

# Node Embedding

소프트웨어 끈대 강의

노기섭 교수

([kafa46@gmail.com](mailto:kafa46@gmail.com))

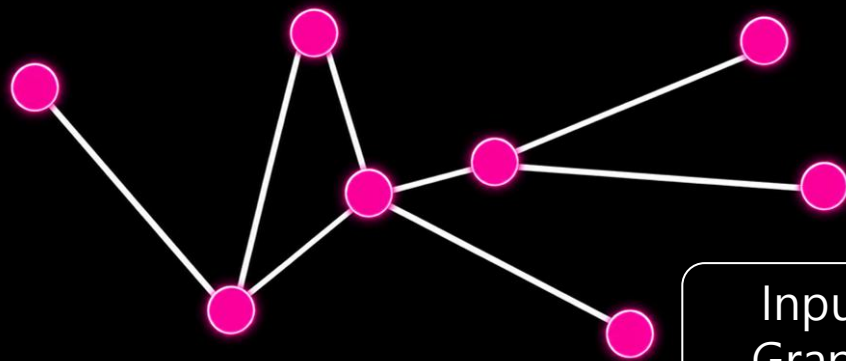
# Traditional ML for Graphs

From an input graph,

$G(N, E)$

Extract node, edge, and graph-level features

Learn a model (Regressor, SVM, NN, etc.)  
that maps features to labels



Input  
Graph

~~Feature Engineering by Human Experts~~

Representation Learning

Features



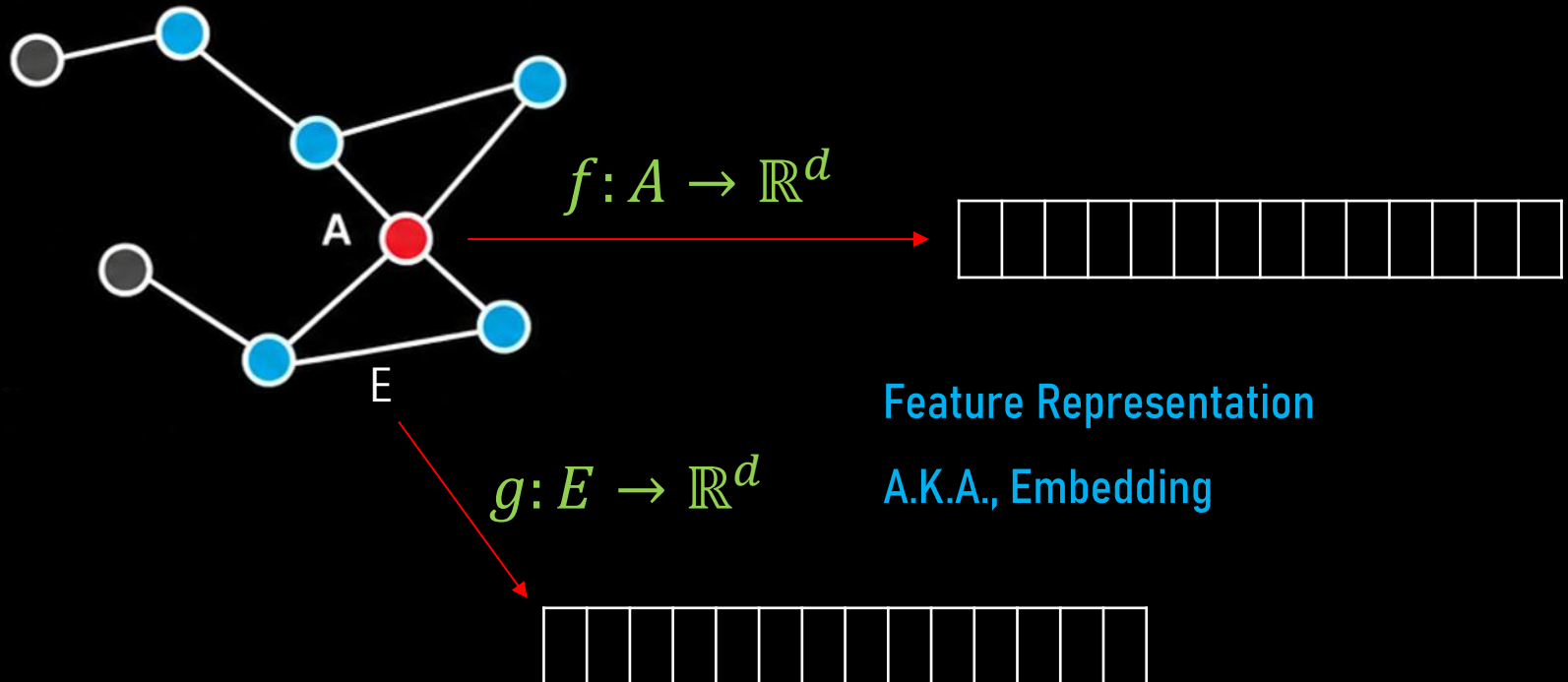
Downstream Task

Model(s)

Prediction

# Goal in Graph Representation Learning

Create a task-independent feature efficiently for ML

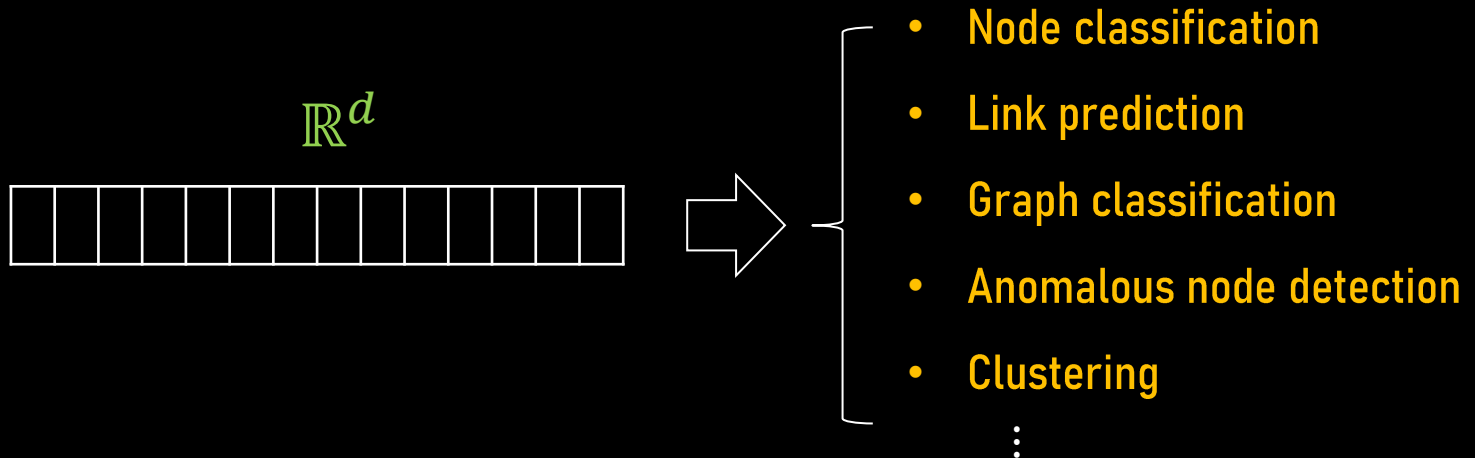


# Why Do We Need the Node Embedding?

Map nodes into an embedding space:

Similarity of embeddings between nodes **indicates** their similarity in the network.

