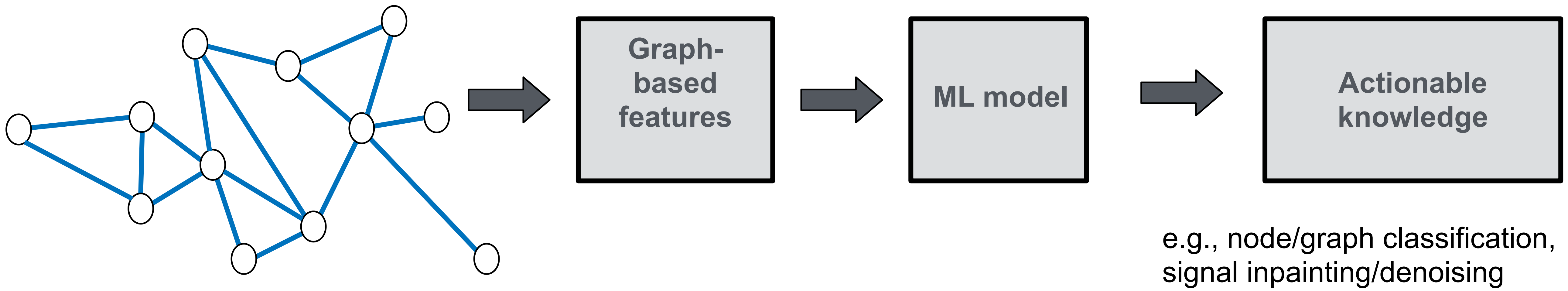
Learning unsupervised embeddings on graphs

Dr Dorina Thanou
18.03.2025

Recap: Traditional ML pipeline on graphs

- **Intuition:** The effectiveness of ML techniques on graphs relies on a good representation (feature set) of data



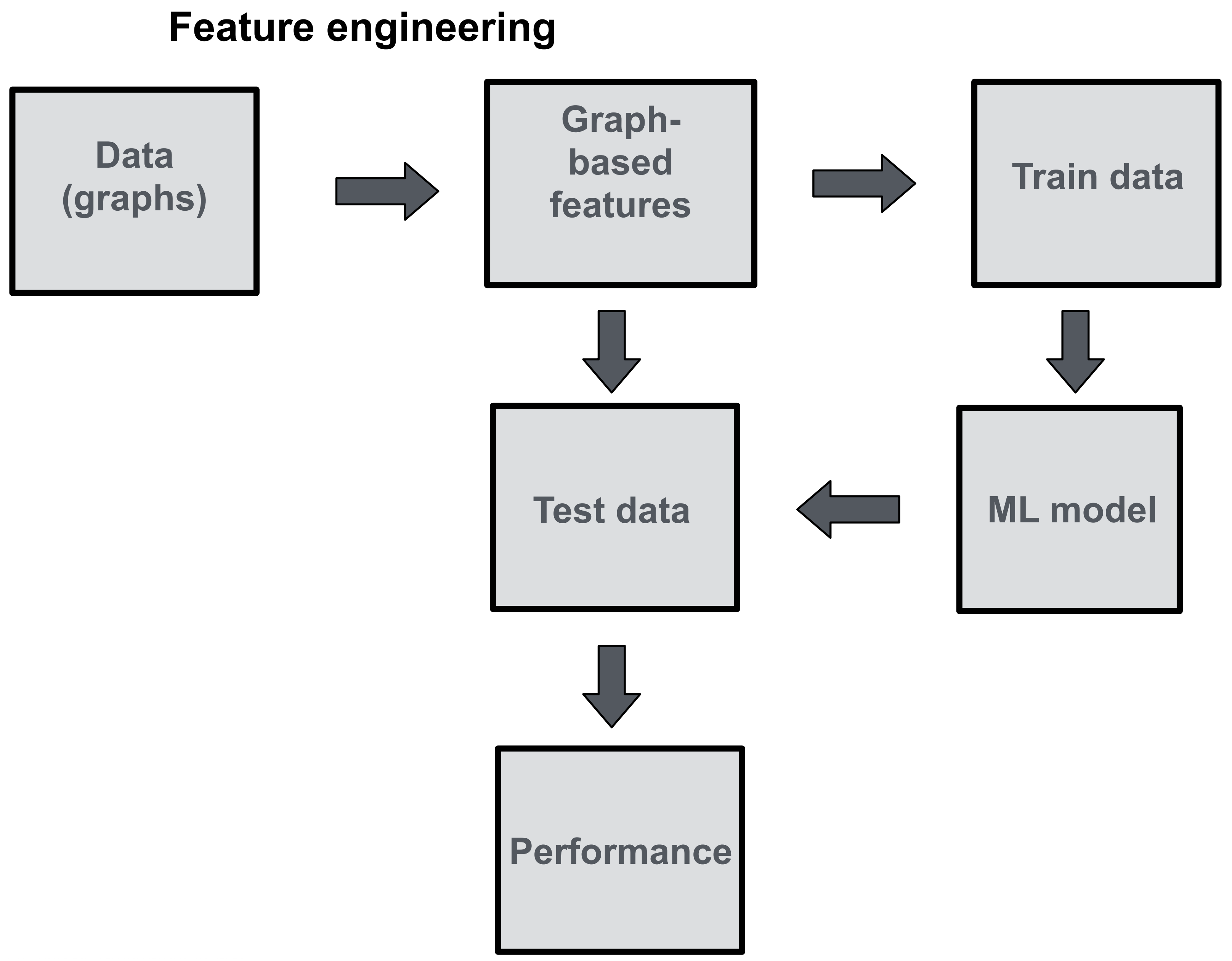
e.g., node/graph classification, signal inpainting/denoising

Feature engineering

$$\begin{tikzcd}\mathcal{G} \arrow[r] & \phi(\mathcal{G}) \arrow[r] & f(\phi(\mathcal{G})) \arrow[r] & Y\end{tikzcd}$$

Recap: Traditional ML pipeline on graphs

- Feature engineering is a way of extracting meaningful information from graphs



Recap: Hand-crafted features on graphs

Graph-based features

$\phi(\mathcal{G})$

- Hand-crafted features are a way to extract discriminative information from graph data by exploiting prior/domain knowledge

- The type of features depends on the final task:
  - **Node level:** generate features for each individual node
    - Node degree, centrality, clustering coefficients, graphlets
  - **Graph level:** generate features for the whole graph
    - Bag of nodes, graphlet kernels, WL kernels
  - **Link level:** generate features that measure a common neighborhood between two nodes
    - Local/global neighborhood overlap

Limitations of hand-crafted features

- Their discriminative power is limited by the effectiveness of the priors: they cannot capture graph patterns different than their design prior
- Hand-crafted features exhibit poor generalization performance across different datasets/graphs
- Real-world network phenomena are usually highly complicated: they require complex and unknown combinations of well-known features

How can we use data to obtain more flexible graph features?

Outline

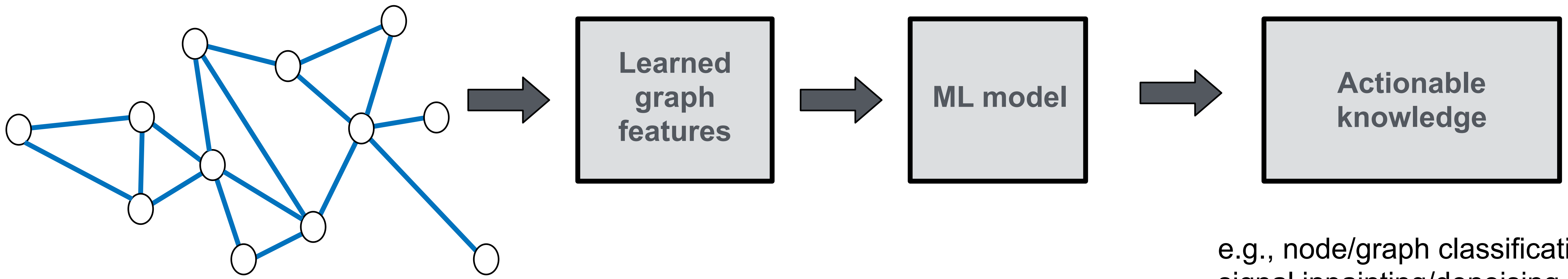
- Graph representation learning
- Unsupervised graph embedding algorithms
- Illustrative applications

Outline

- **Graph representation learning**
- Unsupervised graph embedding algorithms
- Illustrative applications

Graph representation learning

- **Intuition:** Optimize the feature extraction part by adapting it to the specific instances of the graphs/data



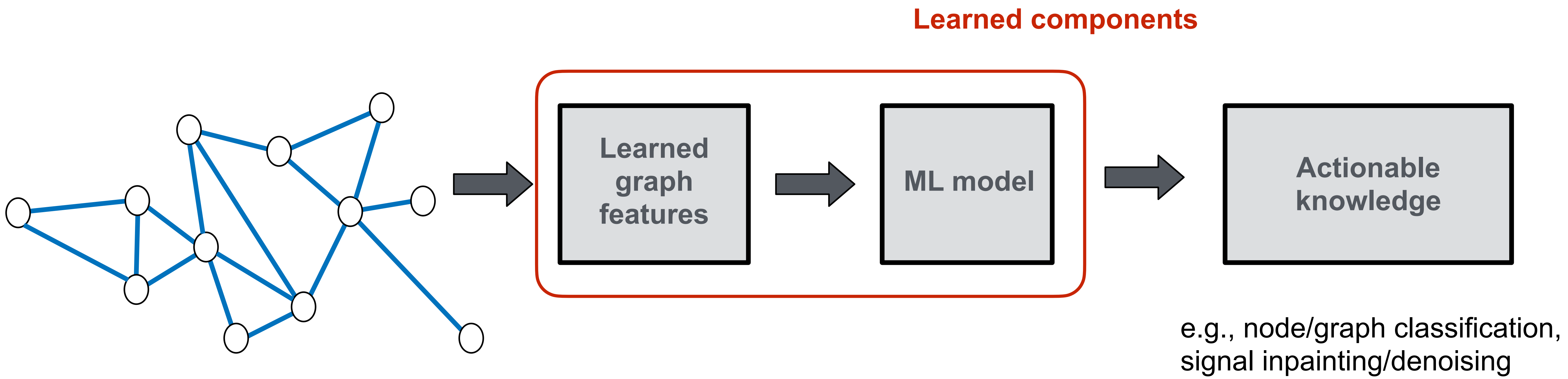
e.g., node/graph classification, signal inpainting/denoising

Feature Learning

$$\mathcal{G} \longrightarrow \phi(\mathcal{G}) \longrightarrow f(\phi(\mathcal{G})) \longrightarrow Y$$

Graph representation learning

- **Intuition:** Optimize the feature extraction part by adapting it to the specific instances of the graphs/data

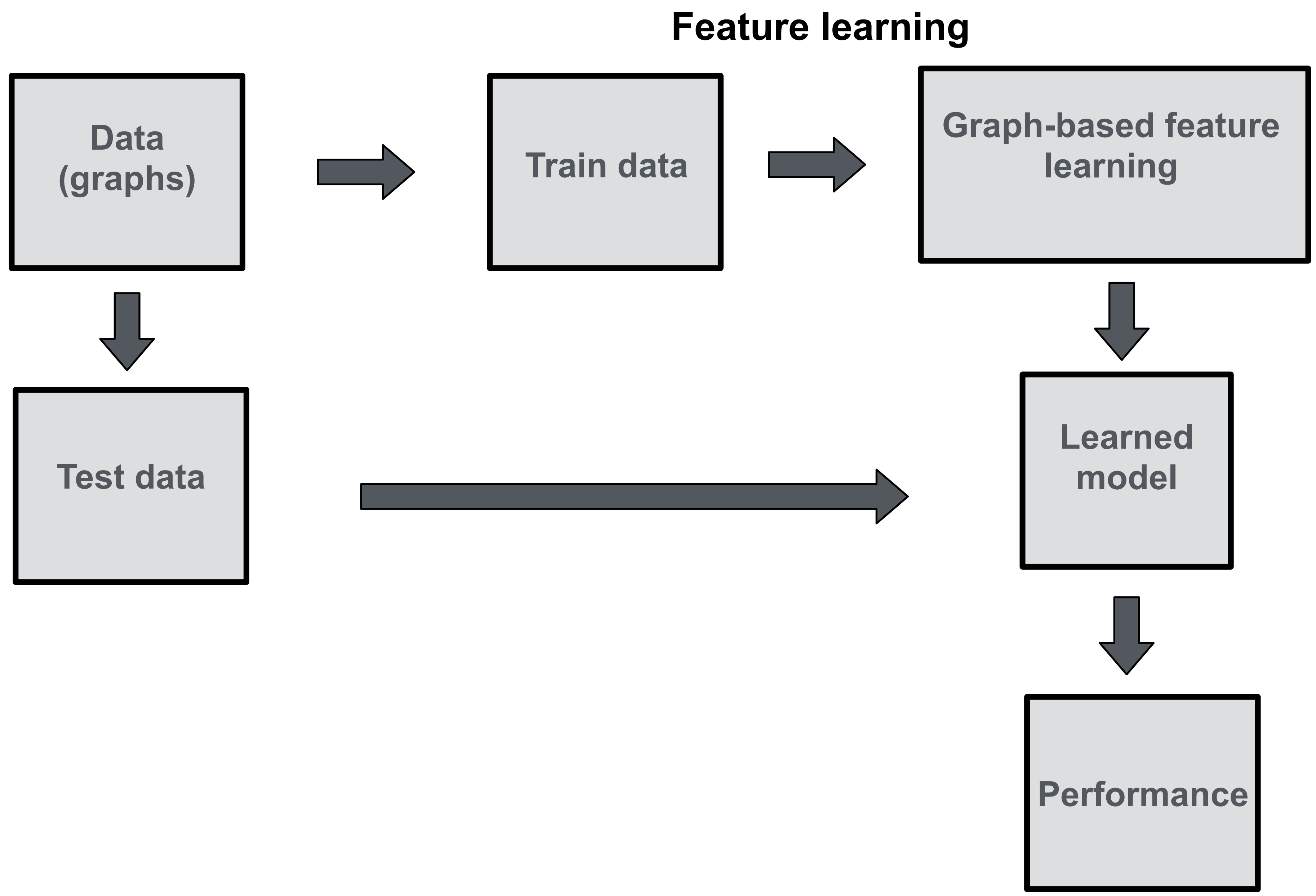


Feature Learning

$$\mathcal{G} \longrightarrow \phi(\mathcal{G}) \longrightarrow f(\phi(\mathcal{G})) \longrightarrow Y$$

