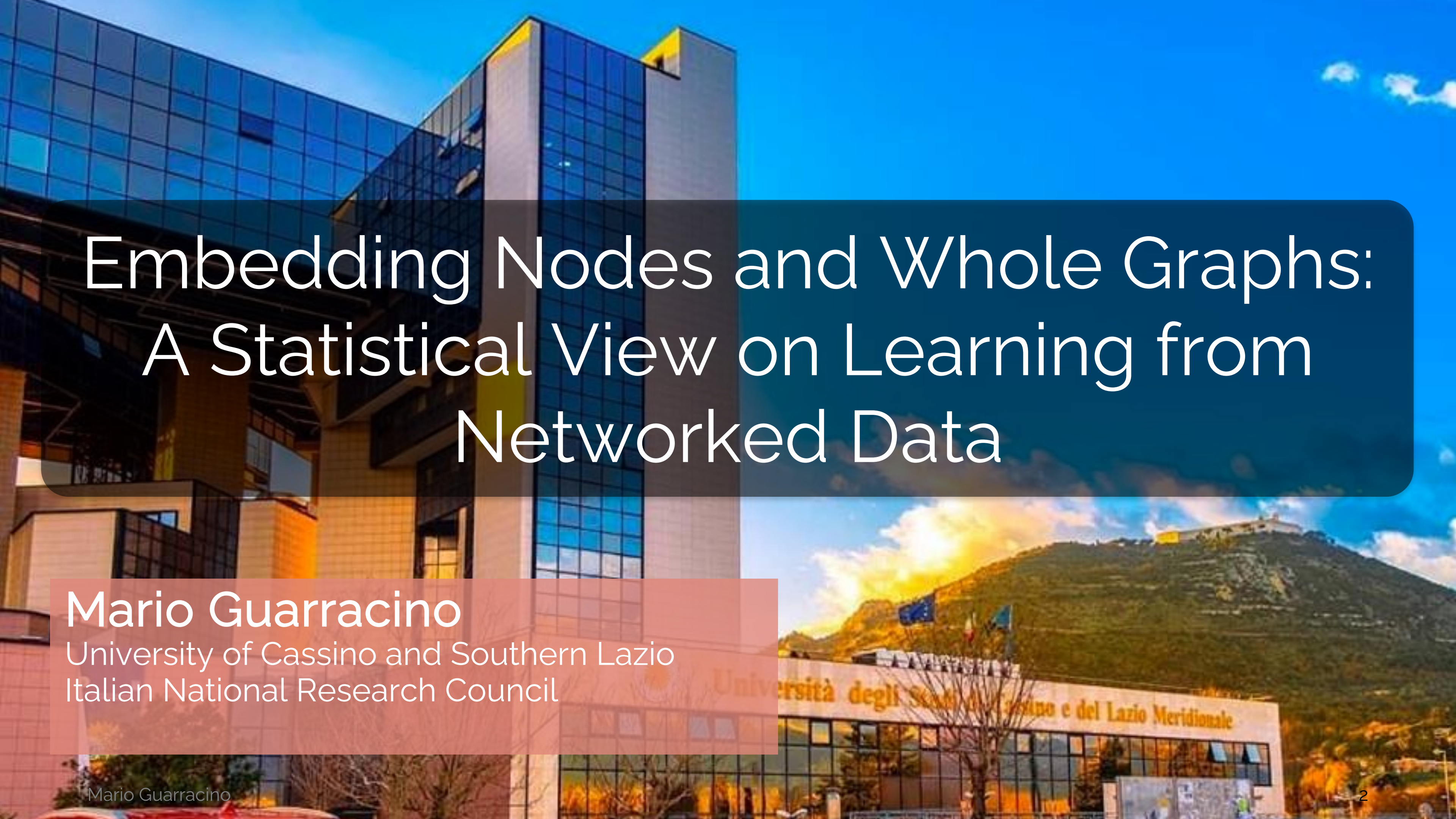


Embedding Nodes and Whole Graphs: A Statistical View on Learning from Networked Data

Mario Guarracino
University of Cassino and Southern Lazio
Italian National Research Council

Introduction

- Network representations capture interactions and dependencies among variables or observations.
  - These can be extended to consider multiple networks, offering flexible and powerful modeling of complex phenomena.
- Graph embedding techniques map nodes, subgraphs, or entire (multiple) graphs into a vector space while preserving structural properties.
- A rich set of methods, including graph kernels, matrix factorization, and deep learning architectures, supports tasks such as feature extraction, graph clustering, and classification.