- Both nodes are close to each other (connected by an edge)
- Encode network information
- Potentially used for many downstream predictions

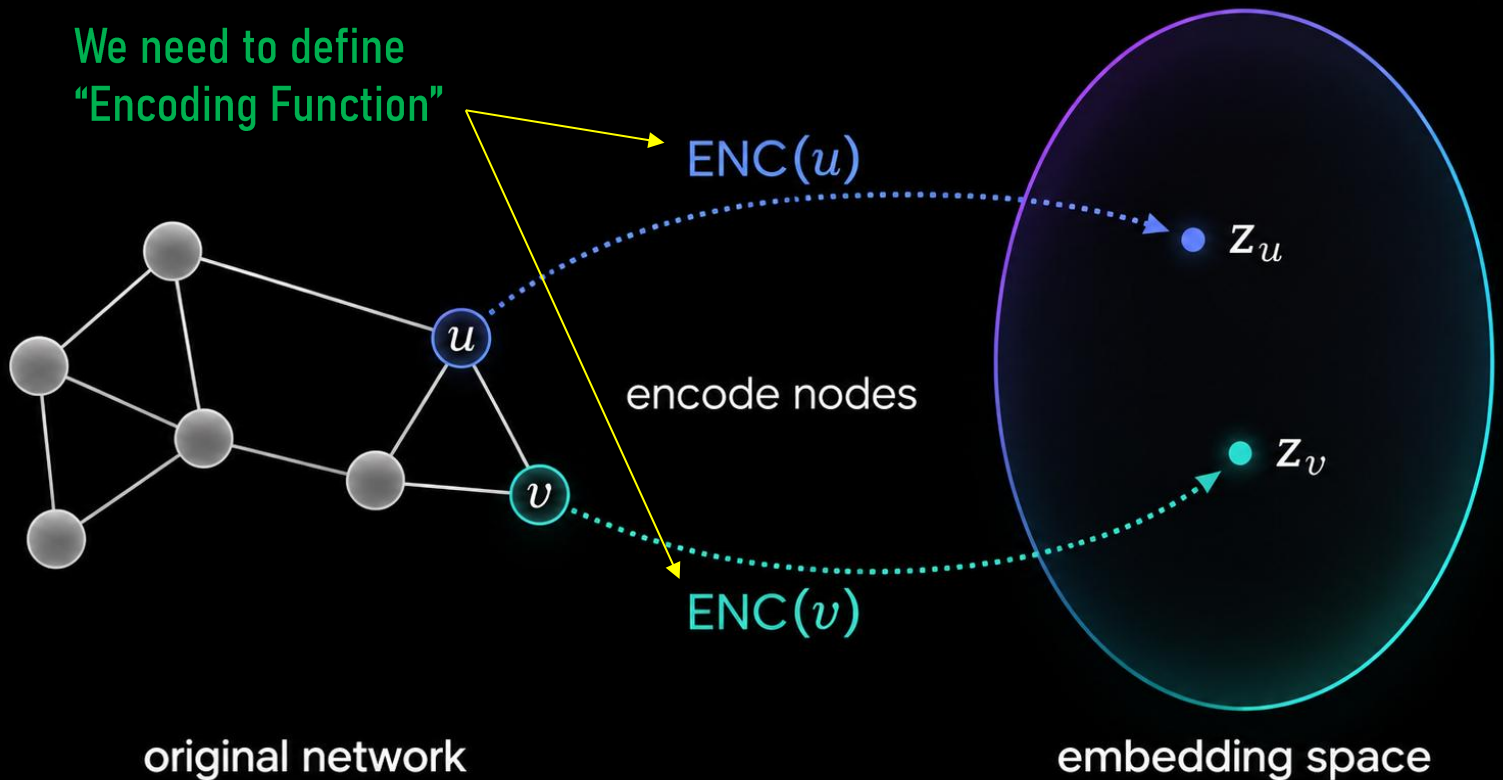


# Goal of Node Embedding

$$\text{Similarity}(u, v) \approx z_v^T \cdot z_u$$

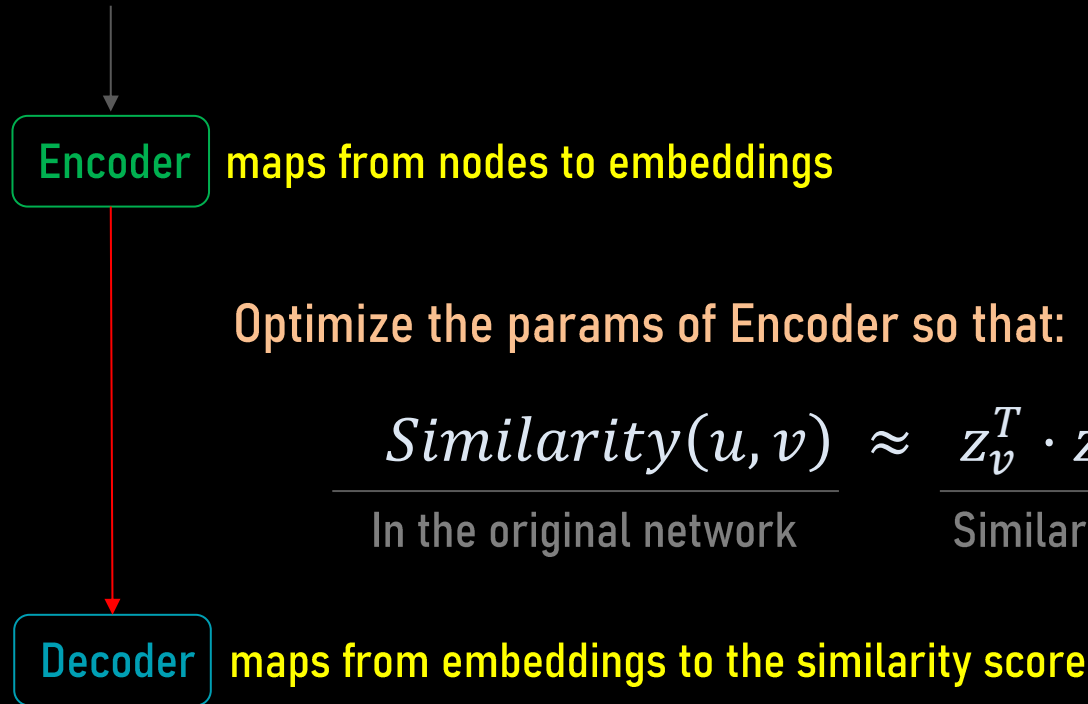
In the original network

Similarity of the embedding



# Learning Node Embedding

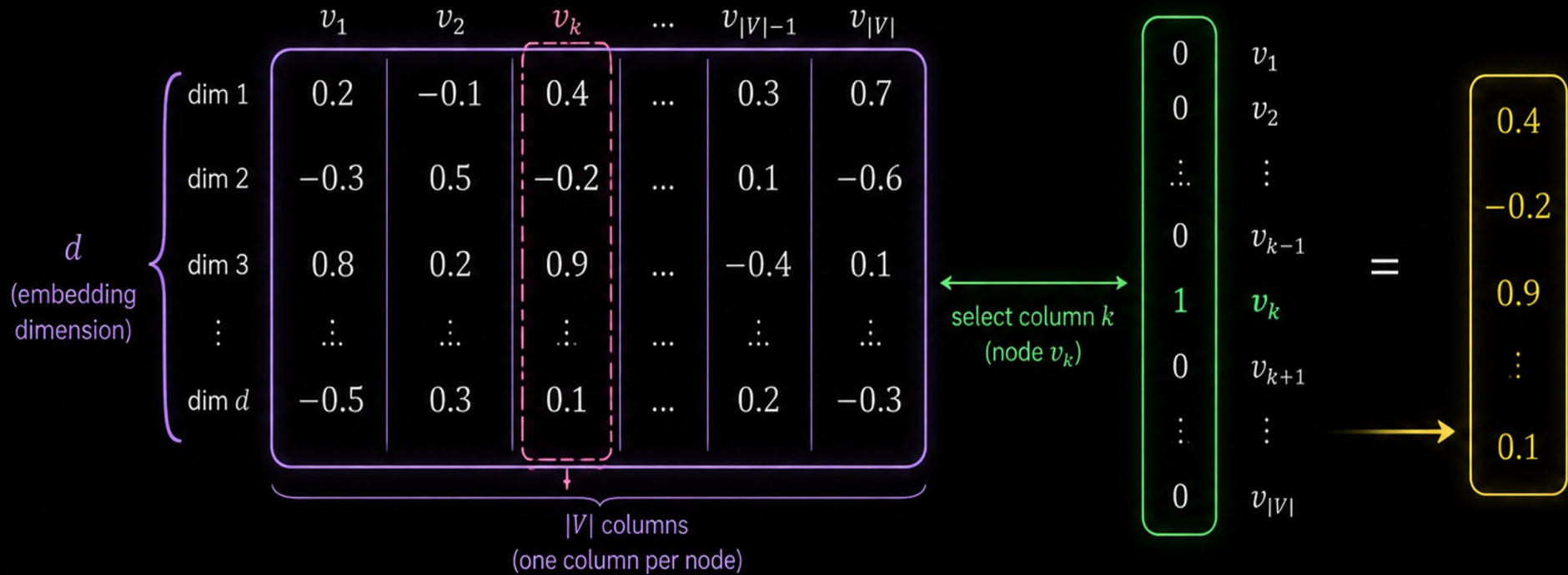
Define a node similarity function  
(A measure of similarity in original graph)