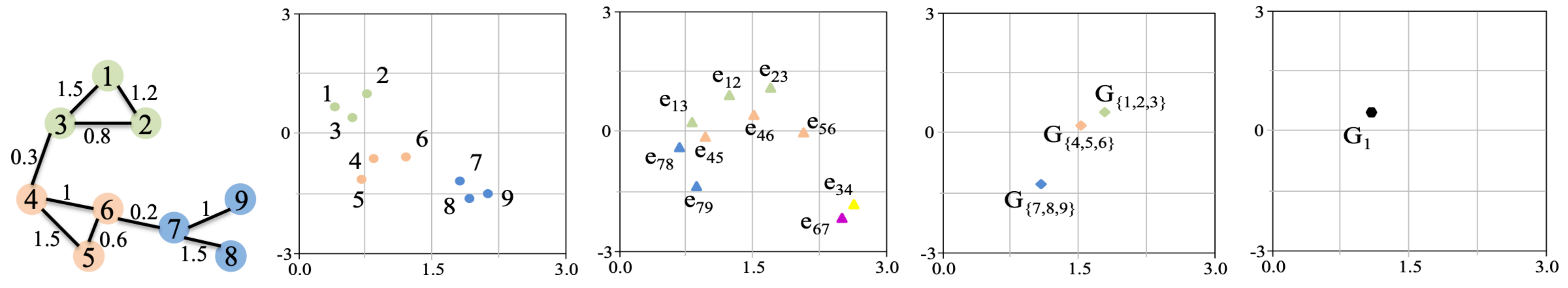
Graph representation learning: basic pipeline

- Feature learning is a way of extracting data-adaptive graph representations



Learning features on graphs

- Learned features convert the graph data in a (low dimensional) latent space (i.e., **embedding space)** where hidden/discriminative information about data is revealed



Original space

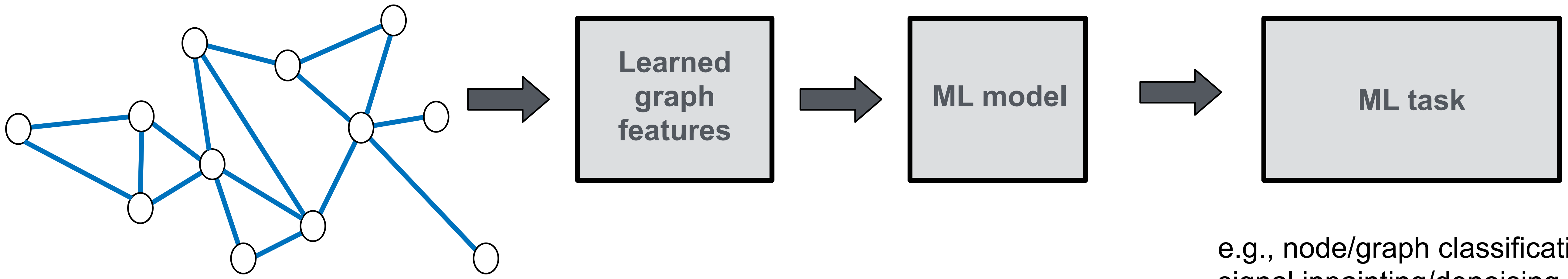
Node embedding **Edge embedding** **Subgraph embedding** **Graph embedding**

Embedding space

How can we learn the embedding space?

Supervised graph representation learning

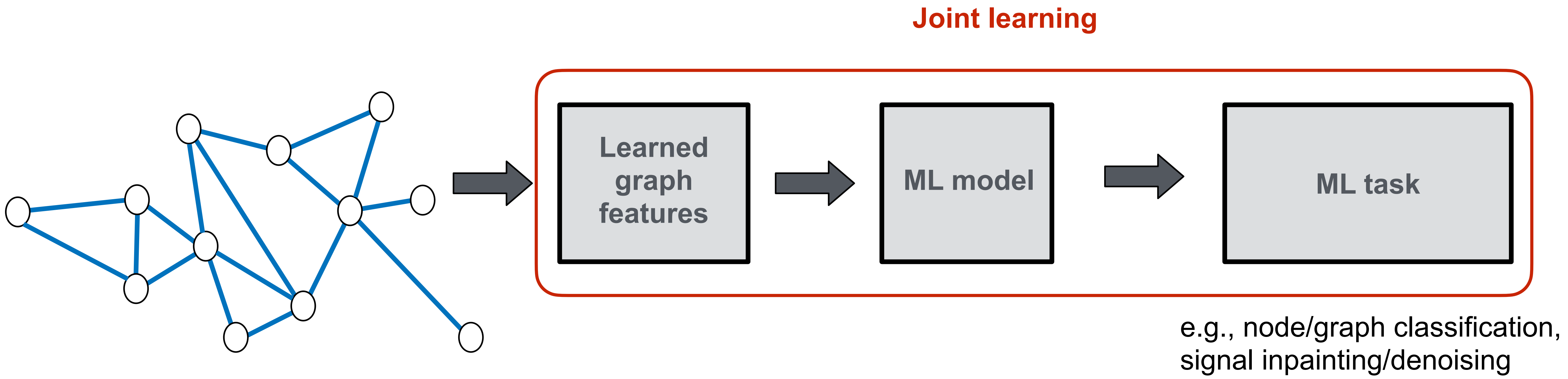
- Learn low-dimensional embeddings for a specific downstream task, e.g., node or graph classification



e.g., node/graph classification, signal inpainting/denoising

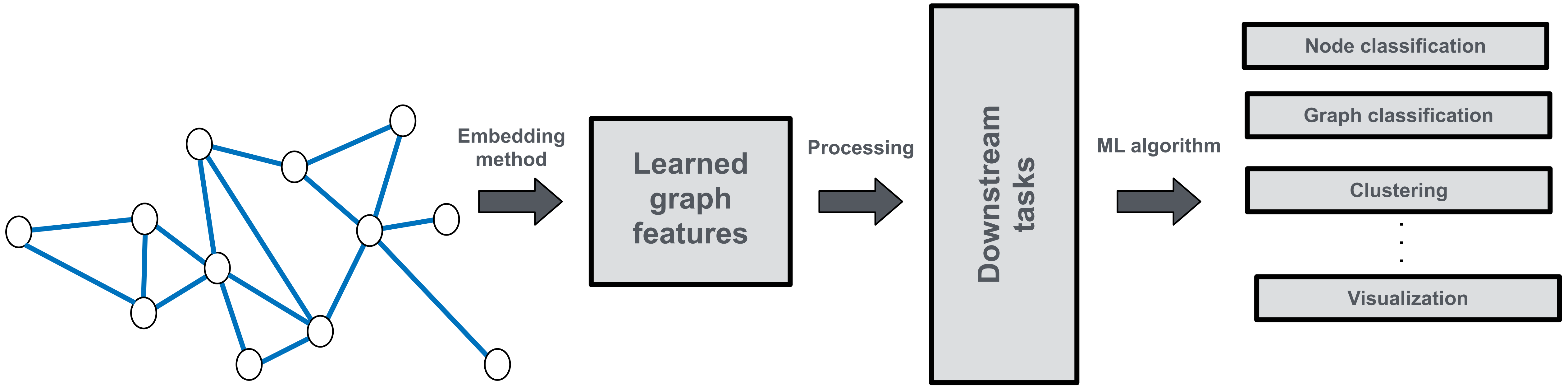
Supervised graph representation learning

- Learn low-dimensional embeddings for a specific downstream task, e.g., node or graph classification


Joint learning
Learned graph features
ML model
ML task
e.g., node/graph classification, signal inpainting/denoising

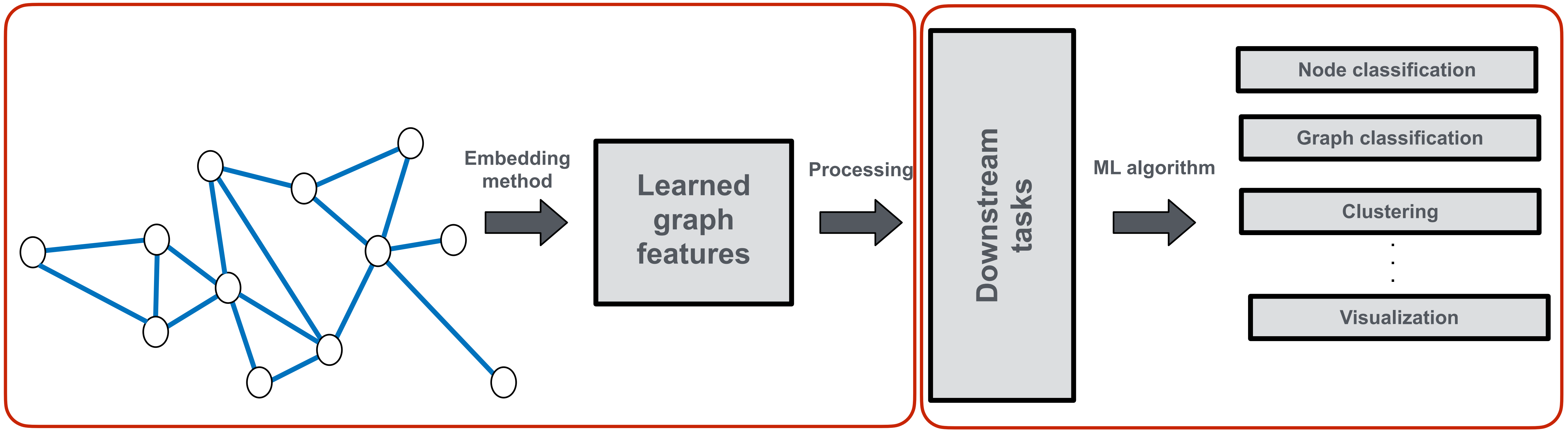
Unsupervised graph representation learning

- Learn low-dimensional embeddings that are not optimized for a specific downstream task
  - They are optimized with respect to some notion of "closeness" in the graph
  - The notion of "closeness" defines the design of the embedding algorithm



Unsupervised graph representation learning

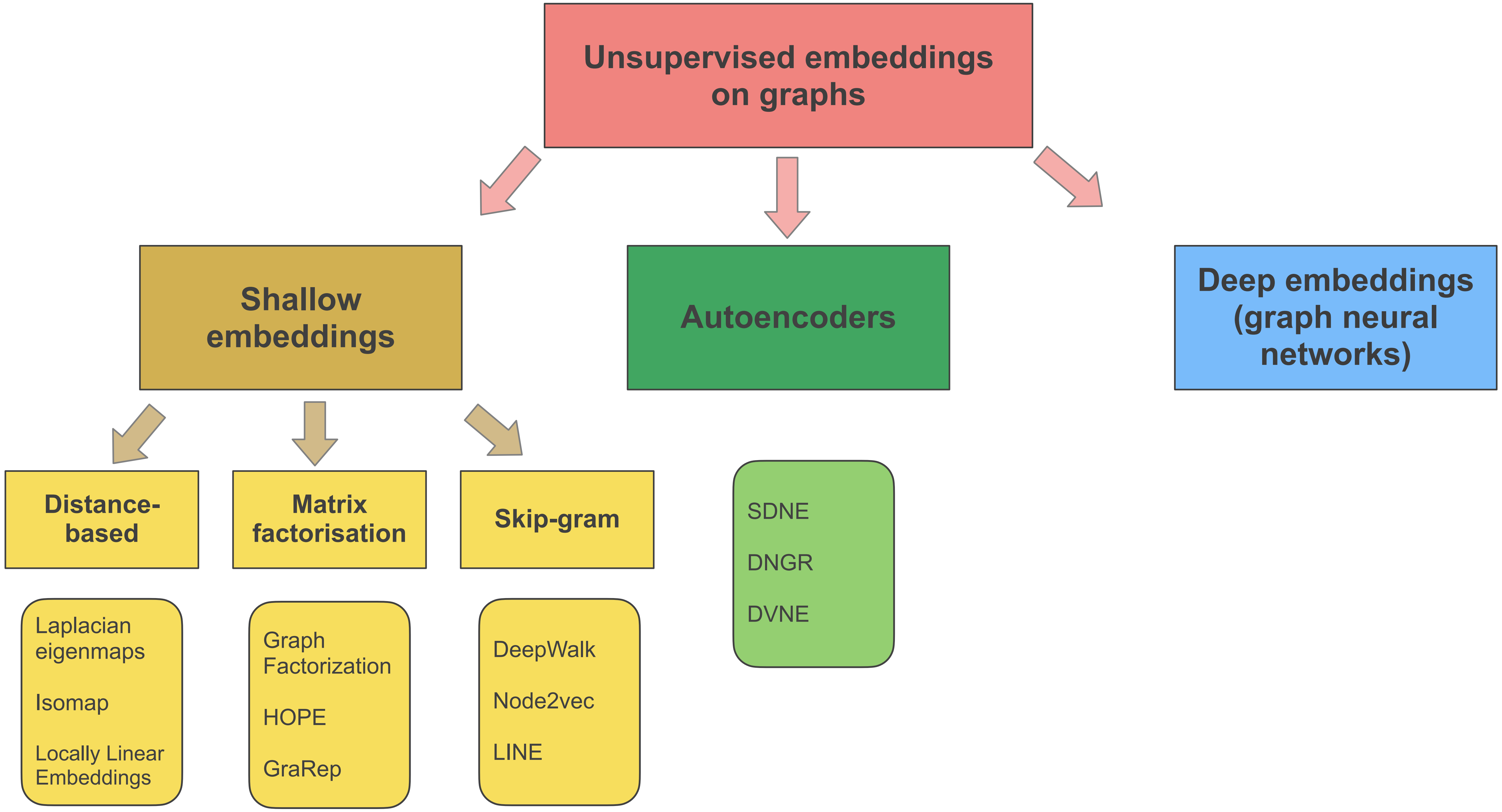
- Learn low-dimensional embeddings that are not optimized for a specific downstream task
  - They are optimized with respect to some notion of “closeness” in the graph
  - The notion of “closeness” defines the design of the embedding algorithm



