Embedding-lookup 방식
  - 임베딩 공간에서 특정 node에 해당하는 column을 읽어오는 것
- 임베딩의 최적화는 encoder의 최적화가 아닌 embedding matrix 자체의 최적화
- 노드 수가 많아지게 되면 가지고 있어야 하는 lookup table이 커지는 단점이 존재

- Other Methods
  - DeepWalk, node2vec



Node Embeddings

#4. Define node similarity?

- Similarity function은 노드 유사도를 정의하는 것에 따라 다른 choice를 한다

- How to define node similarity?
  - 두 노드가 연결되어 있을 때?
  - 공통의 neighbor가 있을때?
  - 구조적으로 비슷한 역할을 할 때?

- node label과 feature를 사용하지 않고 network 구조만 활용한 unsupervised 방식을 고민

Random Walk Approaches for Node

Random Walk Approaches for Node

#1. Notation

- Vector z_u : 노드 u의 임베딩 벡터
- Probability $P(v|z_u)$: 노드 u에서 출발하여 random walk하여 노드 v에 도착할 확률 추정치
- Softmax function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

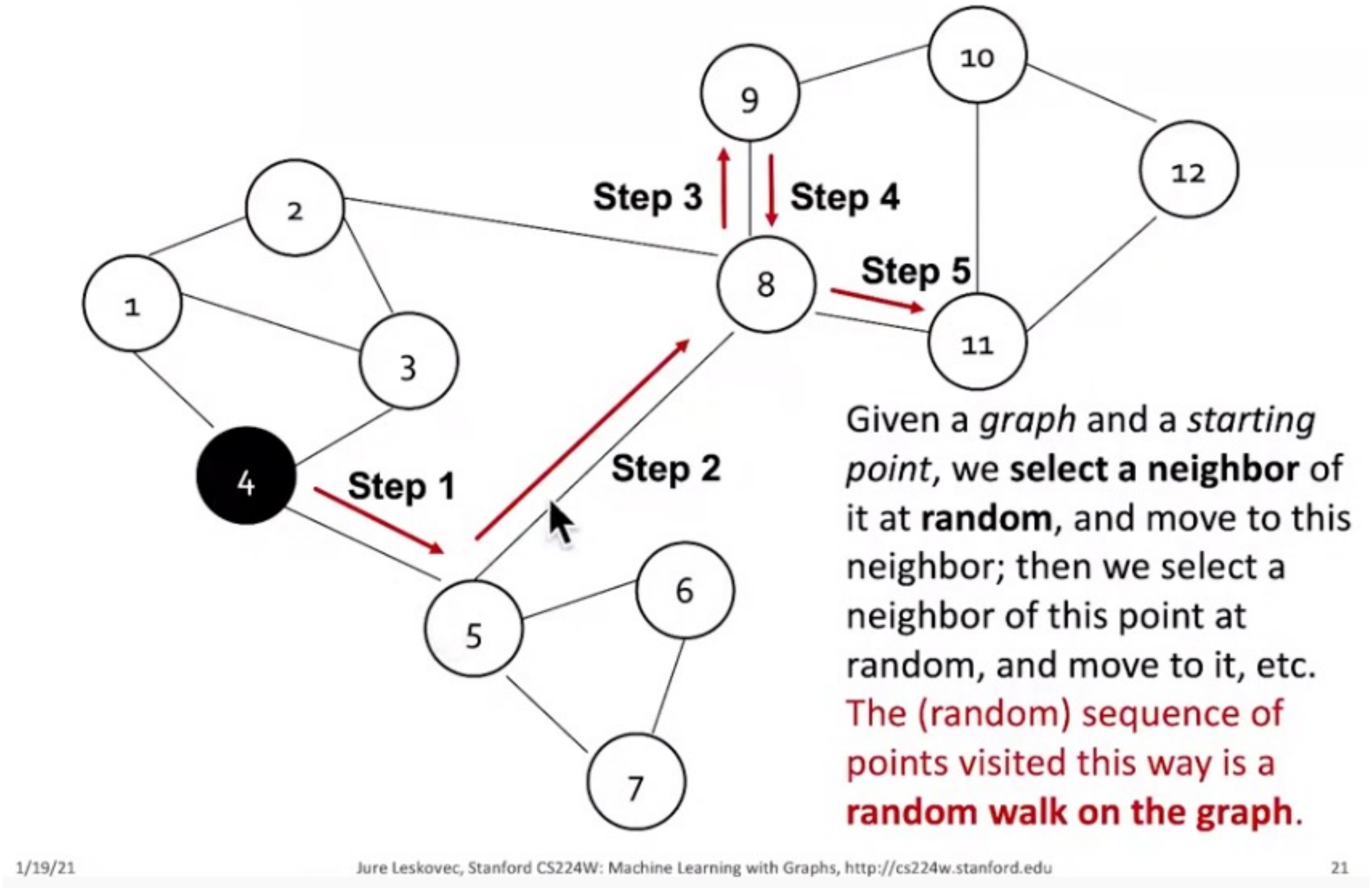
- Sigmoid function

$$S(x) = \frac{1}{1+e^{-x}}.$$

Random Walk Approaches for Node

#2. Random Walk

- Graph와 Starting point가 주어졌을 때 neighbor를 random하게 선택해서 이동하는 sequence
- '네트워크 전반에서 노드 u,v가 random walk 에서 동시에 발생할 확률'로 근사
  → 두 개의 노드가 확률이 유사하다면 서로 가까이 있다는 의미



$$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx \text{probability that } u \text{ and } v \text{ co-occur on a random walk over the graph}$$