# Shallow Encoding

## Encoder : Embedding Lookup Table

$$ENC(v) = Z \cdot v = z_v, \text{ where } Z \in \mathbb{R}^{d \times |V|} \text{ and } v \in \mathbb{I}^{|V|}$$



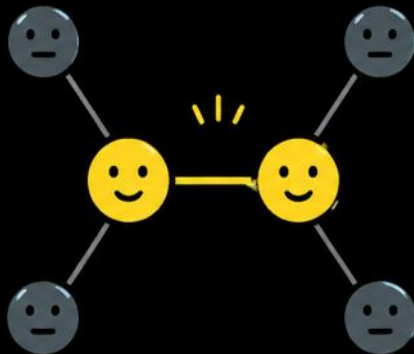
$$\max z_u^T \cdot z_v \\ \text{for } (u, v) \in E$$

Optimization Objective:  
Maximize similarity between  
embeddings of neighboring nodes

# What is the Best in Defining Node Similarity?

Two nodes should be similar if they:

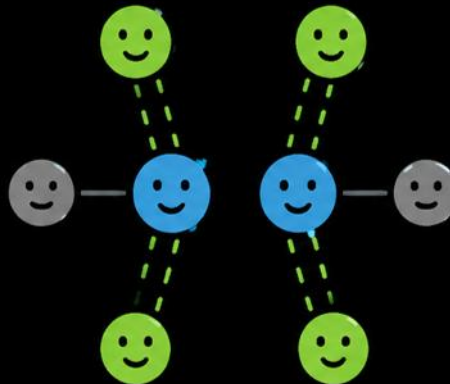
① are linked?



Direct connection  
indicates similarity.

 —  = similar

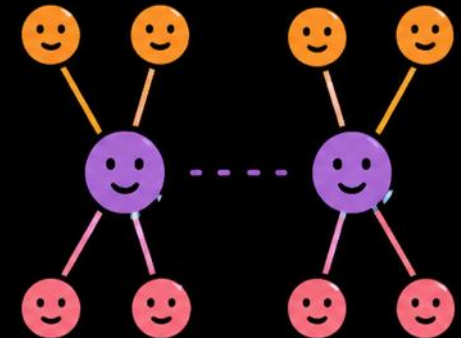
② share neighbors?



Sharing many neighbors  
indicates similarity.

 - -  = similar

③ Have similar  
“structural roles”?



Having similar connection  
patterns (roles) indicates  
similarity.

 ...  = similar

# Possible Solution: Random Walk Approach

**We need an approaches for node embedding s.t.:**

- ✓ Not utilizing node labels
- ✓ Not utilizing node features

**Learn node embeddings while preserving network structure.**

**Also, task independent!**

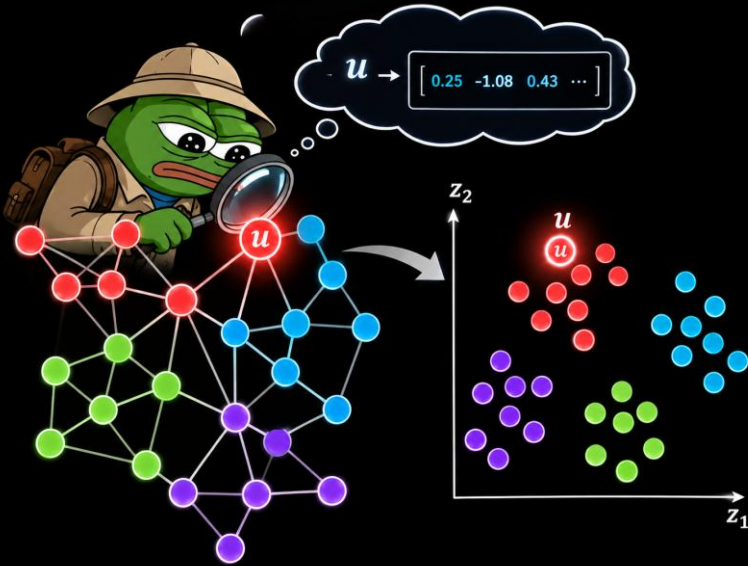
**Random walk approach for node embedding**

or

**Just apply End-to-End fashion**

# Node Embedding: Random Walk Approach

# Notation



$z_u$ : Embedding vector of node  $u$

What we aim to find

$P(v|z_u)$ : Predicted probability of visiting node  $v$  on random walks starting from node  $u$

**Non-linear functions:** Tools for produce predicted probabilities

**Softmax:**

Turns vector of  $K$  real values  
→  $K$  probabilities that sum to 1

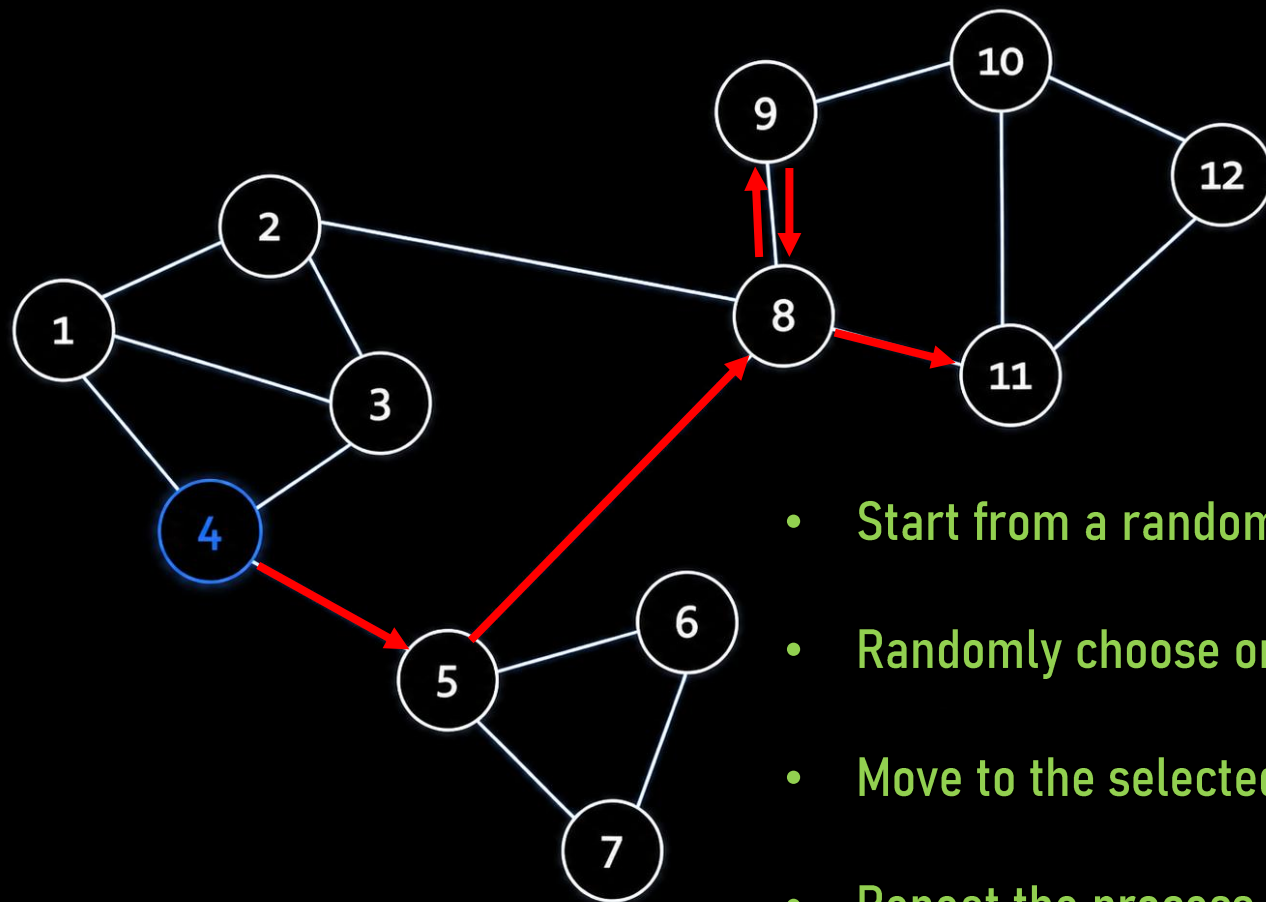
$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