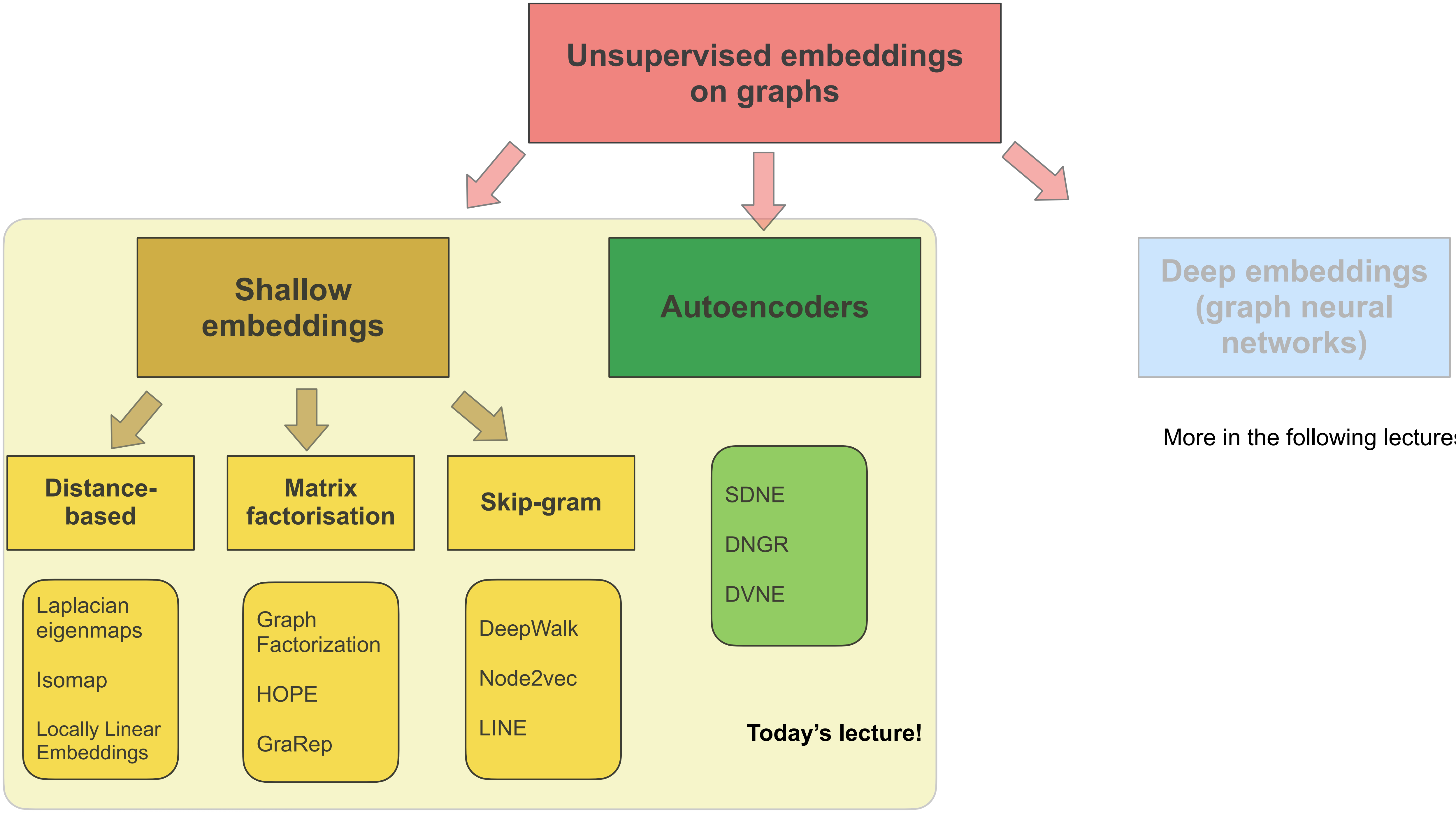
Step 1: Generate embeddings

Step 2: Validate embeddings

Learning unsupervised embeddings on graphs: A (partial) taxonomy



Learning unsupervised embeddings on graphs: A (partial) taxonomy



Outline

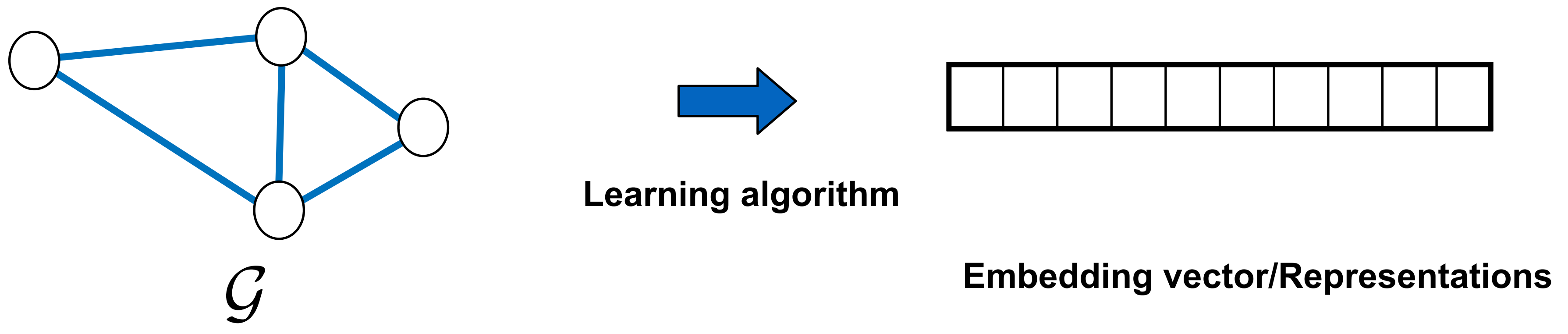
- Graph representation learning

- **Unsupervised graph embedding algorithms**
  - **Shallow embeddings**
  - **Autoencoders**

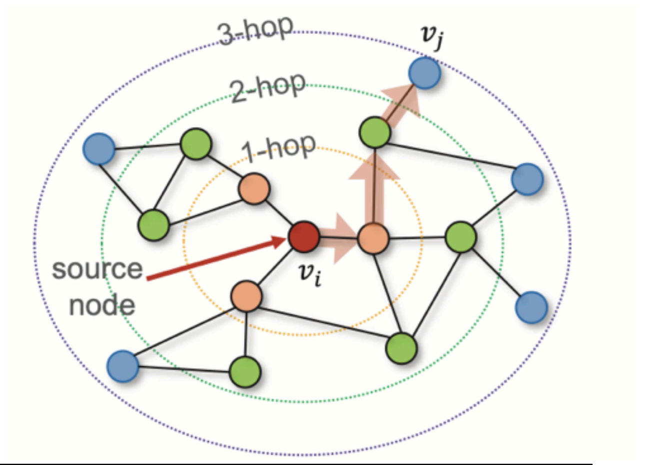
- Illustrative applications

Embeddings on graphs: Definition

- Given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, and a predefined dimensionality of the embedding $d << |\mathcal{V}|$, the goal is to convert $\mathcal{G}$ (or a subgraph, or a node) into a $d-$dimensional space in which graph properties are preserved

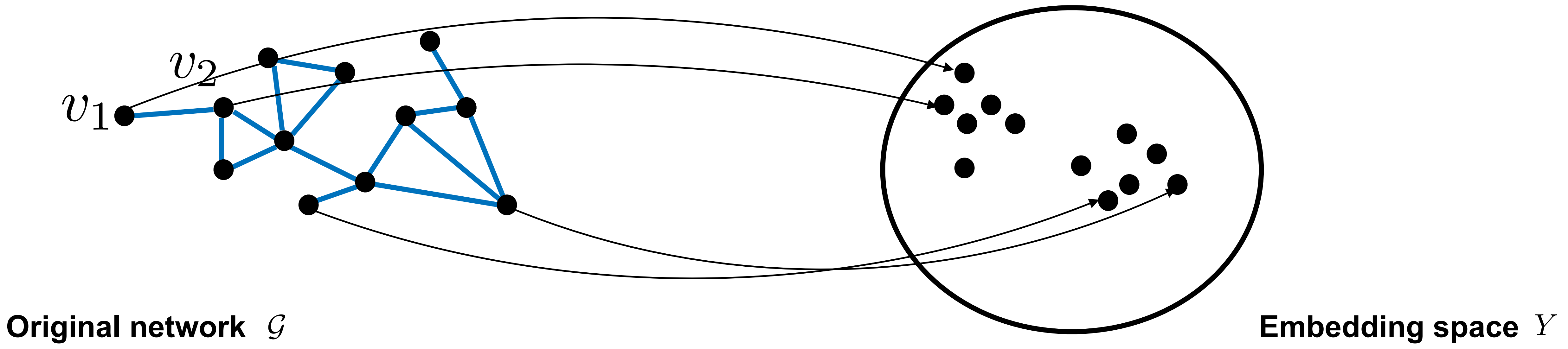


- Graph properties can be quantified using proximity measures on the graph (e.g., $K-$hop neighborhood)



Illustrative example: Node embeddings

- Prior 1: Neighbors on the graph should have similar embeddings (homophily)

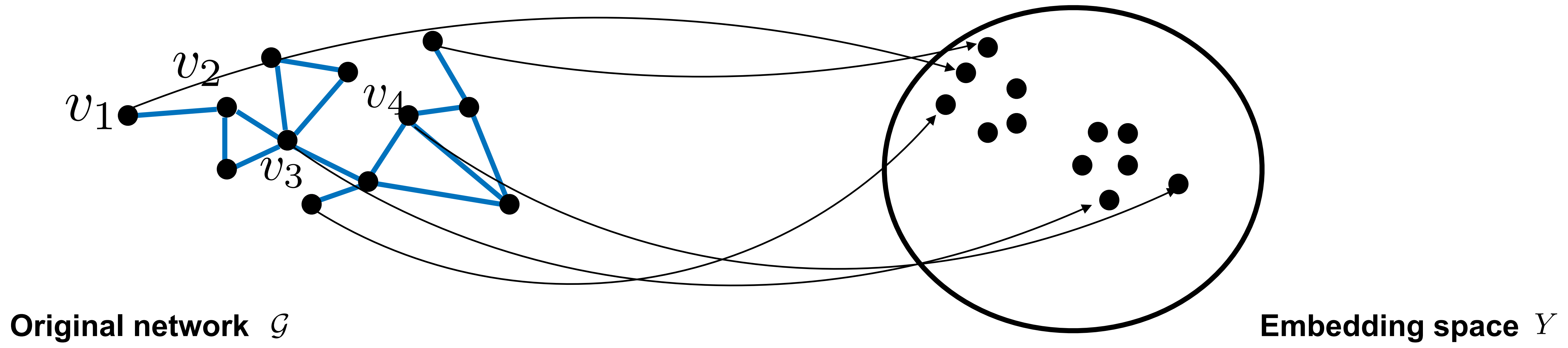


What is the similarity in the graph that should be preserved in the embedding space?

$$sim_{\mathcal{G}}(v_1, v_2) \approx sim_Y(Y_1, Y_2)$$

Illustrative example: Node embeddings

- Prior 2: Nodes on the graph with the same structural role (e.g., hubs) should have similar embeddings (structural equivalence)



What is the similarity in the graph that should be preserved in the embedding space?

$$sim_{\mathcal{G}}(v_3, v_4) \approx sim_Y(Y_3, Y_4)$$

Which graph property should be preserved?

- The choice depends on the application, and the questions we ask about the network

- Widely used examples are the following:

  - 1-hop neighborhood structure
  - high-order neighborhood structure
  - community structures
  - …

- Node embeddings algorithms differ on how to capture the graph structure to be preserved in the embedding

Encoder-decoder framework

- **Encoder:** maps nodes $\mathcal{V}$ to an embedding matrix $Y \in \mathbb{R}^{|\mathcal{V}| \times d}$
- **Similarity function** $sim_{\mathcal{G}}(\cdot, \cdot)$: specifies the similarity between nodes that should be preserved in the original graph
- **Decoder:** maps embeddings Y to a similarity score in the embedding space $sim_Y(\cdot, \cdot)$
- **Learning objective:** Design encoder-decoder such that:

$$sim_{\mathcal{G}}(v_1, v_2) \approx sim_Y(Y_1, Y_2)$$

Optimizing an encoder-decoder model

- Define a loss function $l : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ that measures that discrepancy between the similarity values in the embedding space (decoder), and the similarity between nodes in the original graph

- Learn the embeddings by **minimizing the empirical reconstruction loss** over a set of training data

$$loss = \sum_{(v_i, v_j) \in N_{train}} l(sim_{\mathcal{G}}(v_i, v_j), sim_Y(Y_i, Y_j))$$

- Different choices of $l(\cdot,\cdot),\ sim_{\mathcal{G}}(\cdot,\cdot),\ sim_Y(\cdot,\cdot)$ define different embedding algorithms

Example 1: Laplacian Eigenmaps

- **Intuition:** Preserve pairwise node similarities derived from the adjacency/weight matrix

$$sim_{\mathcal{G}}(v_i, v_j) = W_{ij}$$

- Measure similarity in the embedding space using the mean square error

$$sim_Y(Y_i, Y_j) = \|Y_i - Y_j\|_2^2$$

- Impose larger penalty if two nodes with larger pairwise similarity are embedded far apart

$$\begin{aligned} l(sim_{\mathcal{G}}(v_i, v_j), sim_Y(Y_i, Y_j)) &= sim_{\mathcal{G}}(v_i, v_j) \cdot sim_Y(Y_i, Y_j) \\ &= W_{ij}\|Y_i - Y_j\|_2^2 \end{aligned}$$

Laplacian Eigenmaps: Algorithm

- Compute embeddings that minimize the expected square distance between connected nodes

Centered embeddings

Uncorrelated embedding coordinates

$$\min_{Y \in \mathbb{R}^{N\times K}: Y^T 1 = 0; Y^T Y = I_K} \sum_{(i,j)\in\mathcal{E}} W_{ij} \|Y_i - Y_j\|^2$$

$$L = D - W \Downarrow$$ **Graph smoothness**

$$\min_{Y \in \mathbb{R}^{N\times K}: Y^T 1 = 0; Y^T Y = I_K} \operatorname{tr}(Y^T L Y)$$

$$\Downarrow$$ **Lagrangian**

$$\min_{Y \in \mathbb{R}^{N\times K}; Y^T 1 = 0} \operatorname{tr}(Y^T L Y - (Y^T Y - I_K)\Gamma)$$

$$\Downarrow$$ **Gradient**

$$LY = Y\Gamma \quad \Rightarrow \quad \boxed{u_i \rightarrow (\chi_2(i), ..., \chi_{K+1}(i))}$$

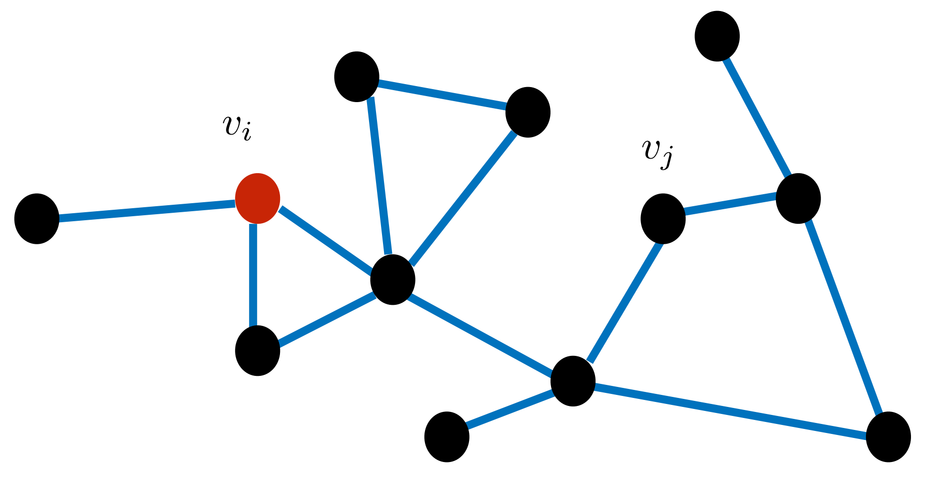
Laplacian Eigenmaps: K first non-trivial eigenvectors of the Laplacian!

