“Shallow Learning” with Graphs

Jiaxuan You

Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall

https://ulab-uiuc.github.io/CS598/

Logistics: Email & Office Hours

- Email: cs598-you@siebelschool.illinois.edu
  - Shared by {jiaxuan, ziruic4}@illiois.edu
- Office Hours:
  - Jiaxuan You: 1:45 PM - 3:00 PM, Wednesday, Siebel Center for Computer Science | **Room 2122**
  - Zirui Cheng: 12:00 PM - 1:00 PM, Tuesday and Thursday, Zoom | link provided on Canvas
  - *We encourage you to reserve your time slots in advance if you think you need much time (max time 10 minutes). The reservation link is provided on Canvas. Otherwise you may need to wait students with reservations.*

Logistics: Writing Task

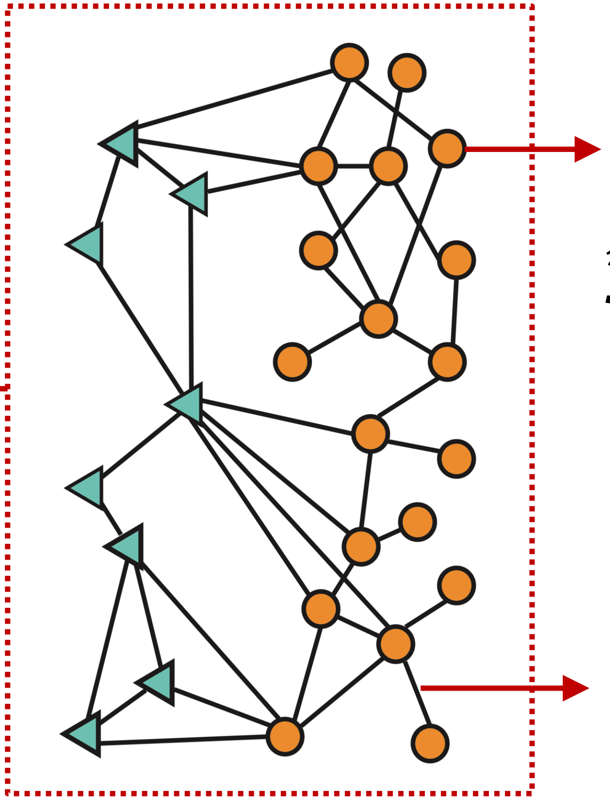
- All the responses should be submitted through **Canvas** in **comma-separated values (CSV)** files. 15% of grade.
- You can **download the CSV template** from a provided link.
- Please submit **one CSV file** for **each paper (3 total)**.
- Please indicate whether you are willing to share your input to the public.
  - It could be shared on our website to help other researchers gain insights, and we will acknowledge your name.
  - Don't worry, it's fully optional, choosing not to share your data will have no effect on your grades.

Recap: Graph Learning Prediction Heads

- **Idea:** Different task levels require different prediction heads
- **Prediction head:** map **node embeddings** to the **predictions of interest**

$$\hat{\boldsymbol{y}}_G = \text{Mean}(\{\mathbf{h}_v \in \mathbb{R}^d, \forall v \in G\})$$

Graph-level prediction



Node-level prediction

$$\hat{\boldsymbol{y}}_v = \text{Head}_{\text{node}}(\mathbf{h}_v) = \mathbf{W} \quad \mathbf{h}_v$$

$$\hat{\boldsymbol{y}}_v = \text{Head}_{\text{node}}(\mathbf{h}_v) = \mathbf{W} \quad \mathbf{h}_v$$

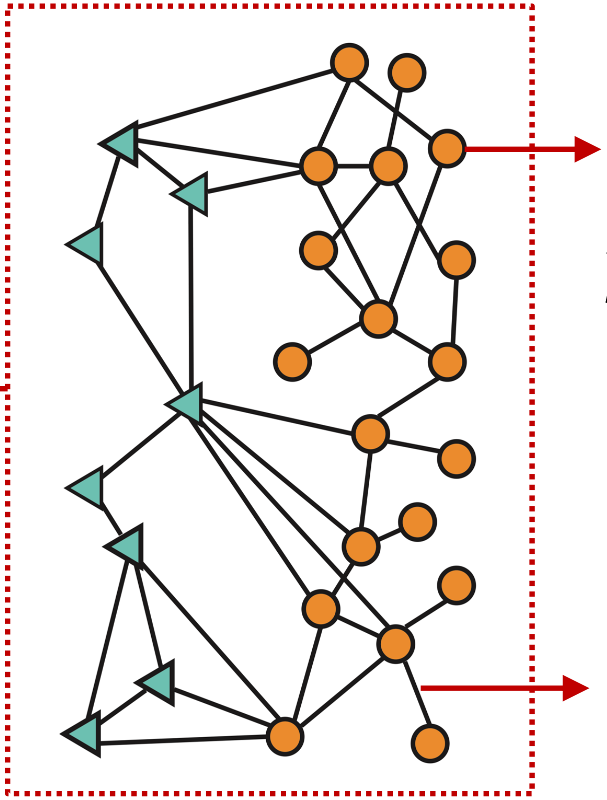
Edge-level prediction

Graph Learning Prediction Heads

- **Today: How do we get h_v in the simplest way?**
  - With **“shallow” node encoders**

$$\hat{y}_G = \mathrm{Mean}(\{\mathbf{h}_v \in \mathbb{R}^d, \forall v \in G\})$$