Random Walk Approaches for Node

#2. Random Walk

- $P_R(v|u)$: node u에서 출발한 random walk R에서 node v를 만날 확률
- 위 확률이 높을 수록 임베딩 벡터 z_u , z_v 의 유사도가 높다
- 특성
  - High expressivity : local and high-order neighborhood 정보를 확률로서 표현하므로
  - High efficienty : 모든 노드를 고려하지 않고 random walk에서 동시에 등장하는 노드쌍만 고려

Random Walk Approaches for Node

#2. Random Walk Optimization

1) short fixed-length random walk를 각 노드마다 진행

2) 각 노드 u에 대한 $N_R(u)$ 수집

이 때 랜덤워크가 진행되면 특정 노드를 반복할 수 있으므로 중복 허용

3) $$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) \mid \mathbf{z}_u)$$ 에 의해 임베딩 최적화

위 과정을 통해 이웃 노드가 등장할 확률이 높아지도록 임베딩 벡터 최적화

이웃 노드는 내적값이 커지고, 이웃하지 않은 노드 간에는 내적값이 작아지게 됨

Random Walk Approaches for Node

#2. Random Walk Optimization

- Random-Walk Optimization

  = Random Walk embedding 최적화

  = L을 최소화하는 임베딩 z_u 찾기

- Loss function에서 전체 노드가 2번 중첩

  → 시간복잡도가 커지는 문제 발생

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

Nested sum over nodes gives
$O(|V|^2)$ complexity!

Putting it all together:

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

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings =

Finding embeddings $\mathbf{z}_u$ that minimize $\mathcal{L}$

Random Walk Approaches for Node

#3. Negative Sampling

- Softmax Function에서 분모 (Normalization term)가 비효율의 원인이므로 정규화 항을 근사

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

- Idea
  - 전체 노드 말고 subset에 대해서만 normalization
  - 노드 u와 이웃하지 않은 노드 중 k개의 random negative samples 추출

Random Walk Approaches for Node

#3. Negative Sampling

- Sigmoid term
  - 1st term : 이웃 노드간 계산되는 이웃노드일 확률로 최대화
  - 2nd term : u와 이웃하지 않은 노드와 계산되는 이웃노드일 확률로 최소화
- Random한 k개의 negative sample
  - k가 클수록 Robust estimates
  - k가 작을수록 Negative Events에 대한 higher bias
  - 보통 5~20 사이가 적절

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k}\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

sigmoid function
(makes each term a "probability" between 0 and 1)

random distribution over all nodes

Instead of normalizing w.r.t. all nodes, just normalize against k random "**negative samples**" n_i

Random Walk Approaches for Node

#4. Stochastic Gradient Descent

- Objective function을 얻은 후에 최적화하는 방법
- Gradient Descent : a simple way to minimize objective function

- Idea
  - 모든 example에 대한 gradient가 아닌, 각각의 training sample에 대한 gradient 계산
  - 여기서는 모든 negative node나 neighborhood에 대하여 계산하는 것이 아니라 given neighborhood에 대해서만 계산

- Initialize z_i at some randomized value for all i.
- Iterate until convergence: $\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
  - Sample a node i, for all j calculate the derivative $\frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$.
  - For all j, update: $z_j \leftarrow z_j - \eta \frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$.

Random Walk Approaches for Node

#5. Random Walks : Summary

1) 적절한 short-fixed length의 random walk를 통해 $N_R(u)$ 수집 (중복 허용)

