

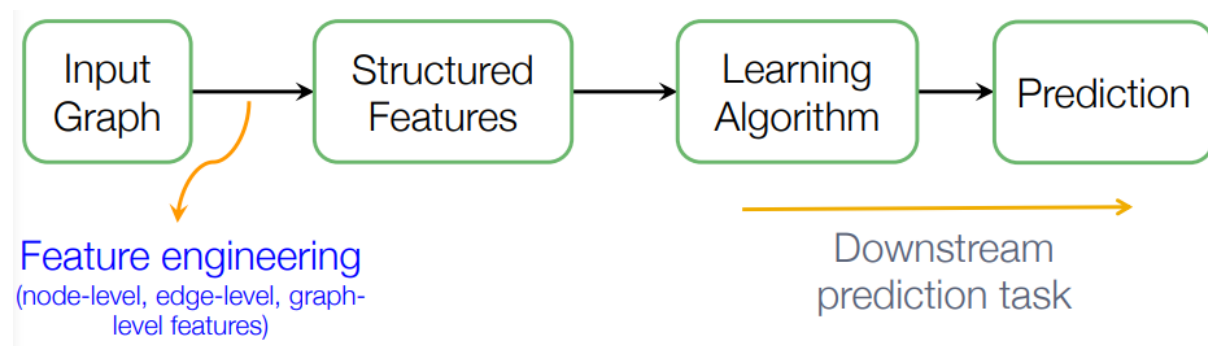
3. Node Embeddings

<https://www.youtube.com/watch?v=rMq21iY61SE>

Contents

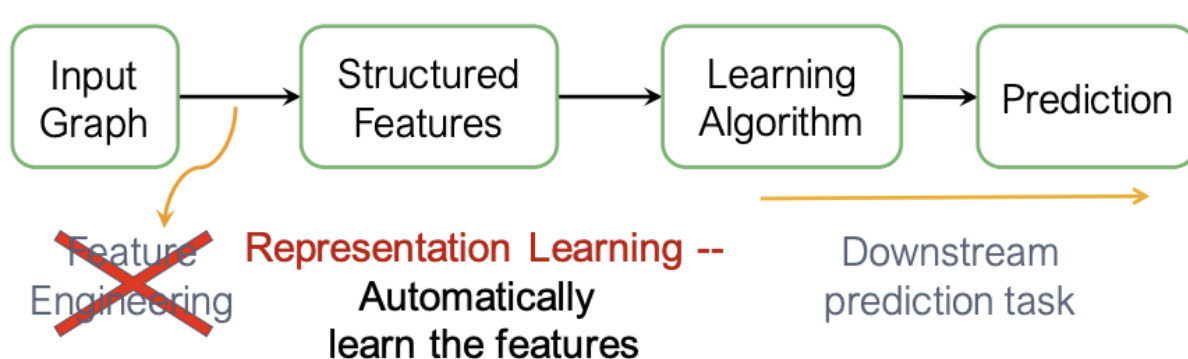
- Recap : Traditional ML for Graphs
- Node Embeddings : Encoder and Decoder
- Random Walk Approaches for Node Embeddings
- Embeddings Entire Graphs

0. Traditional ML for Graphs



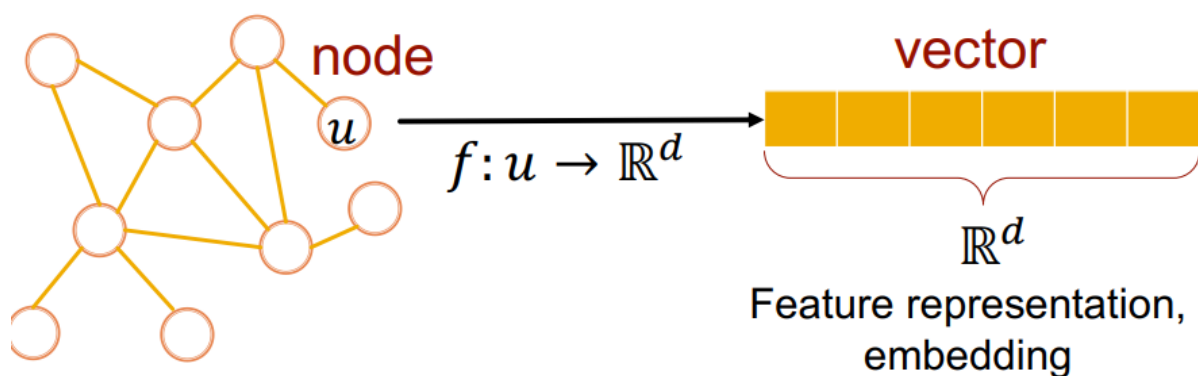
머신러닝 기법들은 그래프를 각각의 도메인과 태스크에 맞게 feature engineering 하여 노드, 링크, 그래프 레벨의 변수들을 생성했다. 그리고 이렇게 만들어진 변수를 이용해 또다시 각각의 태스크와 도메인에 맞게 모델을 정하고, 튜닝하여 예측하였다.

하지만 이렇게 진행되다 보니, 각 태스크와 도메인마다 새로 feature engineering이 진행되어야 했다. 앞에서 잠깐 배웠듯이, 그 과정은 간단하지도, 시간이 적게 들지도 않는다. 그래서 이제는 representation learning을 통해 우리가 배울 모델들이 적절하게 변수들을 뽑아낼 수 있도록 학습시킬 것이다.



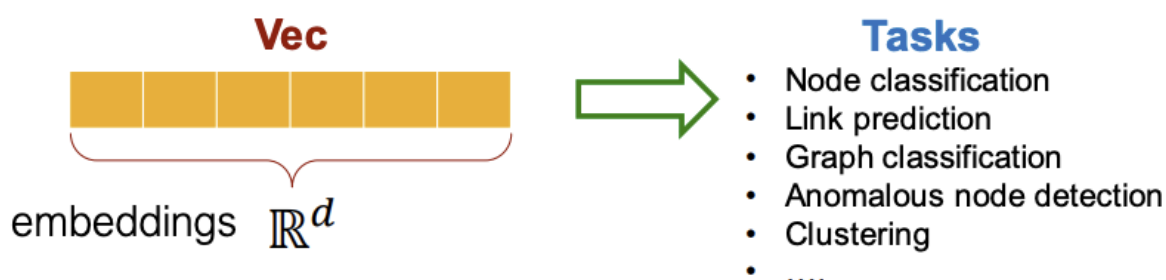
Graph Representation Learning

- Goal : Efficient task-independent feature learning for machine learning with graphs!

