Graph Embedding for Biological Networks

Maurizio Giordano[1] Ilaria Granata [1] Mario Rosario Guarracino[3]
Lucia Maddalena[1] Mario Manzo [2]

Computational Data Science Lab
[1]Institute for High Performance Computing and Networking - CNR
[2]University of Naples L'Orientale
[3]University of Cassino and Southern Lazio

September 13, 2021

Summary

Motivation

- Gathering and expansion of huge amounts of multiform data (Big Data) poses new challenges in bioinformatics investigations
- ML techniques are widely applied to bioinformatics in response to the peculiarities and needs of these data
- In many applications, biological data are constructed as biological networks
- Graph theory is as a solid ground for the representation and analysis of biological heterogeneous data and their relations
- Examples of biological networks are:
  - molecular structure of proteins and RNAs
  - metabolic networks
  - genetic interaction networks
  - cell signalling networks
  - protein-protein interaction networks
  - ...
- ... particularly used in uncovering complex disease mechanisms

Motivation

- Biological networks analysis poses the problem of reducing the complexity of graphs through projections and/or transformation into a more manageable data space.
- Graph Embedding (GE) techniques pursue this scope, by translating large and complex graphs into a reduced vector space called latent space
- We discuss GE methods applied to the task of Graph Classification, i.e. the problem of identifying a categorization of graphs in a dataset
- ... we consider two case studies:
  - MUTAG: molecular graphs [1]
  - KIDNEY: metabolic networks[2]

[1] publicly available at: `https://networkrepository.com/`

[2] developed by CDS lab ICAR-CNR, publicly available at: `https://github.com/cds-group/GraphDatasets`

Graph classification

Definition (Graph classification)

Given a set of graphs $\mathcal{G} = \{G_1, \ldots, G_m\}$ and a set of given properties of graphs $\mathcal{Y} = y_1, \ldots, y_k$, build (learn) a function:

$$f : \mathcal{G} \rightarrow \mathcal{Y}$$

to predict the property of a graph

Graph classification

Definition (Graph classification)

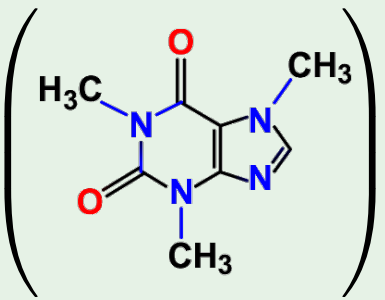
Given a set of graphs $\mathcal{G} = \{G_1, \ldots, G_m\}$ and a set of given properties of graphs $\mathcal{Y} = y_1, \ldots, y_k$, build (learn) a function:

$$f : \mathcal{G} \rightarrow \mathcal{Y}$$

to predict the property of a graph

Example

- $\mathcal{G}$ - set of molecular graphs
- $\mathcal{Y} = \{\texttt{toxic}, \texttt{non-toxic}\}$

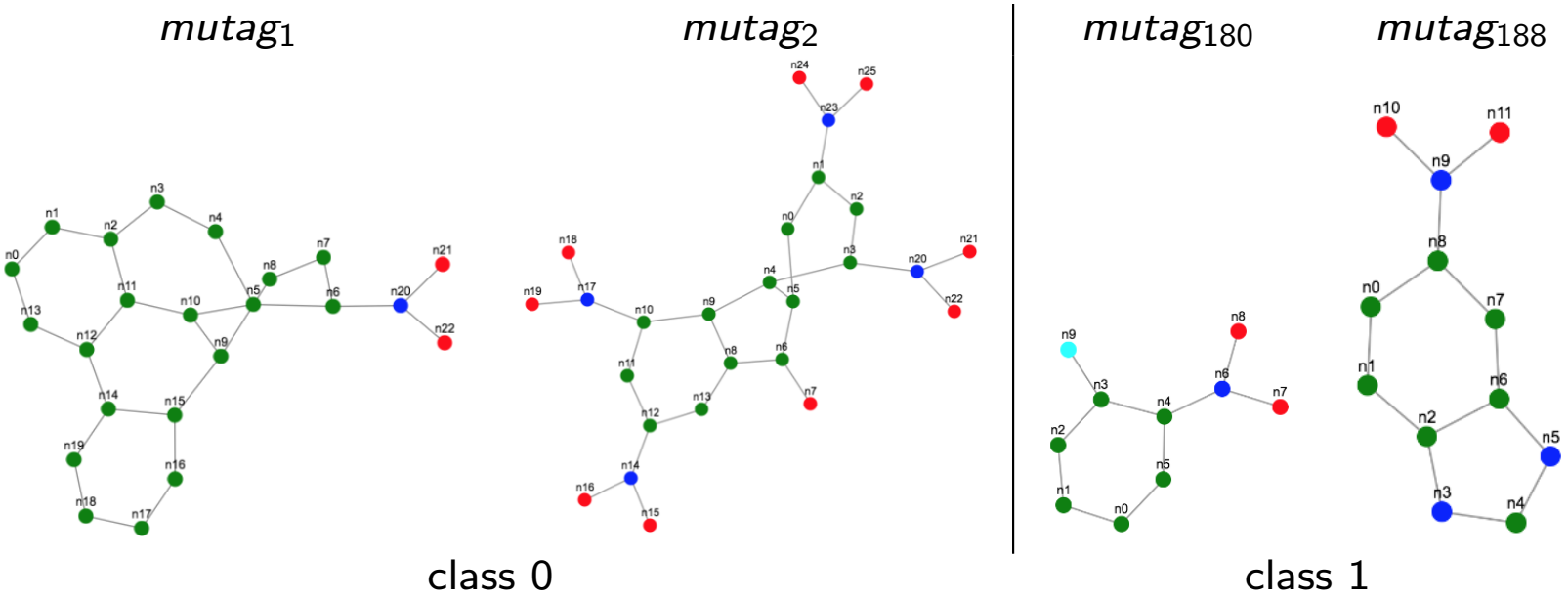
f  $=?$

Graph classification and applications

- An important task with practical applications in several domains:
  - Bioinformatics and Cheminformatics: to predict the function of a protein structure, if cells are cancerous or not, if a protein is enzyme or not, checking the toxicity of a chemical compound
  - Social network analysis: to provide online recommendations for a page or user account, implement newsfeed and calculate page rank [14] - ssers/pages are nodes and the interaction between them are edges
  - Natural Language Processing: to categorize different documents based on the structure of the text [11].
  - Neuroscience: to analyse brain networks. Neurons are represented using nodes and the connection between neurons are represented by edges [3]
  - ...