2) SGD로 objective function 최적화

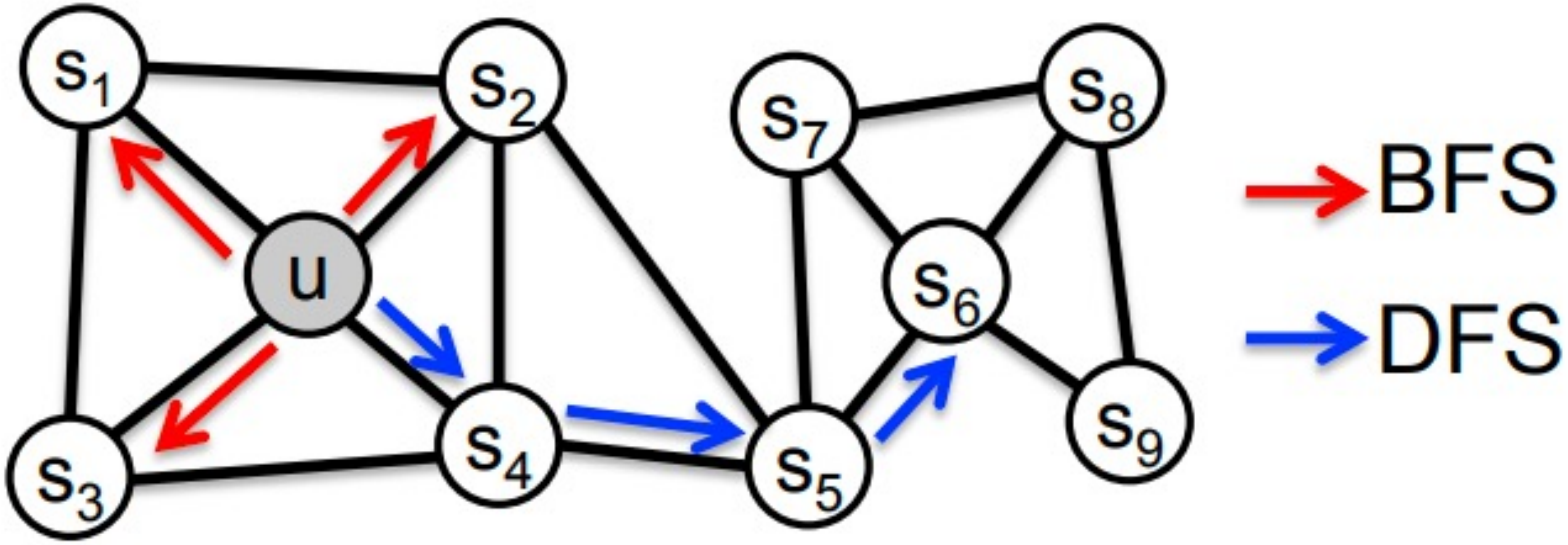
3) 적절한 embedding vector z_u 얻을 수 있음

Node2Vec

#01 Node2Vec이란?

Node2Vec란?

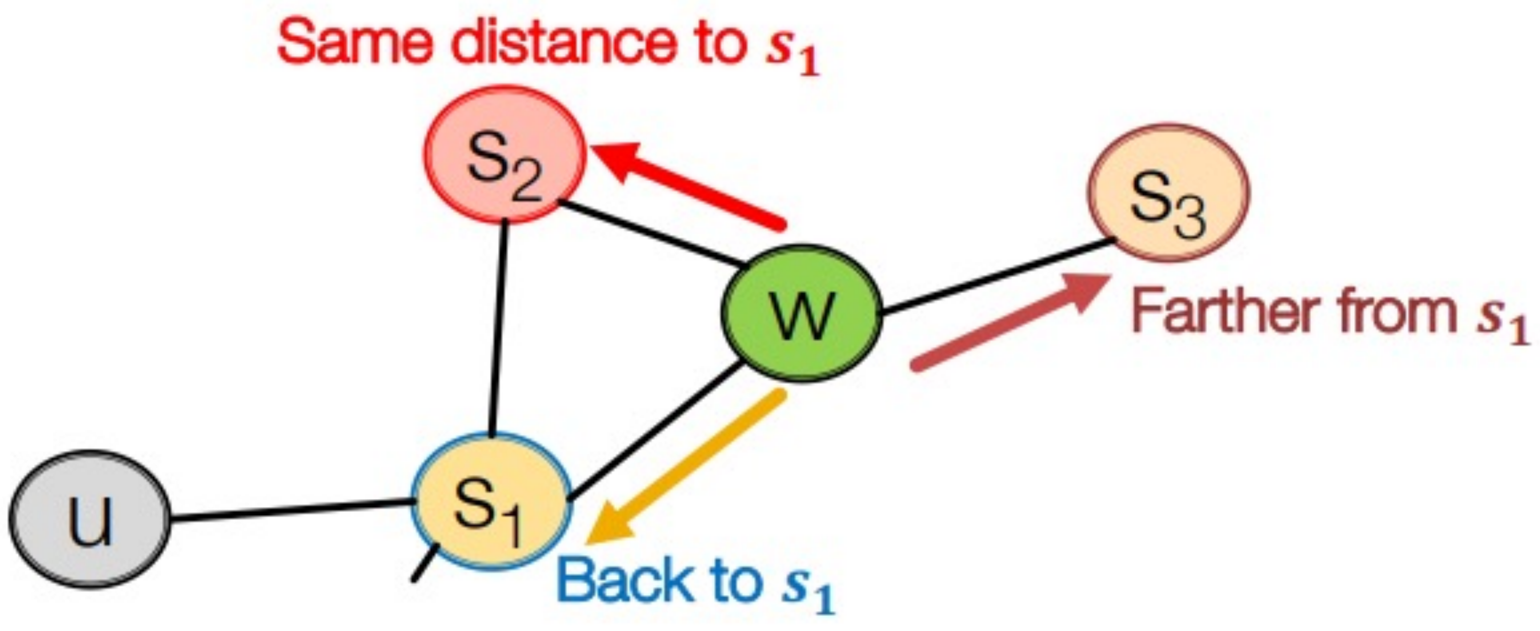
DeepWalk에서 랜덤 워크를 생성하는 방법을 발전시킨 방법



node2vec이 deepwalk와 다른점은 그래프에서 이웃 노드 집합 $N_R(u)$를 찾을 때
비교적 유연하게 대처할 수 있는 전략 R을 이용하여 노드 임베딩에 더욱 풍부한 정보를 인코딩하고자 하는 것