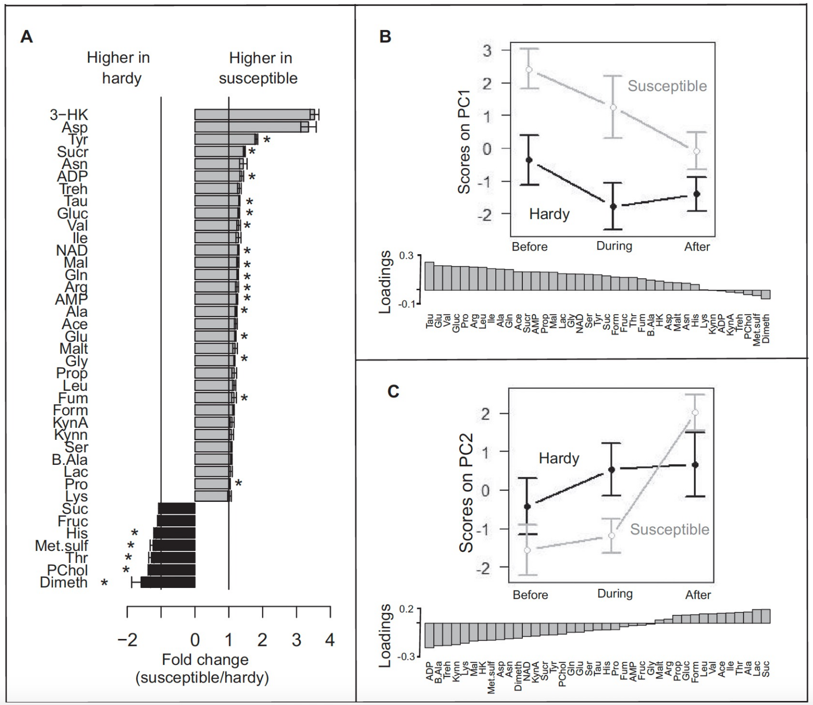
doi:10.1111/evo.12541

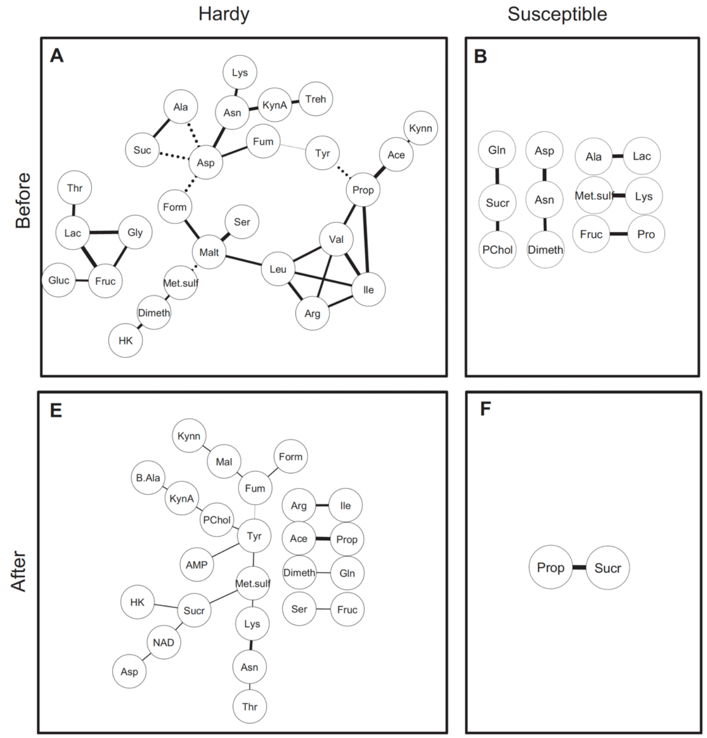
Cold adaptation shapes the robustness of metabolic networks in *Drosophila melanogaster*

EVOLUTION
International Journal of Organic Evolution
OXFORD ACADEMIC

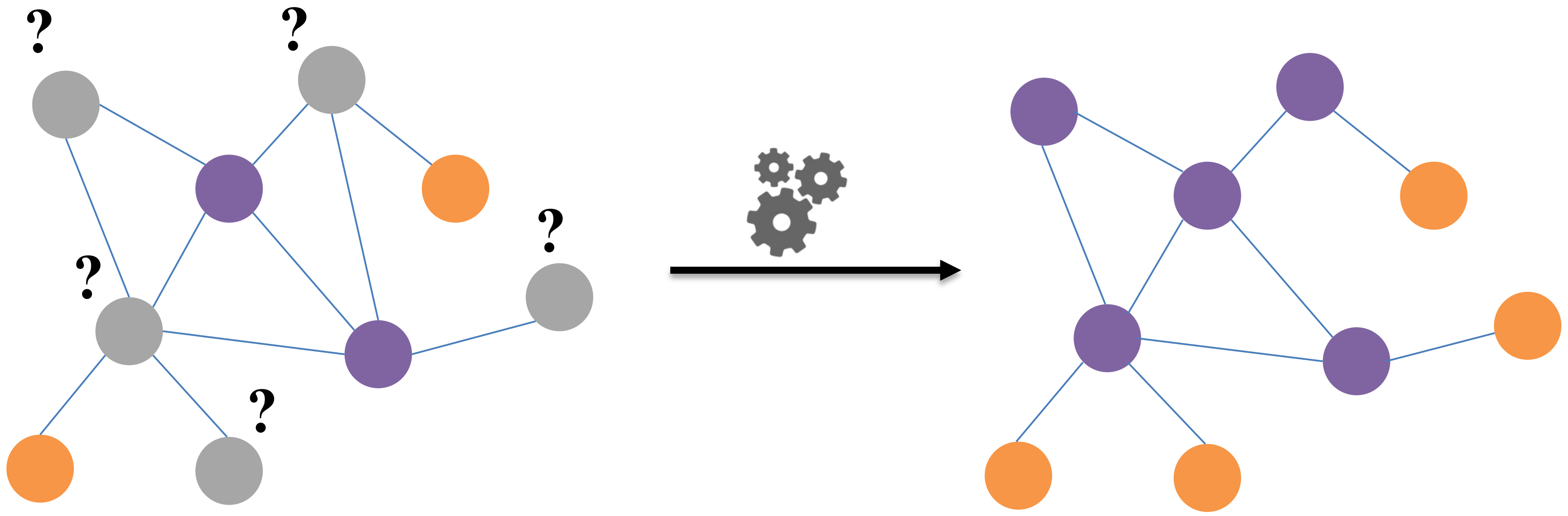
Caroline M. Williams,[1,2] Miki Watanabe,[3] Mario R. Guarracino,[4] Maria B. Ferraro,[5] Arthur S. Edison,[6] Theodore J. Morgan,[7] Arezue F. B. Boroujerdi,[8] and Daniel A. Hahn[9]

