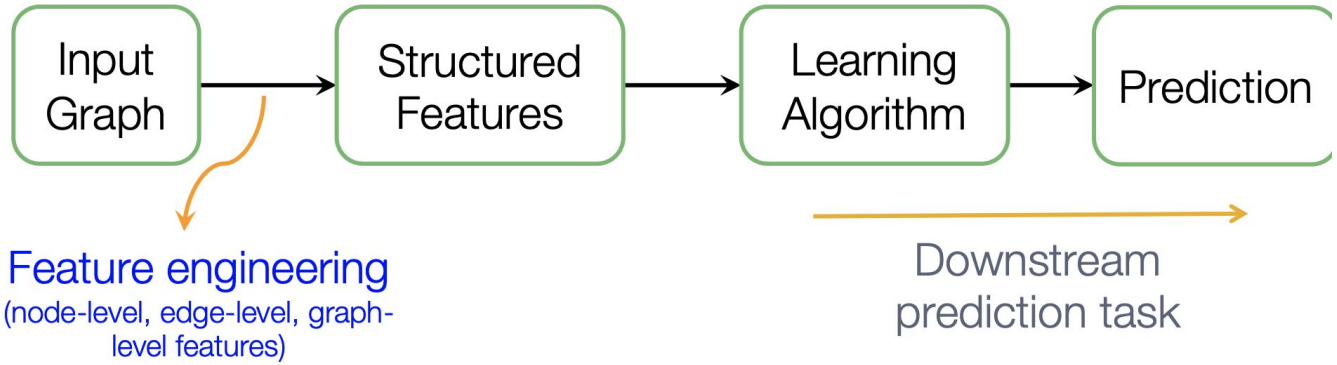
Node Embedding

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Recap: Traditional ML

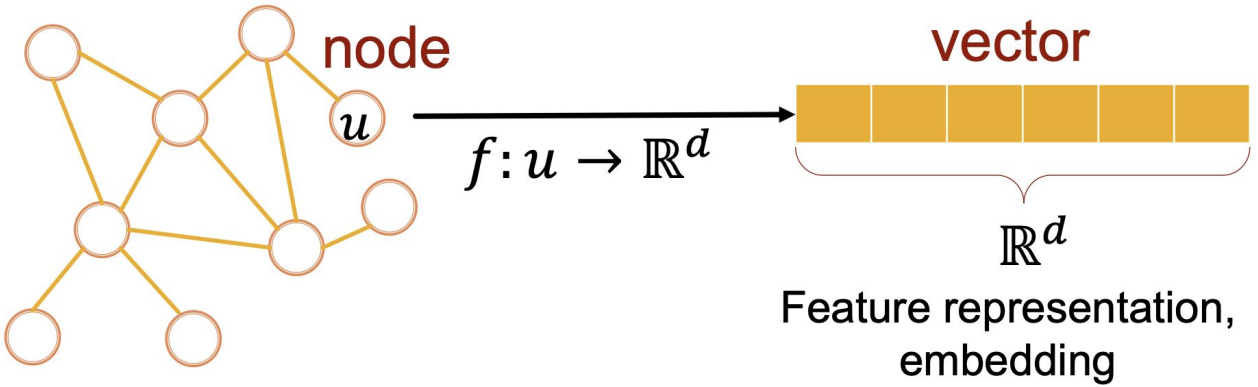
Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels.



Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Graph Representation Learning

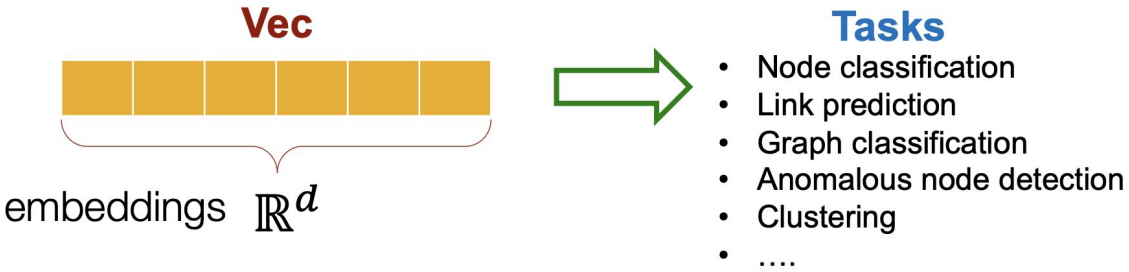
Goal: Efficient task-independent feature learning for machine learning with graphs!



Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

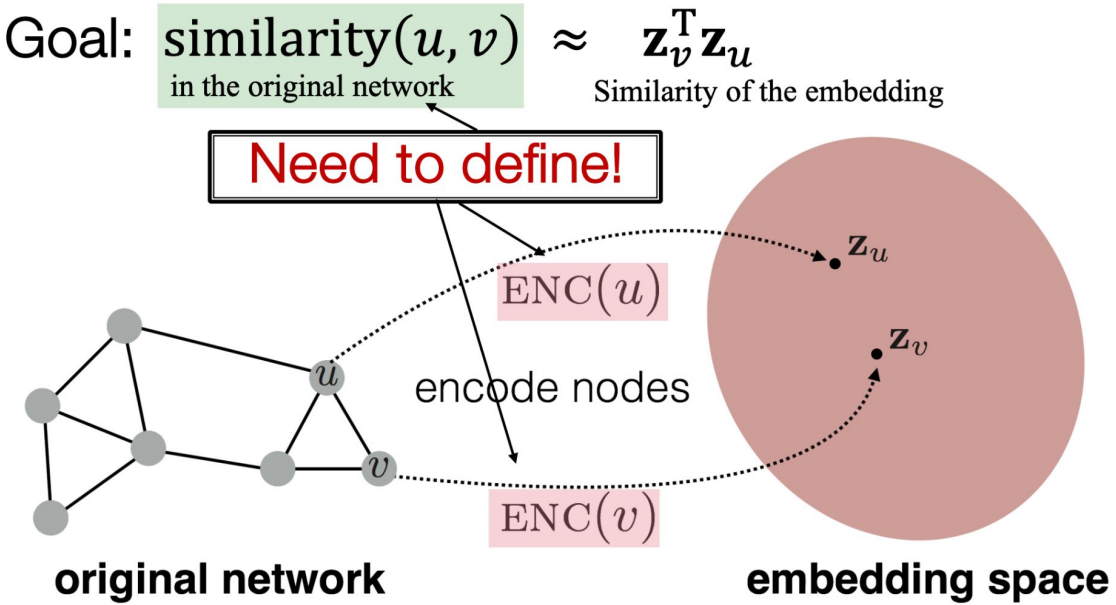
Why Embedding?

- **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



Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Embedding Nodes



Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

“Shallow” Encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

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

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

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

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

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.

- Q: What could be some advantages and disadvantages of each method?
- Q: What could be some potential "structural roles" that we could learn?