**Sigmoid:**

Turns a real values  
→ a value in range of (0, 1)

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

# Random Walk



- Start from a randomly selected node.
- Randomly choose one of its neighbors.
- Move to the selected neighbor.
- Repeat the process for a fixed number of steps.
- Visited node sequence is called a random walk.

# Probability of Co-occur between Nodes

$z_u^T \cdot z_v \approx$  Probability that  $u$  and  $v$  co-occur on a random walk over the graph

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

$$P_R(v \mid u)$$

2. Optimize embeddings to encode these random walk statistics.

$$\text{Dot product} = \cos(\theta)$$

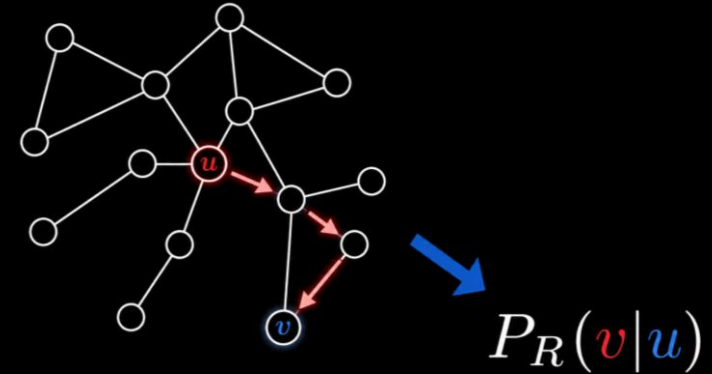
# Advantages on Random Walk

## Expressivity

### Flexible stochastic definition of node similarity

If random walk starting from  $u$  visits  $v$  with high probability,  $u$  and  $v$  are similar.

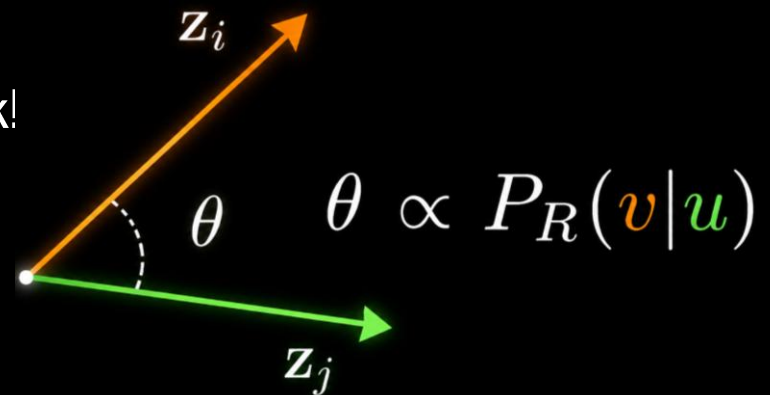
(Repeat until the predefined walk length is reached.)



## Efficiency

### Do not need to consider all node pairs when training

Only need to consider pairs  
that co-occur on random walk!



# Unsupervised Feature Learning

## Goal:

Find embedding of nodes in d-dimensional space  
that preserve similarity

$$G = (V, E)$$

$$f: u \rightarrow \mathbb{R}^d, \text{ where } f(u) = z_u$$

Idea: Nearby nodes are close together in the graph

Define nearby nodes:

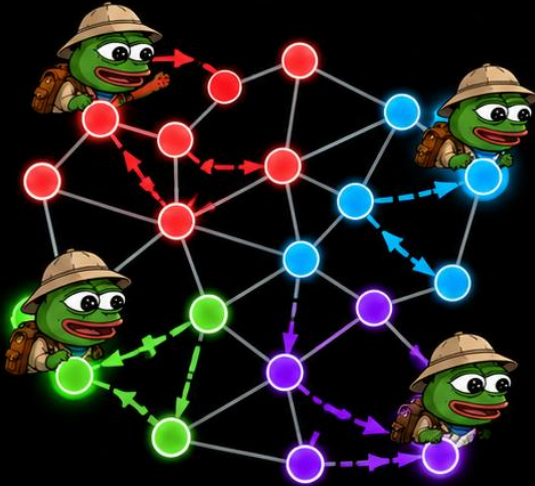
$N_R(u)$ : neighbors of  $u$  obtained by  
some random walk strategy  $R$

Log-likelihood Objective: 
$$\max_f \sum_{u \in V} \log P(N_R(u) \mid z_u)$$

# Random Walk Optimization

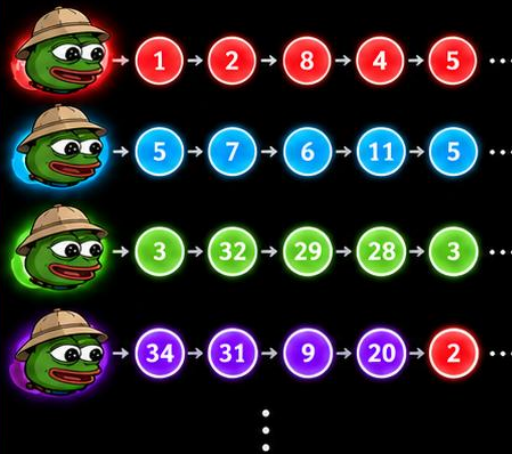
1

Run short random walks from every node.



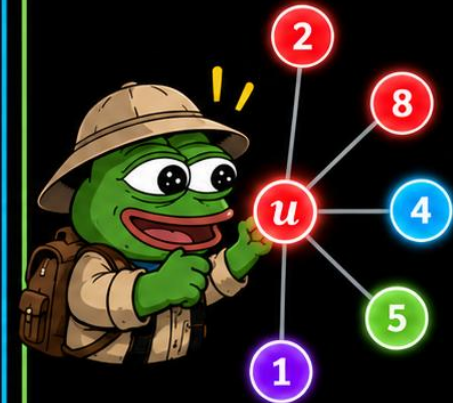
2

Collect nodes visited during the random walks.