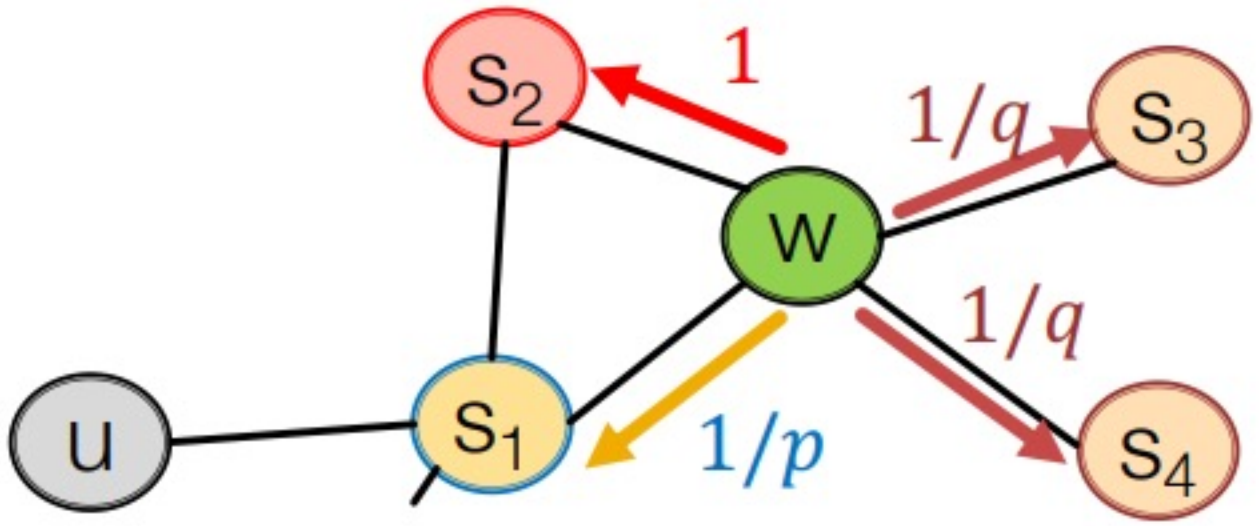
#02 Biased Random Walks

- p : 직전 노드로 돌아갈 확률을 정하는 파라미터
- q : 직전 노드에서 먼 노드로 갈 확률을 정하는 파라미터



<움직일 수 있는 방향>

1. *S1* : 직전 노드로 돌아가기
2. *S2* : 현재 노드와 직전 노드 간의 거리와 동일한 노드로 이동하기
3. *S3* : 현재 노드와 직전 노드 간의 거리보다 먼 노드로 이동하기



랜덤워크지만 그 확률분포가 uniform하지 않고 p와 q에 따라 biased 되기 때문에 biased random walk

- p가 작은 값을 가지게 되면 상대적으로 직전 노드로 돌아갈 확률이 크기 때문에 너비 우선 탐색(BFS)
- q가 작은 값을 가지게 되면 직전 노드와 현재 노드의 거리보다 먼 노드로 이동하기 때문에 깊이 우선 탐색과 비슷한 작동(DFS)

#02 Biased Random Walks

<u>알고리즘</u>

1. 랜덤 워크 확률을 계산
2. R 랜덤워크를 고정된 길이 L에 대해 모든 노드 u에서 실행
3. SGD를 이용하여 node2vec 목적함수를 최적화

<u>장점</u>

1. Linear-time complexity를 가지고 있음.
2. 위의 세 과정 모두 병렬화가 가능해 매우 빠르게 학습이 가능함.

<u>단점</u>

그래프의 크기가 커질수록 임베딩 차원의 수가 커져야 함.