CM Williams



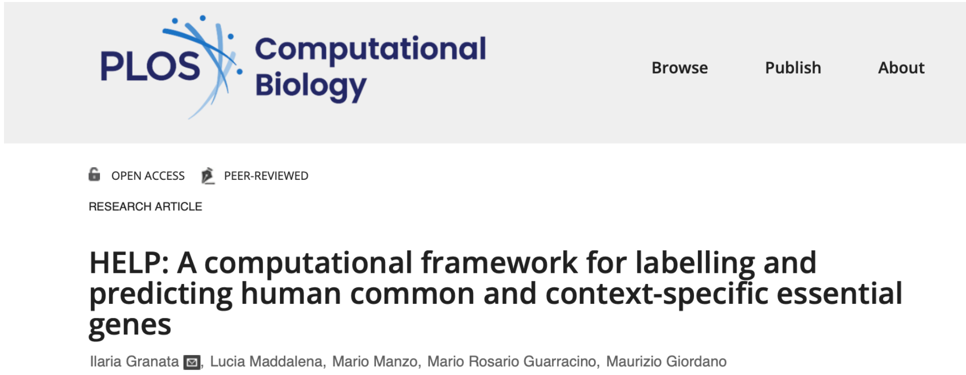


Example: Node Classification



Adapted from *Representation Learning on Networks*, snap.stanford.edu/proj/embeddings-www