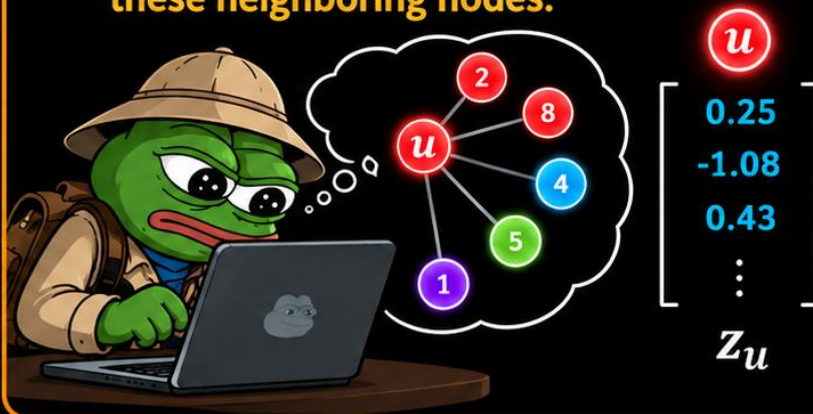
3

Treat visited nodes as the neighborhood of the source node.



4

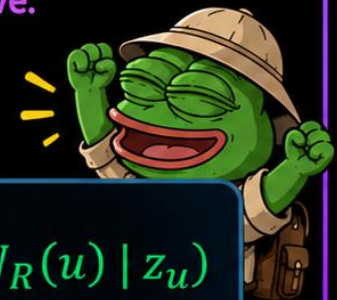
Learn embeddings that predict these neighboring nodes.



5

Optimize using a maximum likelihood objective.

$$\max_f \sum_{u \in V} \log P(N_R(u) | z_u)$$



# Design Objective & Loss

We want to predict the nodes that are likely to be visited during a random walk using only the embedding of node  $u$ .

$$N_R(u) = \{v_1, v_2, \dots, v_n\}$$

$$P(N_R(u) \mid z_u) \xrightarrow[\text{conditionally independent.}]{\text{Assume nodes in } N_R(u) \text{ are}} P(N_R(u) \mid z_u) = \prod_{v \in N_R(u)} P(v \mid z_u)$$

**Taking logarithm**  $\log P(N_R(u) \mid z_u) = \sum_{v \in N_R(u)} \log P(v \mid z_u)$

**Objective:**

$$\max_f \sum_{u \in V} \log P(N_R(u) \mid z_u) \Rightarrow \max_f \sum_{u \in V} \sum_{v \in N_R(u)} \log P(v \mid z_u)$$

**Loss:**

$$\mathcal{L} = \max \log P = \min(-\log P(v \mid z_u)) \rightarrow - \sum_{u \in V} \sum_{v \in N_R(u)} \log P(v \mid z_u)$$

# How to calculate $P(v|z_u)$ ?

We model the probability that node  $u$  is most similar to each node in the graph.

$$P(v|z_u) = \frac{\exp(z_u^T \cdot z_v)}{\sum_{n \in V} \exp(z_u^T \cdot z_n)}$$

Put it all together:

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

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

$O(|V|^2)$  complexity

If the graph is large,  
Too Expensive!

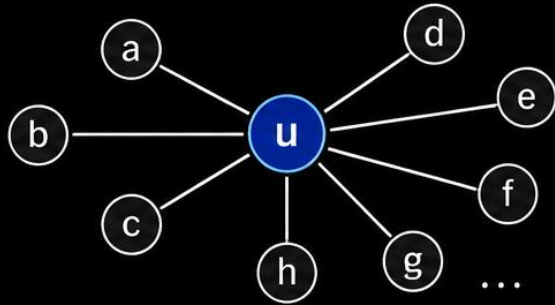
Optimizing random walk embedding:

Finding Embedding  $z_u$

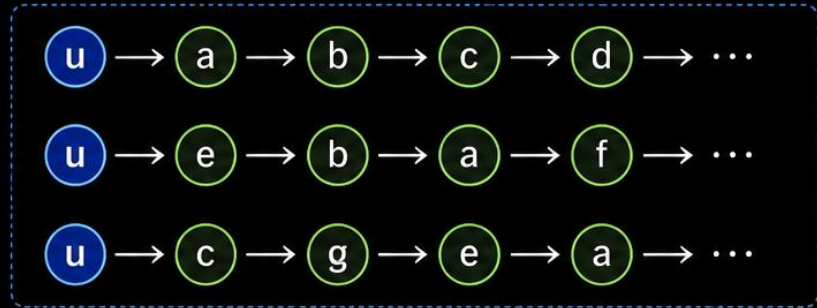
that minimize  $\mathcal{L}$

# How to reduce Complexity?

## 1. Random Walks

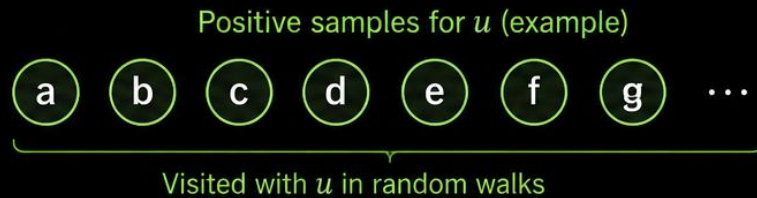


### Example random walks starting from $u$



## 2. Positive Samples

Nodes that co-occur with  $u$  in random walks are **positive samples**.



## 3. Negative Samples

All other nodes that do NOT co-occur with  $u$  are **negative candidates**.



↓ Random sampling

Negative samples for  $u$  (example)



$u$  : target node

○ : positive sample (co-occurs)

○ : negative sample (does not co-occur)



**Positive samples:** nodes that appear with  $u$  in random walks

**Negative samples:** a small number of nodes randomly sampled from the rest (not necessarily never connected to  $u$ )

# Efficient Training with Negative Sampling

**Original Softmax Objective**

$$\sum_{v \in N_R(u)} \log \frac{\exp(z_u^T \cdot z_v)}{\sum_{n \in V} \exp(z_u^T \cdot z_n)}$$

**Randomly sample only  $k$  negative nodes**

**Approximate softmax using sigmoid  $\sigma(\cdot)$**

Randomly sampled  
negative nodes ( $k$ )

$$\approx \log \sigma(z_u^T \cdot z_v) + \sum_{i=1}^k \log \left( \sigma(-z_u^T \cdot z_{n_i}) \right), \text{ where } z_{n_i} \sim P_{V_{\text{negative}}}$$

**Negative Sampling Loss: Maximization  $\rightarrow$  Minimization Objective**

$$\mathcal{L} = -\log \sigma(z_u^T \cdot z_v) - \sum_{i=1}^k \log \left( \sigma(-z_u^T \cdot z_{n_i}) \right)$$

# Optimization of Node Embedding

$$\begin{aligned}\mathcal{L} &= - \sum_{u \in V} \sum_{v \in N_R(u)} \log P(v \mid z_u) \\ &= - \log \sigma(z_u^T \cdot z_v) - \sum_{i=1}^k \log \left( \sigma(-z_u^T \cdot z_{n_i}) \right)\end{aligned}$$

$$z_i = [z_1, z_2, \dots, z_{|V|}]$$

$$\forall i, \text{compute } \frac{\partial \mathcal{L}}{\partial z_i}$$

$$z_i = z_i - \eta \frac{\partial \mathcal{L}}{\partial z_i}$$

# DeepWalk

# DeepWalk (1/2)

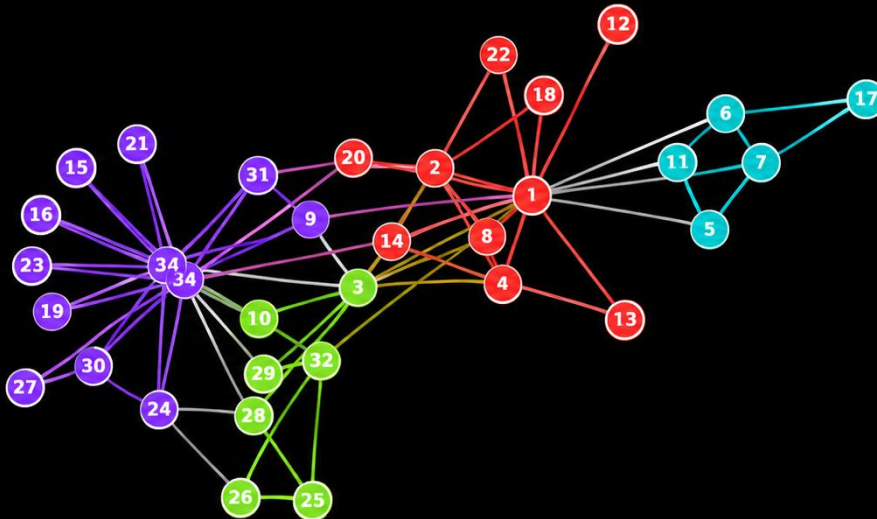
## DeepWalk: Online Learning of Social Representations, ACM KDD 2014