Graph-level prediction



Node-level prediction

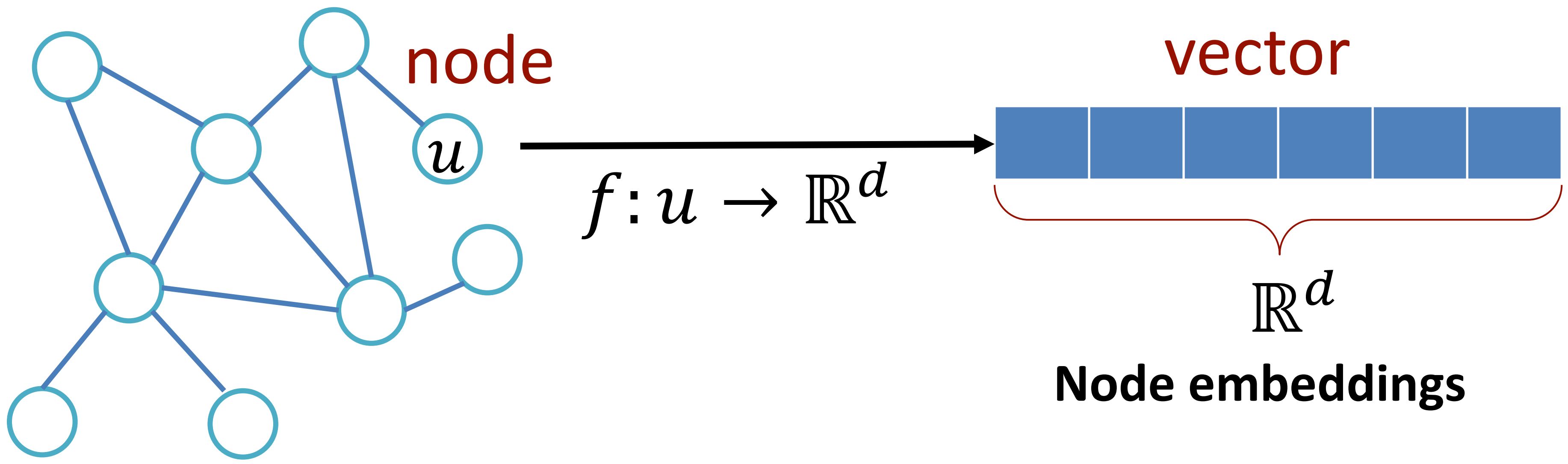
$$\hat{y}_v = \mathrm{Head}_{\mathrm{node}}(\mathbf{h}_v) = \mathbf{W} \;\; \mathbf{h}_v$$

$$\hat{y}_v = \mathrm{Head}_{\mathrm{node}}(\mathbf{h}_v) = \mathbf{W} \;\; \mathbf{h}_v$$

Edge-level prediction

Graph Learning – Node Embeddings

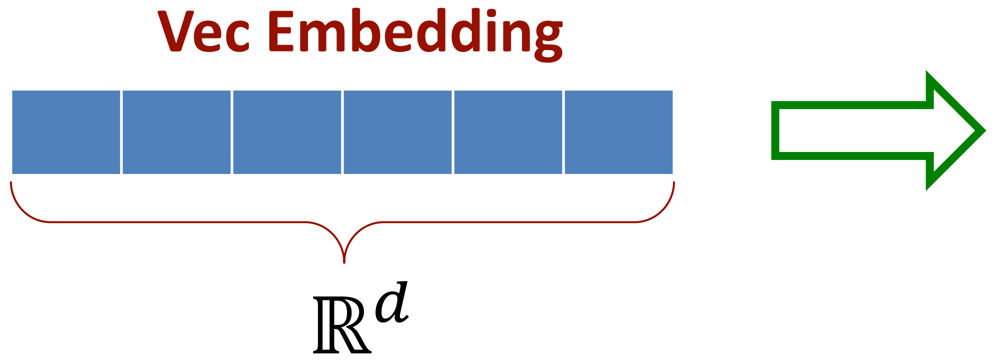
- **Goal**: Get low dimensional vector representation of nodes



- **Why study nodes, not edges?**
  - **Semantics**: usually nodes refers to the entities of interests (users, items, …)
  - **Scalability**: $O(N)$ vs $O(N^2)$
  - **Implementation**: No missing node in a graph; node pair -> edge, node set -> graph

Why Embedding?

- **Task: Map nodes into a vector 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 & compress **feature information**: node, edge, and graph attributes
  - Potentially used for many **downstream predictions**