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

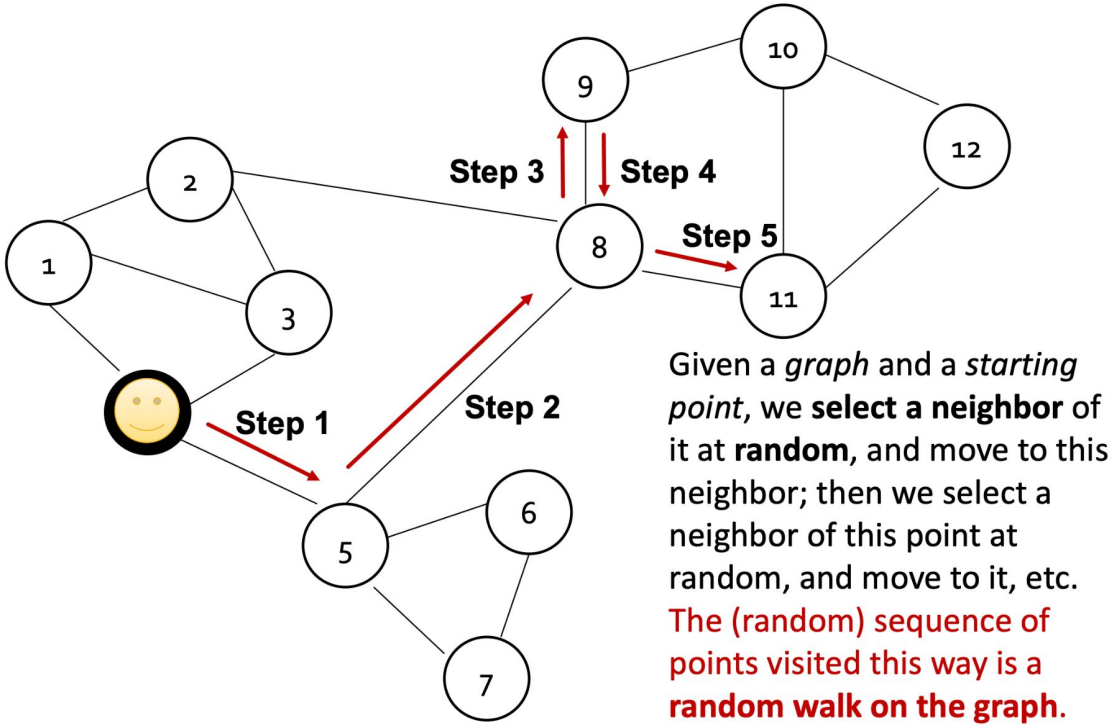
Note on Node Embeddings

- 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.

- Q: What might be some labelled graphs we could use?
- Q: What could be some features of unlabeled graphs we could use?
- Q: What are some tasks for which graphs naturally lend themselves to represent data?

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Random Walk



- Q: How do we pick which neighbor to walk to if there are many neighbors?
- Q: What do we do if a node has no edges leading out of it?
- Note: We could only 'random-walk' until we hit a sequence of fixed length. Then, we restart the walk

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Random-Walk Embeddings

We want to find, for 2 nodes u, v, embeddings Zu and Zv such that

$$\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v \approx$$

probability that u and v co-occur on a random walk over the graph

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Why Random Walks?

1. **Expressivity:** Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information
   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

Q: Why not random walks?
Q: What might be some other methods of grouping similar nodes?

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

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:

$$\arg\max_z \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

  - $N_R(u)$ 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 $N_R(u)$.

Q: What could be some other terms we could add to the objective function?

(E.g. Upweigh nodes with fewer neighbors, penalize norm of embeddings, add term to account for nodes with no neighbors etc.)

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

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 Z using Stochastic Gradient Descent:

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

We can efficiently approximate this using negative sampling!

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

- Q: How do you use Zu in this formula / how is Zu used in P(v|Zu)?

Random Walks: Summary

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

- **Intuition:** Optimize embeddings $\mathbf{z}_u$ to minimize the negative log-likelihood of random walk neighborhoods $N(u)$.
- **Parameterize $P(v|\mathbf{z}_u)$ using softmax:**

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

Why softmax?
We want node v to be most similar to node u (out of all nodes n).
Intuition: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

Q: Why softmax?

Intuitions:

1. It has an inbuilt sparsifying factor - exp of negative number ~ 0
2. Has an inbuilt normalizing factor - plain dot products might explode

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Random Walks: Summary

Putting it all together:

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

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

Q: What is the complexity of this loss function, in number of nodes?

Optimizing random walk embeddings =

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

Problem: |V|^2 Complexity

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Random Walks: Summary

- **Solution:** Negative sampling

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

Why is the approximation valid?
Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. 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.

More at https://arxiv.org/pdf/1402.3722.pdf

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

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

random distribution over nodes

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

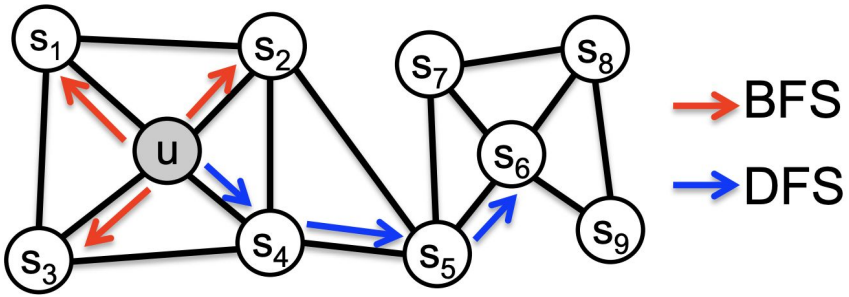
- Negative sampling allows for quick likelihood calculation.

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

node2vec: Biased Walks

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

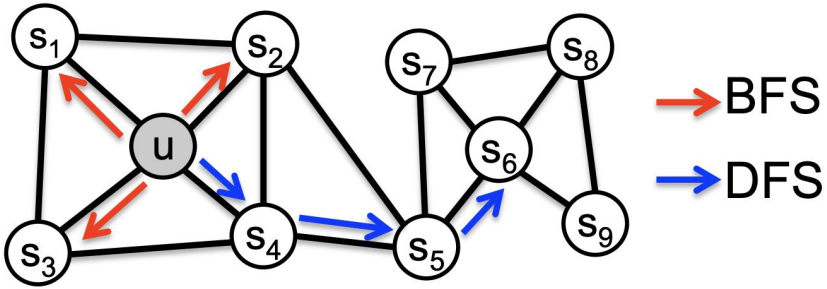
Refresher: What is DFS and BFS?



Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

node2vec: Biased Walks

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):

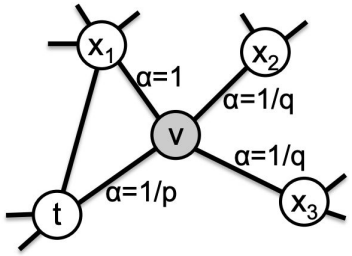
$N_{BFS}(u) = \{s_1, s_2, s_3\}$ Local microscopic view

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

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Node2vec vs. Unbiased Random Walk

Aspect	Unbiased Random Walks	Node2Vec
Walk Strategy	Transition to neighbors equally: $P(u \to v) = \frac{1}{\deg(u)}$	Biased transitions based on p (return) and q (exploration): $P(u \to v) = \frac{\pi_{uv}}{Z}$
Parameters	None.	p (return probability) and q (exploration probability).
Transition Formula	$P(u \to v) = \frac{1}{\deg(u)}$	$\pi_{uv} = \frac{1}{Z} \cdot \begin{cases} 1/p & \text{if } d_{tv} = 0, \\ 1 & \text{if } d_{tv} = 1, \\ 1/q & \text{if } d_{tv} = 2. \end{cases}$
Exploration	Uniform across the graph.	Controlled by p (avoid revisits) and q (balance local vs. global exploration).
Focus	Captures global structure.	Balances local node neighborhoods and global graph structure.
Applications	Simple embeddings (e.g., DeepWalk).	Advanced tasks (node classification, link prediction, community detection).



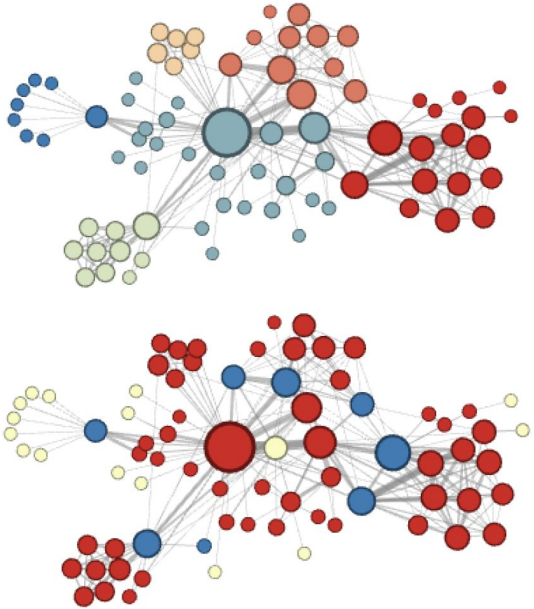
Dtv is the distance between nodes t and v

Effect of changing hyperparameters p and q in Node2vec

Using q = 0.5 (prefering DFS), results in graph embeddings that cluster by network structure

Using q = 2 on the other hand, produce graph embeddings that cluster by structural similarity

Source: https://arxiv.org/pdf/1607.00653

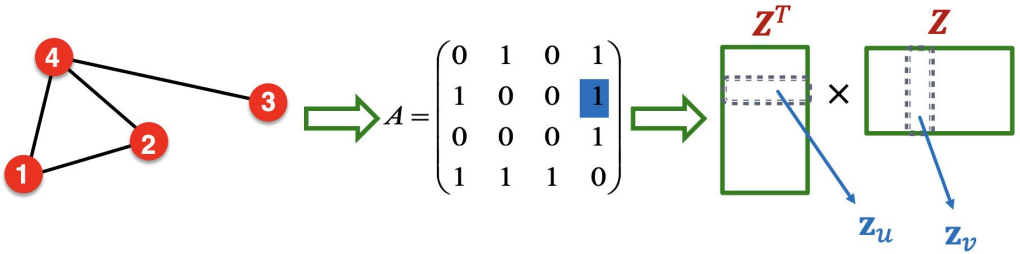


Embedding vs. Matrix Factorization

- Simplest **node similarity**: Nodes u, v are similar if they are connected by an edge
- This means: $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u = A_{u,v}$ which is the (u, v) entry of the graph adjacency matrix A
- Therefore, $\boldsymbol{Z}^T\boldsymbol{Z} = A$