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena

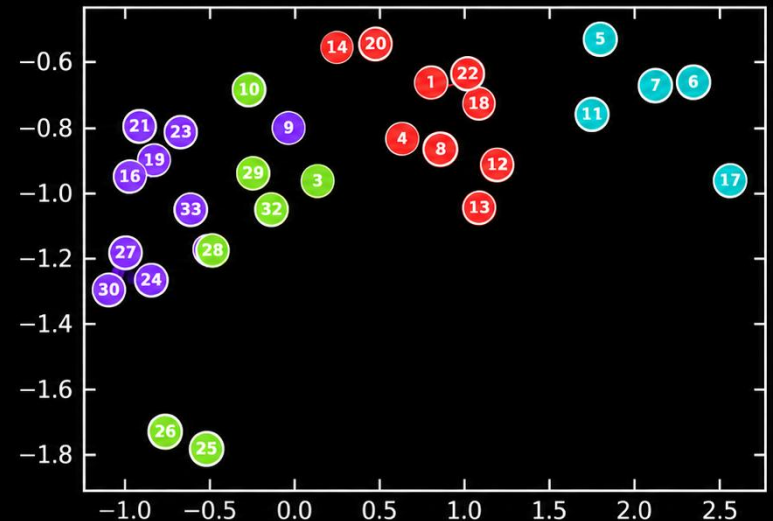
Stony Brook University, NY, USA

<https://arxiv.org/abs/1403.6652>

A good embedding places similar nodes close together.



(a) Input: Karate Graph



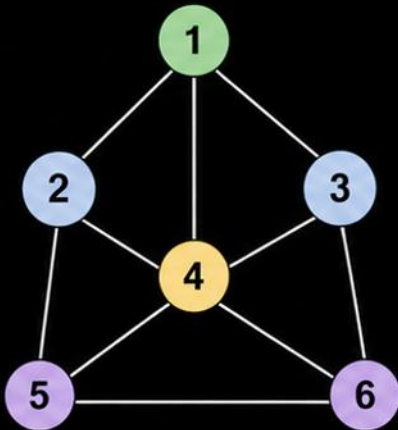
(b) Output: Representation

# DeepWalk Idea: How should We Randomly Walk?

We can optimize embedding with given a random walk strategy  $R$

What strategy should we use for  $R$ ?

Example Graph



Random Walks (length = 5)

- start from 1:



- start from 2:



- start from 3:



## DeepWalk: Key Ideas



### 1. Random Walks

Run multiple fixed-length, **unbiased random walks** from every node.



### 2. Walks as Sentences

Each walk is treated as a **sequence of nodes** (like words in a sentence).



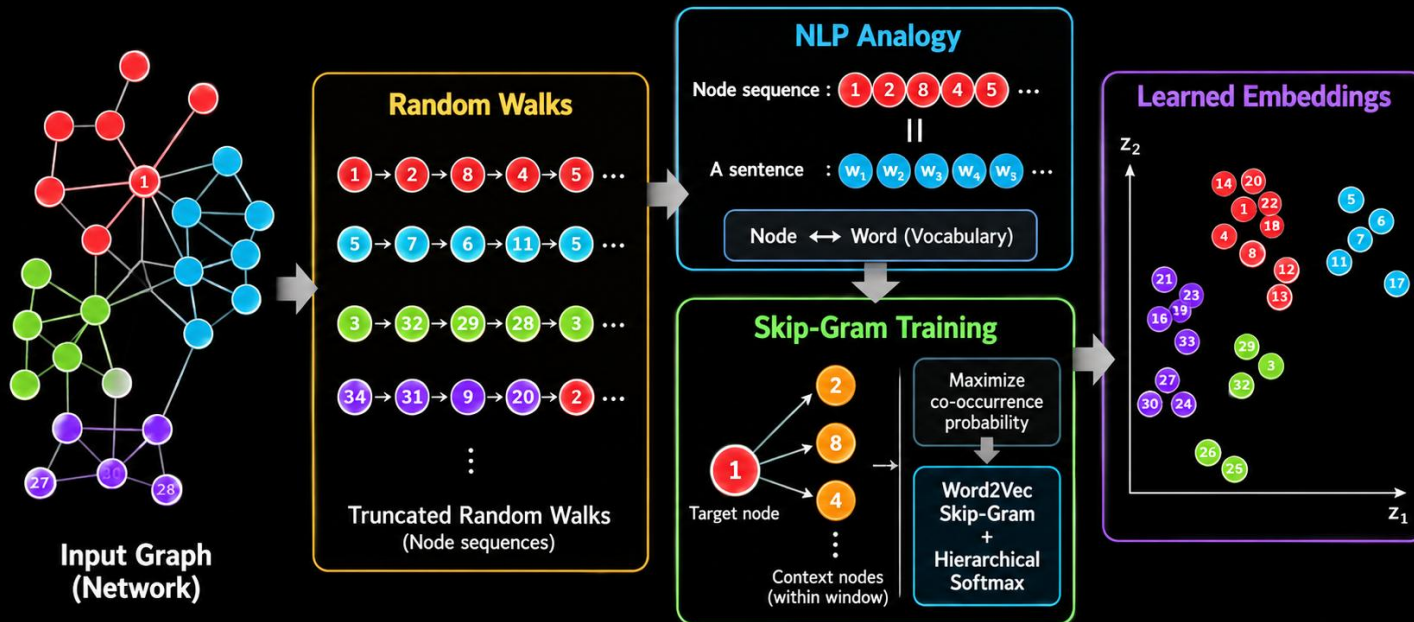
### 3. Learn Embeddings

Use the Skip-gram model with **negative sampling (Word2Vec)** to learn node embeddings.

# DeepWalk (2/2)

## First Deep Learning-Based Node Embedding

Map graph vertices into a low-dimensional continuous vector space.



### Truncated Random Walks:

- Generates short sequences of nodes by traversing the graph randomly, converting structural layout into sequential data.

### Bridge to Natural Language Processing:

- Treats the generated node sequences as "sentences" and individual graph nodes as "words."

### Skip-Gram with Hierarchical Softmax:

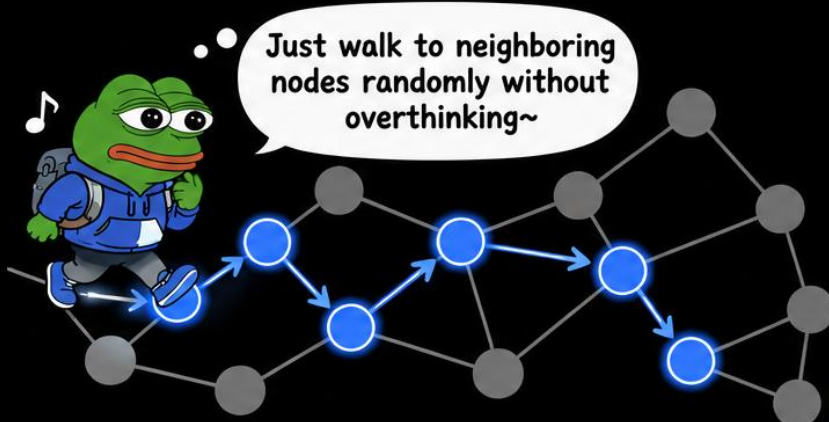
- Leverages Word2Vec's core architecture to maximize the co-occurrence probability of neighboring nodes

# Node2Vec

# Overview on Node2Vec

## DeepWalk

: Just **walk randomly** on the graph.