Embedding Entire Graphs

#01 Simple Approaches

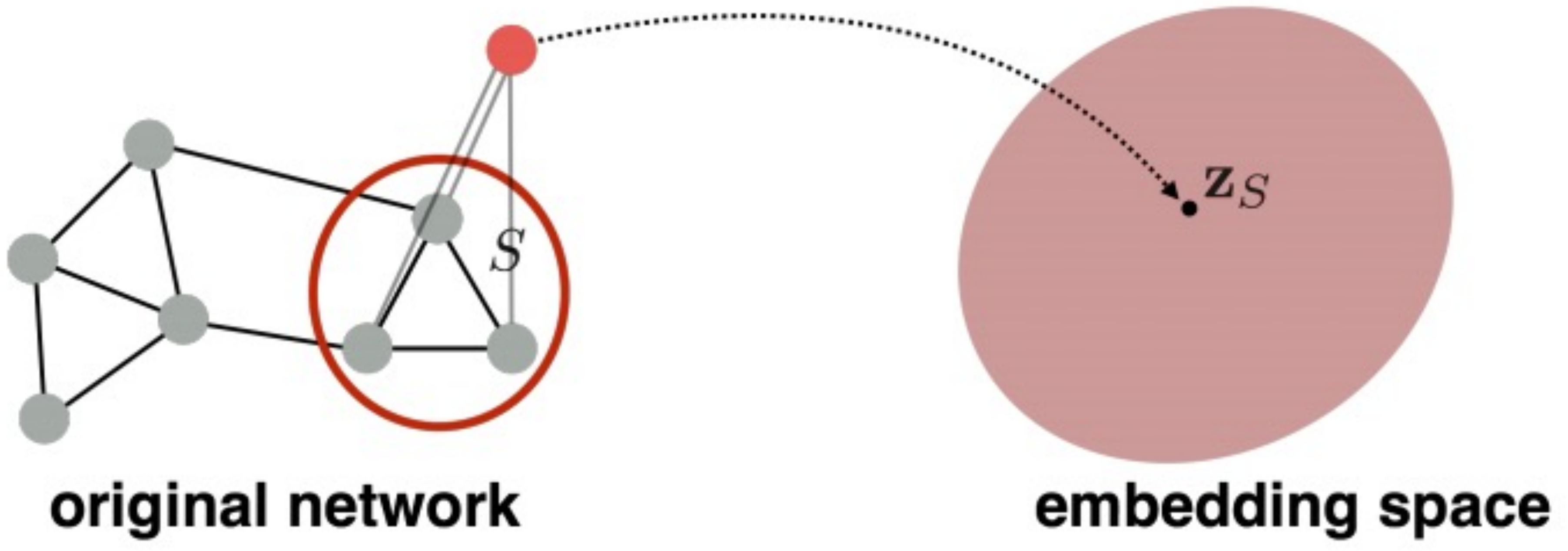
방법1. SUM

앞서 배웠던 일반적인 노드 임베딩 방법을 (sub) graph G에 적용함.
그 결과 해당 (sub) graph 각 노드마다 임베딩 벡터가 생성됩니다.