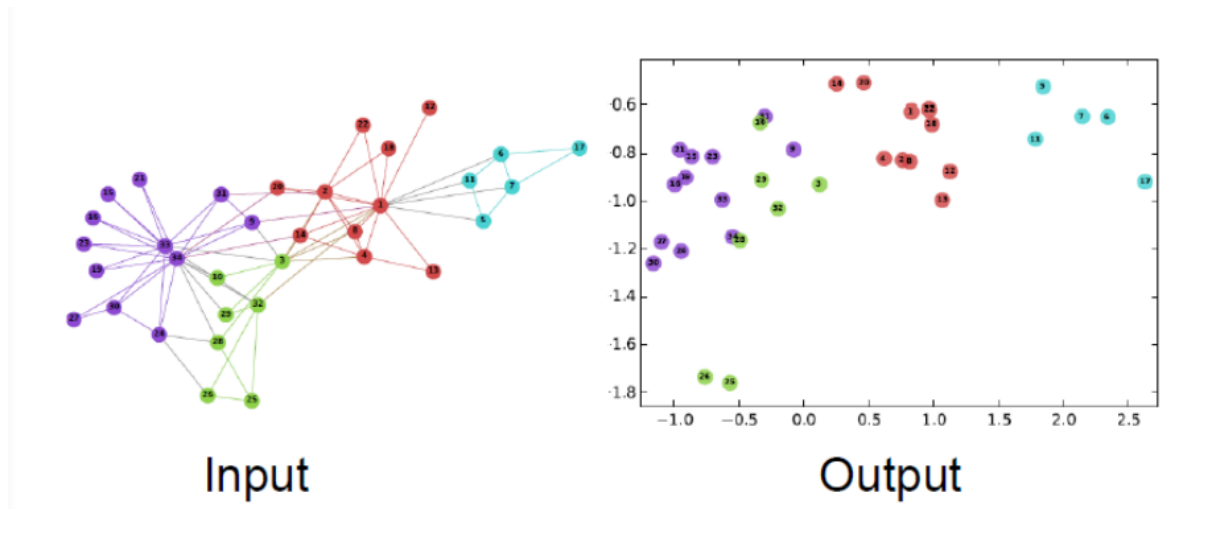


Why Embedding?

- Task : Map nodes into an embedding space
 - Similarity of embeddings between nodes indicates their similarity in the network.
 - For example : Both nodes are close to each other (connected by an edge)
 - Encode network information
 - Potentially used for many downstream predictions



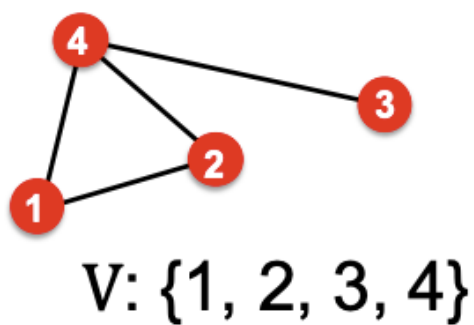
- Example Node Embedding : 2D embedding of nodes of the Zachary's Karate Club network:



1. Node Embeddings : Encoder and Decoder

SetUp

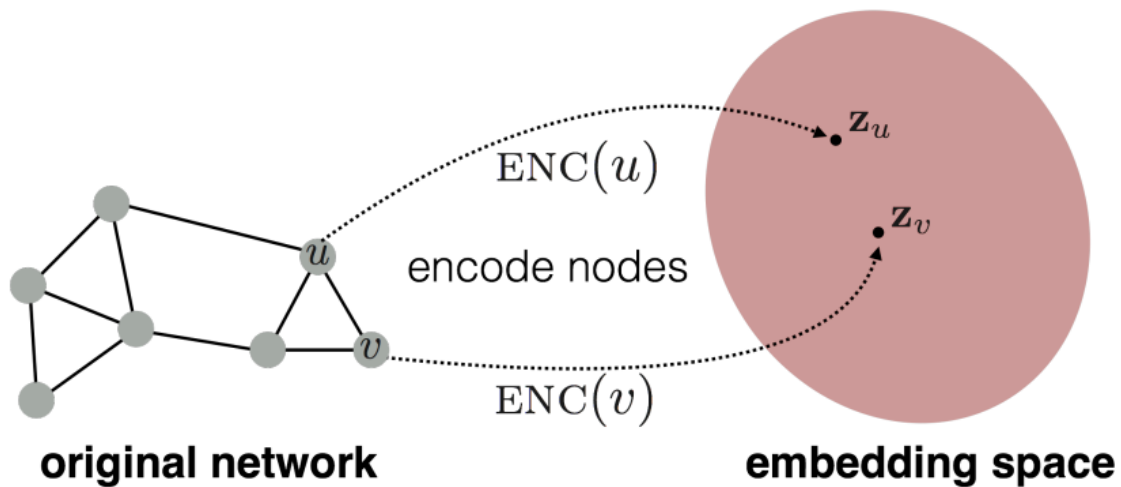
- Assume we have a graph G :
 - V is the vertex set.
 - A is the adjacency matrix(assume binary).
 - For simplicity : No node features or extra information is used



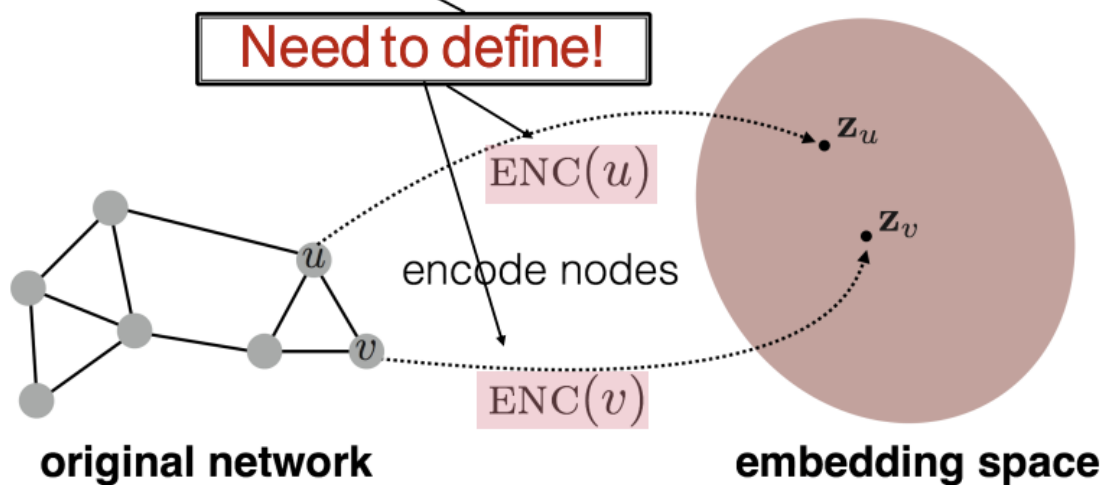
$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Embedding Nodes