MUTAGenic molecular graphs

- MUTAG[3]) is a dataset of nitroaromatic compounds
  - nodes are atoms labeled by the type, while edges represent bonds between the corresponding atoms
  - 188 samples of chemical compounds
  - classes refer to mutagenic effects of compounds on a specific gram negative bacterium (Salmonella typhimurium)

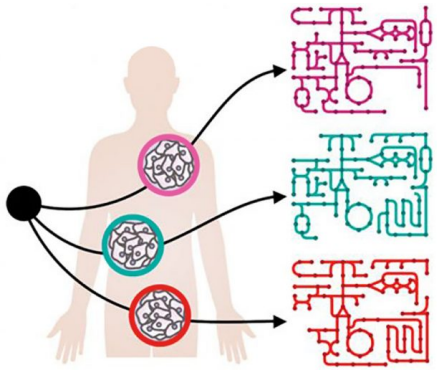


[3]publicly available at: `https://networkrepository.com/`

KIDNEY metabolic graphs

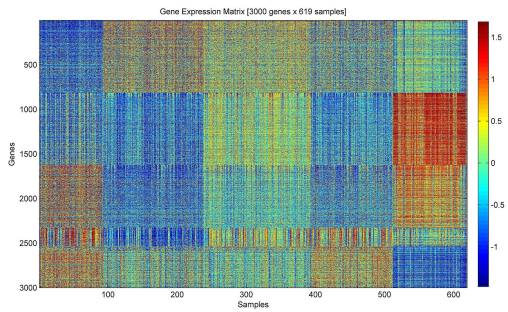
- KIDNEY [8] is a dataset of metabolic models of tissue samples:
  - 299 samples (150 Clear cell carcinoma, 90 Papillary cell carcinoma, 59 Solid tissue normal)
  - raw data from Metabolic Atlas[4]
  - ... enriched by Gene expression values from RNA sequencing data (from NIH Genomic Data Commons data portal [5])

Human specific metabolic models



+

Gene expression data



=

Metabolic networks

[4] https://metabolicatlas.org

[5] https://portal.gdc.cancer.gov

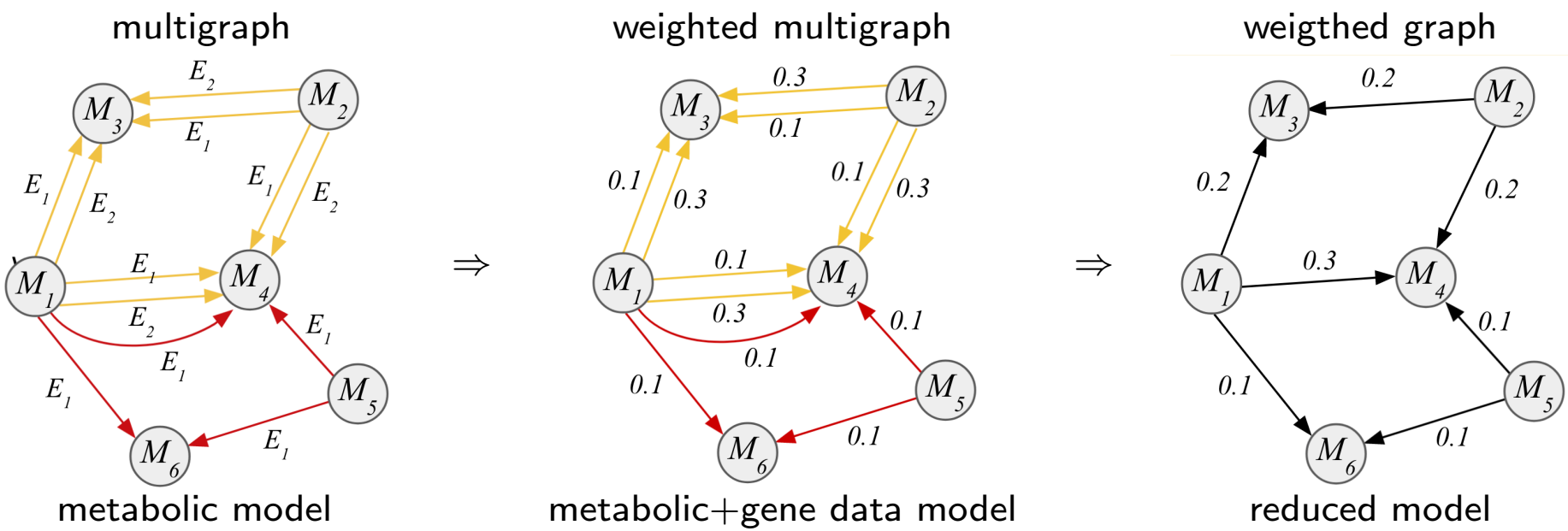
KIDNEY metabolic graphs

- Metabolites are nodes involved in reactions
- Edges connect metabolites involved in the same reaction:
  - one as a reagent and the other one as the product
  - multiple edges for enzymes catalyzing the same reaction
  - .. reduced to one with a weight: avg of gene expression values corresponding to enzymes

KIDNEY metabolic graphs

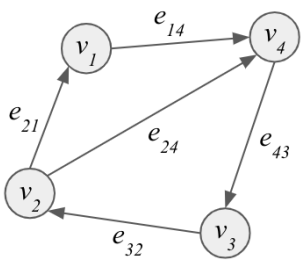
- Metabolites are nodes involved in reactions
- Edges connect metabolites involved in the same reaction:
  - one as a reagent and the other one as the product
  - multiple edges for enzymes catalyzing the same reaction
  - .. reduced to one with a weight: avg of gene expression values corresponding to enzymes

$$R_1 : M_1 + M_2 \xrightarrow{E_1 E_2} M_3 + M_4 \qquad R_2 : M_1 + M_5 \xrightarrow{E_1} M_4 + M_6$$

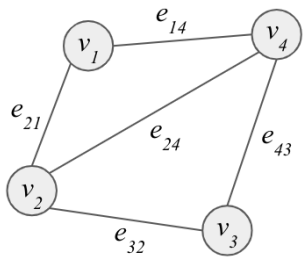


Graph types

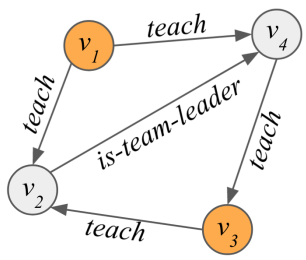
- $G=(V, E)$ is a graph with vertex set $V=\{v_1, \ldots, v_n\}$ and edge set E, such that $(v_i, v_j) \in E$ is a connection from node v_i to node v_j
- Graphs can be:
  - directed: every edge has a specific direction
  - undirected: every edge has no direction
  - homogeneous: all nodes and all edges are of the same type
  - heterogeneous: multiple types of nodes and/or edges are allowed.
  - knowledge graph: a directed heterogeneous graph
  - weigthed: every edge is assigned witha numerical value (weight)
  - binary: no weight associated with edges
  - multigraph: multiple edges with the same end nodes