Essential genes



PLOS Computational Biology

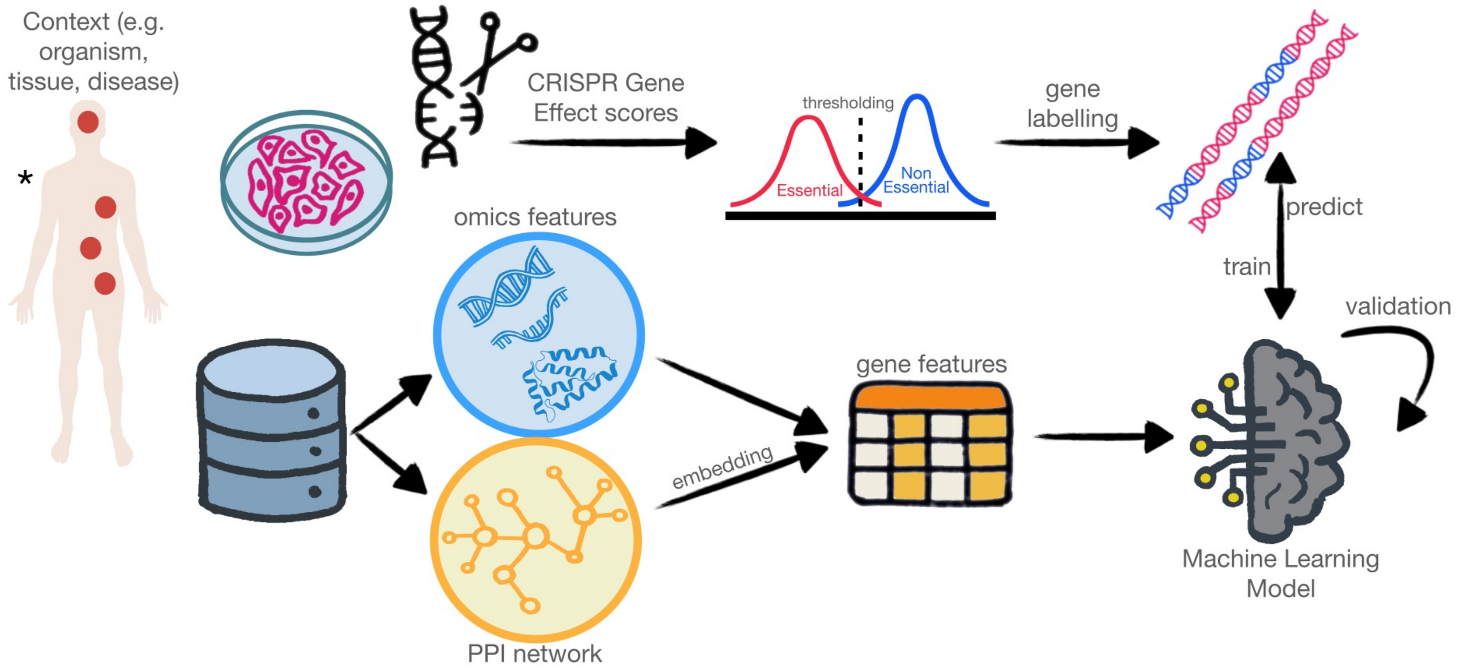
Browse Publish About

OPEN ACCESS PEER-REVIEWED

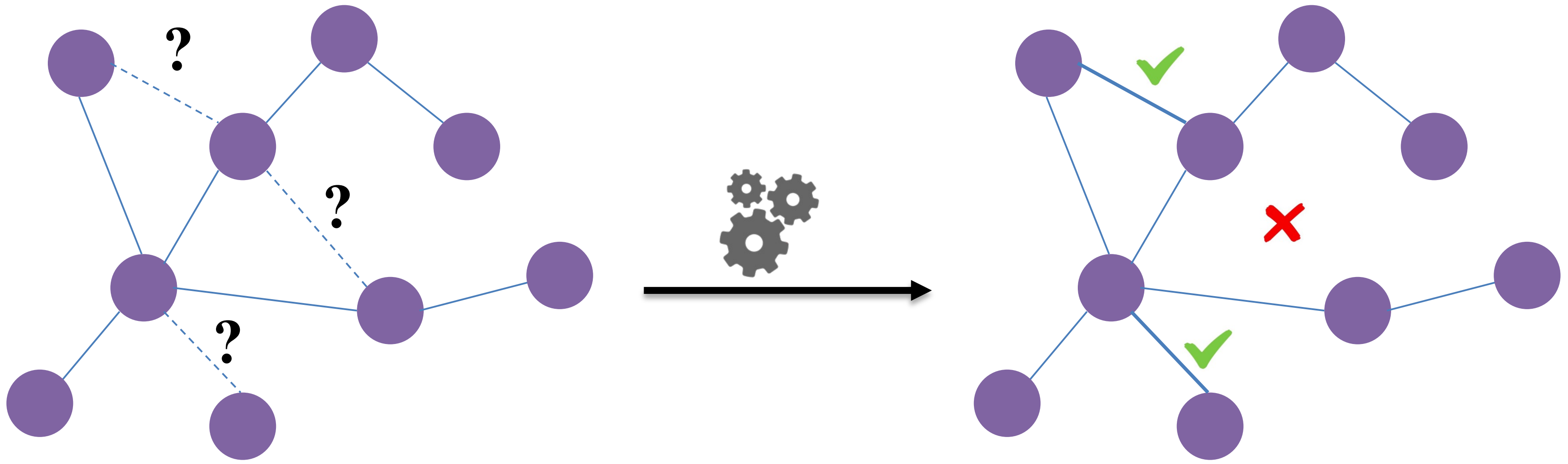
RESEARCH ARTICLE

HELP: A computational framework for labelling and predicting human common and context-specific essential genes

Ilaria Granata, Lucia Maddalena, Mario Manzo, Mario Rosario Guarracino, Maurizio Giordano

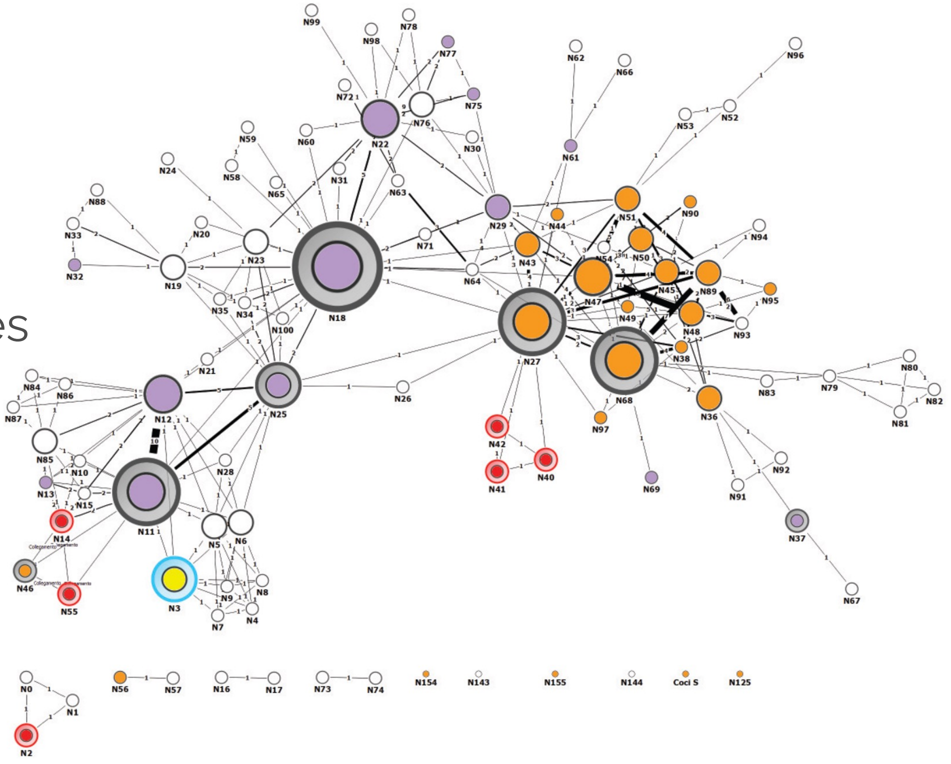


Example: Link Prediction



Example: mafia meetings

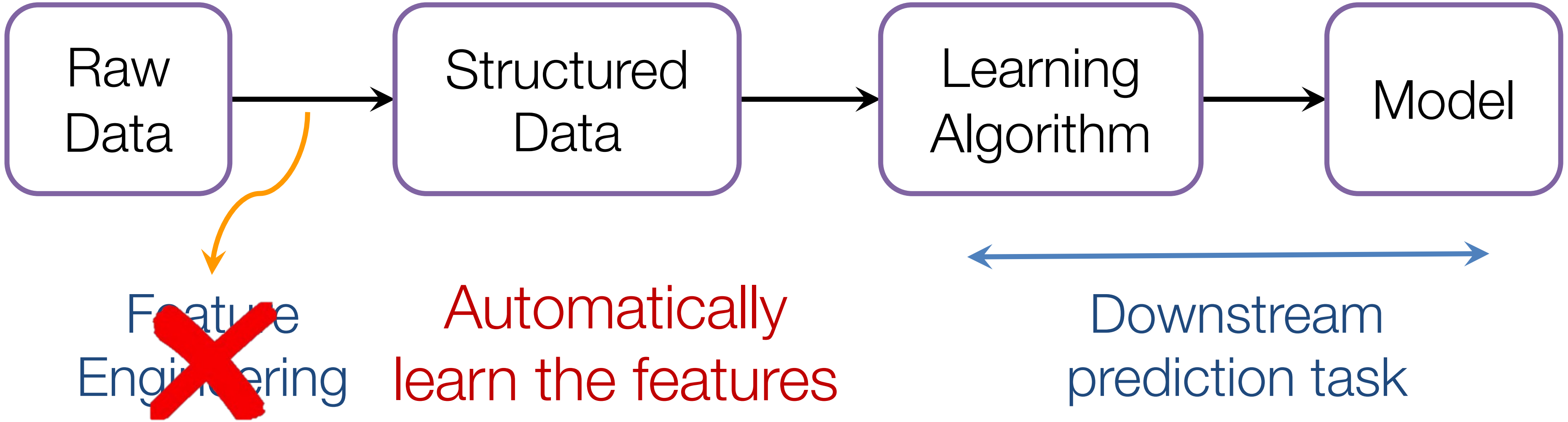
- Nodes show members of the "Mistretta" and "Batanesi" families.
- Circled nodes mark investigated association leaders.
  - The red and yellow circles indicates bosses of other districts.
- White nodes are other relevant associates.
- Edge width reflects meeting frequency, and node size reflects degree.