$$\mathbf{z}_G = \sum_{v \in G} \mathbf{z}_v$$

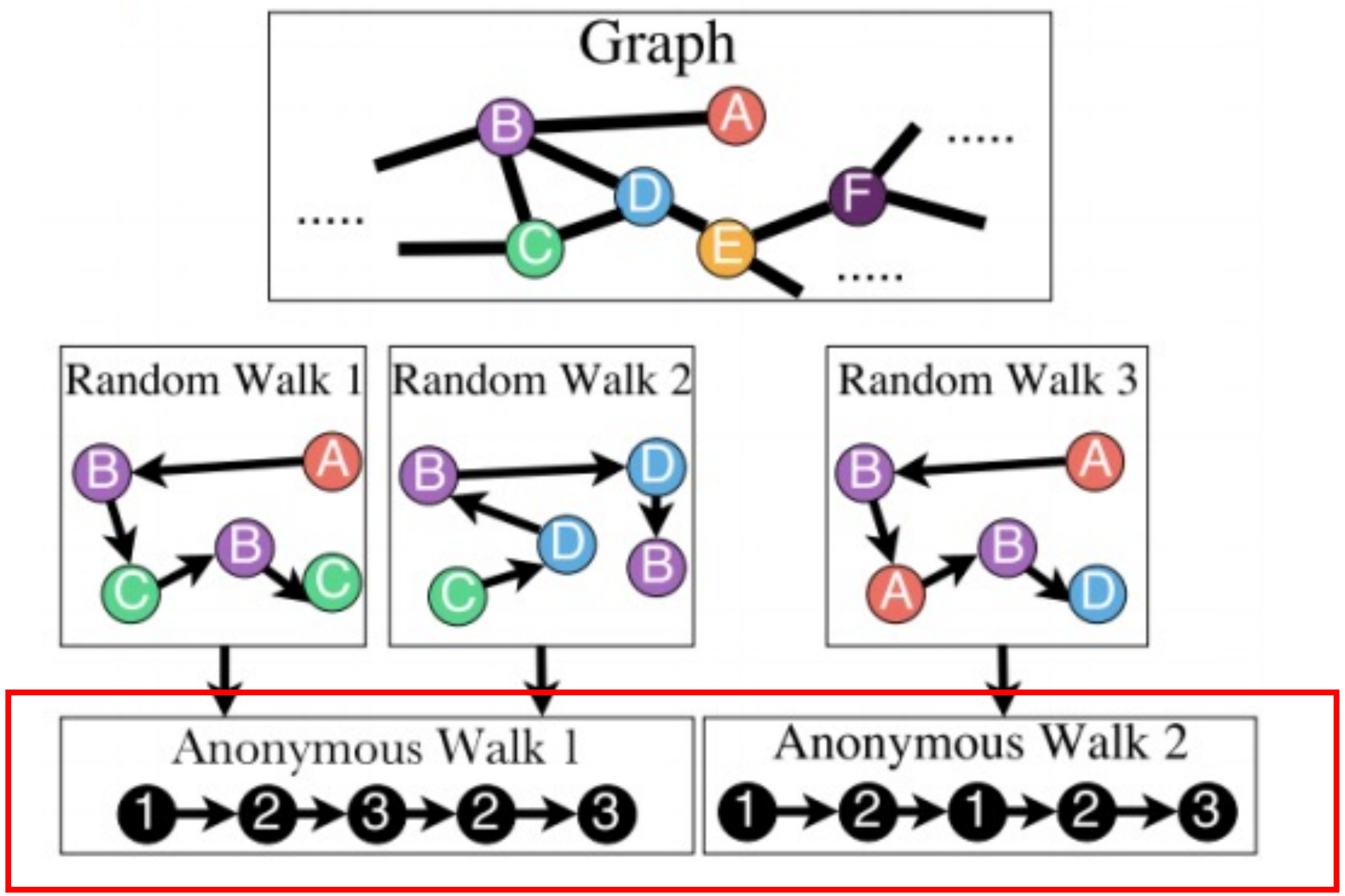
방법2. Virtual Node

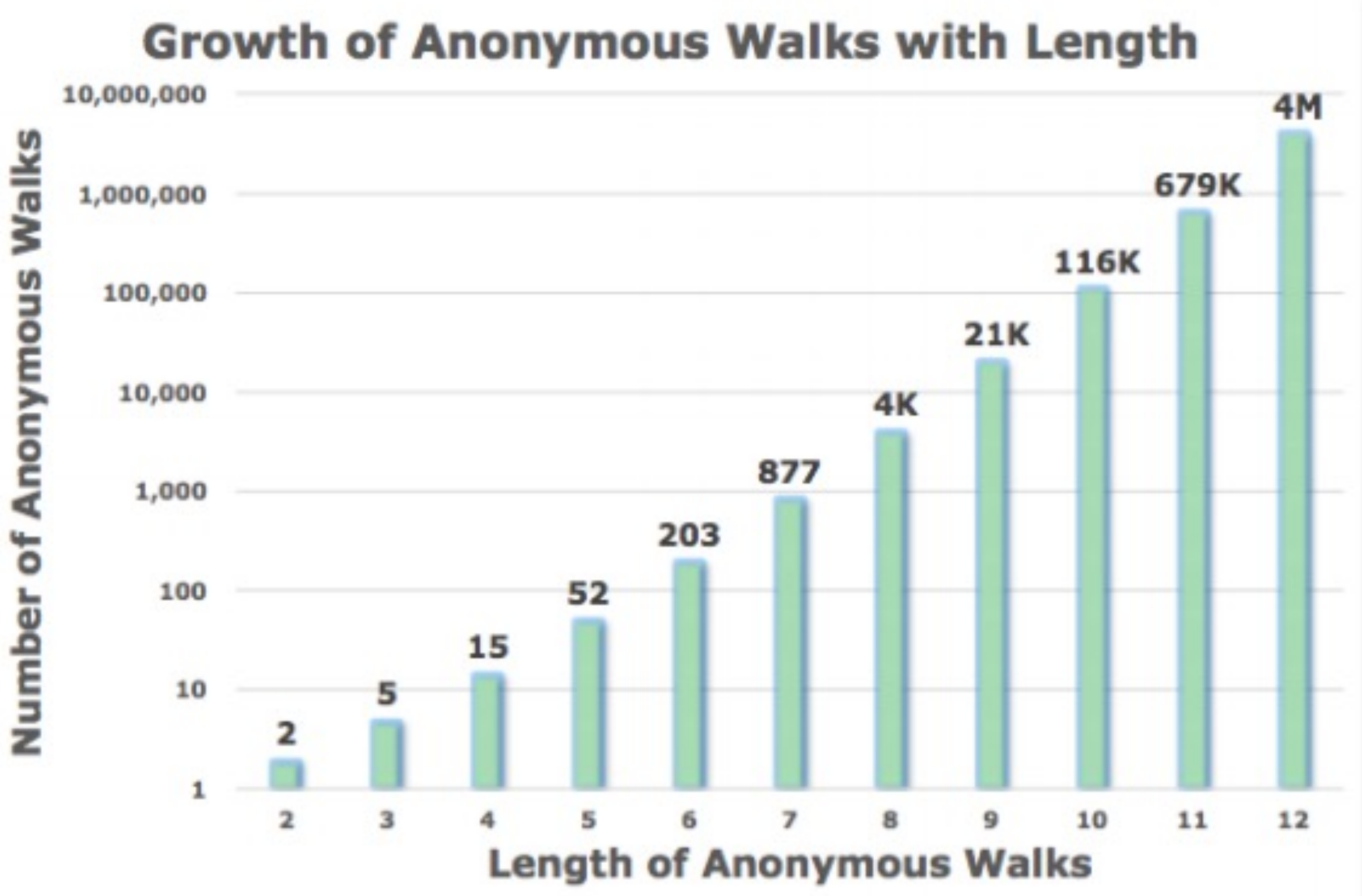
임베딩 하고자 하는 (sub) graph의 모든 노드와 연결된 가상의 노드를 생성함.
이 노드의 임베딩 벡터가 해당 (sub) graph의 임베딩 벡터라 간주하고 노드 임베딩을 실시하게 됨.



#02 Anonymous Walk Embeddings

<u>방법3. Anonymous walks</u>





#02 Anonymous Walk Embeddings

방법3. Anonymous walks

임베딩 벡터는 다음과 같이 생성됨.