Generated node sequence (random walk)

1 → 3 → 7 → 4 → 9 ...

## node2vec

: **Change the walking style** depending on the goal.

By adjusting **p** and **q**, let's walk differently and explore different kinds of nodes!



BFS-like  
(homophily)

DFS-like  
(structural role)

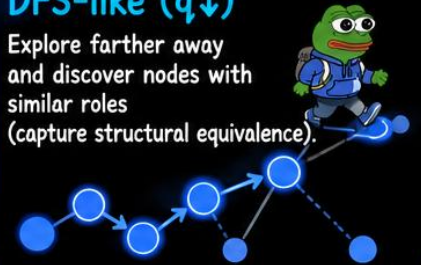
### BFS-like ( $q \uparrow$ )

Stay close to the source and explore nearby nodes (capture local structure).



### DFS-like ( $q \downarrow$ )

Explore farther away and discover nodes with similar roles (capture structural equivalence).



**Result:** We can **control which nodes are considered similar** during learning.

# Overview on Node2Vec

## node2vec: Scalable Feature Learning for Networks

Aditya Grover, Jure Leskovec

Stanford University, CA, USA

<https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf>

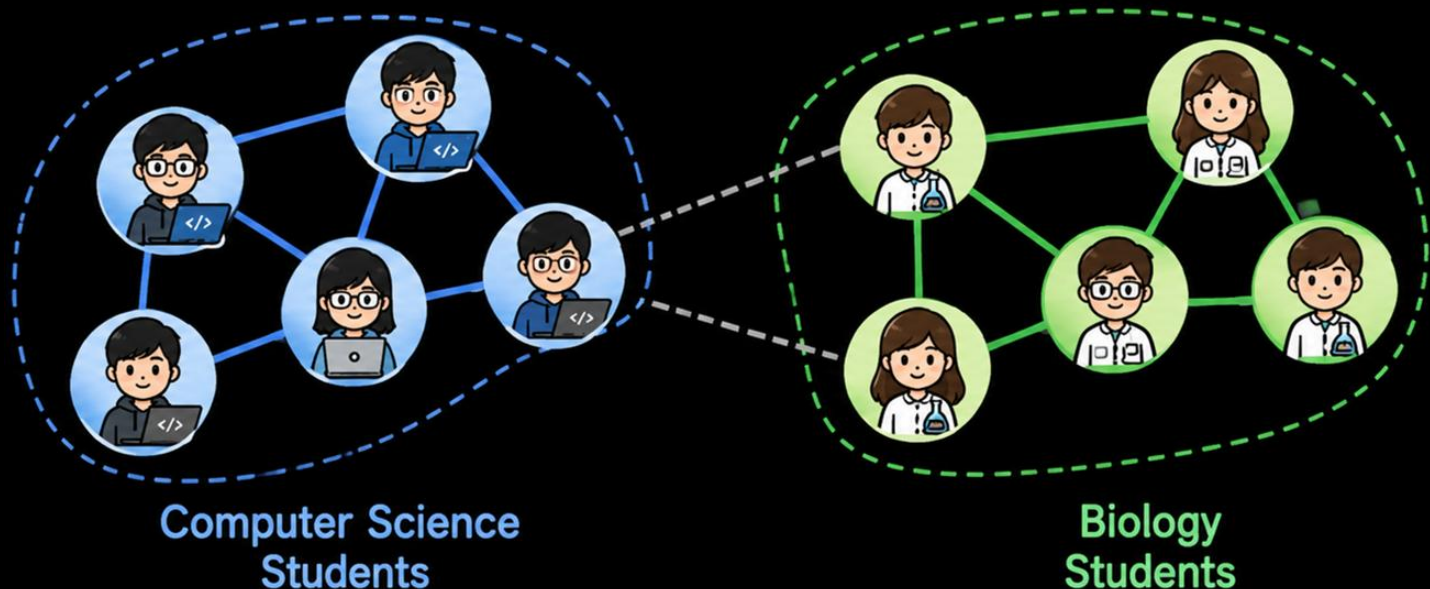
Goal:

Learn low-dimensional node embeddings

such that nodes with similar network structures are located close to each other in the embedding space.

# Homophily?

- Similar nodes tend to connect to each other
- Connected nodes often share similar attributes or interests
- Common in social and citation networks
- BFS-like walks explore nearby nodes
- Helps capture local similarity and community structure

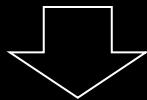


# Node2Vec: Motivation

**Goal:** Embed nodes with similar network neighborhoods close in the feature space.

**Key observation:**

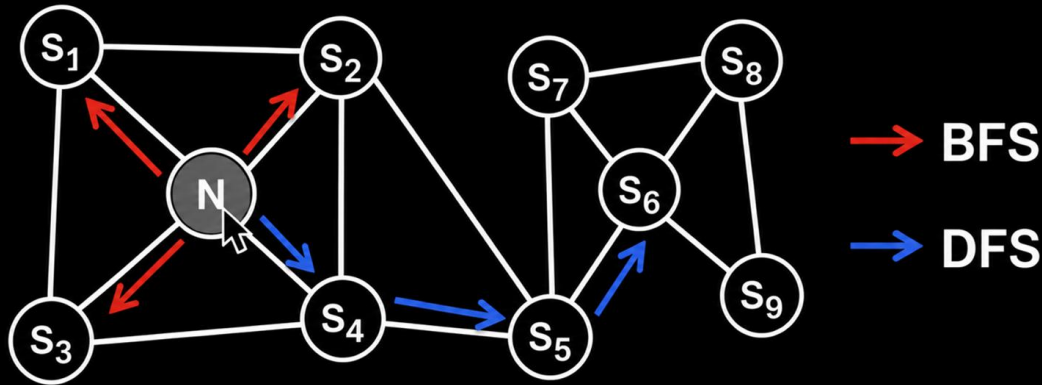
Flexible notion of network neighborhood  $N_R(u)$  of node  $u$  leads to rich node embeddings.



Develop biased 2nd order random walk  $R$  to generate network neighborhood  $N_R(u)$  of node  $u$ .

# Biased Walks

Flexible & biased random walk that can trade off between local and global views



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

$N_{BFS}(u) = \{s_1, s_2, s_3\}$  Local Microscopic View

$N_{DFS}(u) = \{s_4, s_5, s_6\}$  Global Macroscopic View

# Interpolating BFS and DFS

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

Return parameter  $p$

Controls the probability of returning to the previous node.

Large  $p \rightarrow$  Less likely to return.

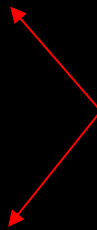
Small  $p \rightarrow$  More likely to return.

In-out parameter  $q$

Controls the trade-off between BFS and DFS exploration.

$q > 1 \rightarrow$  Favors BFS (local neighborhood exploration).

$q < 1 \rightarrow$  Favors DFS (global exploration).



$p$  and  $q$  to smoothly interpolate  
between BFS and DFS.