[Belkin et al, 2003, Laplacian Eigenmaps for Dimensionality Reduction and Data Representation, Neural Comp.]

Example 2: Random walk embeddings

- **Intuition:** Preserve neighborhood structure i.e., higher order similarities, captured by random walks

- Random walk: A random process starting from a node that describes a path that consists of a succession of random steps on a graph (DFS)



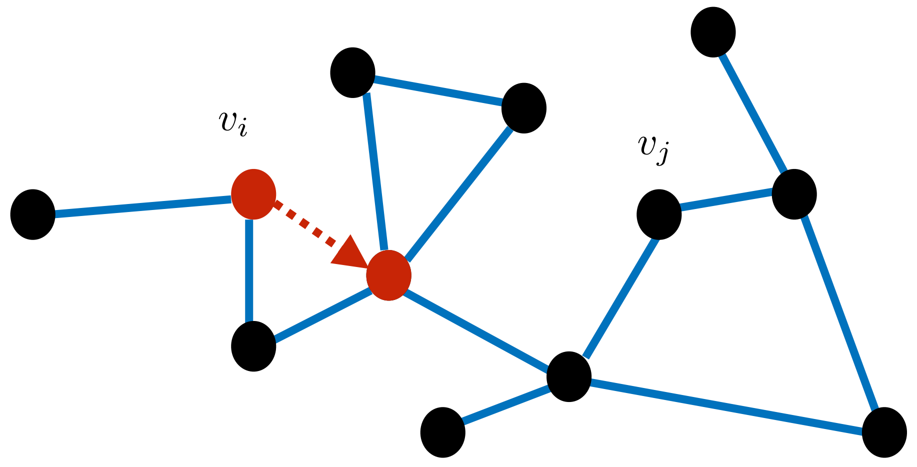
$p(v_j|v_i)$: probability of visiting node v_j on a random walk starting from a node v_i

$\mathcal{N}_{v_i|RW} = \{v_j \in RW\}$: neighbourhood of v_i obtained by some random walk strategy starting from v_i

Example 2: Random walk embeddings

- **Intuition:** Preserve neighborhood structure i.e., higher order similarities, captured by random walks

- Random walk: A random process starting from a node that describes a path that consists of a succession of random steps on a graph (DFS)



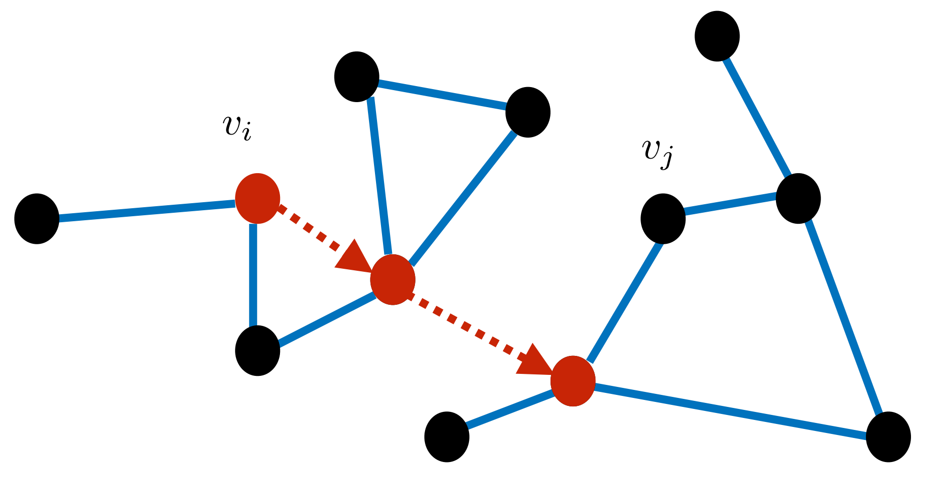
$p(v_j|v_i)$: probability of visiting node v_j on a random walk starting from a node v_i

$\mathcal{N}_{v_i|RW} = \{v_j \in RW\}$: neighbourhood of v_i obtained by some random walk strategy starting from v_i

Example 2: Random walk embeddings

- **Intuition:** Preserve neighborhood structure i.e., higher order similarities, captured by random walks

- Random walk: A random process starting from a node that describes a path that consists of a succession of random steps on a graph (DFS)



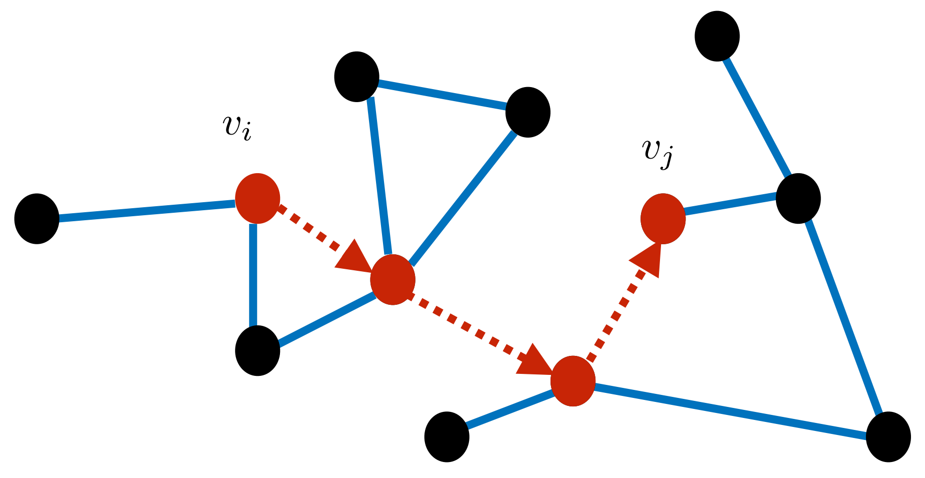
$p(v_j|v_i)$: probability of visiting node v_j on a random walk starting from a node v_i

$\mathcal{N}_{v_i|RW} = \{v_j \in RW\}$: neighbourhood of v_i obtained by some random walk strategy starting from v_i

Example 2: Random walk embeddings

- **Intuition:** Preserve neighborhood structure i.e., higher order similarities, captured by random walks

- Random walk: A random process starting from a node that describes a path that consists of a succession of random steps on a graph (DFS)



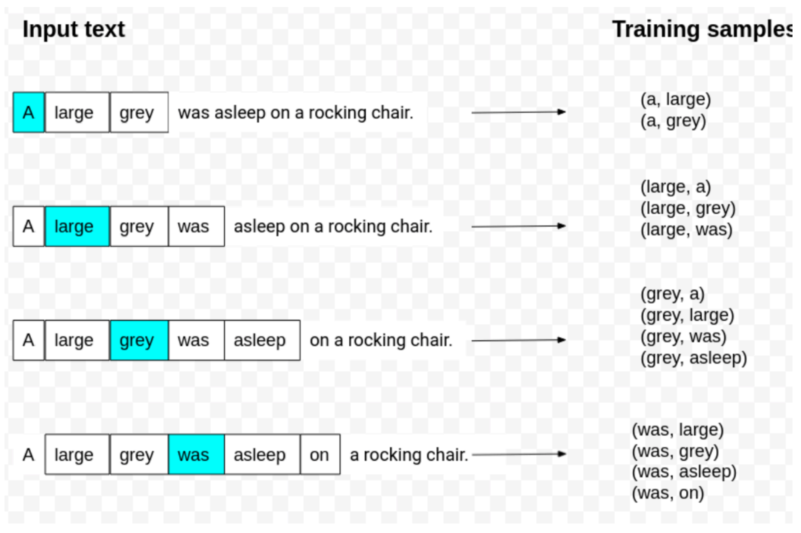
$p(v_j|v_i)$: probability of visiting node v_j on a random walk starting from a node v_i

$\mathcal{N}_{v_i|RW} = \{v_j \in RW\}$: neighbourhood of v_i obtained by some random walk strategy starting from v_i

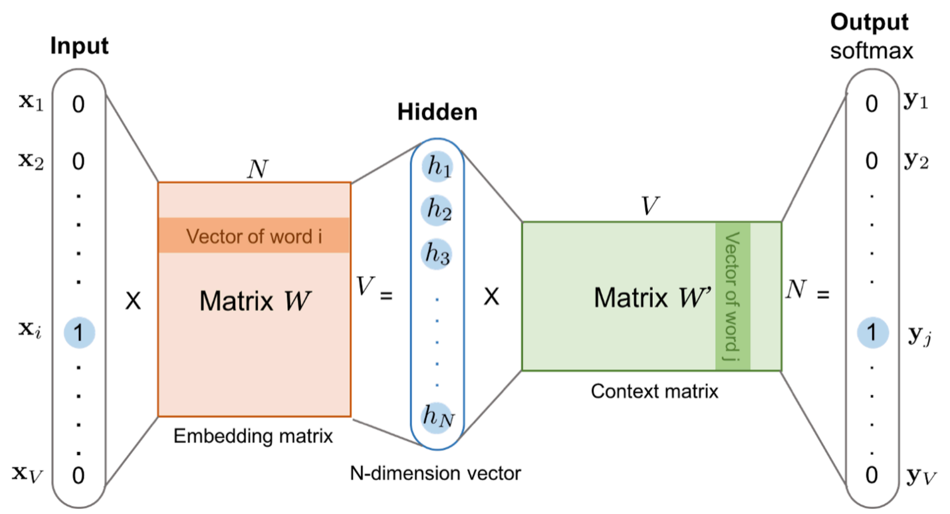
Random walk embeddings inspired from language modeling

- State-of-the-art methods learn a representation of a word from documents (word co-occurrence)

- **word2vec** embedding algorithm:
  - words appearing in similar contexts have similar meaning
  - embedding is achieved by looking at words appearing close to each other as defined by context windows


Input text
Training samples
A large grey was asleep on a rocking chair.
(a, large)
(a, grey)
A large grey was asleep on a rocking chair.
(large, a)
(large, grey)
(large, was)
A large grey was asleep on a rocking chair.
(grey, a)
(grey, large)
(grey, was)
(grey, asleep)
A large grey was asleep on a rocking chair.
(was, large)
(was, grey)
(was, asleep)
(was, on)

input   **the skip-gram model**   context

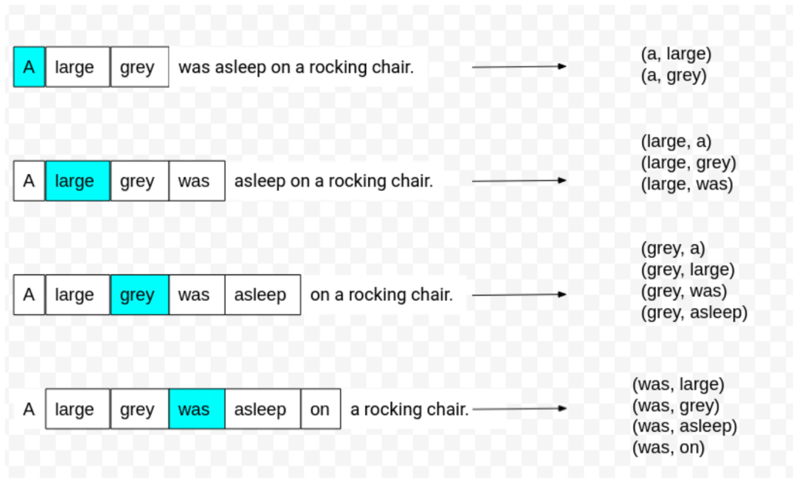


(input, context)

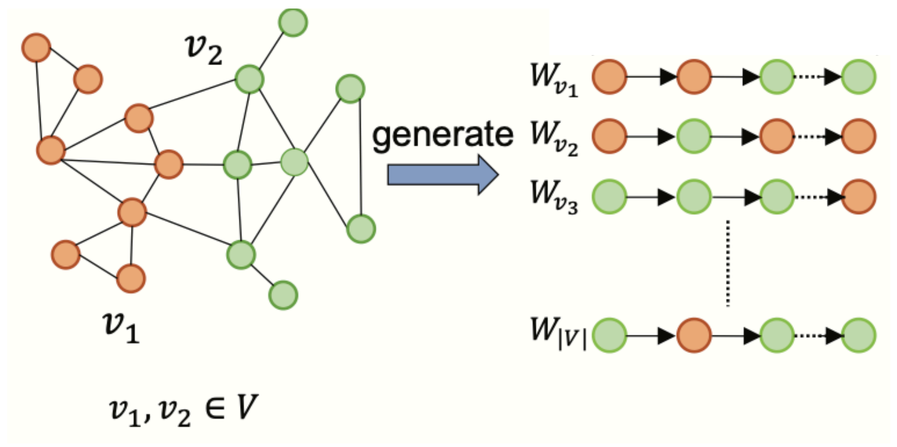
[Mikolov et al. 2013. Distributed Representations of Words and Phrases and their Compositionality, NeurIPS]

From NLP to node embeddings

- Generalization to graphs:
  - nodes: words
  - node sequences: sentences


A large grey was asleep on a rocking chair.
(a, large)
(a, grey)
A large grey was asleep on a rocking chair.
(large, a)
(large, grey)
(large, was)
A large grey was asleep on a rocking chair.
(grey, a)
(grey, large)
(grey, was)
(grey, asleep)
A large grey was asleep on a rocking chair.
(was, large)
(was, grey)
(was, asleep)
(was, on)

words, sentences



nodes, node sequences

- Random walks are a flexible way to generate node sequences
  - Random walk embedding algorithms: DeepWalk, node2vec

Example 2A: DeepWalk

- **Intuition:** Nodes have similar embeddings if they tend to cooccur on short random walks over the graph

$$sim_{\mathcal{G}}(v_i, v_j) = p(v_j|v_i)$$

- **Objective:** Given node v_i learn a mapping $\phi : v_i \rightarrow \mathbb{R}^d;\ \phi(v_i) = Y_i$ such that the feature representation Y_i are predictive of the nodes in its random walk neighborhood $\mathcal{N}_{v_i|RW}$

$$\max_{\phi} \sum_{v_i \in \mathcal{V}} \log sim_Y(Y_i, Y_j) = \max_{\phi} \sum_{v_i \in \mathcal{V}} \log P(Y_j \text{ for } v_j \in R_{i|RW}|Y_i)$$