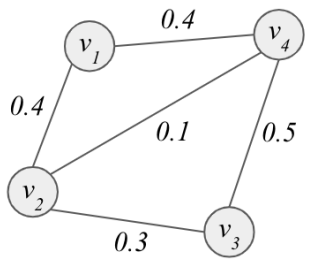
directed graph



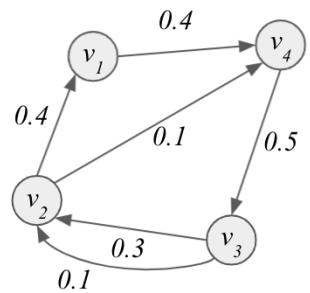
undirected graph



knowlege graph

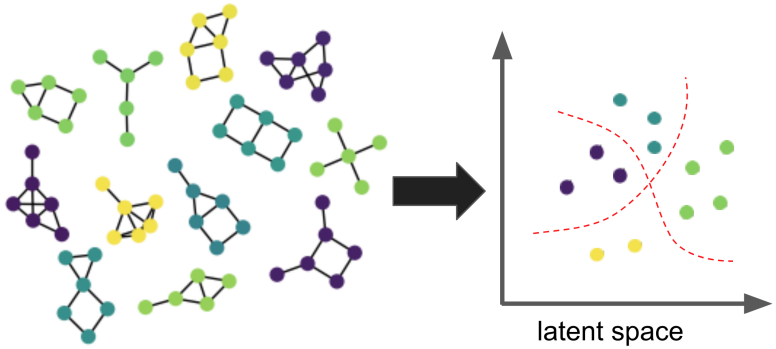


weighted graph



multigraph

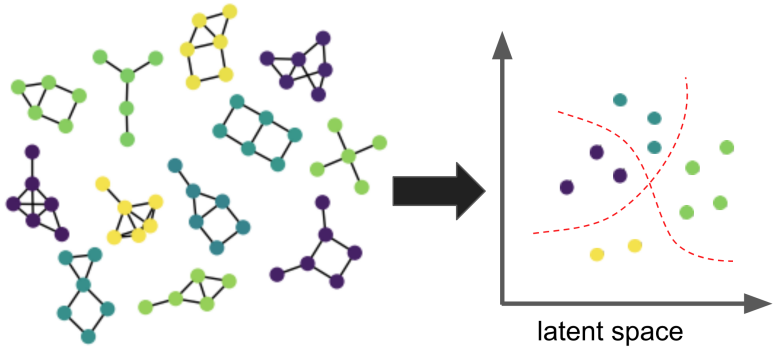
Graph Embedding



Definition (Graph Embedding)

Given a set of graphs $\mathcal{G} = \{G_1, \ldots, G_m\}$ with the same set of vertices V a whole-graph embedding is a mapping function $f : \mathcal{G} \to \mathbb{R}^d$ where $d \in \mathbb{N}$, such that f preserves some proximity measure defined on $\mathcal{G}$

Graph Embedding



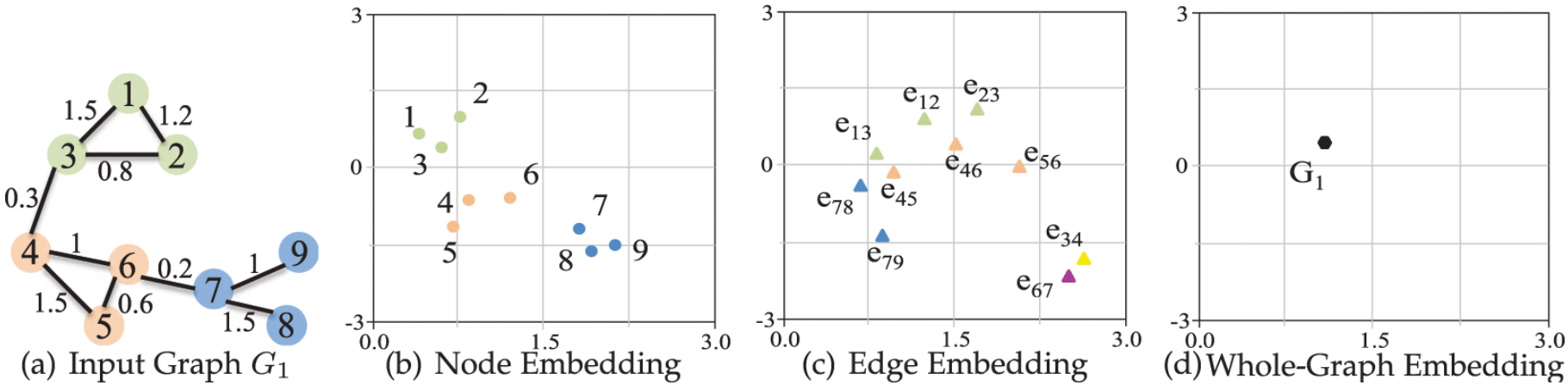
Definition (Graph Embedding)

Given a set of graphs $\mathcal{G} = \{G_1, \ldots, G_m\}$ with the same set of vertices V a whole-graph embedding is a mapping function $f : \mathcal{G} \to \mathbb{R}^d$ where $d \in \mathbb{N}$, such that f preserves some proximity measure defined on $\mathcal{G}$

- Graph Embedding (GE) methods that translate large and complex graphs into a reduced vector space, which is often called latent space.
- Choosing an appropriate embedding dimension d is challenging but necessary to generate embeddings applicable to a multitude of tasks
  - small enough to be efficient and large enough to be effective ($d \ll |V|$)

Embedding types for graphs

- What aspects of the graph are we trying to represent:
  - vertex embeddings: describe connectivity of each node. It targets node prediction, reconstruction, and graph clustering
  - edge/path embeddings: describe traversals across the graph. It targets edge prediction, reconstruction, and graph clustering
  - graph embeddings: encode the entire graph into a single vector. It targets graph classification, graph matching