Calderoni et al, Robust link prediction in criminal networks: A case study of the Sicilian Mafia, Expert Syst. App., 2020.

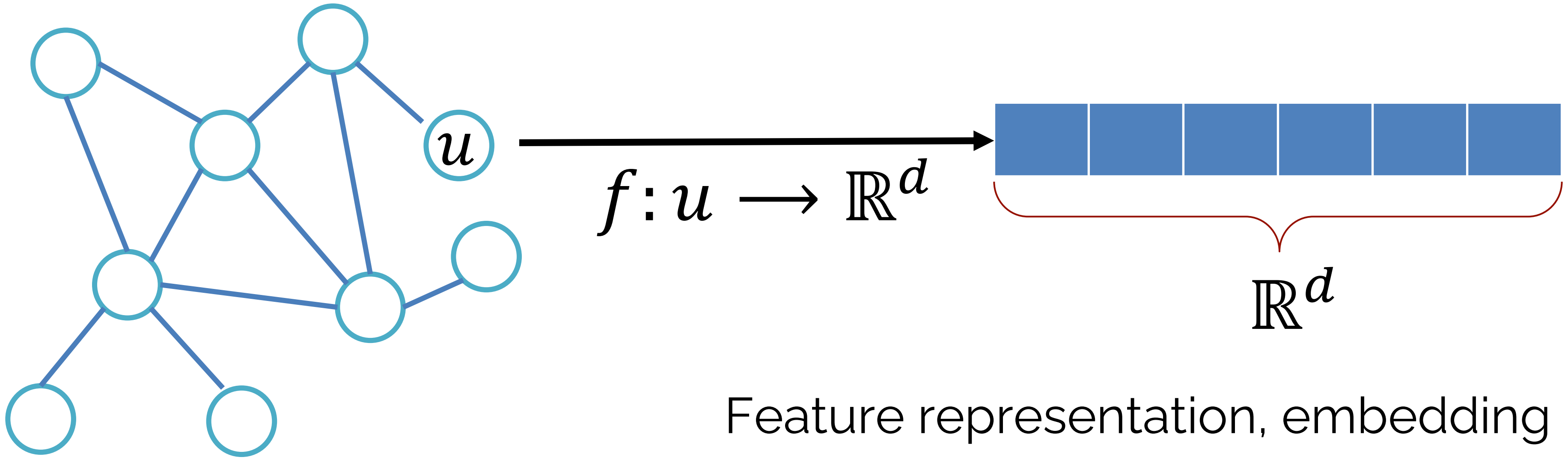
SL Lifecycle

- (Supervised) Statistical Learning Lifecycle: This feature, that feature. Every single time!



Feature learning on graphs

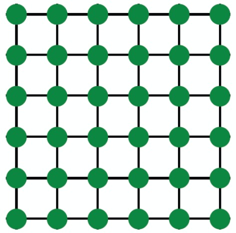
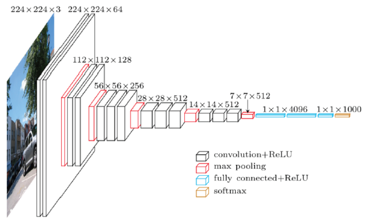
Goal: Efficient task-independent feature learning in networks!



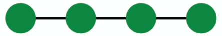
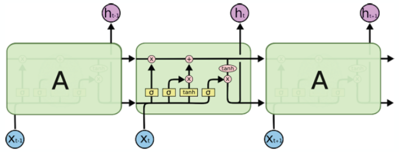
Feature representation, embedding

Why is it hard?

- Modern deep learning toolboxes are designed for simple sequences or grids.
  - CNNs for fixed-size images/grids….

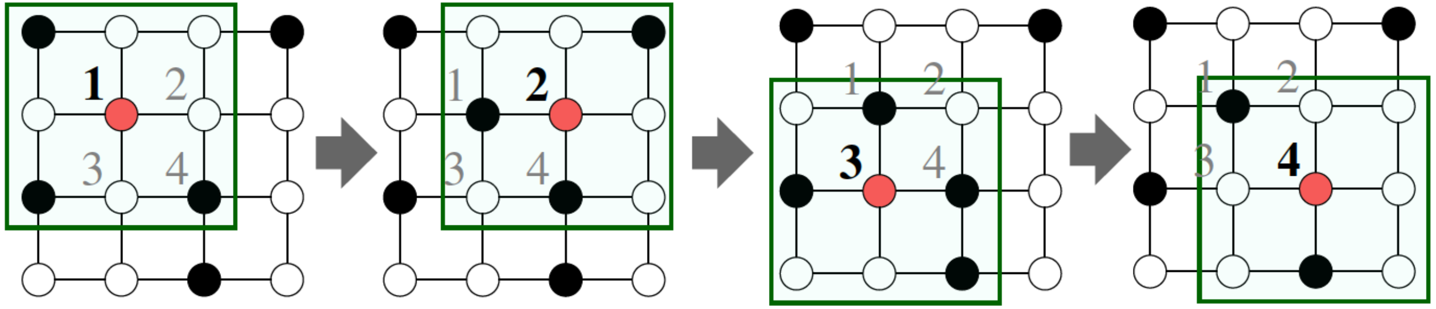



  - RNNs or word2vec for text/sequences…

Networks are complex!

- Complex topographical structure (i.e., no spatial locality like grids).