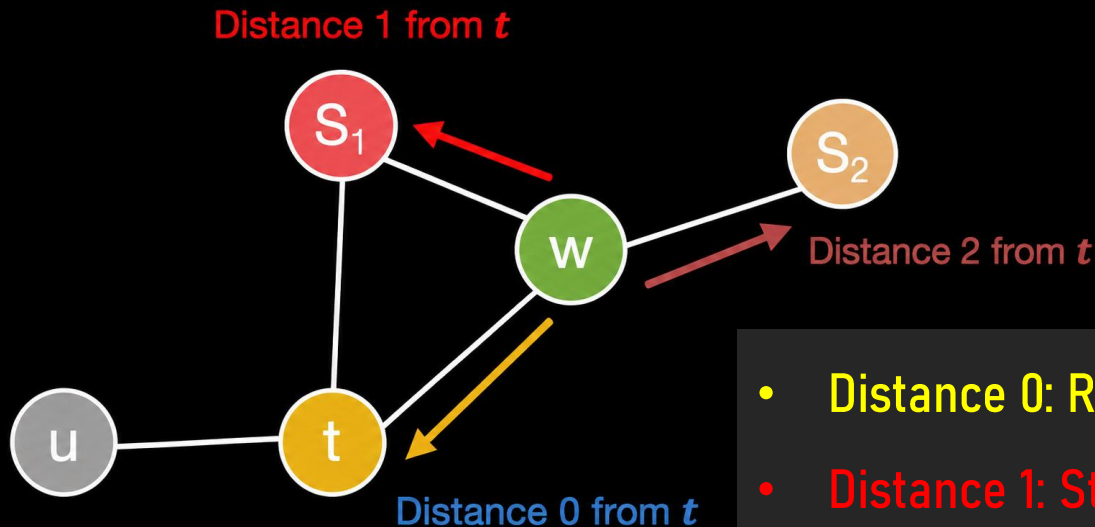
# One Step of Biased Random Walk

## Transition Probabilities in node2vec

The random walk has just moved from  $t \rightarrow w$ .

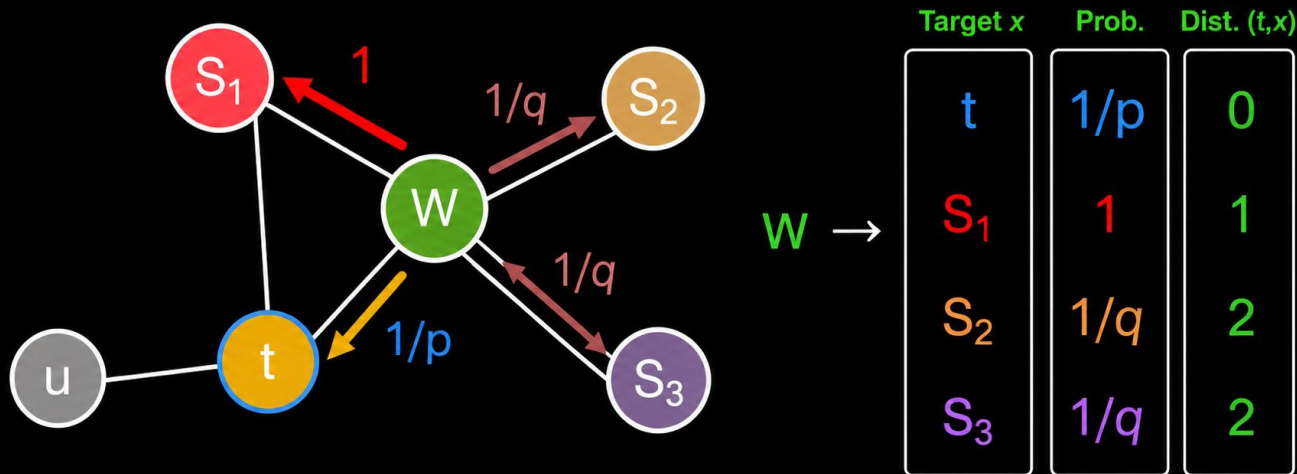
The next node is selected among the neighbors of  $w$ .

Each neighbor is categorized by its distance from the previous node  $t$ :



- **Distance 0:** Return to the previous node.
- **Distance 1:** Stay close to the previous node.
- **Distance 2:** Explore farther from the previous node.

# Example of Biased Walk



BFS-like walk: Low value of  $p$

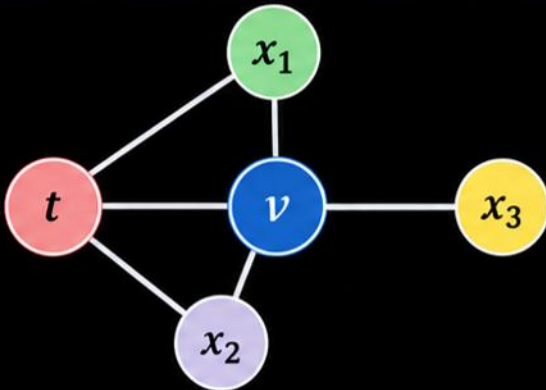
DFS-like walk: Low value of  $q$

The values  $1/p$ ,  $1$ , and  $1/q$  are unnormalized transition weights and are normalized into probabilities before sampling.

$$P(c_i|w) = \frac{\pi_{wc_i}}{\sum_{x \in N(w)} \pi_{wx}}$$

# How to Choose the Next Node in Node2Vec?

## 1 Graph structure ( $t \rightarrow v \rightarrow ?$ )



- Walk has just moved  $t \rightarrow v$ .
- Candidates from  $v$ :  $\{t, x_1, x_2, x_3\} = N(v)$ .
- Edges  $(t, x_1)$  and  $(t, x_2)$  exist, but  $(t, x_3)$  does not.

## 2 Distance $d(t, x)$ for each candidate

| Candidate node $x$ | Relation to $t$                             | $d(t, x)$ |
|--------------------|---------------------------------------------|-----------|
| $t$ (return)       | $x = t$                                     | 0         |
| $x_1$              | neighbor of $t$ (edge $t - x_1$ )           | 1         |
| $x_2$              | neighbor of $t$ (edge $t - x_2$ )           | 1         |
| $x_3$              | not a neighbor of $t$ (path $t - v - x_3$ ) | 2         |

## 3 Compute unnormalized weights

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx} \quad (\text{assume } w_{vx} = 1)$$

| Candidate node $x$ | $\alpha_{pq}(t, x)$ | $\pi_{vx}$ |
|--------------------|---------------------|------------|
| $t$                | $1/p$               | $1/p$      |
| $x_1$              | 1                   | 1          |
| $x_2$              | 1                   | 1          |
| $x_3$              | $1/q$               | $1/q$      |

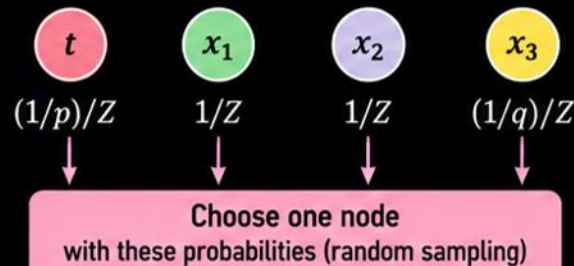
$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & (x = t) \\ 1 & (x \in N(t) \setminus \{t\}) \\ \frac{1}{q} & (\text{otherwise}) \end{cases}$$

## 4 Normalize to get transition probabilities

$$Z = \frac{1}{p} + 1 + 1 + \frac{1}{q}$$

| Candidate node $x$ | $P(v \rightarrow x) = \frac{\pi_{vx}}{Z}$ |
|--------------------|-------------------------------------------|
| $t$                | $(1/p) / Z$                               |
| $x_1$              | $1 / Z$                                   |
| $x_2$              | $1 / Z$                                   |
| $x_3$              | $(1/q) / Z$                               |

## 5 Randomly sample according to the probabilities



## 6 Effect of $p$ and $q$

### $p$ (return parameter)

- Controls the probability of returning to  $t$ .
- **Smaller  $p$**   $\rightarrow$  **higher** chance to return ( $1/p$  is large).
- **Larger  $p$**   $\rightarrow$  **lower** chance to return.

### $q$ (in-out parameter)

- Controls the preference between BFS and DFS.
- **Smaller  $q$**   $\rightarrow$  **higher** chance to visit  $x_3$  (outward,  $d = 2$ ).
- **Larger  $q$**   $\rightarrow$  **lower** chance to visit  $x_3$ .



Thank you ^^.