Deepwalk is equivalent to Matrix Factorization of

$$log\left(vol(G)\left(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r\right)D^{-1}\right) - \log b$$

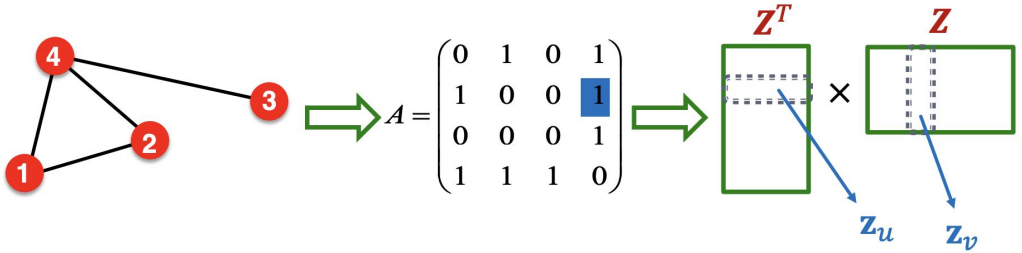


Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Embedding vs. Matrix Factorization

An alternate approach to calculating node embeddings

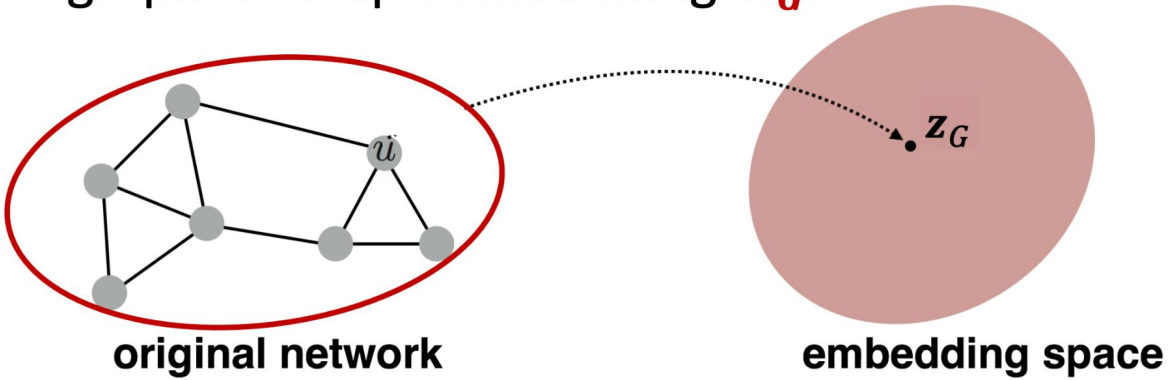
- Simplest **node similarity**: Nodes u, v are similar if they are connected by an edge
- This means: $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u = A_{u,v}$ which is the (u, v) entry of the graph adjacency matrix A
- Therefore, $\boldsymbol{Z}^T \boldsymbol{Z} = A$



Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Embedding Entire Graphs

- **Goal:** Want to embed a subgraph or an entire graph G. Graph embedding: $\mathbf{z}_G$.



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

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

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.

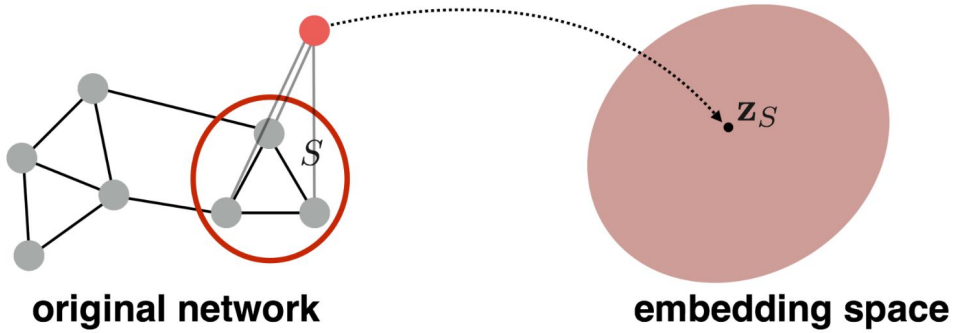
$$\mathbf{z}_G = \sum_{v \in G} \mathbf{z}_v$$