(a) Input Graph G_1 (b) Node Embedding (c) Edge Embedding (d) Whole-Graph Embedding

Transductive vs inductive embedding

- When talking about GE techniques, it is important to be aware of another distinction:
  - Transductive embedding: the vector representation (embedding) for a new graph is obtained by re-applying the process jointly with previous graphs. The embedding of older graphs changes when we perform the embedding of a new graph
  - Inductive embedding: the vector representation (embedding) for a new graph is obtained by applying a pre-trained model of only on the new graph. The embedding for older graphs does not change when we perform the embedding of a new graph
- The GE process can be unsupervised or supervised
  - transductive supervised embedding methods cannot be used as models for the prediction on unknown graphs

Graph embedding methods

- **Graph kernels**: similarity functions among graphs, typically performing a transformation of graph structure to compare two graphs (Weisfeiler-Lehman [13] Shortest-path [1], Randowm Walk [4]).
- **Statistical Representations**: generate an one-off graph signature vector, based on statistical properties, and use it in subsequent inter-graph comparisons (FGSD [17], FeatherGraph [12])
- **Graph textualization**: represent graphs as documents and reduce graph similarity to a NLP problem. (Graph2Vec [10], GL2vec [16], Netpro2vec [9])
- **Spectral Representations**: use spectral graph theory as a solid ground for graph comparison (NetLSD [15], IGE [2])
- **Graph Neural Networks**: a family of neural networks models that automatically learn embedding (features) for graphs: GNN, GCN, GAE, ...

Graph kernels

Definition

Given a set of graphs $\mathcal{G} = \{G_1, \dots G_m\}$, a function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is a graph kernel if there is a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{G} \to \mathcal{H}$ such that $k(G_i, G_j) = \langle \phi(G_i), \phi(G_j) \rangle$ for $G_i, G_j \in \mathcal{G}$

Graph kernels

Definition

Given a set of graphs $\mathcal{G} = \{G_1, \dots G_m\}$, a function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is a graph kernel if there is a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{G} \to \mathcal{H}$ such that $k(G_i, G_j) = \langle \phi(G_i), \phi(G_j) \rangle$ for $G_i, G_j \in \mathcal{G}$

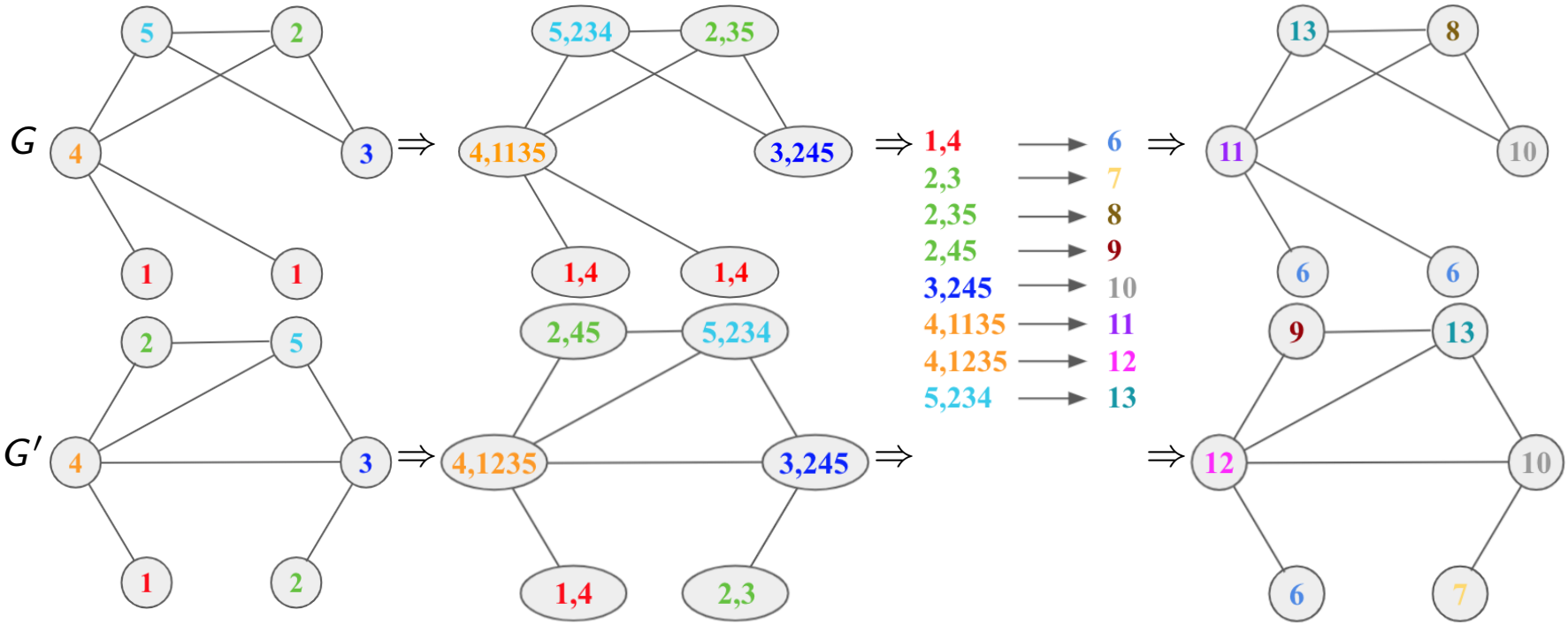
- Where $\langle , \rangle$ is an inner product
- $\phi(G)$ is the embedding of G in the feature space $\mathcal{H}$
- $k(G_i, G_j)$ is a similarity measure for graphs G_i, G_j, e.g. a real number equal to the inner product between G_i and G_j in the feature space $\mathcal{H}$
  - the feature space is assumed to be more computationally manageable, either in terms of space dimensionality or algorithmic complexity
- The kernel matrix (Gram) on the set of graphs $\mathcal{G}$ is defined as: $K_{ij} = k(G_i, G_j)$
  - used for graph classification,
  - not used as graph-level representation (e.g., not applicable to graph matching)

Weisfeiler–Lehman subtree kernel

- An iterative of graph relabeling algorithm [13]:
  1. generation/sorting of multiset-label: node-rooted subtree patterns
  2. label compression: multiset-label are encoded in new unique labels
  3. graph relabeling: nodes are assigned with the new labels
- Embedding vector: i-th element is the no. of occurrences of label i