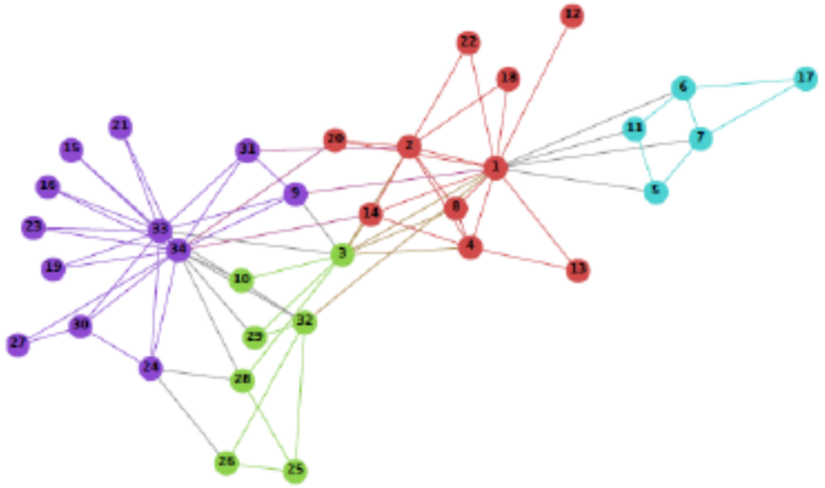


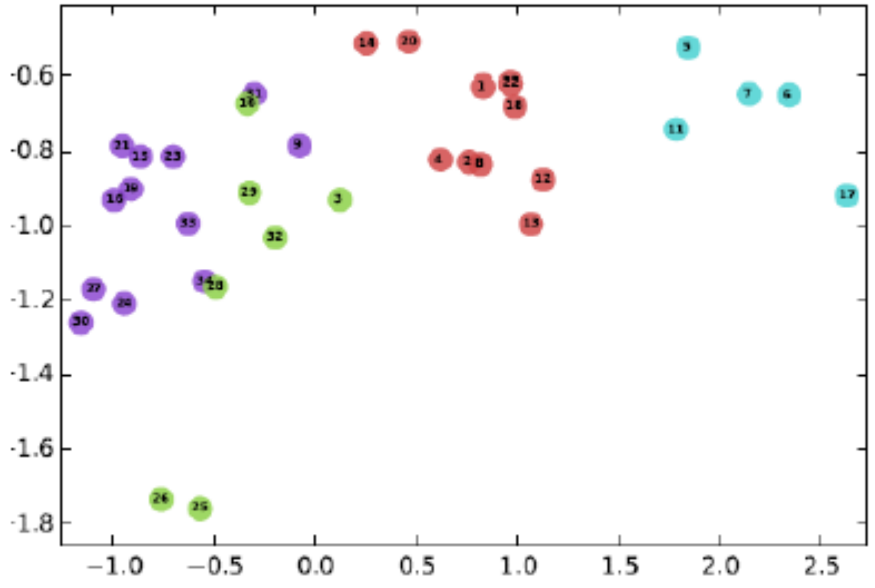
Tasks

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

Example Node Embedding

- **2D embedding of nodes of the Zachary's Karate Club network:**





Input

Output

Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. *KDD 2014.*

“Shallow” Graph Learning

Shallow Node Encoders

“Shallow” Encoding

- **Goal of node encoder**: Define $\mathrm{ENC}(v) = \mathbf{z}_v$
- Simplest encoding approach: **Encoder is just an embedding-lookup**

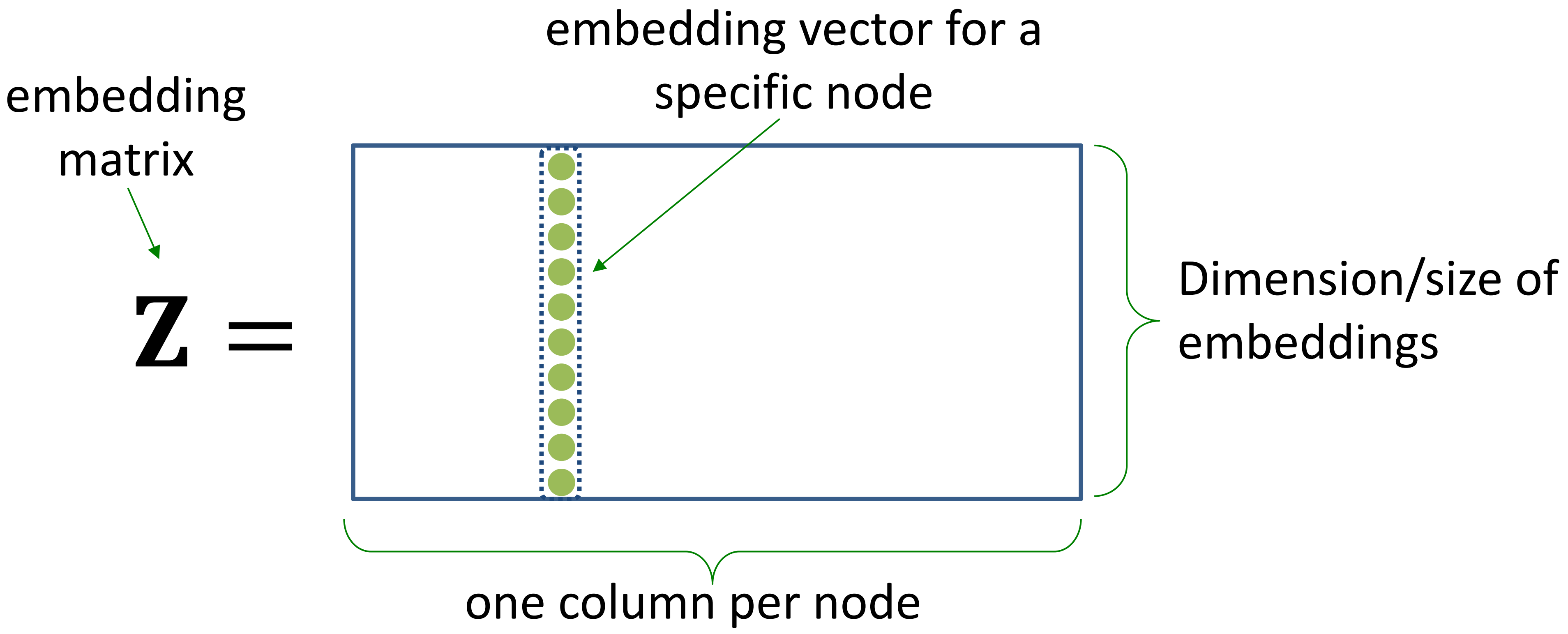
$$\mathrm{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ Embedding matrix, each column is a node embedding [will be optimized with chosen prediction heads]

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

“Shallow” Encoding

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


embedding matrix
$\mathbf{Z} =$
embedding vector for a specific node
Dimension/size of embeddings
one column per node

“Shallow” Encoding

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

Each node is assigned a unique embedding vector

(i.e., we directly optimize the embedding of each node)