- Used by Duvenaud et al. to classify molecules based on their graph structure

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Approach 2

- **Approach 2:** Introduce a **"virtual node"** to represent the (sub)graph and run a standard graph embedding technique



- Proposed by Li et al. as a general technique for subgraph embedding

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

How to Use Embeddings

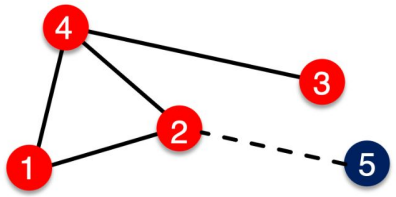
- **How to use embeddings z_i of nodes:**
  - **Clustering/community detection:** Cluster points $\boldsymbol{z_i}$
  - **Node classification:** Predict label of node i based on $\boldsymbol{z_i}$
  - **Link prediction:** Predict edge (i, j) based on $(\boldsymbol{z_i}, \boldsymbol{z_j})$
    - Where we can: concatenate, avg, product, or take a difference between the embeddings:
      - Concatenate: $f(\boldsymbol{z}_i, \boldsymbol{z}_j) = g([\boldsymbol{z}_i, \boldsymbol{z}_j])$
      - Hadamard: $f(\boldsymbol{z}_i, \boldsymbol{z}_j) = g(\boldsymbol{z}_i * \boldsymbol{z}_j)$ (per coordinate product)
      - Sum/Avg: $f(\boldsymbol{z}_i, \boldsymbol{z}_j) = g(\boldsymbol{z}_i + \boldsymbol{z}_j)$
      - Distance: $f(\boldsymbol{z}_i, \boldsymbol{z}_j) = g(||\boldsymbol{z}_i - \boldsymbol{z}_j||_2)$
  - **Graph classification**: Graph embedding $\boldsymbol{z_G}$ via aggregating node embeddings or virtual-node.
    Predict label based on graph embedding $\boldsymbol{z_G}$.

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Limitations (1)

Limitations of node embeddings via matrix factorization and random walks

- **Transductive (not inductive) method:**
  - Cannot obtain embeddings for nodes not in the training set
  - Cannot apply to new graphs
    - If you apply DeepWalk to the same graph multiple times, each time you'll get a different embedding each time. Why?



Training set

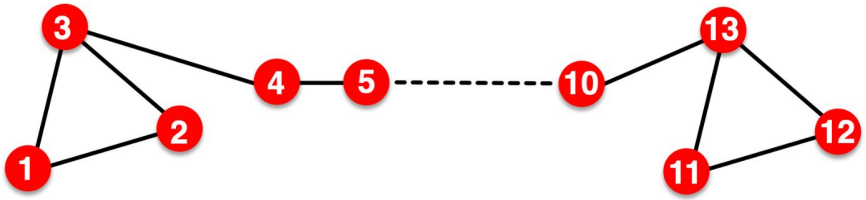
A newly added node 5 at test time (e.g., new user in a social network)

Cannot compute its embedding with DeepWalk / node2vec. Need to recompute all node embeddings.

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Limitations (2)

- Cannot capture **structural similarity**:



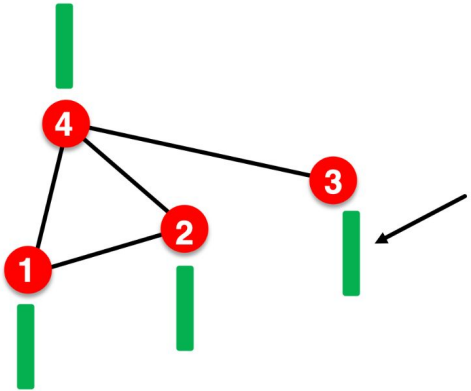
  - Node 1 and 11 are **structurally similar** – part of one triangle, degree 2, …
  - However, they have very **different** embeddings.
    - It's unlikely that a random walk will reach node 11 from node 1.

- **DeepWalk and node2vec do not capture structural similarity.**

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf

Limitations (3)

- Cannot utilize node, edge and graph features



Feature vector
(e.g. protein properties in a protein-protein interaction graph)

DeepWalk / node2vec embeddings do not incorporate such node features

Solution to these limitations: Deep Representation Learning and Graph Neural Networks
(To be covered in depth next)

Source: https://web.stanford.edu/class/cs224w/slides/02-nodeemb.pdf