Maximum likelihood

- Measure the similarity in the embedding space in a probabilistic manner

$$sim_Y(Y_i, Y_j) = \frac{e^{Y_i^T Y_j}}{\sum_{k \in \mathcal{V}} e^{Y_i^T Y_k}}$$

Softmax

DeepWalk - Algorithm

- Run fixed length random walks starting from each node of the graph

- For each node v_i define its random walk neighborhood $\mathcal{N}_{v_i|RW}$

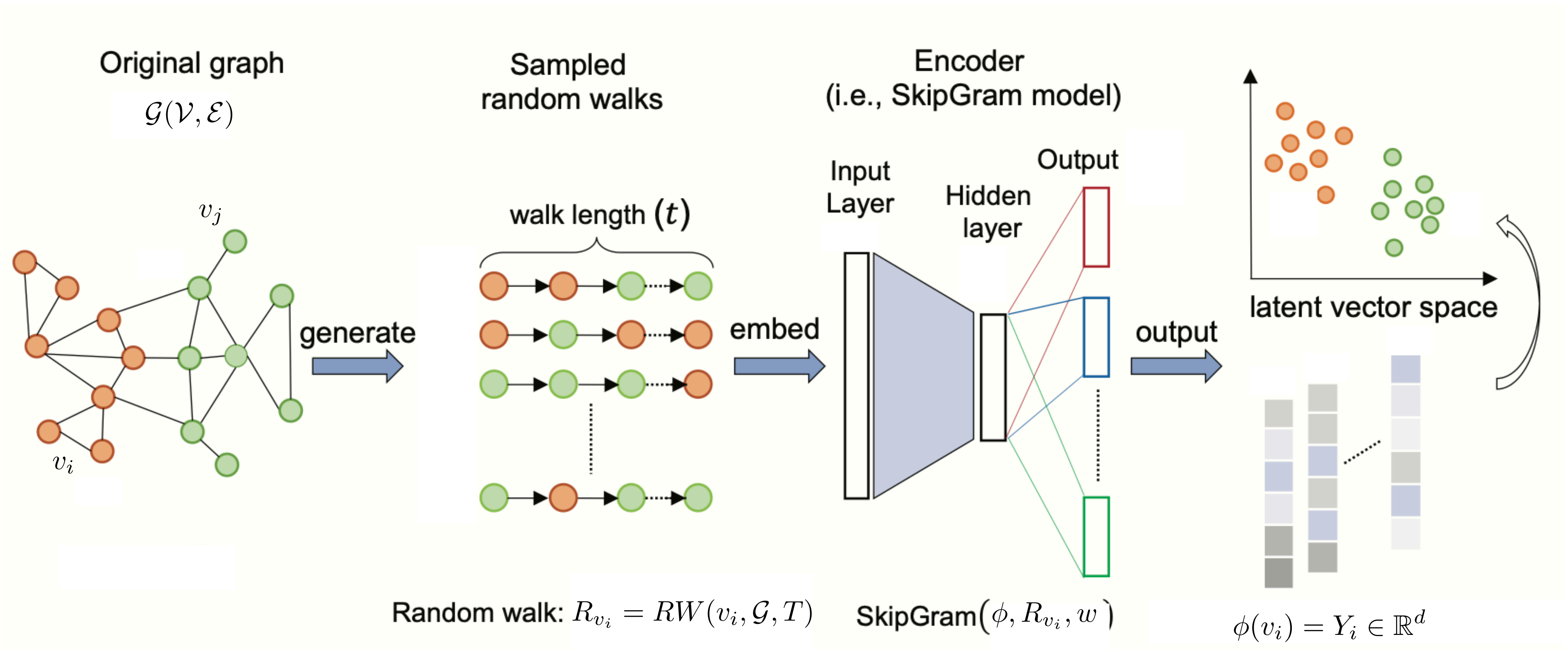
- Find embeddings to maximize the likelihood of random walk co-occurrences

$$loss_{(v_i,v_j)\in N_{Train}} = \sum_{v_i\in N_{Train}} \sum_{v_j\in R_{v_i|RW}} -log\left(\frac{e^{Y_i^T Y_j}}{\sum_{v_k\in N_{Train}} e^{Y_i^T Y_k}}\right)$$

Predicted probability of two nodes co-occurring in a random walk

- Embeddings are optimized using stochastic gradient descent

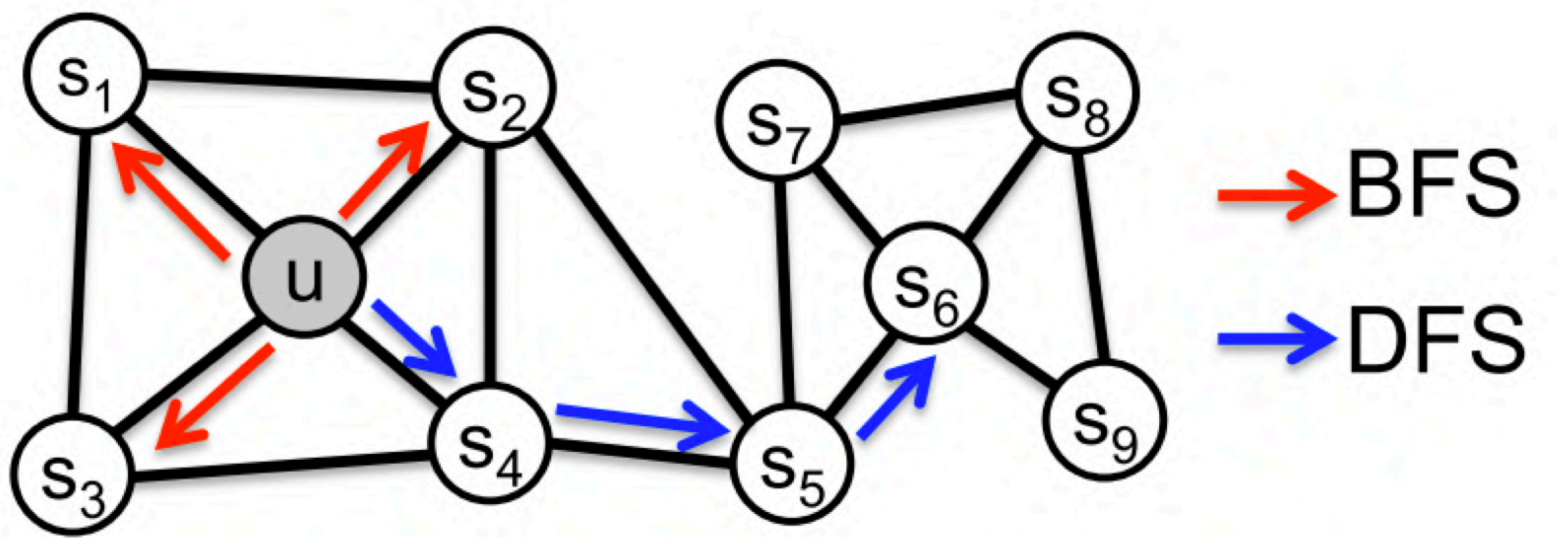
DeepWalk - Schematic overview


Original graph
$\mathcal{G}(\mathcal{V},\mathcal{E})$
v_j
v_i
generate
Sampled random walks
walk length (t)
Random walk: $R_{v_i} = RW(v_i, \mathcal{G}, T)$
embed
Encoder (i.e., SkipGram model)
Input Layer
Hidden layer
Output
SkipGram(ϕ, R_{v_i}, w)
output
latent vector space
$\phi(v_i) = Y_i \in \mathbb{R}^d$

[Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD*]

Example 2B: Node2vec

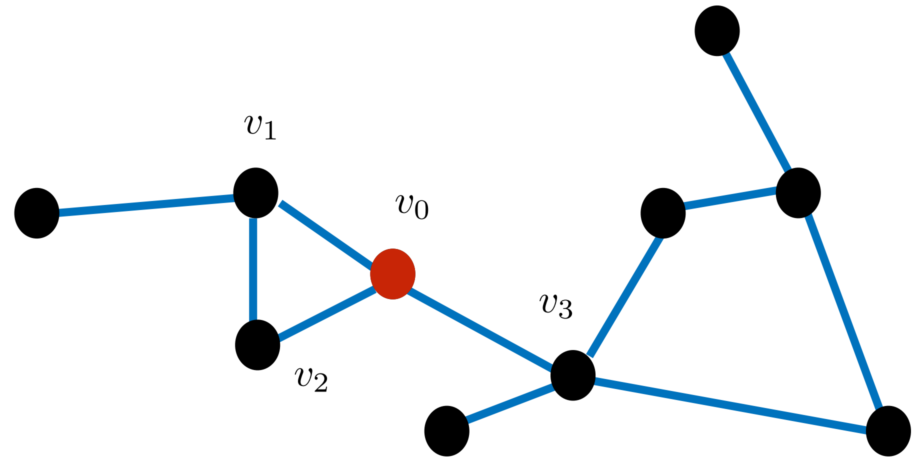
- Intuition: More flexible representations by obtaining similar embeddings for
  - Nodes that share similar role; Structural equivalence (local view of the network: **B**reath-**F**irst **S**earch)
  - Nodes belonging to similar network clusters; Homophily (global view of the network: **D**epth-**F**irst **S**earch)



- In real-world, networks commonly exhibit both behaviours

Node2vec: Random walks (I)

- Biased random walk: Interpolate between BFS and DFS
- Given a starting node, the neighborhood is generated based on two parameters:
  - Return parameter p: controls the likelihood of immediately revisiting a node in the random walk; microscopic view around the node
  - In-out parameter q: controls how fast the next walk explores or leaves the neighborhood of a starting node; moving inwards (BFS) versus outwards (DFS)
- The rest of the algorithm is similar to DeepWalk



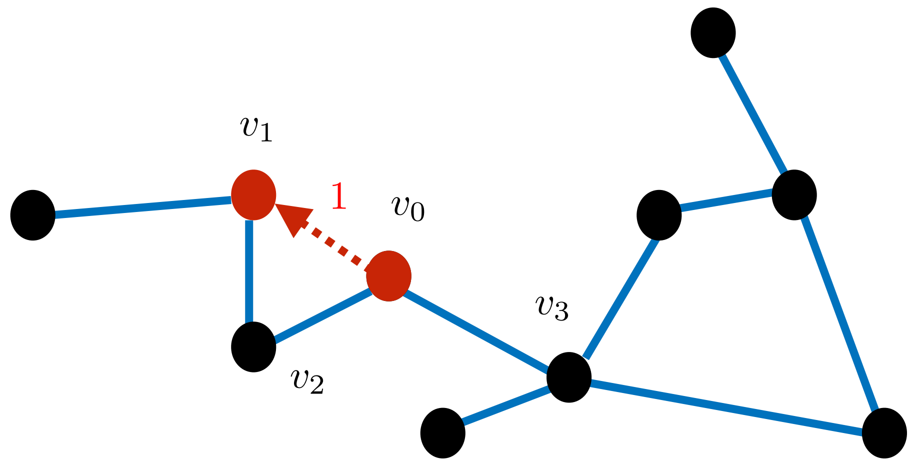
Current state of the walker: v_0
Previous state of the walker: v_2
Node close to the previous state: v_1

Where to go next?

[Grover et al. 2016. node2vec: Scalable Feature Learning for Networks, *KDD*]

Node2vec: Random walks (I)

- Biased random walk: Interpolate between BFS and DFS
- Given a starting node, the neighborhood is generated based on two parameters:
  - Return parameter p: controls the likelihood of immediately revisiting a node in the random walk; microscopic view around the node
  - In-out parameter q: controls how fast the next walk explores or leaves the neighborhood of a starting node; moving inwards (BFS) versus outwards (DFS)
- The rest of the algorithm is similar to DeepWalk



Current state of the walker: v_0
Previous state of the walker: v_2
Node close to the previous state: v_1

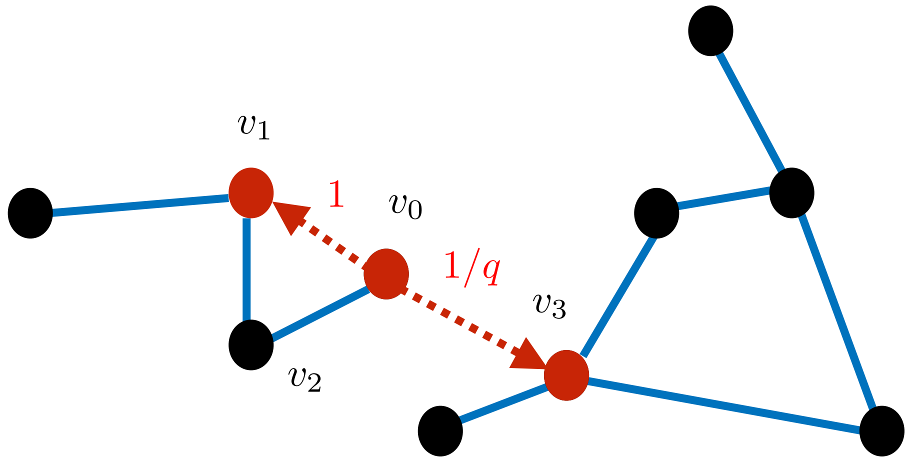
Where to go next?

[Grover et al. 2016. node2vec: Scalable Feature Learning for Networks, *KDD*]

Node2vec: Random walks (I)

- Biased random walk: Interpolate between BFS and DFS

- Given a starting node, the neighborhood is generated based on two parameters:
  - Return parameter p: controls the likelihood of immediately revisiting a node in the random walk; microscopic view around the node
  - In-out parameter q: controls how fast the next walk explores or leaves the neighborhood of a starting node; moving inwards (BFS) versus outwards (DFS)
- The rest of the algorithm is similar to DeepWalk



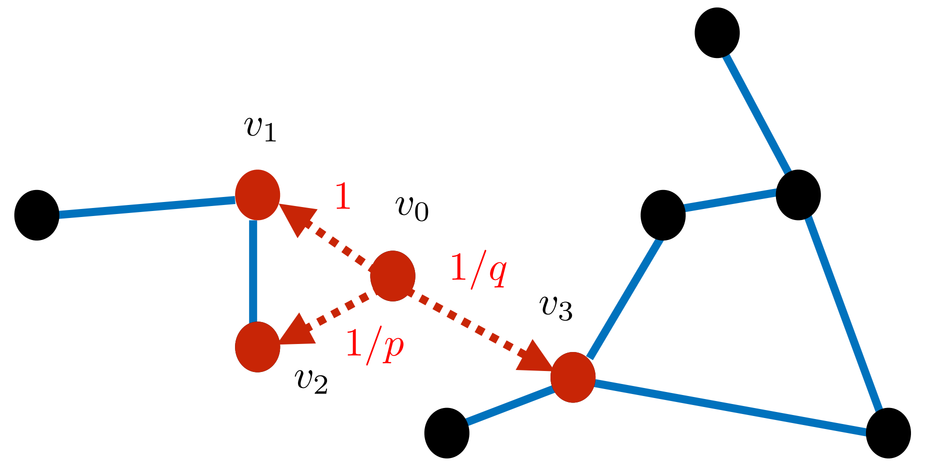
Current state of the walker: v_0
Previous state of the walker: v_2
Node close to the previous state: v_1

Where to go next?