- No fixed node ordering or reference point (i.e., the isomorphism problem)
- Often dynamic and with multimodal features.

Traditional approaches

- Traditional graph-based methods follow the pre–deep learning paradigm:
  - Compute hand-crafted features or statistics first, often guided by heuristics or domain expertise;
  - Then feed those features into a standard learning algorithm such as logistic regression.

Graph statistics

- *Node degree*: d_u simply counts the number of edges incident to a node $u \in \mathcal{V}$:

$$d_u = \sum_{v \in \mathcal{V}} \mathbf{A}[u, v]$$

- *Node eigenvector centrality*: e_u is recursively defined by a relation in which a node's score is proportional to the sum of its neighbors' centralities:

$$e_u = 1/\lambda \sum_{v \in V} A[u, v] e_v \; \forall u \in \mathcal{V},$$

- where λ is a constant.

Graph statistics

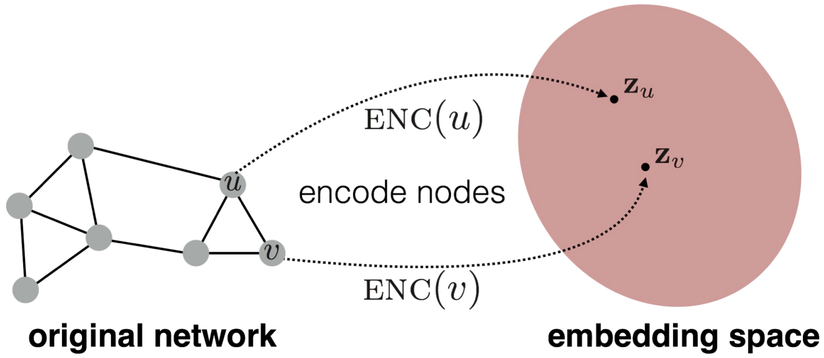
- Clustering coefficient:

$$c_u = \frac{|(v_1, v_2) \in \mathcal{E}: v_1, v_2 \in \mathcal{N}(u)|}{\binom{d_u}{2}}$$

  - Number of edges between neighbors of node u in $\mathcal{N}(u) = \{v \in \mathcal{V} : (u, v) \in \mathcal{E}\}$ divided by the total pairs of nodes in u's neighborhood.
- The clustering coefficient counts the number of closed triangles within each node's local neighborhood.
- We can consider more complex structures, such as cycles of fixed length, and characterize nodes by counts in their ego graph.

Node embeddings

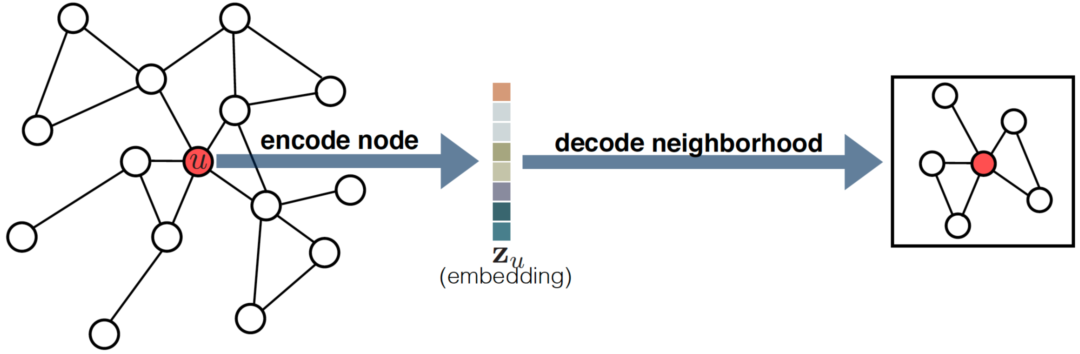
- These methods encode nodes as low-dimensional vectors, summarizing the structure of their local graph neighborhood.
- In other words, they project nodes into a latent space, where geometric relations correspond to relationships (e.g., edges) in the original graph.
- Node embeddings can be explained in the framework of encoding and decoding graphs.



Graphics from WL Hamilton Graph Representation, 2020.

Encoding and decoding graphs

- First, an encoder model maps each node in the graph into a low-dimensional vector or embedding.
- Next, a decoder model takes the low-dimensional node embeddings and uses them to reconstruct information about each node's neighborhood in the original graph.



The encoder

- The encoder maps nodes $v \in V$ to vector embedding $z_v \in \mathbb{R}^d$, where z_v corresponds to the embedding for node $v \in V$.
- In the simplest case, the encoder has the following form:

$$\mathrm{ENC} : V \to \mathbb{R}^d$$

- The encoder often relies on what we call the *shallow embedding* approach, where this encoder is simply an embedding lookup based on the node ID:

$$\mathrm{ENC}(v) = \mathbf{Z}[v]$$

- where $\mathbf{Z} \in \mathbb{R}^{|V| \times d}$ is a matrix containing the embedding vectors for all nodes and $\mathbf{Z}[v]$ denotes the row of $\mathbf{Z}$ corresponding to node v.

The decoder

- The role of the decoder is to reconstruct some graph characteristics from the node embeddings that are generated by the encoder.
  - For example, given a node u embedding $\mathbf{z}_u$, the decoder might attempt to predict the neighborhood $\mathcal{N}(u)$ of u.
- It is standard to define pairwise decoders, which have the following signature:

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+.$$

- Pairwise decoders can be interpreted as predicting the relationship or similarity between pairs of nodes.