Weisfeiler–Lehman subtree kernel

- An iterative of graph relabeling algorithm [13]:
  1. generation/sorting of multiset-label: node-rooted subtree patterns
  2. label compression: multiset-label are encoded in new unique labels
  3. graph relabeling: nodes are assigned with the new labels
- Embedding vector: i-th element is the no. of occurrences of label i



$$\phi(G) = [2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1]$$
$$\phi(G') = [1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1]$$
$$\Rightarrow k(G, G') = \langle \phi(G), \phi(G') \rangle = 11$$

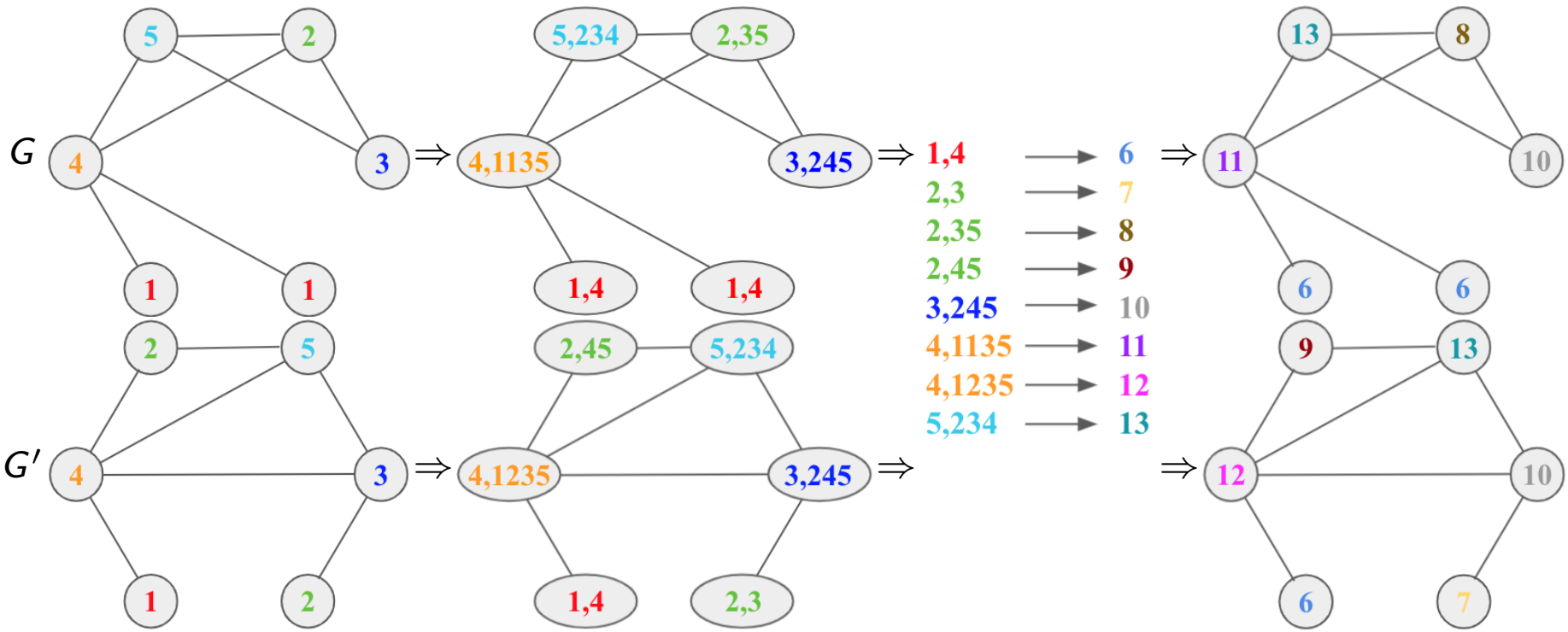
Graph2Vec

- The first approach to graph textualization is **Graph2vec** [10]:
  1. graph relabeling by means of the Weisfeiler–Lehman algorithm
  2. WL output is a sequence of node labels, i.e. a document representation of graph structure (node-rooted subtree patterns):

Graph2Vec

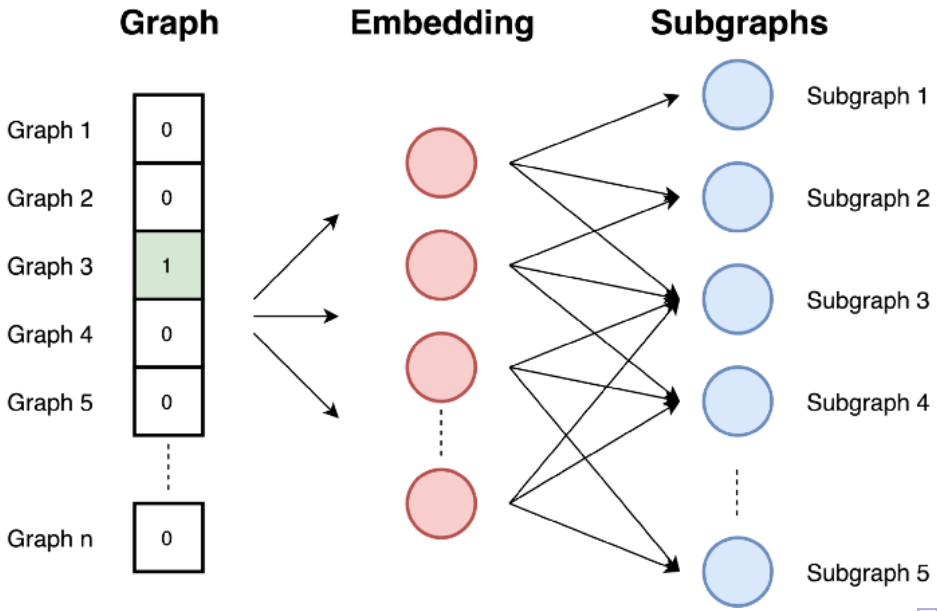
- The first approach to graph textualization is **Graph2vec** [10]:
  1. graph relabeling by means of the Weisfeiler–Lehman algorithm
  2. WL output is a sequence of node labels, i.e. a document representation of graph structure (node-rooted subtree patterns):

$$\phi_{WL}(G) = [\,1, 1, 4, 5, 2, 3, 6, 6, 11, 13, 8, 10\,]$$
$$\phi_{WL}(G') = [\,2, 1, 4, 2, 5, 3, 7, 6, 12, 9, 13, 10\,]$$