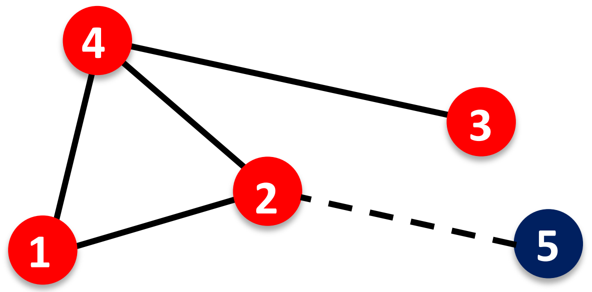
Many early graph ML methods: DeepWalk, node2vec

Shallow Encoding: Benefits

- Simplest encoding approach: **Encoder is just an embedding-lookup**
- **Benefits:**
  - Efficient
    - Easily scale to web-scale data (billions of nodes/edges)
    - Does not rely on fancy deep learning frameworks and hardware
  - Easy to implement: basic Python/C++
  - Expressive – each node can have a unique representation
    - We will visit the concept of expressive power later in the course
  - Therefore, shallow embedding based methods are still widely adopted in industry (even beyond the scope of graph ML)

Shallow Encoding Limitations (1)

- **Limited generalization:** Cannot obtain meaningful embeddings for **unobserved/new nodes** (nodes not in the training set)
  - Can only assigned default/random embeddings to new nodes



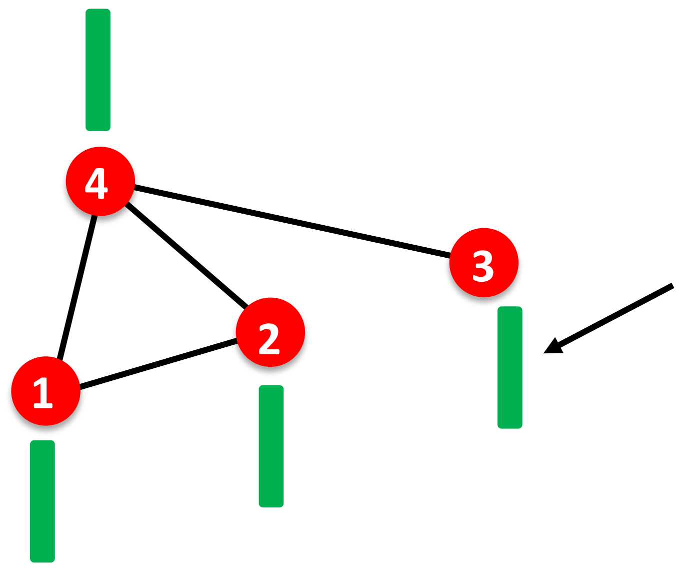
Red nodes: Training set
Blue node: Test set

A newly added node 5 at test time (e.g., new user in a social network)
Have to assign random/default embedding to the new node

- **Implication:** Shallow encoders cannot be used when **new nodes occur**
  - ☹ **Node-level tasks when graph structure changes, graph-level tasks**
  - ☺ **Node/edge-level tasks when graph structure do not change**

Shallow Encoding Limitations (2)

- **No feature information:** Cannot utilize node, edge and graph features



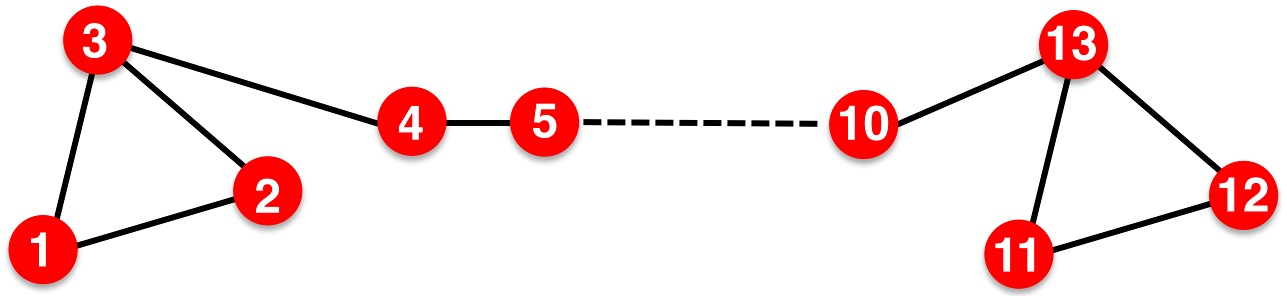
Feature vector
(e.g. protein properties in a protein-protein interaction graph)

Shallow encoder do not incorporate such node features
(Note: you may concatenate node features, but that’s not “encoded”)

- **Solution to these limitations:**
- **Deep Graph Learning - Graph Neural Networks** – major focus of this course later

Shallow Encoding Limitations (3)