The decoder

- Applying the pairwise decoder to a pair of embeddings $(\mathbf{z}_u, \mathbf{z}_v)$ results in reconstructing the relationship between u and v.
- The goal is optimizing the encoder and decoder to minimize the reconstruction loss, so that:

$$\text{DEC}\big(\text{ENC}(u); \text{ENC}(v)\big) = \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \approx \mathbf{S}[u; v]$$

- Here, we assume that $\mathbf{S}[u; v]$ is a graph-based similarity measure between nodes.
- For example, a simple reconstruction objective of predicting whether two nodes are neighbors would correspond to

$$\mathbf{S}[u; v] \triangleq \mathbf{A}[u, v].$$

Optimizing an Encoder-Decoder

- The standard practice is to minimize an empirical reconstruction loss $\mathcal{L}$ over a set of training node pairs $\mathcal{D}$:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \ell(\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u; v]),$$

- where $\ell: \mathbb{R}\times\mathbb{R} \rightarrow \mathbb{R}$ is a loss function measuring the discrepancy between the estimated $\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v)$ and the true values $\mathbf{S}[u; v]$.
- Depending on the definition of DEC and $\mathbf{S}$, the loss function ℓ can be a mean-squared error or even a classification loss.
- Most approaches minimize the loss using stochastic gradient descent, but matrix factorization can be also used.

Encoder-Decoder Approaches

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u,v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u,v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u,v], ..., \mathbf{A}^k[u,v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
DeepWalk	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k\in\mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$	$-\mathbf{S}[u,v]\log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$
node2vec	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k\in\mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$ (biased)	$-\mathbf{S}[u,v]\log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

Limits

- Shallow embedding methods do not share any parameters between nodes in the encoder, since the encoder directly optimizes a unique embedding vector for each node.
- This lack of parameter sharing is both statistically and computationally inefficient.
  - From a statistical perspective, parameter sharing can improve the efficiency of learning and also act as a powerful form of regularization.
  - From the computational perspective, the number of parameters necessarily grows as $\mathcal{O}(|V|)$, which can be intractable in massive graphs.
- Shallow embedding approaches not leverage node features in the encoder.

Graph Neural Networks

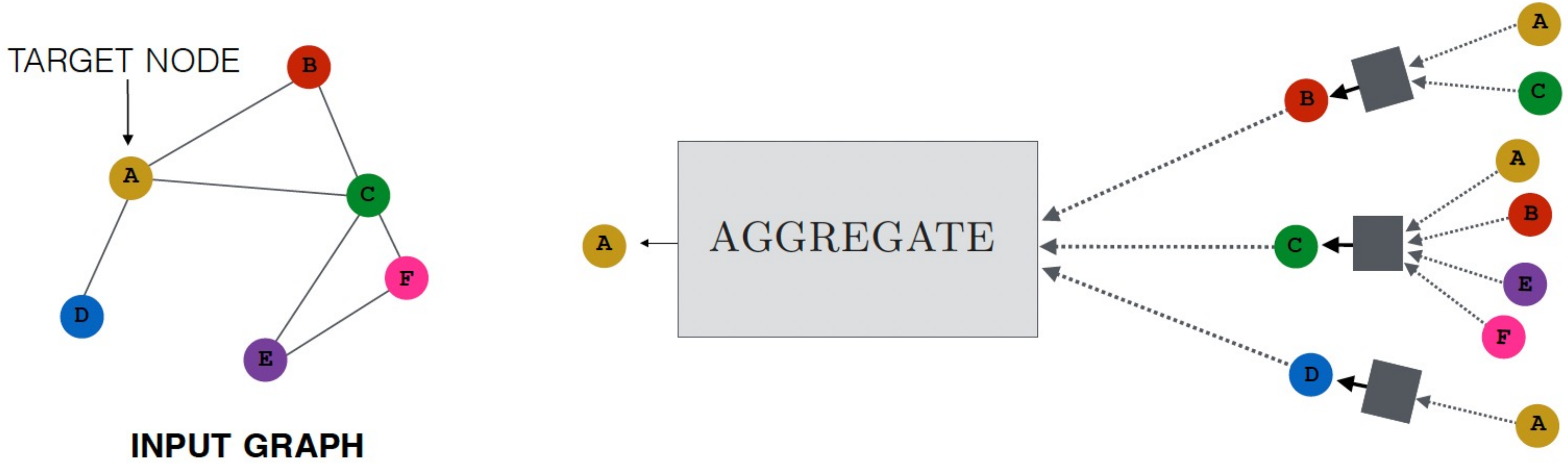
- Graph neural network (GNN) formalism is a general framework for defining deep neural networks on graph data.
- The key idea is to generate representations of nodes that actually depend on the structure of the graph, as well as any feature information we might have.

Neural Message Passing

- The defining feature of a GNN is that it uses a form of neural message passing in which vector messages are exchanged between nodes and updated using neural networks [Gilmer et al., 2017].
- We will now focus on the message passing framework itself and describe how we can take an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, along with a set of node features $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|}$, and use this information to generate node embeddings $\mathbf{z}_u, u \in \mathcal{V}$.

Overview of the Message Passing

- During each message-passing iteration in a GNN, a *hidden embedding* $\mathbf{h}_u^{(k)}$ corresponding to each node $u \in \mathcal{V}$ is updated according to information aggregated from u's graph neighborhood $\mathcal{N}(u)$



Overview of the Message Passing

- This message-passing update can be expressed as follows:

$$\begin{aligned}\mathbf{h}_u^{(k+1)} &= \mathrm{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)};\ \mathrm{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_v^{(k)}, \forall\, v \in \mathcal{N}(u)\right\}\right)\right) \\ &= \mathrm{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right)\end{aligned}$$

- where UPDATE and $\mathrm{AGGREGATE}$ are arbitrary differentiable functions and $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ is the “message" that is aggregated from u's graph neighborhood $\mathcal{N}(u)$.