- Goal is to encode nodes so that **similarity in the embedding space**(e.g., dot product) approximates **similarity in the graph**



Goal: $\text{similarity}(u, v)$ in the original network $\approx \mathbf{z}_v^T \mathbf{z}_u$ Similarity of the embedding



Learning Node Embeddings

1. **Encoder** maps from nodes to embeddings
2. **Define a node similarity function**(i.e., a measure of similarity in the original network)
3. **Decoder DEC** maps from embeddings to the similarity score
4. **Optimize the parameters of the encoder so that:**

Two Key Components

- **Encoder:** maps each node to a low-dimensional vector

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

node in the input graph

d -dimensional embedding

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

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

Similarity of u and v in the original network

dot product between node embeddings

Decoder

“Shallow” Encoding

가장 단순한 인코딩 방식은 인코더가 단순히 embedding-lookup하는 것이다.

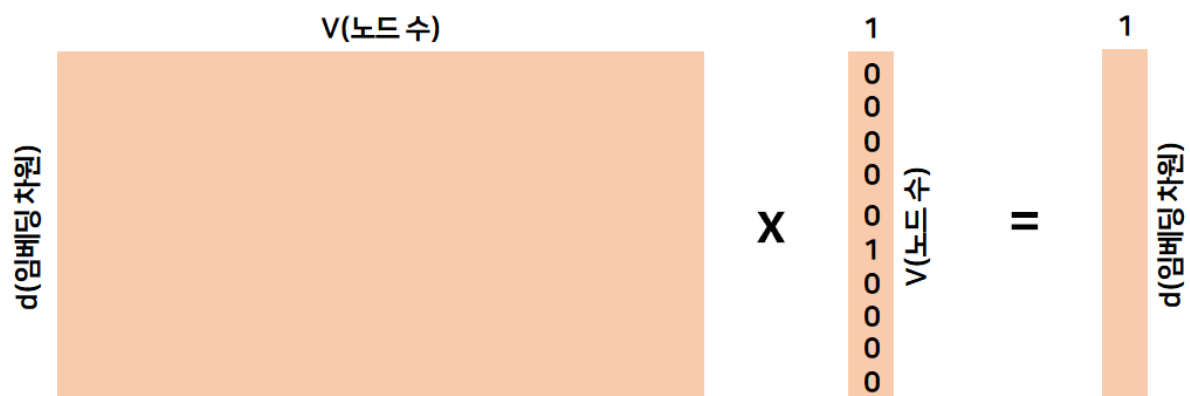
embedding-lookup이란 룩업 테이블에서 입력으로 주어진 인덱스의 열만 출력하는 것을 의미한다.

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

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn / optimize]

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node v

Encoder is just an embedding-lookup



Each node is assigned a unique embedding vector(i.e., we directly optimize the embedding of each node)

하지만 노드 수가 많아지게 될 경우 가지고 있어야 하는 룩업 테이블이 무척 커지게 되는 단점이 있다.

Many methods : **DeepWalk, node2vec**

Framework Summary

■ Encoder + Decoder Framework

- Shallow encoder: embedding lookup
- Parameters to optimize: \mathbf{Z} which contains node embeddings \mathbf{z}_u for all nodes $u \in V$
- We will cover deep encoders (GNNs) in Lecture 6
- **Decoder:** based on node similarity.
- **Objective:** maximize $\mathbf{z}_v^T \mathbf{z}_u$ for node pairs (u, v) that are **similar**

How to Define Node Similarity?


- Key choice of methods is **how they define node similarity**.
- Should two nodes have a similar embedding if they...
 - are linked?
 - share neighbors?
 - have similar “structural roles”?
- We will now learn **node similarity definition** that uses **random walks**, and how to optimize embeddings for such a similarity measure.

Note on Node Embeddings

- This is **unsupervised/self-supervised** way of learning node embeddings.
 - We are **not** utilizing node labels
 - We are **not** utilizing node features
 - The goal is to directly estimate a set of coordinates(i.e., the embedding) of a node so that some aspect of the network structure (captured by DEC) is preserved.
 - These embeddings are **task independent**
 - They are not trained for a specific task but can be used for any task.
-

2. Random Walk Approaches for Node Embeddings

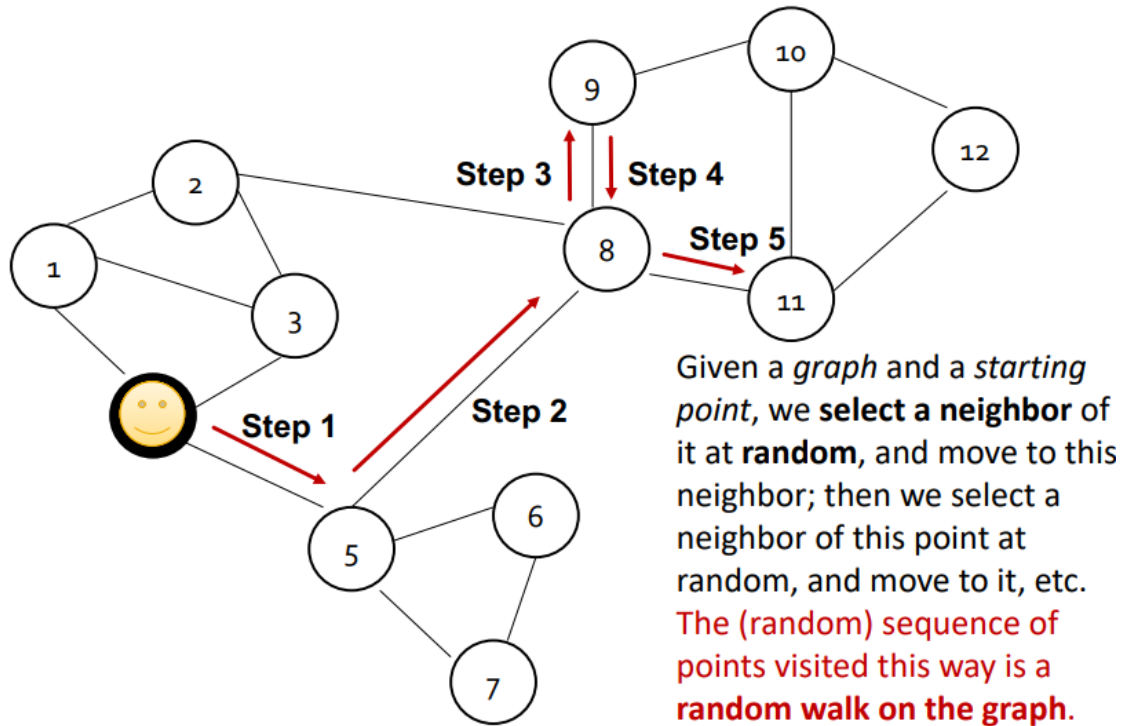
Notation

- **Vector** \mathbf{z}_u :
 - The embedding of node u (what we aim to find).
 - **Probability** $P(v | \mathbf{z}_u)$:  Our model prediction based on \mathbf{z}_u
 - The **(predicted) probability** of visiting node v on random walks starting from node u .
-