- Does not retain **structural information**



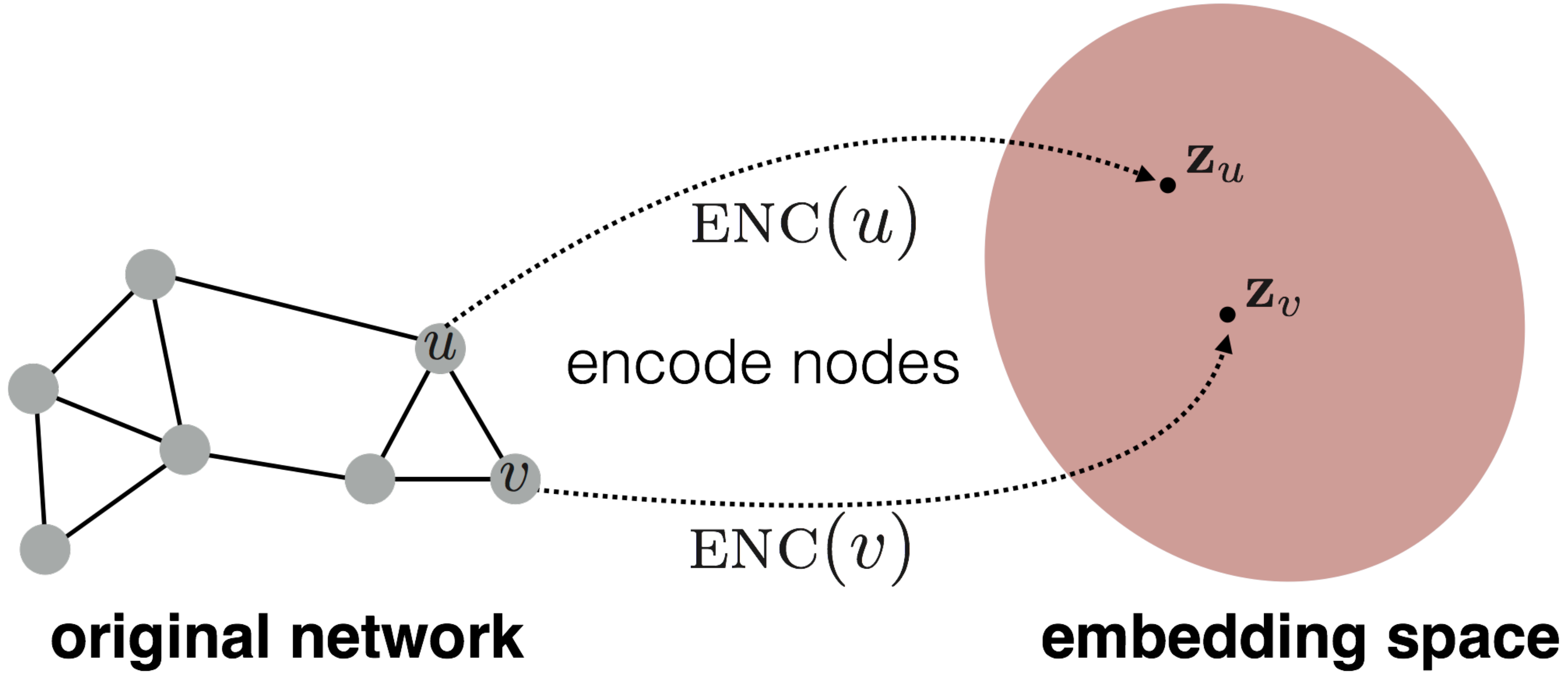
  - Node 1 and 2 are **positional similar** – direct neighbors
  - Node 1 and 11 are **structurally similar** – part of one triangle, degree 2, …
- Remember that the shallow embedding matrix is **trainable**
- **Solution: Define proper objective function to train embedding matrix!**

“Shallow” Graph Learning

Similarity Based Objective Function

Similarity Based Objective Function

- **Objective function:** encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph



Similarity Based Objective Function

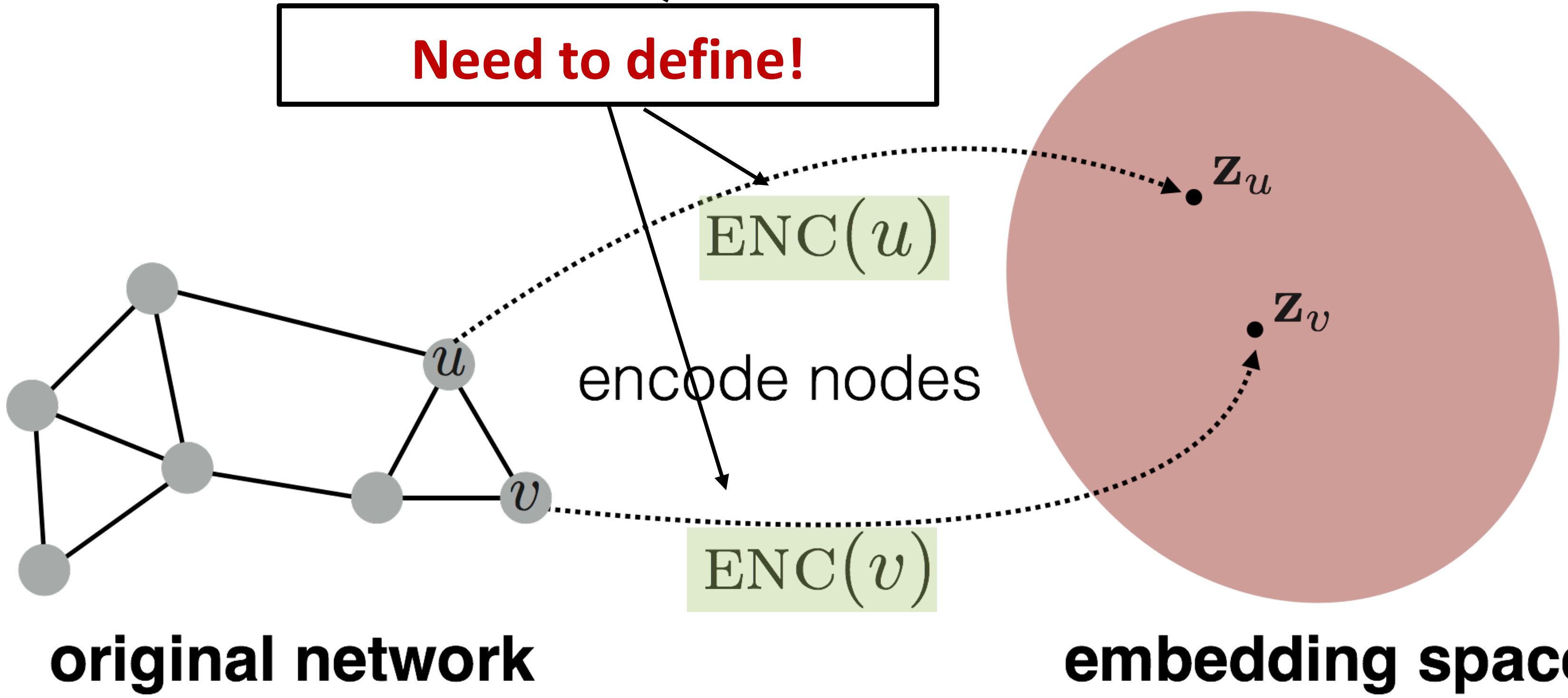
- Requires an **edge-level prediction head**