1. L 스텝에 대해 anonymous walk w_i를 실행하고 그 빈도를 기록한다.
2. 임베딩 벡터 Z_G의 i번째 element로 w_i의 빈도를 사용한다.

- **For example:**
  - Set $l = 3$
  - Then we can represent the graph as a 5-dim vector
    - Since there are 5 anonymous walks w_i of length 3: 111, 112, 121, 122, 123
  - $\mathbf{z_G}[i]$ = probability of anonymous walk w_i in graph G.

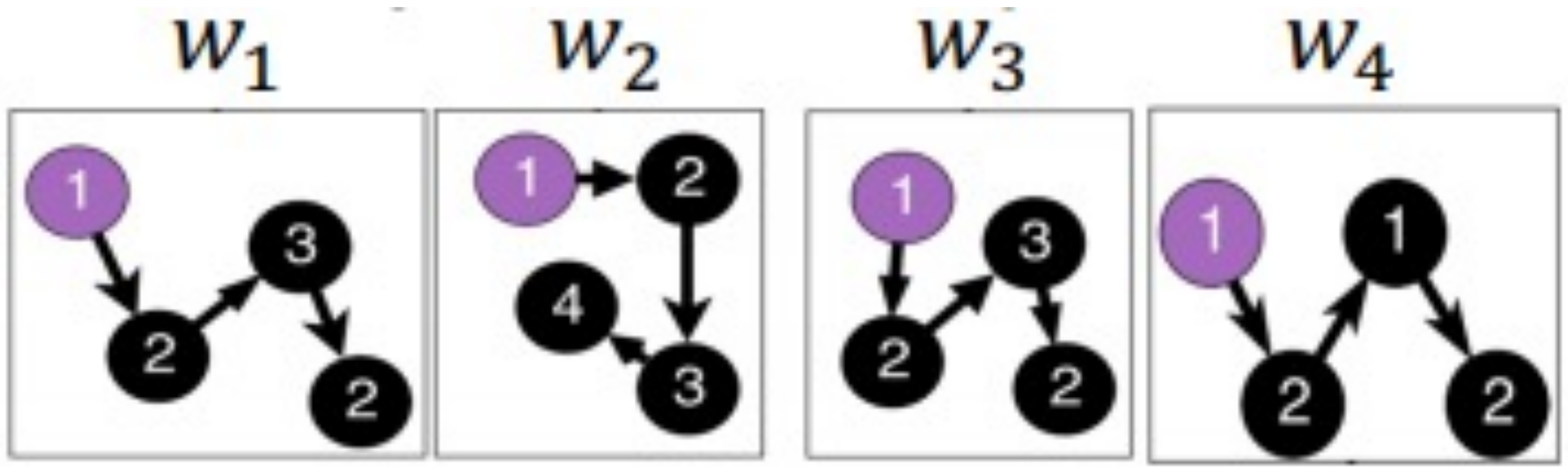
<Sampling anonymous walks>

$$m = \left\lceil \frac{2}{\varepsilon^2}\left(\log(2^{\eta} - 2) - \log(\delta)\right) \right\rceil$$

ϵ: 오차 하한
δ: 오차 상한
η: 길이가 l일 때의 총 anonyomous walk 수

#03 Learn Walk Embeddings

각 walk도 임베딩하여 사용해보자!



window size를 Δ라고 할 때, 동일한 노드에서 출발한 $w_t - \Delta, \ldots, w_t + \Delta$를 이용해 w_t를 예측하자.
여기서 인접한 walk란 아래 그림과 같이 동일한 노드에서 출발한 walk를 의미함.

Objective function:

$$\max_{\mathbf{z}_G} \sum_{t=\Delta+1}^{T} \log P(w_t | w_{t-\Delta}, \ldots, w_{t-1}, \mathbf{z}_G)$$

Where T is the total number of walks

#03 Learn Walk Embeddings

<전체 학습과정>

1. 노드 u, 길이 l에 대해 개별적인 T번의 랜덤 워크를 수행하여 다음과 같은 N_R을 구한다.
이때의 $N_R(u)$은 이전의 이웃 노드 집합이 아니라 동일한 노드 u에서 시작한 random walk의 집합이다.

$$N_R(u) = \{w_1^u, w_2^u \dots w_T^u\}$$

2. η 사이즈의 윈도우를 이용해 해당 랜덤워크를 예측하는 태스크를 수행한다. 이때의 목적함수는 아래와 같다.