Non-linear functions used to produce predicted probabilities

- **Softmax** function:
 - Turns vector of K real values (model predictions) into K probabilities that sum to 1: $\sigma(\mathbf{z})[i] = \frac{e^{z[i]}}{\sum_{j=1}^K e^{z[j]}}$
- **Sigmoid** function:
 - S-shaped function that turns real values into the range of (0, 1).
Written as $S(x) = \frac{1}{1+e^{-x}}$.

Random Walk

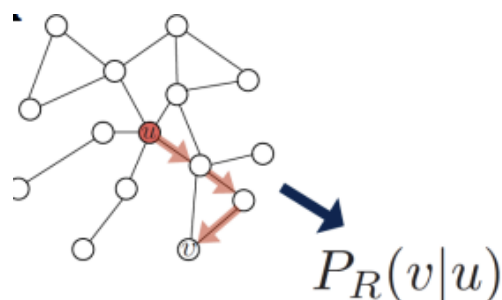


위 그림과 같이 그래프가 주어지고 특정 노드에서 시작하여 한번에 하나씩 이웃 노드로 랜덤하게 옮겨간다고 할 때, 연속적으로 옮겨가는 노드 시퀀스를 랜덤워크라고 한다. 또한 옮기는 횟수는 고정되거나 특정 전략을 통해 이뤄지는데 이를 R 로 표기한다. 즉, 출발 노드에서 움직이면서 그 기록을 남긴 것을 랜덤워크라고 한다. 위의 랜덤워크는 [5, 8, 9, 8, 11]이 될 것이다.

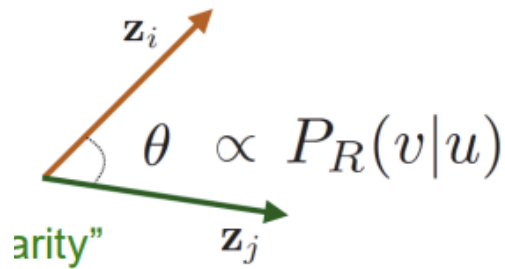
그래프에서 각 노드에 대해 랜덤워크를 기록하게 되면, 총 V 개의 랜덤워크가 있게 된다. 만약 노드 u 와 노드 v 가 전체 랜덤워크에서 같이 등장하는 확률이 높다면, 두 노드는 서로 가까이 있다는 의미가 될 것이다. 두 노드 사이에 엣지가 많거나 경로가 짧아서 자주 같이 등장한 것이기 때문이다. 이러면 랜덤워크를 이용해 그래프에서 두 노드 (u, v) 의 유사도를 측정할 수 있게 된 것이라고 할 수 있을 것이다. 이전에 그래프에서 근처에 있는 노드를 유사도가 높다고 생각하자고 했고, 근처에 있는 노드면 확률이 높을 것이기 때문이다.

Random-Walk Embeddings

1. Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R



2. Optimize embeddings to encode these random walk statistics:



Similarity in embedding space(Here : dot product= $\cos(\theta)$) encodes random walk “similarity”

Why Random Walks?

1. Expressivity : 확률로 표현되기 때문에 두 노드의 경로가 짧은 경우는 물론이고 경로가 긴 경우에도 이웃 정보를 잡아낼 수 있다. **Idea : if random walk starting from node u visits v with high probability, u and v are similar(high-order multi-hop information)**
2. Efficiency : Do not need to consider all node pairs when training; **only need to consider pairs that co-occur on random walks**

Unsupervised Feature Learning

- **Intuition** : Find embedding of nodes in d -dimensional space that **preserves similarity**
 - **Idea** : Learn node embedding such that **nearby** nodes are close together in the network
 - **Given a node u , how do we define nearby nodes?**
- $N_R(u)$... neighbourhood of u obtained by some **random walk strategy R**

Feature Learning as Optimization

- Given $G = (V, E)$,
- Our goal is to learn a mapping $f: u \rightarrow \mathbb{R}^d$:
 $f(u) = \mathbf{z}_u$

- Log-likelihood objective:

$$\max_f \sum_{u \in V} \log P(N_R(u) | \mathbf{z}_u)$$

- $N_R(u)$ is the neighborhood of node u by strategy R
- Given node u , we want to learn feature representations that are predictive of the nodes in its random walk neighborhood $N_R(u)$.

How should we randomly walk?

1. 짧은 거리의 랜덤워크를 고정하여 R 에 따라 각 노드마다 진행한다.
2. 각 노드 u 마다 $N_R(u)$ 를 수집한다. 이때 $N_R(u)$ 는 일반적인 집합과 다르게 중복이 허용된다. 랜덤워크시 반복하여 특정노드를 반복할 수도 있기 때문이다.
3. 다음 식에 따라 임베딩을 최적화한다.

$$\max_f \sum_{u \in V} \log P(N_R(u) | \mathbf{z}_u)$$

손실함수를 구하면 다음과 같이 될 것이다.

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

이때 확률은 소프트맥스 함수를 통해 구하게 된다.

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

즉, 이웃노드가 등장할 확률이 높아지도록 임베딩 벡터가 최적화될 것이다. 즉, 이웃 노드는 내적값이 커지고, 이웃하지 않은 노드 간에는 내적값이 작아지게 된다.

하지만 이와 같은 손실함수를 사용하는 것은 문제가 있다.

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

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

위와 같이 전체 노드가 두번 중첩되어 시간복잡도가 너무 커지게 된다.

Negative Sampling

But doing this naively is too expensive!


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

The normalization term from the softmax is the culprit... can we approximate it?

손실함수 식을 살펴보면 맨 앞의 v 는 해결할 수 없다. 모든 노드에 대해 손실 함수가 계산되긴 해야한다. 하지만 두번째 v 는 수정할 수 있다. 두번째 v 는 정규화를 위한 항이기 때문이다. 이를 근사하여 계산량을 줄일 수 있다.