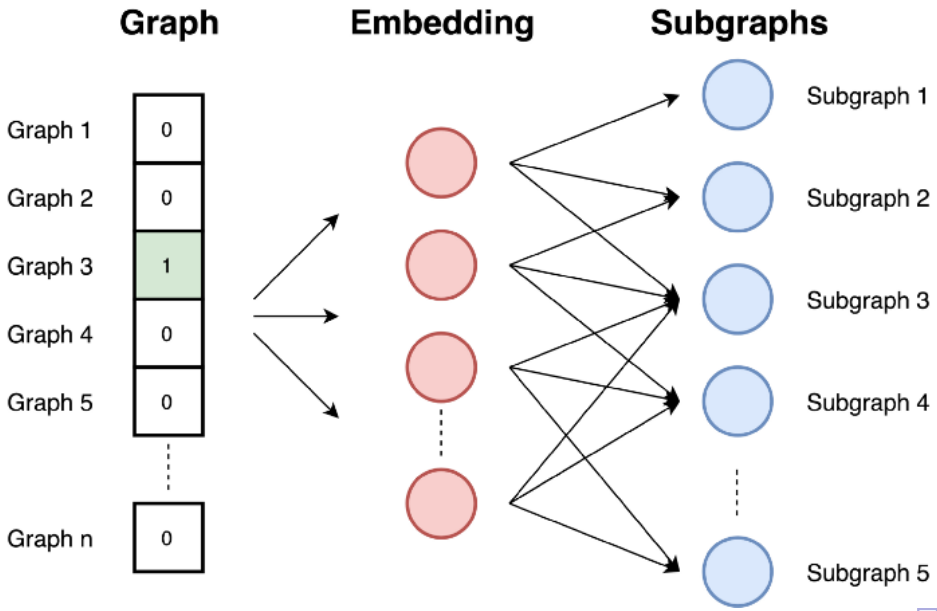
Graph2Vec

- The first approach to graph textualization is **Graph2vec** [10]:
  1. graph relabeling by means of the Weisfeiler–Lehman's algorithm
  2. WL output is a sequence of node labels, i.e. a document representation of graph structure (node-rooted subtree patterns)
  3. train the skip-gram model (PV-DBOW) on graphs (documents) to maximize the probability of predicting a sub-graph that exists in the input graph



Graph2Vec

- The first approach to graph textualization is **Graph2vec** [10]:
  1. graph relabeling by means of the Weisfeiler–Lehman's algorithm
  2. WL output is a sequence of node labels, i.e. a document representation of graph structure (node-rooted subtree patterns)
  3. train the skip-gram model (PV-DBOW) on graphs (documents) to maximize the probability of predicting a sub-graph that exists in the input graph
  4. the embedding of each input graph is the result of the hidden layer of the skip-gram neural model



Netpro2vec

- **Netpro2vec** [9] differs from Graph2Vec in the way graphs are transformed to documents:
  1. graph relabeling by using a set of node-proximity metrics (NDD, TM)
  2. the output is a sequence of node labels, i.e. a document representation of graph structure (node proximity measures):
  3. Train the skip-gram model (PV-DBOW) on graphs (documents) to maximize the probability of predicting a sub-graph that exists in the input graph
  4. the embedding of each input graph is the result of the hidden layer of the skip-gram neural model

Node Distance

Definition (Node Distance)

the Node Distance Distribution (NDD) of node v in graph $G = (V, E)$, namely $N_v(s)$, is the fraction of nodes reachable from v within a shortest path of length s from node v

- maximal depth s_{max} is set to the maximal diameter in all graphs
- nodes in disconnected components are considered at infinite distance

Node Distance

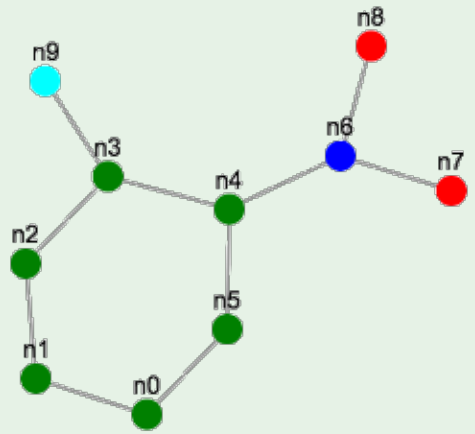
Definition (Node Distance)

the Node Distance Distribution (NDD) of node v in graph $G = (V, E)$, namely $N_v(s)$, is the fraction of nodes reachable from v within a shortest path of length s from node v

- maximal depth s_{max} is set to the maximal diameter in all graphs
- nodes in disconnected components are considered at infinite distance

Example (node distance of $mutag_{180}$)

$$NDD_{mutag_{180}} = \begin{array}{c} \\ n0 \\ n1 \\ n2 \\ n3 \\ n4 \\ n5 \\ n6 \\ n7 \\ n8 \\ n9 \end{array} \begin{array}{c} \begin{array}{cccccc} s=0 & s=1 & s=2 & s=3 & s=4 & s=5 \end{array} \\ \begin{pmatrix} 0.11 & 0.22 & 0.22 & 0.22 & 0.33 & 0.00 \\ 0.11 & 0.22 & 0.22 & 0.22 & 0.11 & 0.22 \\ 0.11 & 0.22 & 0.33 & 0.22 & 0.22 & 0.00 \\ 0.11 & 0.33 & 0.33 & 0.33 & 0.00 & 0.00 \\ 0.11 & 0.33 & 0.56 & 0.11 & 0.00 & 0.00 \\ 0.11 & 0.22 & 0.33 & 0.44 & 0.00 & 0.00 \\ 0.11 & 0.33 & 0.22 & 0.33 & 0.11 & 0.00 \\ 0.11 & 0.11 & 0.22 & 0.22 & 0.33 & 0.11 \\ 0.11 & 0.11 & 0.22 & 0.22 & 0.33 & 0.11 \\ 0.11 & 0.11 & 0.22 & 0.33 & 0.33 & 0.00 \end{pmatrix} \end{array}$$



Transition Matrix

Definition (Transition Matrix of order s)

The Transition Matrix of order s for graph $G = (V, E)$, namely $T_{vw}(s)$. is the probability of a node w to be reached in s steps by a random walker located in node v

- $T(1)$ is the adjacency matrix of graph G re-scaled by the degree of each node ($T(s) = T(1)^s$)