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

in the original network

similarity of the embedding

Need to define!



Similarity Based Objective Function: Pipeline

1. **Shallow Encoder** maps from nodes to embeddings
2. **Define a node similarity function** (i.e., a measure of similarity in the original network)
3. **Edge-level prediction head** maps from embeddings to the similarity score: $\hat{\boldsymbol{y}}_{\boldsymbol{uv}} = \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$
4. **Optimize the parameters of the encoder so that:**

   maximize $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$ for node pairs (u, v) that are **similar**

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

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 Similarity

- 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.

“Shallow” Graph Learning

Random Walk based Node Similarity

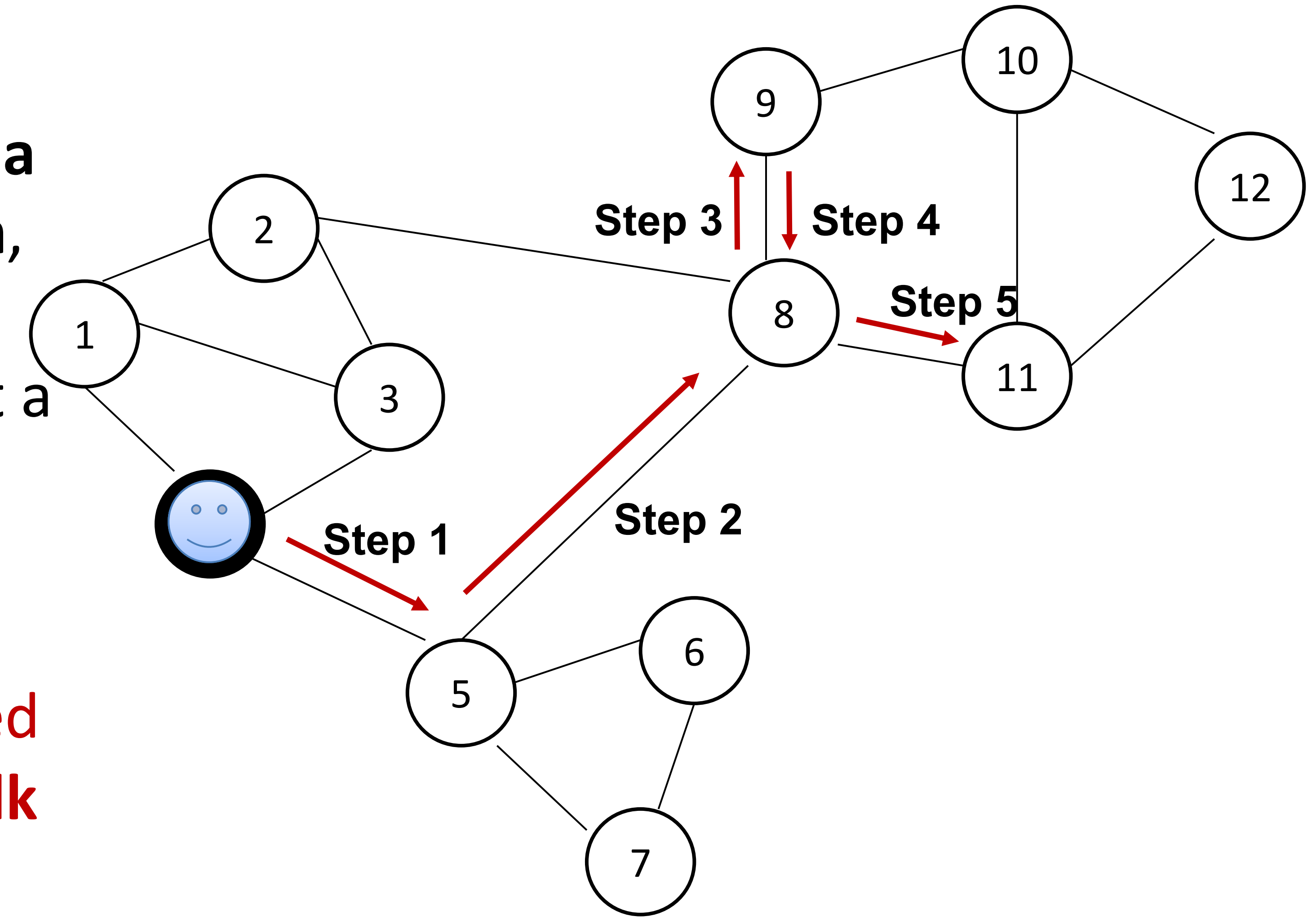
Notation

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

- **Softmax** function:
  - Turns vector of K real values (model predictions) into K probabilities that sum to 1: $\sigma(\boldsymbol{z})[i] = \frac{e^{z[i]}}{\sum_{j=1}^{K} e^{z[j]}}$
- **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

- Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.