소프트 맥스에서 정규화 항을 근사하는 방법이 네거티브 샘플링이다.

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

random distribution
over nodes


※ Why is the approximation valid?

Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approximate maximizes the log probability of softmax.

New formulation corresponds to using a logistic regression(sigmoid func) to distinguish the target node v from nodes n_i sampled from background distribution P_v

- Sample K negative nodes each with prob proportional to its degree
- Higher k gives more robust estimates
- Higher k corresponds to higher bias on negative events (주로 5~20사이로~~)

※ Can negative sample be any node or only the nodes not on the walk?

People often use any nodes(for efficiency). However, **the most “correct” way is to use nodes not on the walk.**

네거티브 샘플링은 위와 같은 방법으로 이뤄진다. 전체 노드에 대해 내적값을 구하는 대신 노드 **u와 이웃하지 않은 노드 중 k개를 샘플링**하여 사용하게 된다. 이 때 앞의 시그모이드 항은 이웃 노드간 계산되는 이웃노드일 확률로 최대화 된다. 뒤에 나오는 시그모이드 항은 u와 이웃하지 않은 노드와 계산되는 이웃노드일 확률로 최소화하게 된다.

즉, 이웃 노드 간의 확률은 높아지고, 이웃하지 않은 노드 간 확률은 낮아지도록 최적화하게 된다.

Stochastic Gradient Descent

- After we obtained the objective function, how do we optimize it?

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

- **Gradient Descent** : a simple way to minimize L:

- Initialize \mathbf{z}_u at some randomized value for all nodes u .
- Iterate until convergence:

- For all u , compute the derivative $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_u}$.

η : learning rate

- For all u , make a step in reverse direction of derivative: $\mathbf{z}_u \leftarrow \mathbf{z}_u - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{z}_u}$.

- **Stochastic Gradient Descent** : Instead of evaluating gradients over all examples, evaluate it for each **individual** training example.

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

Random walks : Summary

1. Run **short fixed-length** random walks starting from each node on the graph.
2. For each node u collect $N_{R(u)}$, the multiset of nodes visited on random walks starting from u .
3. Optimize embeddings using SGD

How should we randomly walk?

- So far we have described how to optimize embeddings given a random walk strategy R
- **What strategies should we use to run these random walks?**
 - Simplest idea : **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk from Perozzi et al., 2013)
 - The issue is that such notion of similarity is too constrained
- **How can we generalize this?**

Overview of node2vec

- **Goal** : Embed nodes with similar network neighborhoods close in the feature space.
- We frame this goal as a maximum likelihood optimization problem, independent to the downstream prediction task.
- **Key observation** : Flexible notion of network neighborhood $N_{R(u)}$ of node u leads to rich node embeddings
- Develop biased 2nd order random walk R to generate network neighborhood $N_{R(u)}$ of node u
- paper :

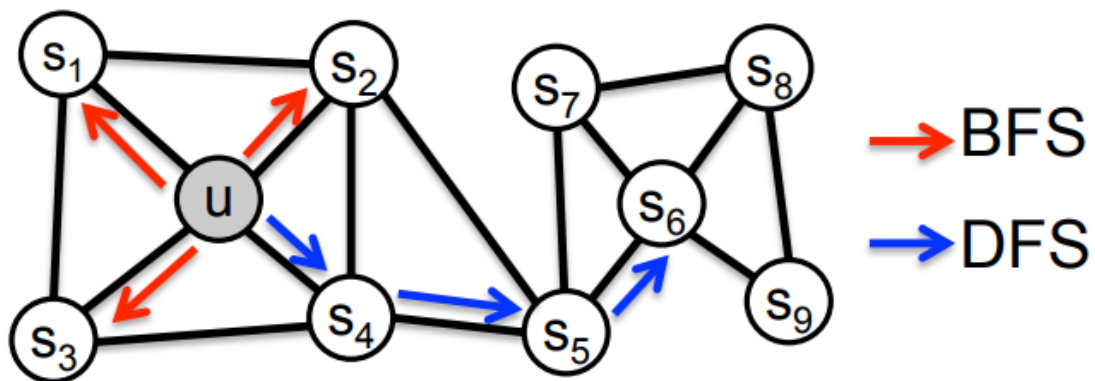
node2vec : **Biased** Walks

Idea: use flexible, biased random walks that can trade off between **local** and **global** views of the network (Grover and Leskovec, 2016).

앞서 DeepWalk를 설명하면서 strategy R 이 어떻게 구성되는지 자세한 설명이 없었다. 단순히 각 시점마다 uniform dist를 통해 이동한다고 했다. 하지만 분명히 이보다 효과적으로 그래프의 정보를 임베딩할 수 있는 전략이 있다. node2vec는 그 전략에 관한 설명이 주를 이룬다.

node2vec이 deepwalk와 달라지는 점은 그래프에서 이웃 노드 집합 $N_{R(u)}$ 를 찾을 때 비교적 유연하게 대처할 수 있는 전략 R 을 이용하여 노드 임베딩에 더욱 풍부한 정보를 인코딩하고자 하는 것이다.

Two classic strategies to define a neighborhood $N_{R(u)}$ of a given node u :



Walk of length 3 ($N_R(u)$ of size 3):

$$N_{BFS}(u) = \{s_1, s_2, s_3\} \quad \text{Local microscopic view}$$

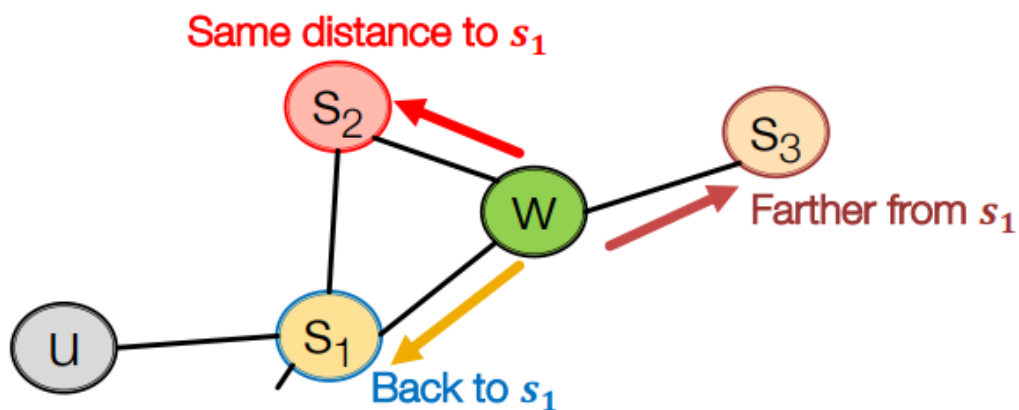
$$N_{DFS}(u) = \{s_4, s_5, s_6\} \quad \text{Global macroscopic view}$$