Transition Matrix

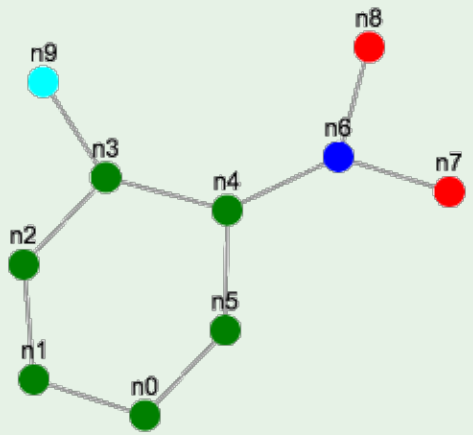
Definition (Transition Matrix of order s)

The Transition Matrix of order s for graph $G = (V, E)$, namely $T_{vw}(s)$. is the probability of a node w to be reached in s steps by a random walker located in node v

- $T(1)$ is the adjacency matrix of graph G re-scaled by the degree of each node ($T(s) = T(1)^s$)

Example (Transition matrix of order 1 of $mutag_{180}$)

$$TM^s_{mutag_{180}} = \begin{array}{c} \\ n0 \\ n1 \\ n2 \\ n3 \\ n4 \\ n5 \\ n6 \\ n7 \\ n8 \\ n9 \end{array} \begin{array}{c} \begin{array}{cccccccccc} n0 & n1 & n2 & n3 & n4 & n5 & n6 & n7 & n8 & n9 \end{array} \\ \begin{pmatrix} 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3 & 0.0 & 0.3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.3 & 0.0 & 0.3 & 0.3 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.3 & 0.0 & 0.0 & 0.3 & 0.3 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \end{array}$$



Netpro2vec: graphs relabeling

- Compute NDD distributions for all graphs in the dataset
- For each node v, get s values ordered by decreasing values of $N_v(s)$
- Keep only s values such taht $N_v(s)$ is greater than a threshold (cutoff)
- Every node is represented by a word, i.e. a string concatenation of node id v with the selected s values
- ... the sequence of words is the graph document (same node order)

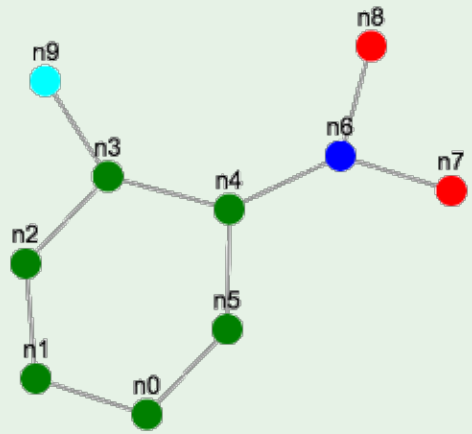
Netpro2vec: graphs relabeling

- Compute NDD distributions for all graphs in the dataset
- For each node v, get s values ordered by decreasing values of $N_v(s)$
- Keep only s values such taht $N_v(s)$ is greater than a threshold (cutoff)
- Every node is represented by a word, i.e. a string concatenation of node id v with the selected s values
- ... the sequence of words is the graph document (same node order)

Example (NDD words for $mutag_{180}$)

cut_off $= 0.25$

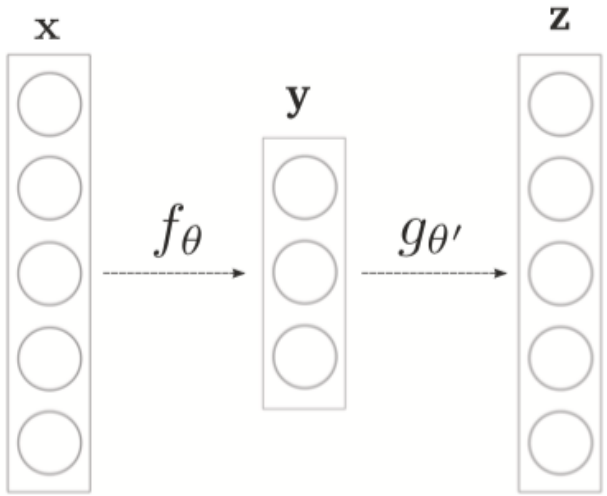
$$NDD_{mutag_{180}} = \begin{array}{c} \\ n0 \\ n1 \\ n2 \\ n3 \\ n4 \\ n5 \\ n6 \\ n7 \\ n8 \\ n9 \end{array} \begin{array}{c} \begin{array}{cccccc} s=0 & s=1 & s=2 & s=3 & s=4 & s=5 \end{array} \\ \begin{pmatrix} 0.11 & 0.22 & 0.22 & 0.22 & 0.33 & 0.00 \\ 0.11 & 0.22 & 0.22 & 0.22 & 0.11 & 0.22 \\ 0.11 & 0.22 & 0.33 & 0.22 & 0.22 & 0.00 \\ 0.11 & 0.33 & 0.33 & 0.33 & 0.00 & 0.00 \\ 0.11 & 0.33 & 0.56 & 0.11 & 0.00 & 0.00 \\ 0.11 & 0.22 & 0.33 & 0.44 & 0.00 & 0.00 \\ 0.11 & 0.33 & 0.22 & 0.33 & 0.11 & 0.00 \\ 0.11 & 0.11 & 0.22 & 0.22 & 0.33 & 0.11 \\ 0.11 & 0.11 & 0.22 & 0.22 & 0.33 & 0.11 \\ 0.11 & 0.11 & 0.22 & 0.33 & 0.33 & 0.00 \end{pmatrix} \end{array} \rightarrow \begin{array}{l} ndd\_n0\_4 \\ - \\ ndd\_n2\_2 \\ ndd\_n3\_3\_2\_1 \\ ndd\_n4\_2\_1 \\ ndd\_n5\_3\_2 \\ ndd\_n6\_3\_1 \\ ndd\_n7\_4 \\ ndd\_n8\_4 \\ ndd\_n9\_4\_3 \end{array}$$



Graph Auto-Encoders

- Graph Auto-Encoders (GAEs) [6] learn a compact representation of a graph and then re-construct it by using the decoder
  - used to learn graph embeddings, hence for predicting embeddings for un-seen graphs and to classify new graphs
  - auto-encoders parameters are optimized by minimizing the average reconstruction error over the training set:

$$\theta^*, \theta'^* = \underset{\theta,\theta'}{arg\,\min} \frac{1}{m} \sum_{i=1}^{m} \left\| \mathbf{x}^{(i)} - \mathbf{z}^{(i)} \right\|_2^2$$