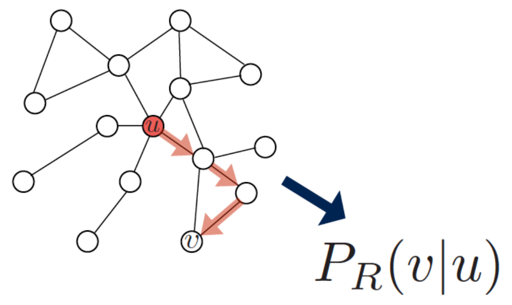
Random-Walk Embeddings

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

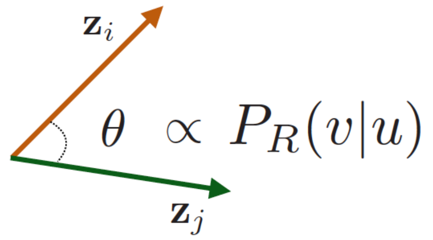
probability that *u* and *v* co-occur on a random walk over the graph

Random-Walk Embeddings

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



- **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:** 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

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 $f: u \to \mathbb{R}^d$:
  $f(u) = \mathbf{z}_u$
- Log-likelihood objective:

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

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

Random Walk Optimization

1. Run **short fixed-length random walks** starting from each node u in the graph using some random walk strategy R.
2. For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u.
3. Optimize embeddings according to: Given node u, predict its neighbors $N_{\mathrm{R}}(u)$.

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \, \mathbf{z}_u) \Longrightarrow \text{Maximum likelihood objective}$$

*$N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks

Random Walk Optimization

- More derivation:

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

- **Intuition:** Optimize embeddings $\boldsymbol{z}_u$ to maximize the likelihood of random walk co-occurrences.

- **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: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

Random Walk Optimization

- **Putting it all together:**

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

sum over all nodes u

sum over nodes v seen on random walks starting from u

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

Optimizing random walk embeddings =Finding embeddings z_u that minimize $\mathcal{L}$

Random Walk Optimization

- But doing this naively is too expensive!

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

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

Random Walk Optimization

- But doing this naively is too expensive!

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

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

Negative Sampling

- **Solution:** Negative sampling

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

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

sigmoid function

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

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 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

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

- Negative sampling allows for quick likelihood calculation.

Negative Sampling

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

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

random distribution over nodes

- Sample k negative nodes n_i each with prob. proportional to its degree.
- Two considerations for k (# negative samples):
  1. Higher k gives more robust estimates
  2. Higher k corresponds to higher bias on negative events

  In practice k =5-20.

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.

Stochastic Gradient Descent

- After obtaining the objective function, how do we optimize (minimize it?

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

- **Stochastic Gradient Descent**: evaluating gradient for individual training examples(s) – basis of deep learning
  - Initialize z_u at some randomized value for all nodes u.
  - Iterate until convergence:
    - Sample a node u, for all v calculate the derivative $\frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$.
    - For all v, update: $z_v \leftarrow z_v - \underset{\uparrow}{\eta} \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$.

η: learning rate

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 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!

Other Random Walk Ideas

- **Different kinds of random walks:**
  - node2vec: Proposing the idea of biased random walks (Grover et al., 2016)
  - 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).

Summary so far

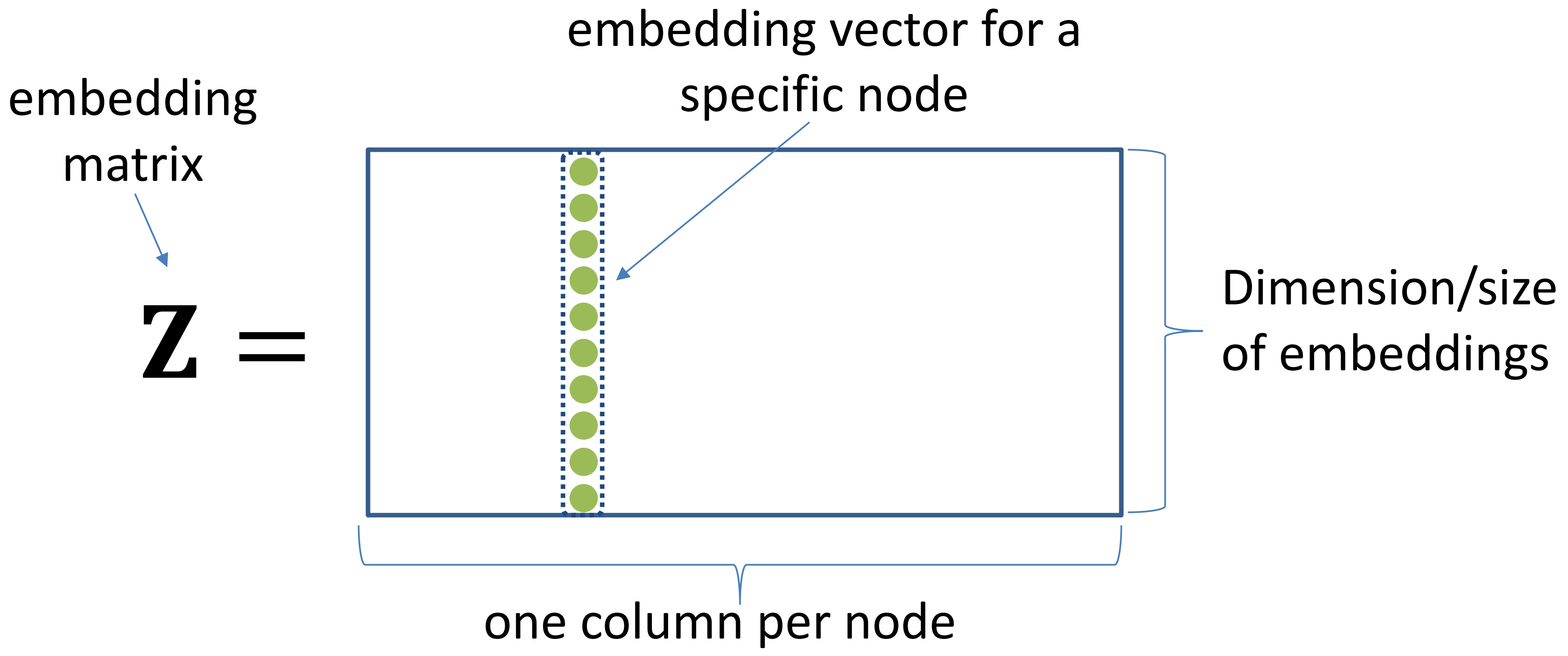
- **Core idea:** Embed nodes so that distances in embedding space reflect node similarities in the original network.
- **Different notions of node similarity:**
  - Naïve: Similar if two nodes are connected
  - Neighborhood overlap
  - Random walk approaches
- **So what method should I use..?**
  - No one method wins in all cases….
  - Random walk approaches are generally more efficient.
- **In general:** Must choose definition of node similarity that matches your application.