노드 u 에서 시작하여 랜덤워크를 한다고 할 때 너비 우선 탐색 방식으로 진행한다면 노드 u 주변의 정보는 풍부하게 담을 수 있지만, 전체적인 구조를 담지 못한다. 즉, 위 그림에서 오른쪽의 부분 그래프에 대한 정보를 u 는 담지 못하게 된다. 반대로 깊이 우선 탐색 방식으로 진행하면 전체적인 구조에 대한 정보를 담을 수는 있지만 노드 u 주변의 정보가 부실해지게 된다. 결국 **노드 주변을 얼마나 탐색하는지에 따라 local과 global 정보가 트레이드 오프 관계에 놓이는 것이다. 그리고 node2vec은 이를 조절하여 탐색하는 전략 R 을 세우는 것이다.**

Interpolating BFS and DFS

Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$

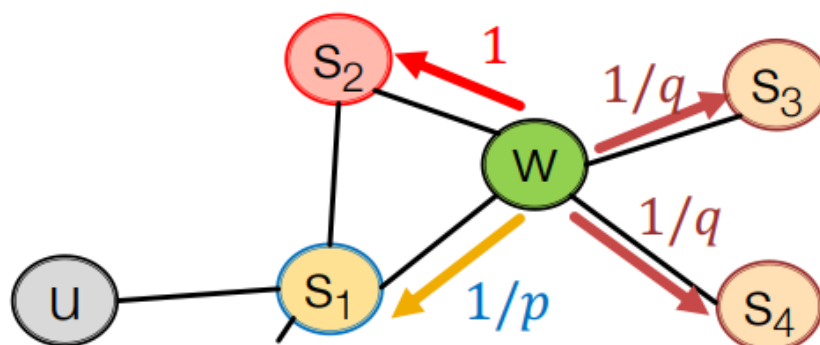
- Two parameters:
 - Return parameter p :
 - Return back to the previous node
 - In-out parameter q :
 - Moving outwards(DFS) vs. inwards(BFS)
 - Intuitively, q is the “ratio” of BFS vs. DFS



위 그림에서 랜덤워크는 s_1 에서 w 로 이동한 상태라고 할 때, 움직일 수 있는 방향은 세 가지 이다.

1. s_1 : 직전 노드로 돌아가기
2. s_2 : 현재 노드와 직전 노드 간의 거리와 동일한 노드로 이동하기
3. s_3 : 현재 노드와 직전 노드간의 거리보다 먼 노드로 이동하기

그리고 각 노드로 이동할 확률 분포는 p 와 q 를 통해 정해지게 된다.



p 와 q 는 위 그림처럼 정규화되지 않은 확률로 작동하게 된다. 구체적으로 w 에서 각 노드까지 거리를 계산하고, 거리마다 위의 세가지 방향 중 어떤 방향인지 분류하고, p 와 q 를 이용한 확률을 정규화하고, 그 확률에 따라 이동하는 것이다. 이때 랜덤워크지만 그 확률분포가 uniform하지 않고 p 와 q 에 따라 biased 되기 때문에 biased random walk라고 한다.

이때 만약 p 가 작은 값을 가지게 되면 상대적으로 직전 노드로 돌아갈 확률이 크기 때문에 너비 우선 탐색과 비슷한 작동이 된다. 만약 q 가 작은 값을 가지게 되면 직전 노드와 현재 노드의 거리보다 먼 노드로 이동하기 때문에 깊이 우선 탐색과 비슷한 작동이 되게 된다.

node2vec algorithm

1. Compute random walk probabilities
2. Simulate r random walks of length l starting from each node u
3. Optimize the node2vec objective using SGD

node2vec의 장점은 선형적인 복잡도를 가지고 있다는 점이다. 또한 위의 세 과정 모두 병렬화가 가능해 매우 빠르게 학습이 가능하다.

다만, 그래프의 크기가 커질수록 임베딩 차원의 수가 커져야하는 단점이 있다.

Other Random Walk Ideas


- **Different kinds of biased random walks:**
 - Based on node attributes ([Dong et al., 2017](#)).
 - Based on learned weights ([Abu-El-Haija et al., 2017](#))
- **Alternative optimization schemes:**
 - Directly optimize based on 1-hop and 2-hop random walk probabilities (as in [LINE from Tang et al. 2015](#)).
- **Network preprocessing techniques:**
 - Run random walks on modified versions of the original network (e.g., [Ribeiro et al. 2017's struct2vec](#), [Chen et al. 2016's HARP](#)).

Sumarry so far

- **Core idea** : Embed nodes so that distances in embedding space reflect node similarities in the original network.
- Different notions of node similarity:
 - Naïve : similar if two nodes are connected
 - Neighborhood overlap(covered in Lecture 2)
 - Random walk approaches(covered this lecture)
- **So what method should I use?**
- No one method wins in all cases...
 - E.g., node2vec performs better on node classification while alternative methods perform better on link prediction

Graph Embedding Techniques, Applications, and Performance: A Survey

Graphs, such as social networks, word co-occurrence networks, and communication networks, occur naturally in various real-world applications. Analyzing them yields insight into the structure of society, language, and different

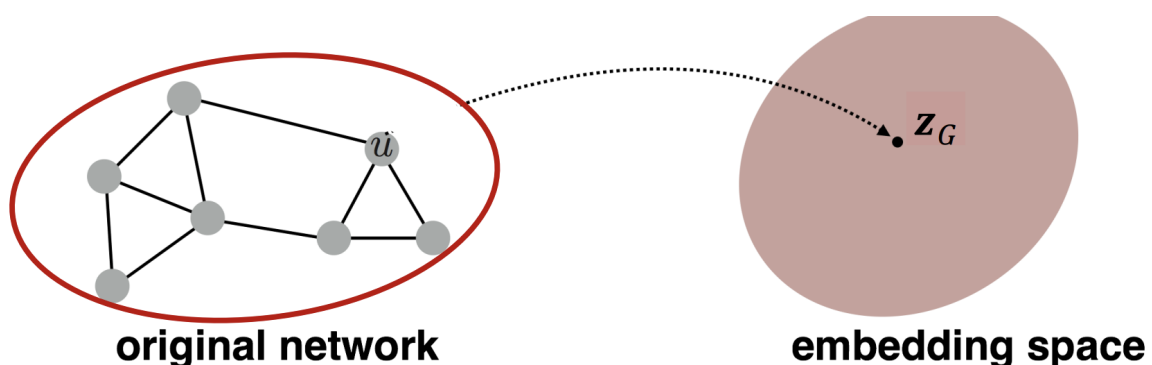
 <https://arxiv.org/abs/1705.02801>

arXiv

- Random walk approaches are generally more efficient.
- **In general** : Must choose definition of node similarity that matches your application.