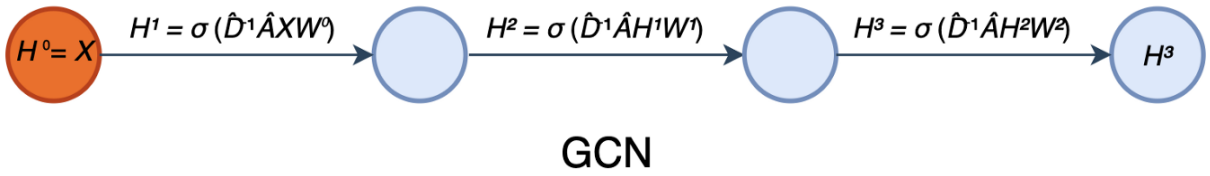


Graph Convolutional Network

- Graph Convolutional Networks (GCNs) [5] are adaptation of Convolutional Neural Networks (used in image recognition) to the graph domain

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

  - H - hidden state (or node attributes when $l = 0$)
  - $\tilde{D}$ - degree matrix
  - $\tilde{A}$ - adjacency matrix (with self-loops)
  - W - trainable weights
  - σ - activation function
  - l - layer number
- H_0 is the network input (e.g. node attributes)
- the embedding is the output of the last hidden layer



Results

	method	MUTAG	KIDNEY
GNN	DAE	-	90.33*
GK	WL-OA	84.59±9.59	55.51±2.35
	SP	84.59±9.59	53.17±0.48†
Spectral	NetLSD	86.48±6.57	53.84±6.20
Stat.	FGSD	94.11±5.32	86.66±6.29
	FeatherGraph	82.66±7.85	81.47±5.62
GT	Graph2Vec	80.99±7.63	53.17±0.48†
	Netpro2vec^{ndd}	99.47±1.59	92.44±5.43
	Netpro2vec^{tm1}	99.57±1.79	95.58±3.45

Table: Accuracy of 10-fold cross-validation

- Graph2vec, NetLSD, WL-OA and SP fail on KIDNEY: they don't use weights (graphs have same topology with different edge weights)
- Netpro2vec shows top-most performance in both case studies
  - † is null classification
  - .. see other results in [9]

*results reported in [7]

Conclusions

- Netpro2vec exploits node proximity measures to transform graphs into documents, while preserving their significant structural properties
- Netpro2vec relies on a Natural Language Processing method[7] to extract, from each document-based graph, the meaningful features in terms of vector (embedding).
- Graph embeddings produced by Netpro2vec can be used for multiple ML tasks (clustering, classification, matching, etc.)
- PROS:
  - efficient embeddings in different graph data domains
  - shallow architecture
  - inductive learning
- CONS: performance depends on:
  - fine tuning of skipgram NN parameters
  - appropriate choice of the proximity information (NDD, $TM(s)$) to extract as well as its level of details

[7] skipgram model

Considerations

- Efficiency/scalability:
  - due to a pair-wise similarity calculation, GKs suffer significantly from computational bottlenecks (poor scalability)
  - some GNNs are more efficient since they can directly perform graph classification based on the extracted graph representations
  - GNNs on small datasets may results in poor approx. (over-fitting)
  - GT* methods are based on shallow learning: faster in training (less parameters) wrt GNNs
- Embedding properties:
  - GNNs exploit more levels of embeddings of input data, which is expected to result in more powerful feature extraction
  - GT/GNNs embedding are learnable, while GK embedding are deterministic (hand-crafted)
  - GK embedding size grows with number of samples
  - GT/GNN embedding size is fixed
    - ★ best choice: small enough to be efficient and large enough to be effective in representing the graph proximity

*here discussed: Graph2vec, GL2Vec, Netpro2vec

Thank you ...

References I

[1] K. Borgwardt and H. Kriegel.
Shortest-path kernels on graphs.
In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–, 2005.

[2] A. Galland and M. Lelarge.
Invariant embedding for graph classification.
In *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*, 2019.

[3] J. O. Garcia, A. Ashourvan, S. Muldoon, J. M. Vettel, and D. S. Bassett.
Applications of community detection techniques to brain graphs: Algorithmic considerations and implications for neural function.
Proceedings of the IEEE, 106(5):846–867, 2018.

[4] T. Gärtner, P. Flach, and S. Wrobel.
On graph kernels: Hardness results and efficient alternatives.
In B. Schölkopf and M. K. Warmuth, editors, *Learning Theory and Kernel Machines*, pages 129–143, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[5] T. N. Kipf and M. Welling.
Semi-supervised classification with graph convolutional networks.
arXiv preprint arXiv:1609.02907, 2016.

[6] T. N. Kipf and M. Welling.
Variational graph auto-encoders, 2016.

[7] L. Maddalena, I. Manipur, M. Manzo, and M. R. Guarracino.
On Whole-Graph Embedding Techniques, pages 115–131.
Springer International Publishing, Cham, 2021.

[8] I. Manipur, I. Granata, L. Maddalena, and M. R. Guarracino.
Clustering analysis of tumor metabolic networks.
BMC Bioinformatics, 21(10):349, 2020.

References II

[9] I. Manipur, M. Manzo, I. Granata, M. Giordano, L. Maddalena, and M. Guarracino.
Netpro2vec: a graph embedding framework for biomedical applications.
IEEE/ACM transactions on computational biology and bioinformatics, PP, 2021.

[10] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal.
graph2vec: Learning distributed representations of graphs.
CoRR, abs/1707.05005, 2017.

[11] G. Nikolentzos, P. Meladianos, A. J.-P. Tixier, K. Skianis, and M. Vazirgiannis.
Kernel graph convolutional neural networks.
In V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 22–32, Cham, 2018. Springer International Publishing.

[12] B. Rozemberczki and R. Sarkar.
Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models, page 1325–1334.
Association for Computing Machinery, New York, NY, USA, 2020.

[13] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt.
Weisfeiler-lehman graph kernels.
Journal of Machine Learning Research, 12(77):2539–2561, 2011.

[14] L. Tang and H. Liu.
Graph Mining Applications to Social Network Analysis, pages 487–513.
Springer US, Boston, MA, 2010.

[15] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller.
Netlsd: Hearing the shape of a graph.
In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &; Data Mining*, KDD '18, page 2347–2356, New York, NY, USA, 2018. Association for Computing Machinery.

References III

[16] K. Tu, J. Li, D. Towsley, D. Braines, and L. D. Turner.
gl2vec: Learning feature representation using graphlets for directed networks.
In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 216–221, 2019.

[17] S. Verma and Z.-L. Zhang.
Hunt for the unique, stable, sparse and fast feature learning on graphs.
In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 87–97, Red Hook, NY, USA, 2017. Curran Associates Inc.