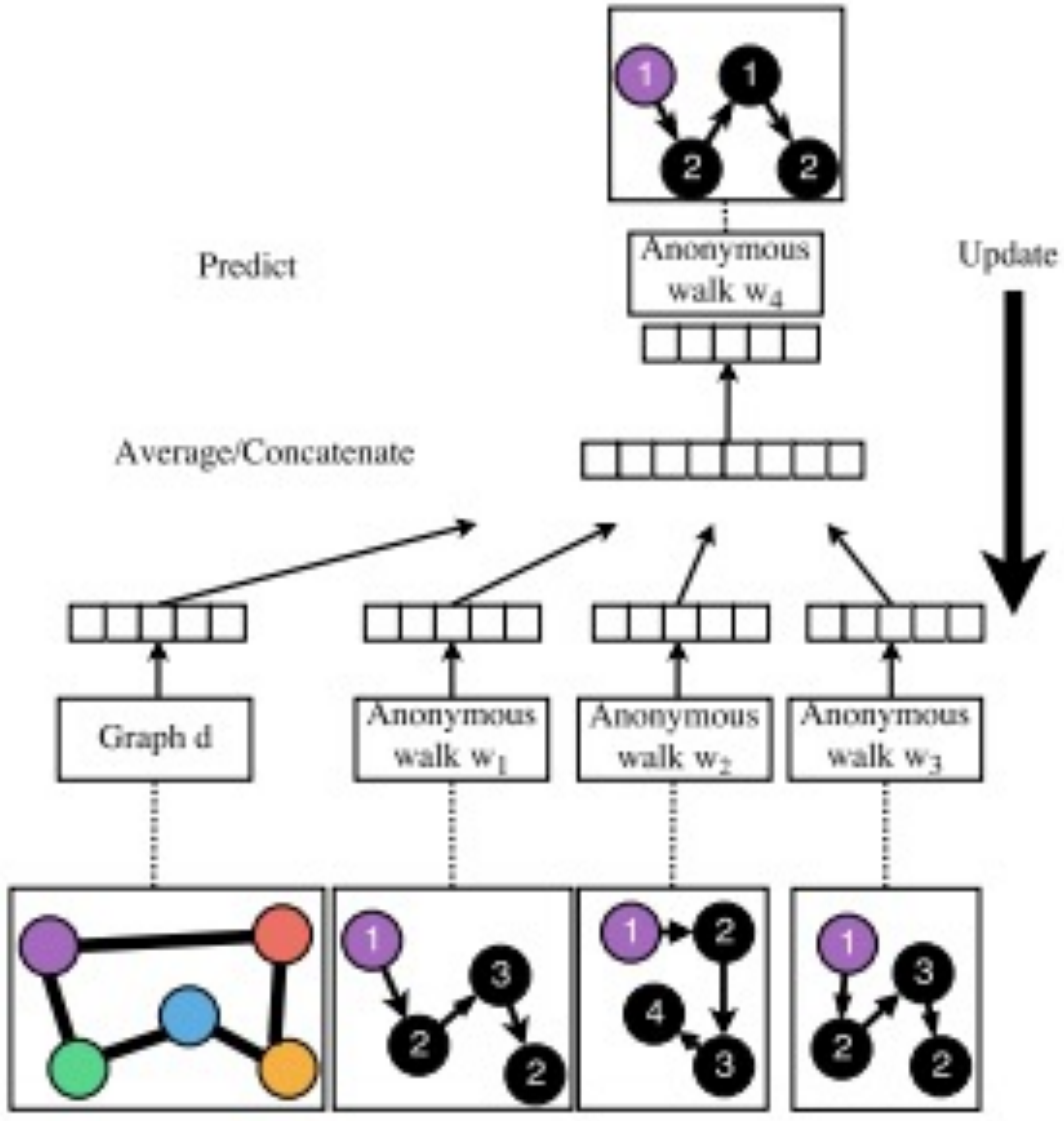
$$\textbf{Objective:} \quad \max_{z_i, z_G} \frac{1}{T} \sum_{t=\Delta}^{T} \log P(w_t | \{w_{t-\Delta}, \dots, w_{t-1}, \boldsymbol{z_G}\})$$

3. 소프트 맥스 함수를 이용해 예측이 진행된다. 이때 랜덤워크 벡터들은 평균을 내어 그래프 벡터와 concat되어 입력값으로 사용된다.

$$P(w_t | \{w_{t-\Delta}, \dots, w_{t-1}, \boldsymbol{z_G}\}) = \frac{\exp(y(w_t))}{\sum_{i=1}^{\eta} \exp(y(w_i))}$$

#03 Learn Walk Embeddings

<전체 학습과정>



Z_G를 이용해 실제 태스크를 수행하는 방법:

- **Option1**: Inner product Kernel $z_{G_1}^T z_{G_2}$ (Lecture 2)
- **Option2**: Use a neural network that takes z_G as input to classify G.

#04 How to Use Embeddings

임베딩 벡터 Z_i를 사용하는 방법:

1. 클러스터링, 소셜 네트워크 군집화 z_i를 하나의 점으로 간주하고 군집화를 실행
2. 노드 분류: z_i를 이용해 i 노드의 레이블을 예측
3. 링크 예측 : (i, j)의 엣지를 두 노드의 임베딩 벡터 (z_i, z_j)를 이용해 예측할 수 있음

- Concatenate: $f(z_i, z_j) = g([z_i, z_j])$
- Hadamard: $f(z_i, z_j) = g(z_i * z_j)$ (per coordinate product)
- Sum/Avg: $f(z_i, z_j) = g(z_i + z_j)$
- Distance: $f(z_i, z_j) = g(||z_i - z_j||_2)$

4. 그래프 분류 : 노드 임베딩을 결합하여 그래프 임베딩을 생성하거나 anonymous random walks를 이용해 만든 그래프 임베딩을 통해 분류 태스크를 수행할 수 있음.

THANK YOU