[Grover et al. 2016. node2vec: Scalable Feature Learning for Networks, *KDD*]

Node2vec: Random walks (I)

- Biased random walk: Interpolate between BFS and DFS
- Given a starting node, the neighborhood is generated based on two parameters:
  - Return parameter p: controls the likelihood of immediately revisiting a node in the random walk; microscopic view around the node
  - In-out parameter q: controls how fast the next walk explores or leaves the neighborhood of a starting node; moving inwards (BFS) versus outwards (DFS)
- The rest of the algorithm is similar to DeepWalk

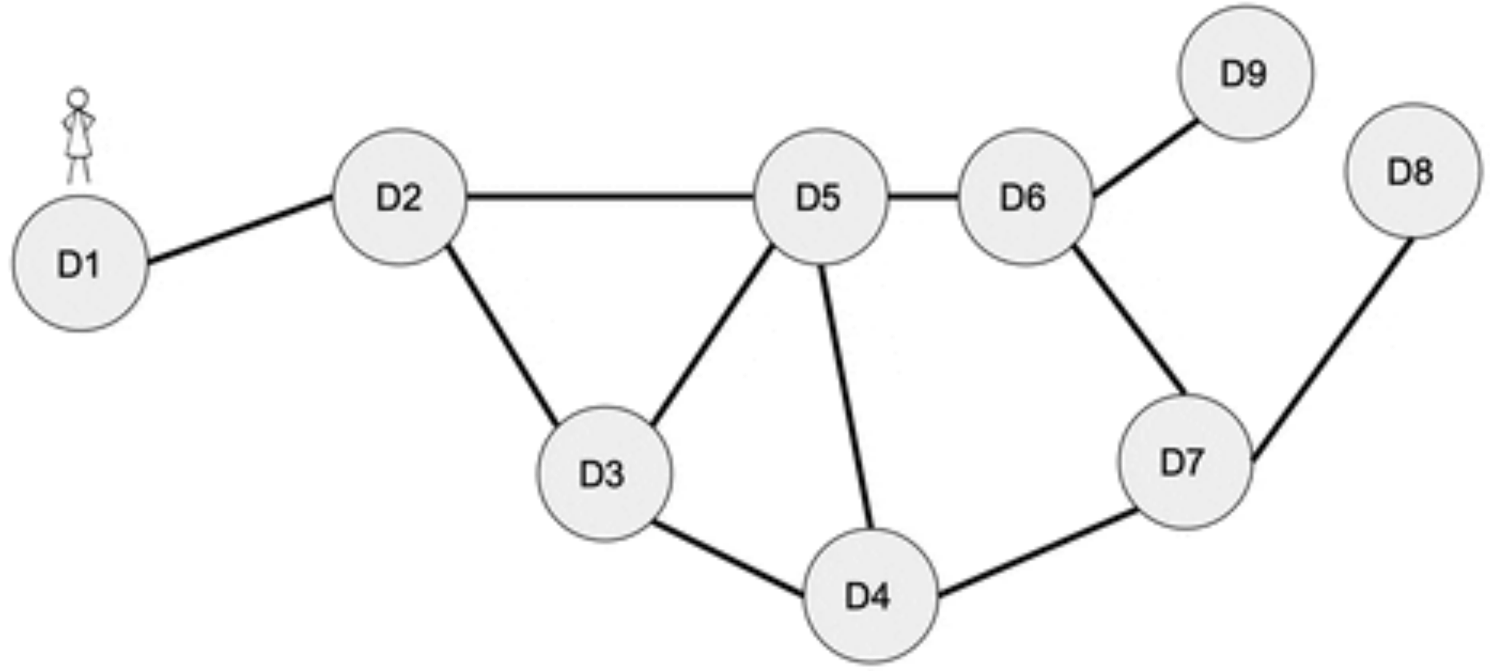


Current state of the walker: v_0
Previous state of the walker: v_2
Node close to the previous state: v_1

Where to go next?

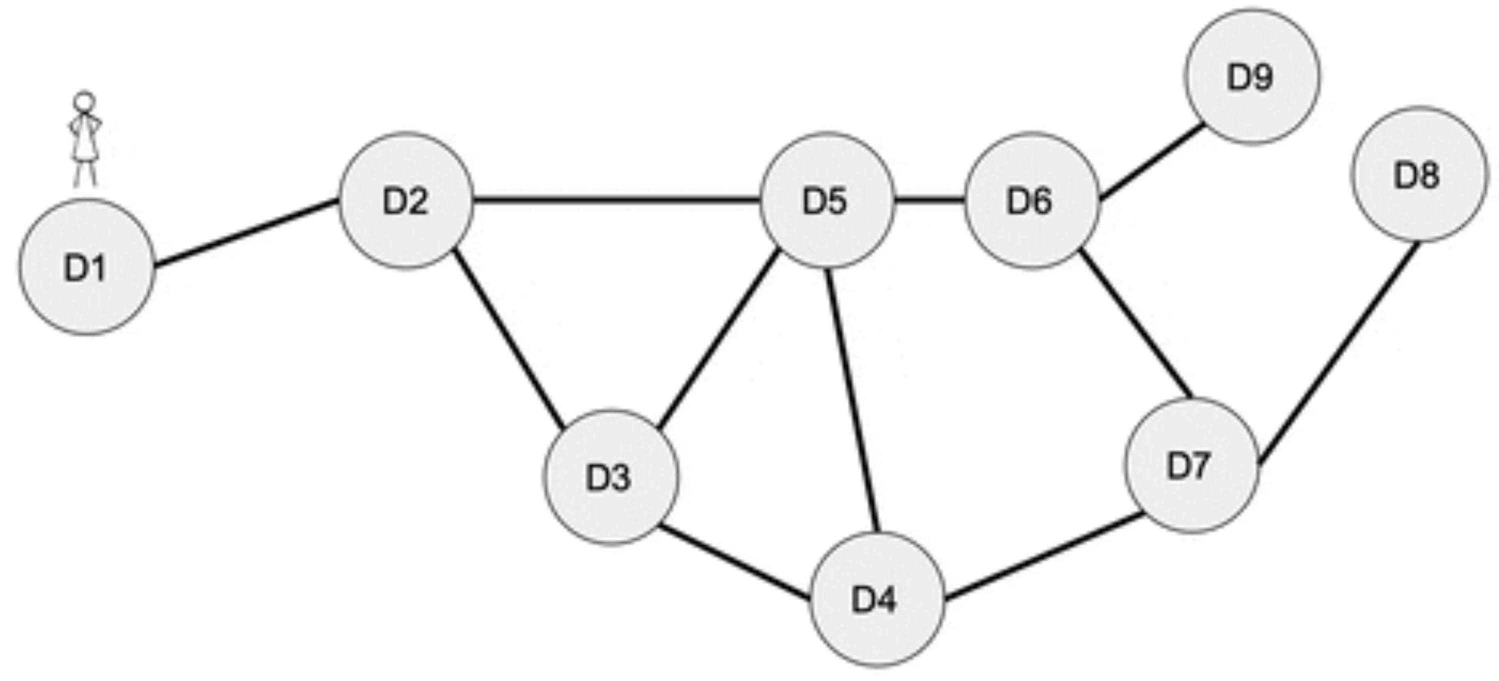
[Grover et al. 2016. node2vec: Scalable Feature Learning for Networks, *KDD*]

Node2vec: Random walks (II)



Node2Vec Random Walk: D1

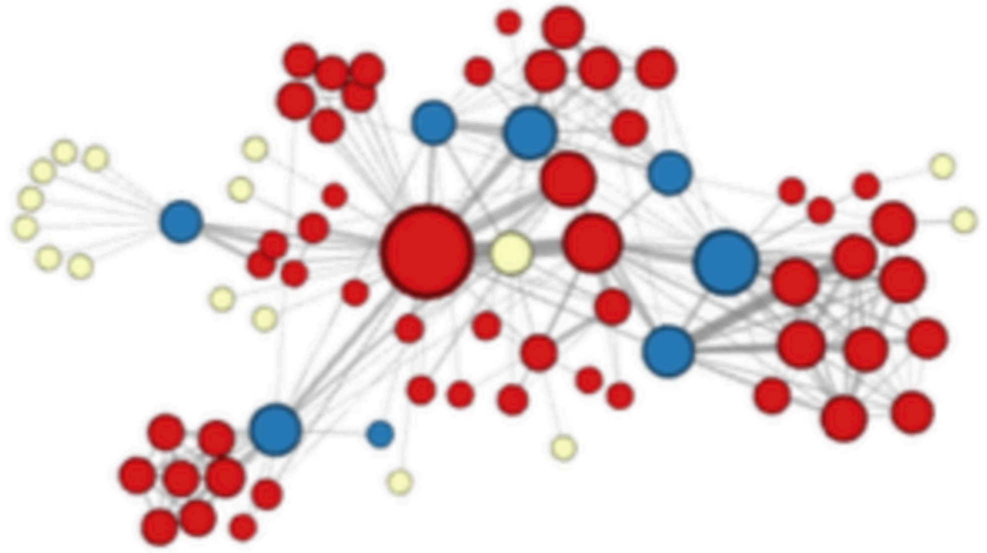
Node2vec: Random walks (II)



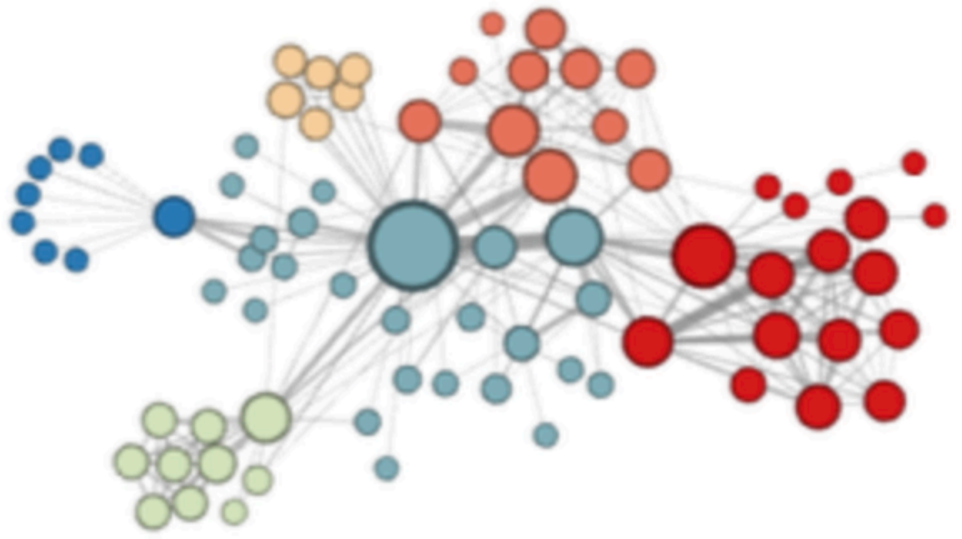
Node2Vec Random Walk: D1

Node2vec: example

- Clustering of the node embeddings from ‘Les Misérables’
  - Embeddings obtained with structural and homophily priors



BFS-based (p = 1, q=2)

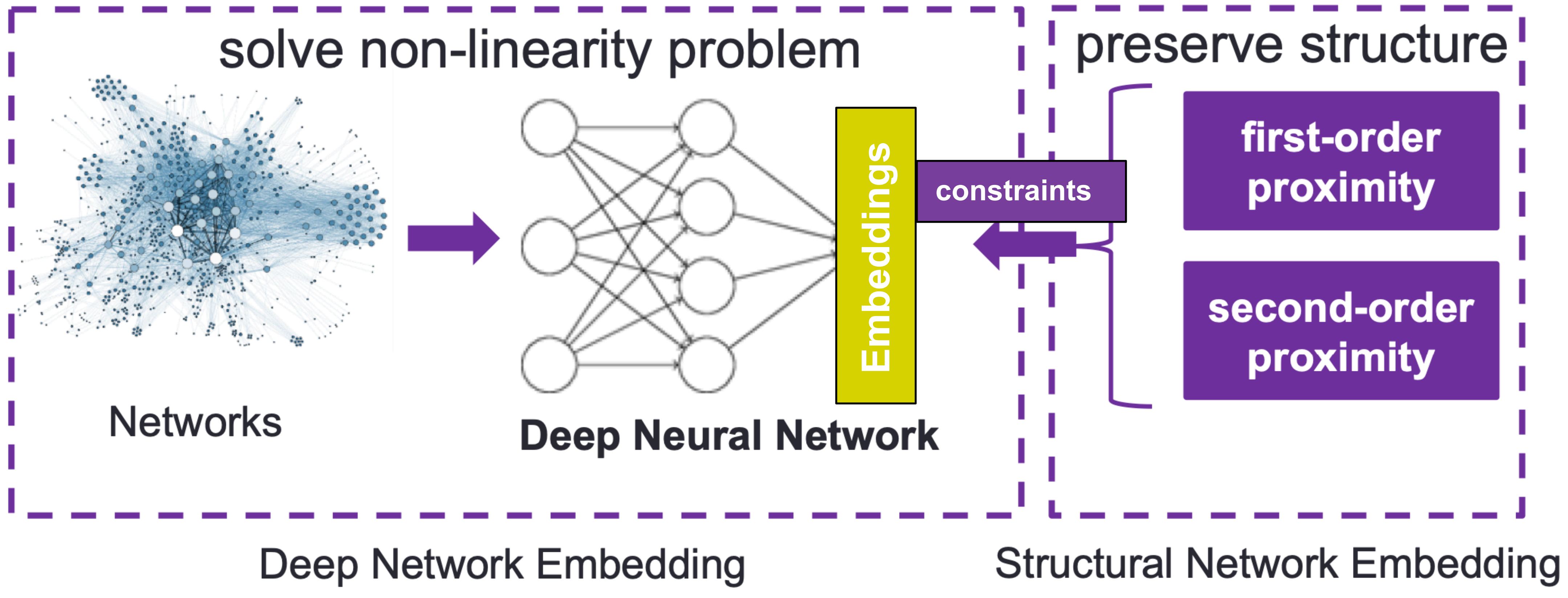


DFS-based (p = 1, q = 0.5)

- BFS: better for capturing structural nodes, e.g., hubs, outliers (higher value of q)
- DFS: better for capturing communities (smaller value of q)

Example 3: Structural deep network embeddings (SDNE)