3. Embedding Entire Graphs

- Goal : Want to embed a subgraph or an entire graph G . Graph embedding : z_G



- Tasks:
 - Classifying toxic vs. non-toxic molecules
 - Identifying anomalous graphs

Approach 1

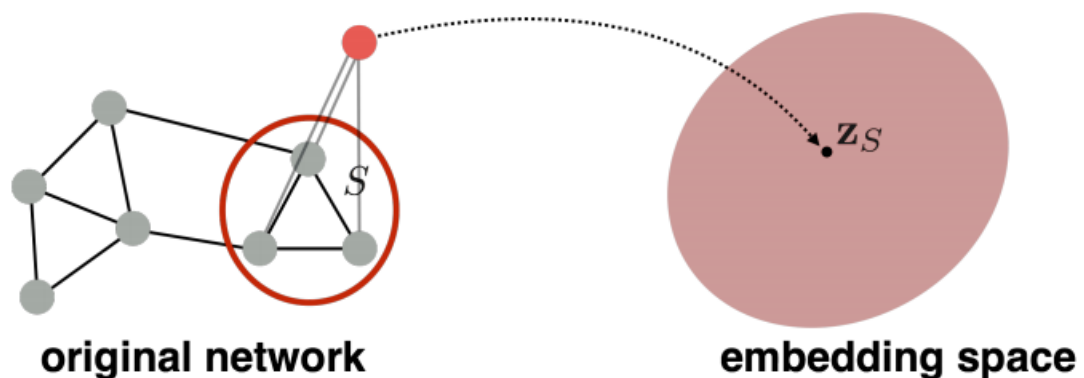
Simple(but effective) approach 1:

- Run a standard graph embedding technique on the (sub)graph G .
- Then just sum(or average) the node embeddings in the (sub)graph G .

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

Approach 2

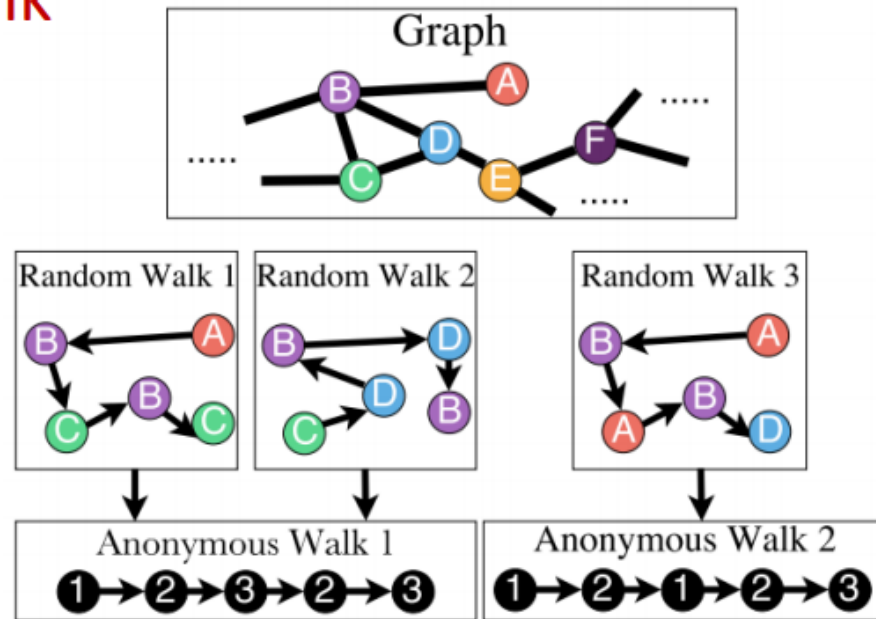
Introduce a “**virtual node**” to represent the (sub)graph and run a standard graph embedding technique



Approach 3 : Anonymous Walk Embeddings

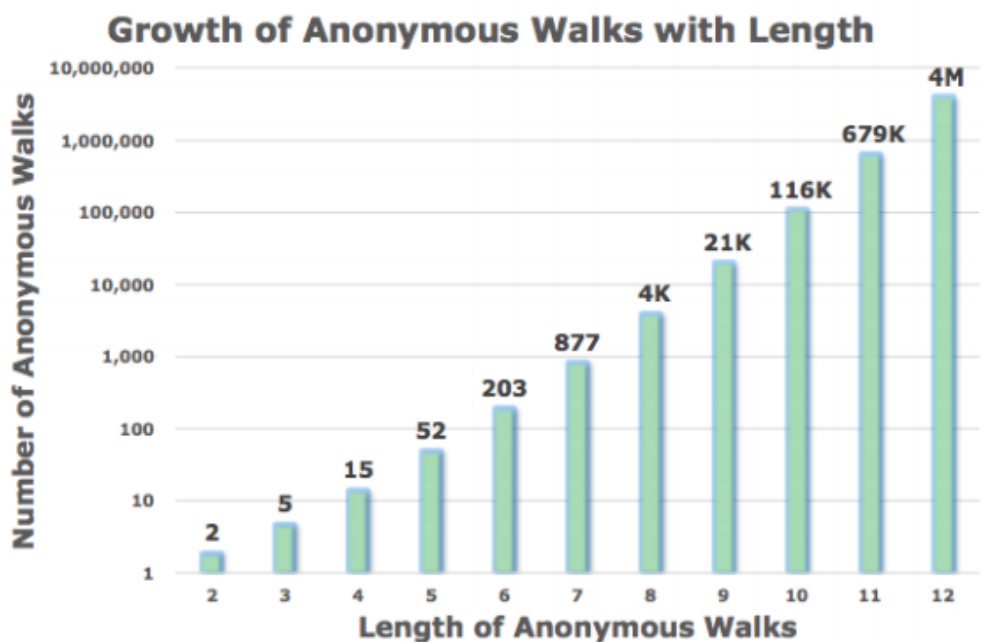
States in anonymous walks correspond to the index of the first time we visited the node in a random walk

walk



이는 위 그림과 같이 정해진 길이의 랜덤워크에 대해 각 노드의 인덱스를 지워 노드를 익명으로 만든다. 대신 랜덤워크에서 중복으로 등장한 노드에 대해서는 처음 등장시 매겼던 인덱스를 부여하게 된다. 그 결과 위 그림과 같이 Random Walk 1과 2는 각기 다른 노드를 지났지만, 노드의 인덱스를 지우면 동일한 anonymous walk를 가지게 된다. 그래프의 입장에서 각 노드의 정보는 활용할 수 없고, 그 구조만 파악 가능하기 때문에 이와 같이 노드의 정보(인덱스)를 지우고 처리하는 것이 유의미해진다.