“Shallow” Graph Learning

Matrix Factorization and Random Walks

Embeddings & Matrix Factorization

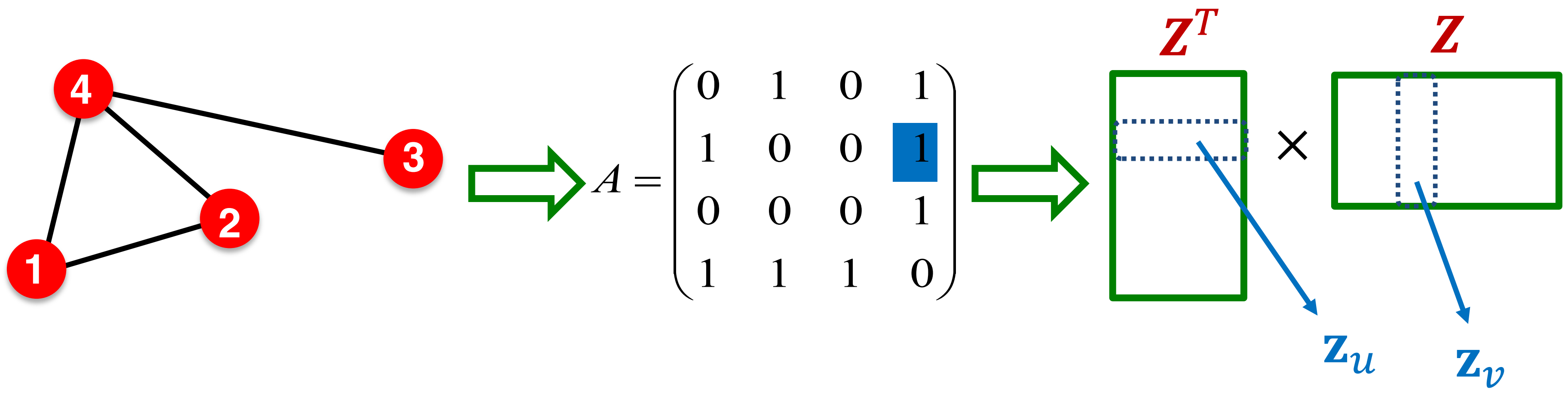
- **Recall:** encoder as an embedding lookup


embedding matrix
$\mathbf{Z} =$
embedding vector for a specific node
Dimension/size of embeddings
one column per node

Objective: maximize $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ for node pairs (u, v) that are **similar**

Connection to 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, $\boldsymbol{Z}^T \boldsymbol{Z} = A$



4
3
2
1
$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & \boxed{1} \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$
$\boldsymbol{Z}^T$
$\boldsymbol{Z}$
$\times$
$\mathbf{z}_u$
$\mathbf{z}_v$

Matrix Factorization

- The embedding dimension d (number of rows in $\boldsymbol{Z}$) is much smaller than number of nodes n.
- Exact factorization $A = \boldsymbol{Z}^T\boldsymbol{Z}$ is generally not possible
- However, we can learn $\boldsymbol{Z}$ approximately
- **Objective**:$\min_{\mathbf{Z}} \parallel \mathrm{A} - \boldsymbol{Z}^T\boldsymbol{Z} \parallel_2$
  - We optimize $\boldsymbol{Z}$ such that it minimizes the L2 norm (Frobenius norm) of $\mathrm{A} - \boldsymbol{Z}^T\boldsymbol{Z}$
  - Note today we used softmax instead of L2. But the goal to approximate A with $\boldsymbol{Z}^T\boldsymbol{Z}$ is the same.
- Conclusion: **Inner product decoder with node similarity defined by edge connectivity is equivalent to matrix factorization of A.**

Random Walk-based Similarity

- **DeepWalk** has a more complex **node similarity** definition based on random walks
- **DeepWalk** is equivalent to matrix factorization of the following complex matrix expression:

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

  - Explanation of this equation is on the next slide.

Random Walk-based Similarity

Volume of graph

$vol(G) = \sum_i \sum_j A_{i,j}$

Diagonal matrix D

$D_{u,u} = \deg(u)$

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

context window size

$T = |N_R(u)|$

Power of normalized adjacency matrix

Number of negative samples

- Refer to the paper for more details

How to Use Embeddings

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

Today's Summary

We discussed "shallow" **graph representation learning**

- **Shallow node encoders**
  - **Encoder: embedding lookup**
  - **Objective: predict score based on embedding to match node similarity**

- **Node similarity measure: random walk**
  - **Examples: DeepWalk, node2Vec**