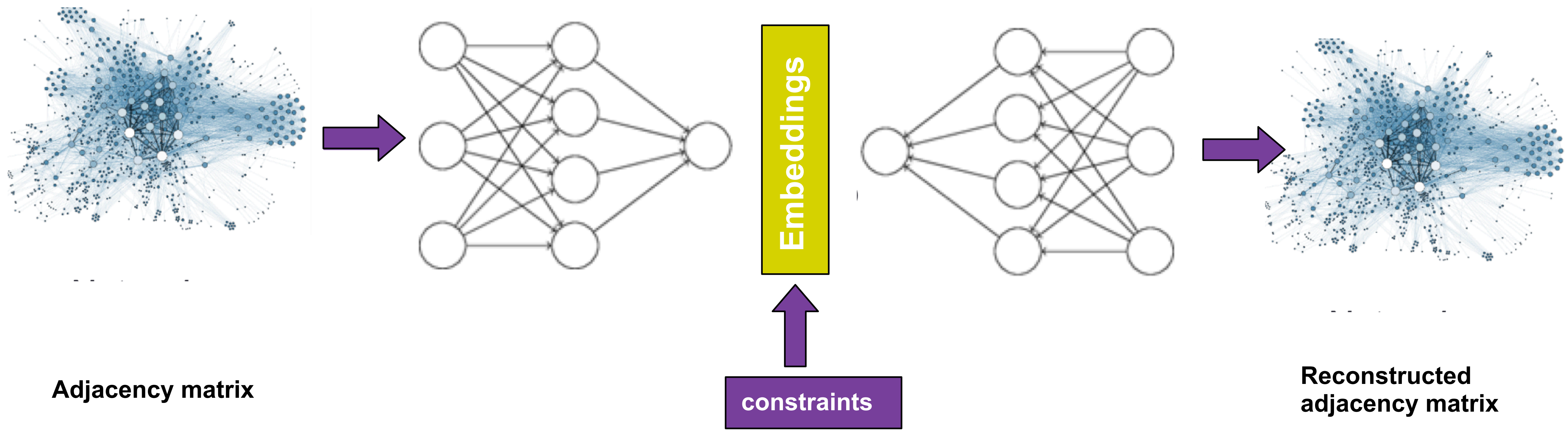
- A deep learning approach to network embeddings
  - Shallow models (e.g., deepwalk, node2vec) cannot capture the non-linear network structure
  - Encoder: A deep network



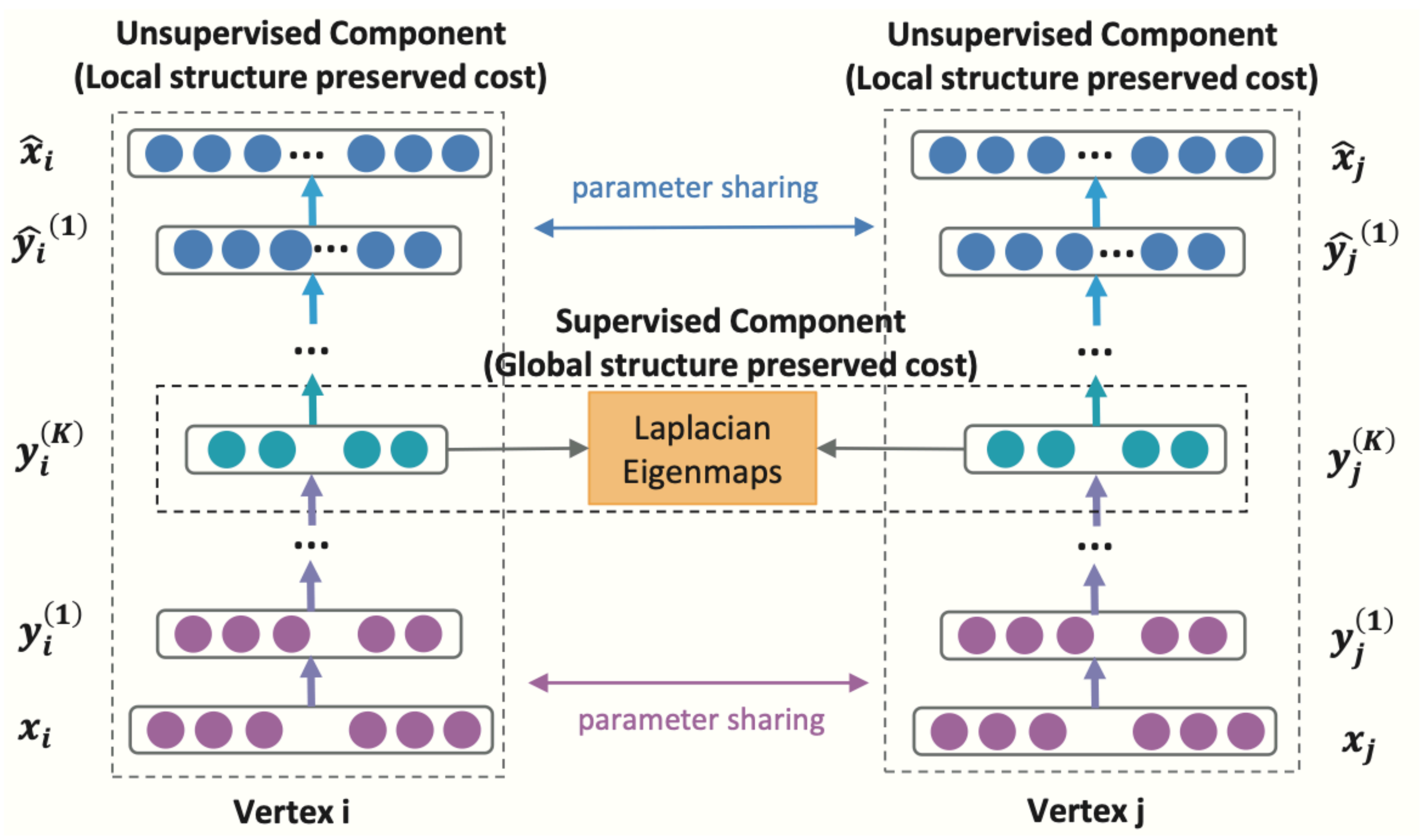
[Wang et al. 2016.Structural Deep Network Embedding, KDD]

SDNE: An autoencoder approach

- **Intuition:** Find embeddings that minimize the reconstruction error of the input from the low dimensional embedding



SDNE: constrained embeddings



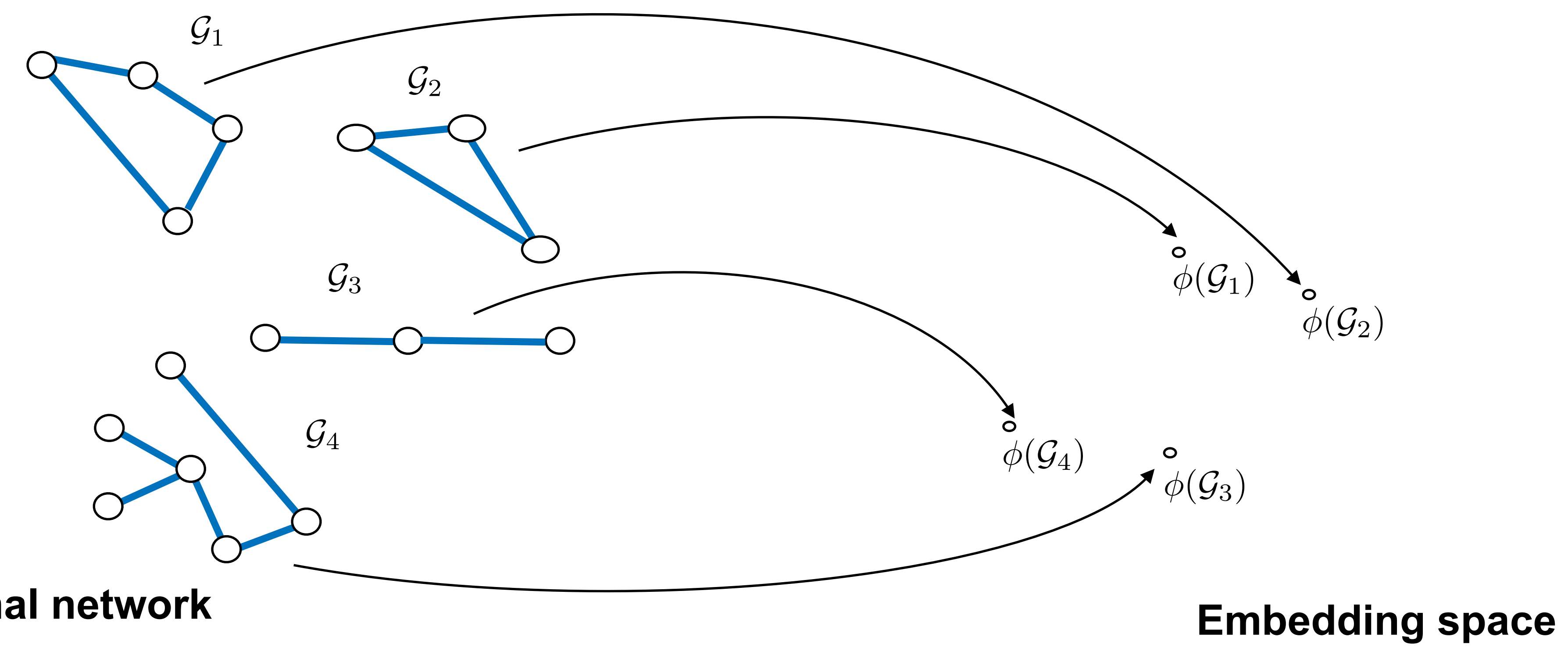
Optimization problem:

$$Y^* = \underset{Y \in \mathbb{R}^{N \times d}}{\operatorname{argmin}} \|(X - \hat{X}) \circ B\|_F^2 + \alpha \sum_{v_i, v_j \in \mathcal{V}} W_{ij} \|Y_i - Y_j\|_2^2$$

Reconstruction term

Laplacian eigenmaps

From node embedding to graph embedding

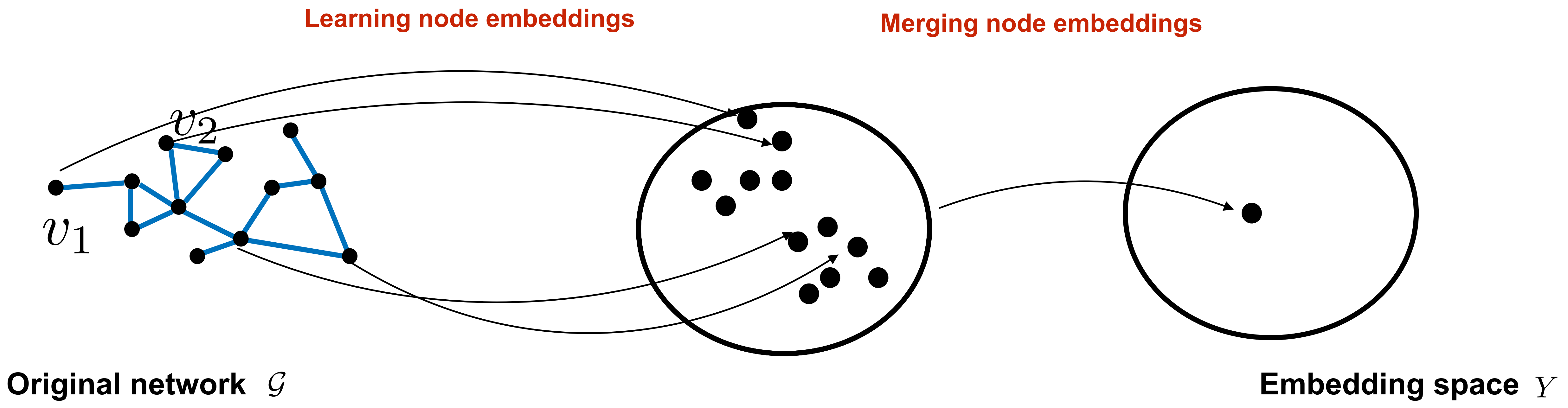


Original network

How can we embed an entire graph or a subgraph?

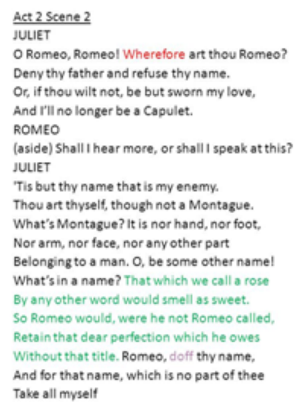
Graph embedding: A pooling approach

- Compute a single embedding per graph
- Usually achieved by generating a graph/subgraph representation from the individual embeddings of each node
  - sum, average, max operator
  - virtual nodes
  - hierarchical approaches based on graph coarsening (more in the following lectures)



Graph embedding: A sub-graph based approach

- **Graph2vec:** View the graph as a document and the rooted subgraphs around every node as words



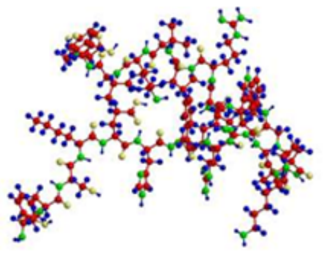
<u>Act 2 Scene 2</u>
JULIET
O Romeo, Romeo! Wherefore art thou Romeo?
Deny thy father and refuse thy name.
Or, if thou wilt not, be but sworn my love,
And I'll no longer be a Capulet.
ROMEO
(aside) Shall I hear more, or shall I speak at this?
JULIET
'Tis but thy name that is my enemy.
Thou art thyself, though not a Montague.
What's Montague? It is nor hand, nor foot,
Nor arm, nor face, nor any other part
Belonging to a man. O, be some other name!
What's in a name? That which we call a rose
By any other word would smell as sweet.
So Romeo would, were he not Romeo called,
Retain that dear perfection which he owes
Without that title. Romeo, doff thy name,
And for that name, which is no part of thee
Take all myself

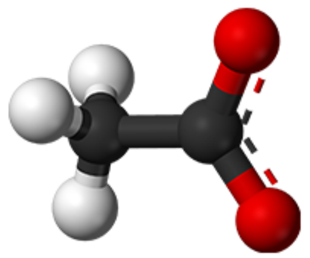
"O Romeo, Romeo, wherefore

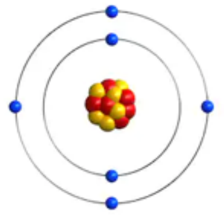
art thou Romeo?"