Number of Walks Grows



Number of anonymous walks grows exponentially:

- There are 5 anon. walks w_i of length 3:
 $w_1=111, w_2=112, w_3=121, w_4=122, w_5=123$

- Simulate anonymous walks w_i of l steps and record their counts.
- Represent the graph as a probability distribution over these walks.

■ 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** : Generate independently a set of m random walks.
- How many random walks m do we need?
 - We want the distribution to have error of more than ϵ with prob. less than δ :

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

η : 길이가 l 일 때의 총 anonymous walk 수

New idea : Learn Walk Embeddings

단순히 각 walk의 빈도수를 임베딩 벡터로 사용하기 보단 각 walk역시 임베딩하여 사용하는 방법

We learn embedding \mathbf{z}_i of anonymous walk w_i .

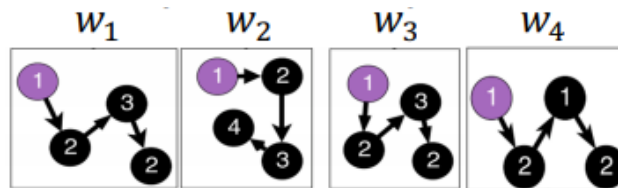
- Learn a graph embedding \mathbf{z}_G together with all the anonymous walk embeddings \mathbf{z}_i
 $\mathbf{Z} = \{\mathbf{z}_i : i = 1 \dots \eta\}$, where η is the number of sampled anonymous walks.

How to embed walks?

- Idea : Embed walks s.t. the next walk can be predicted.

Learn Walk Embeddings

- Output : A vector \mathbf{z}_G for input graph G
 - The embedding of entire graph to be learned
- Starting from node 1 : Sample anonymous random walks, e.g.



- Learn to predict walks that co-occur in Δ -size window (e.g., predict w_3 given w_1, w_2 if $\Delta=2$)
- Objective:

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

- Where T is the total number of walks

전체 학습 과정은 다음과 같다.

1. 노드 u , 길이 l 에 대해 개별적인 T 번의 랜덤 워크를 수행하여 다음과 같은 N_R 을 구한다. 이때의 $N_R(u)$ 는 동일한 노드 u 에서 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.

Objective: $\max_{\mathbf{z}_i, \mathbf{z}_G} \frac{1}{T} \sum_{t=\Delta}^T \log P(w_t | \{w_{t-\Delta}, \dots, w_{t-1}, \mathbf{z}_G\})$

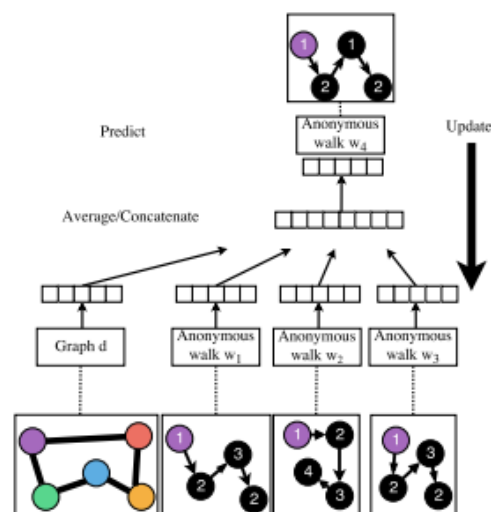
- $P(w_t | \{w_{t-\Delta}, \dots, w_{t-1}, \mathbf{z}_G\}) = \frac{\exp(y(w_t))}{\sum_{i=1}^{\eta} \exp(y(w_i))}$
- $y(w_t) = b + U \cdot \left(\text{cat}\left(\frac{1}{\Delta} \sum_{i=1}^{\Delta} \mathbf{z}_i, \mathbf{z}_G\right) \right)$
 - $\text{cat}\left(\frac{1}{\Delta} \sum_{i=1}^{\Delta} \mathbf{z}_i, \mathbf{z}_G\right)$ means an average of anonymous walk embeddings \mathbf{z}_i in the window, concatenated with the graph embedding \mathbf{z}_G .
 - $b \in \mathbb{R}, U \in \mathbb{R}^D$ are learnable parameters. This represents a linear layer.

All possible anonymous walks
(requires negative sampling)

Anonymous Walk Embeddings, ICML 2018 <https://arxiv.org/pdf/1805.11921.pdf>

- We obtain the graph embedding \mathbf{z}_G (learnable parameter) after the optimization.
- Is \mathbf{z}_G simply the sum over walk embeddings \mathbf{z}_i ? Or is \mathbf{z}_G the residual embedding next to \mathbf{z}_i ?
- According to the paper, \mathbf{z}_G is a separately optimized vector parameter, just like other \mathbf{z}_i 's.
- Use \mathbf{z}_G to make predictions(e.g., graph classification):

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



Summary

We discussed 3 ideas to graph embeddings:

- **Approach 1** : Embed nodes and sum/avg them
- **Approach 2** : Create super-node that spans the (sub)graph and then embed that node.
- **Approach 3** : Anonymous Walk Embeddings
 - Idea 1 : Sample the anon. Walks and represent the graph as fraction of times each anon walk occurs.
 - Idea 2 : Learn graph embedding together with anonymous walk embeddings.

How to Use Embeddings

- How to use embeddings \mathbf{z}_i of nodes:

- Clustering/community detection : Cluster points z_i
 - Node classification : Predict label of node i based on z_i
 - Link prediction : predict edge (i,j) based on (z_i, z_j)
- 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: $f(z_i, z_j) = g(\|z_i - z_j\|_2)$

hadamard나 sum/avg는 교환법칙이 성립하므로 undirected graph에 적용하기 좋다.

- Graph classification : Graph embedding z_G via aggregating node embedding or anonymous random walks. Predict label based on graph embedding z_G .

Total Summary

We discussed graph representation learning, a way to learn node and graph embeddings for downstream tasks, without feature engineering.

- **Encoder-decoder framework:**
 - Encoder : embedding lookup
 - Decoder : predict score based on embedding to match node similarity
- **Node similarity measure : (biased) random walk**
 - Examples : DeepWalk, Node2Vec
- **Extension to Graph embedding** : Node embedding aggregation and Anonymous Walk Embeddings

References

<http://web.stanford.edu/class/cs224w/>

<https://velog.io/@stapers/Lecture-3-Node-Embeddings>