wherefore

$$\max \sum_{j=1}^{length_i} \log P(word_j | document_i)$$





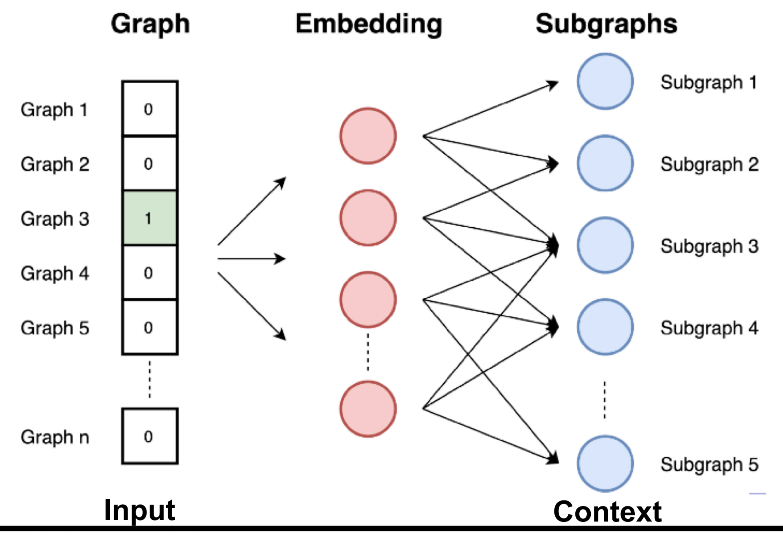


$$\max \sum_{j=1}^{length_i} \log P(subgraph_j | graph_i)$$

- The embeddings of two graphs are close if they are composed of similar rooted subgraphs

Graph2vec in a slide

- Given a set of graphs, consider the set of all rooted subgraphs (i.e., neighborhoods) around every node (up to a certain degree) as vocabulary
  - Compute rooted subgraphs (i.e., a neighbourhood of certain degree) with WL kernel (from previous lecture!)
  - WL kernels lead to non-linear substructures (contrary to random walks that are linear): better representation of the structure
- Train embeddings by maximizing the probability of predicting subgraphs that exist in the input graph



[Narayanan et al. 2017. graph2vec: Learning Distributed Representations of Graphs]

Outline

- Graph representation learning
- Unsupervised graph embedding algorithms
- **Illustrative applications**

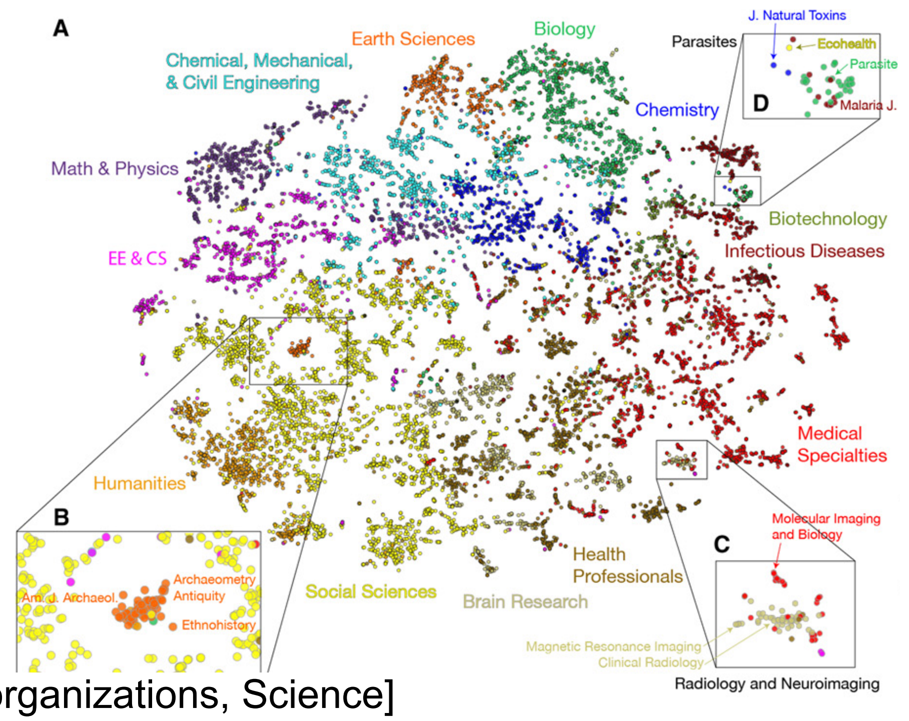
Applications

- Visualization
- Node related applications
  - Node clustering
  - Node classification
  - Node ranking
- Edge related applications
  - Link prediction
- Graph related applications
  - Graph classification
  - Graph clustering

Visualization

- We usually visualise the embeddings on a two-dimensional (2D) space
  - Dimensionality reduction by PCA, t-SNE
  - Visualize each vector as a point in 2D space

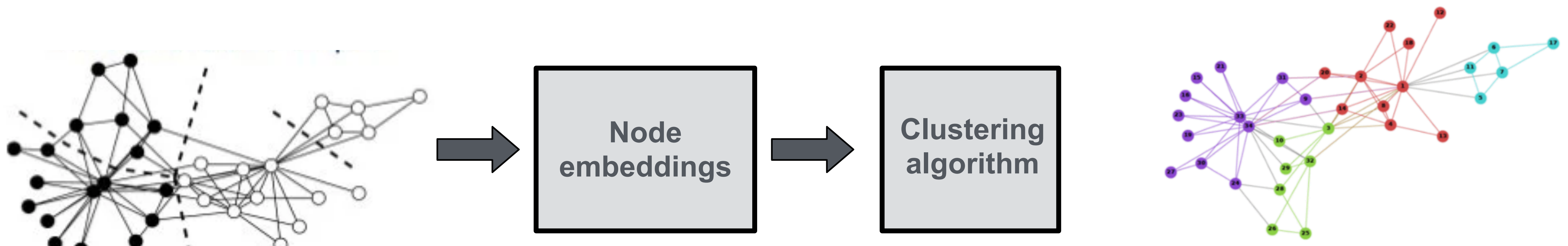
- Visualizations can be used to infer latent dependencies in the data
  - Example:
    - 2D representation of 12780 journals
    - Each dot represent a journal
    - Each color denotes its discipline
    - Embeddings reveal journal similarities across disciplines



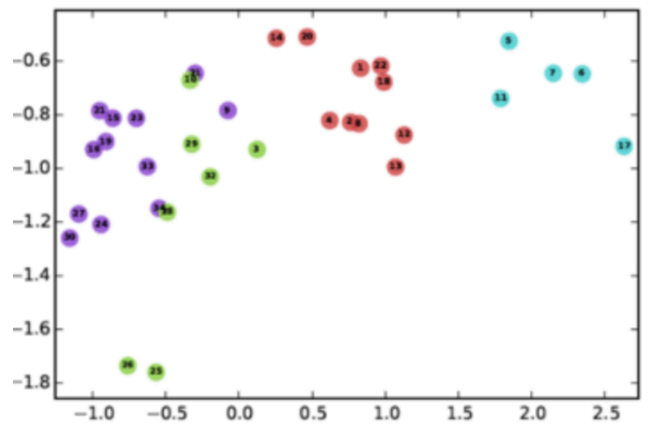
[Peng et al. 2021. Neural embeddings of scholarly periodicals reveal complex disciplinary organizations, Science]

Node clustering/Community detection

- The karate-club example
  - Compute node embeddings
  - Apply any clustering algorithm (e.g., K-means) on the learned embeddings

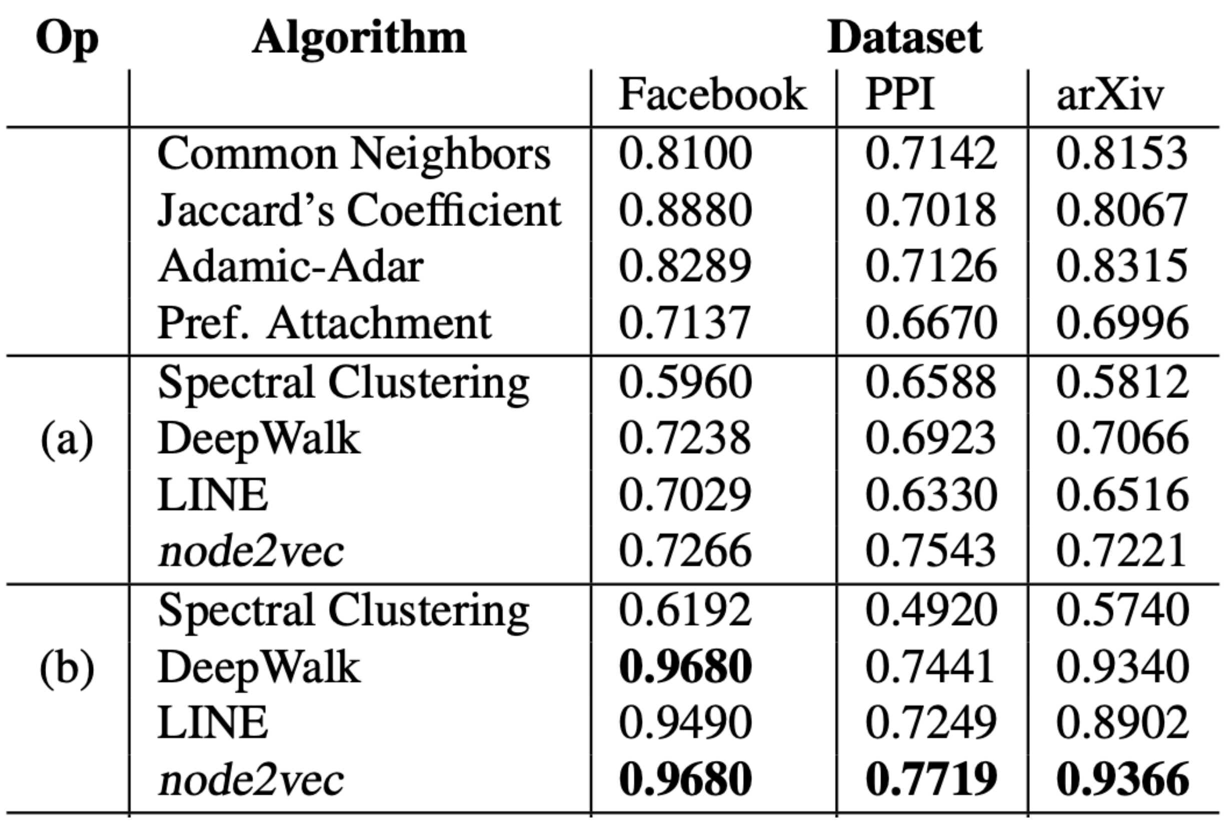


e.g., K-means



Link prediction

- Node embeddings encode rich information about the network structure: they can be used to predict missing links
- Embeddings of two nodes can be combined by different operators such as (a) average, (b) Hadamard product

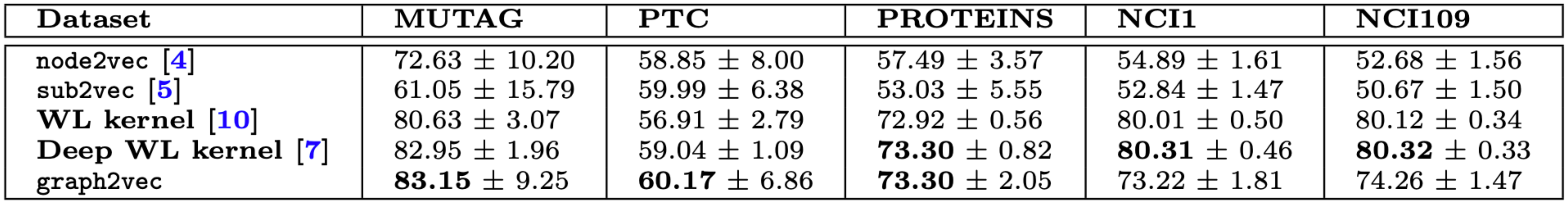


Op	Algorithm	Dataset		
		Facebook	PPI	arXiv
	Common Neighbors	0.8100	0.7142	0.8153
	Jaccard's Coefficient	0.8880	0.7018	0.8067
	Adamic-Adar	0.8289	0.7126	0.8315
	Pref. Attachment	0.7137	0.6670	0.6996
(a)	Spectral Clustering	0.5960	0.6588	0.5812
	DeepWalk	0.7238	0.6923	0.7066
	LINE	0.7029	0.6330	0.6516
	node2vec	0.7266	0.7543	0.7221
(b)	Spectral Clustering	0.6192	0.4920	0.5740
	DeepWalk	**0.9680**	0.7441	0.9340
	LINE	0.9490	0.7249	0.8902
	node2vec	**0.9680**	**0.7719**	**0.9366**

[Grover et al. 2016. node2vec: Scalable Feature Learning for Networks, KDD]

Graph classification

- Many applications in chemo/bioinformatics
- Node/graph embeddings are followed by a classifier (e.g., SVM)
- Classical datasets:
  - MUTAG: chemical compounds labeled according to whether of not they have a mutagenic effect on a specific bacteria
  - PROTEINS: collection of graphs whose nodes represent secondary structure elements and edges indicate neighborhood in the amino-acid sequence



Dataset	MUTAG	PTC	PROTEINS	NCI1	NCI109
node2vec [4]	72.63 ± 10.20	58.85 ± 8.00	57.49 ± 3.57	54.89 ± 1.61	52.68 ± 1.56
sub2vec [5]	61.05 ± 15.79	59.99 ± 6.38	53.03 ± 5.55	52.84 ± 1.47	50.67 ± 1.50
WL kernel [10]	80.63 ± 3.07	56.91 ± 2.79	72.92 ± 0.56	80.01 ± 0.50	80.12 ± 0.34
Deep WL kernel [7]	82.95 ± 1.96	59.04 ± 1.09	**73.30** ± 0.82	**80.31** ± 0.46	**80.32** ± 0.33
graph2vec	**83.15** ± 9.25	**60.17** ± 6.86	**73.30** ± 2.05	73.22 ± 1.81	74.26 ± 1.47

[Narayanan et al. 2017. graph2vec: Learning Distributed Representations of Graphs]

Summary

- Feature learning on graphs is a data-driven (and ofter more flexible) alternative to designing hand-crafted features
- Unsupervised learning on graphs provides representations i.e., embeddings, that are not adapted to specific tasks
- Different assumptions lead to different ways of preserving information from the original graph in the embedding space (e.g., weight matrix, random walks…)
- The choice of what structure information to preserve depends on the application

Limitations of the (discussed) embedding algorithms

- Usually transductive not inductive
  - Learned embedding models often do not generalize to new nodes
- Do not incorporate node attributes
- Independent of downstream tasks
- No parameter sharing:
  - Every node has its own unique embedding
- Graph neural networks: an alternative to (deeper) node/graph embeddings!

References

1. Graph representation learning (chap 3), William Hamilton

2. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications, Cai et al., 2017

   - https://arxiv.org/pdf/1709.07604.pdf