S. Scardapane Alice's Adventures in a Differentiable Wonderland (2024)

Node features

- Unlike the shallow embedding methods, the GNN require node features $\mathbf{x}_u, \forall\, u \in \mathcal{V}$ as input to the model.
- In many applications, we will have rich node features, such as gene expression in biological networks or text features in social networks.
- When no node features are available, we can use node statistics, such as node degree or centrality, to define features.
- Another popular approach is to use identity features, where we associate each node with a one-hot indicator feature, which uniquely identifies that node.

Intuition

- In addition to structural information, the other key information captured by GNN node embedding is *feature-based*.
- After k iterations, the embedding can encode information about the degrees of all the nodes in u's k-hop neighborhood.
- This local behavior of GNN is similar to that of the convolutional kernels in convolutional neural networks (CNN).
  - While CNNs aggregate features from spatially-defined patches in an image, GNNs aggregate information based on local graph neighborhoods.

Graph Attention Network (GAT)

- A popular strategy for improving the aggregation layer in GNN is to apply attention [Bahdanau et al., 2015].
- The basic idea is to assign an attention weight to each neighbor, to weight this neighbor's influence during the aggregation step.
- A GNN model applying attention is Graph Attention Network (GAT) [Veličković et al., 2018], which uses attention weights to define a weighted sum of the neighbors:

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v$$

Neighborhood attention

- Any standard attention model from the deep learning literature at large can be used.
- Popular variants of attention include the bilinear attention model

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{h}_u^\top \mathbf{W} \mathbf{h}_v\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{h}_u^\top \mathbf{W} \mathbf{h}_{v'}\right)},$$

- as well as variations of attention layers using MLPs

$$\alpha_{u,v} = \frac{\exp\left(\mathrm{MLP}(\mathbf{h}_u, \mathbf{h}_v)\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathrm{MLP}(\mathbf{h}_u, \mathbf{h}_{v'})\right)},$$

GAT with multiple heads

- In addition, while it is less common in the GNN literature, it is also possible to add multiple attention "heads", in the style of the popular transformer architecture [Vaswani et al., 2017].
- In this approach, one computes K distinct attention weights $\alpha_{u,v,k}$, using independently parameterized attention layers.
- The messages aggregated using the different attention weights are then transformed and combined in the aggregation step, usually with a linear projection followed by a concatenation operation, e.g.

$$\mathbf{m}_{\mathcal{N}(u)} = [\mathbf{a}_1 \oplus \mathbf{a}_2 \oplus \ldots \oplus \mathbf{a}_K]$$

$$\mathbf{a}_k = \mathbf{W}_k \sum_{v \in \mathcal{N}(u)} \alpha_{u,v,k} \mathbf{h}_v$$

Concatenation and Skip-Connections

- One issue with GNN is over-smoothing: after several iterations of GNN message passing, the representations for all the nodes can become similar to each another.
- Over-smoothing is expected when the information aggregated from the node neighbors during message passing begins to dominate the updated node representations.
  - $\mathbf{h}_u^{(k+1)}$ will strongly depend on the incoming $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ message aggregated from the neighbors at the expense of the node representations $\mathbf{h}_u^{(k)}$ from the previous layers.

GraphSAGE

- A natural way to alleviate this issue is to directly preserve information from previous rounds of message passing during the update step using *skip connections*.
- The connection was proposed in the GraphSAGE and employs a concatenation to preserve node-level information during message passing:

$$\text{UPDATE}_{\text{concat}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = [\text{UPDATE}_{\text{base}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) \oplus \mathbf{h}_u],$$

- where we simply concatenate the output of the base update function with the node's previous-layer representation.

Heterophily

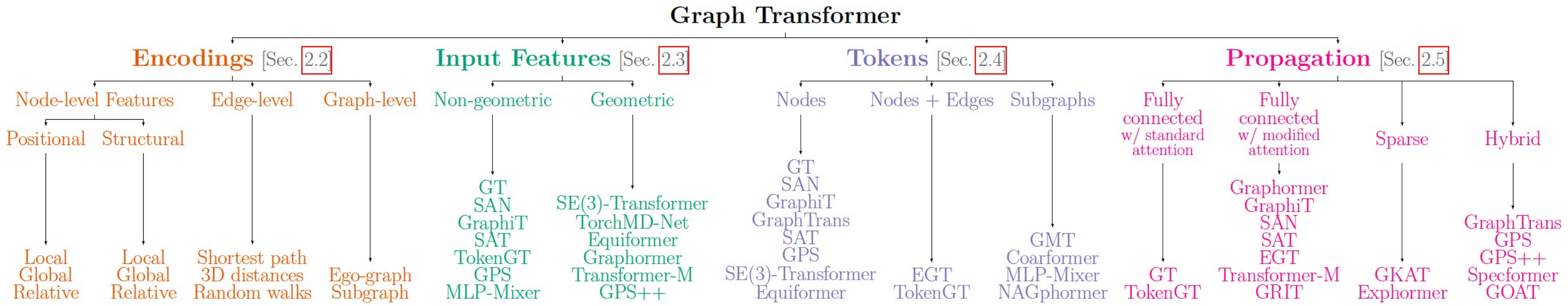
- Many popular GNNs fail to generalize to networks where connected nodes may have different class labels and dissimilar features.
- It can happen that a GNN is outperformed by models that ignore the graph structure!
- Possible solutions [Zhu 2020]:
  - Ego and neighbor-embedding separation,
  - Higher-order neighborhoods,
  - Combination of intermediate representations.
  - Attention-line mechanisms
    - More on this tomorrow – session 9, at 15,15.

Explainability vs interpretability

- Most *explainability* approaches for GNNs are local and post-hoc
  - GNNExplainer, PGExplainer, SubgraphX, offering limited global insights.
- Interpretability focuses on models that can provide information on which nodes are influencing the class predictions [Bechler-Speicher, 2024].

Graph Transformers

- Can we do better?
  – Over-smoothing and over-squashing, structural and positional encodings, explainability and interpretability.

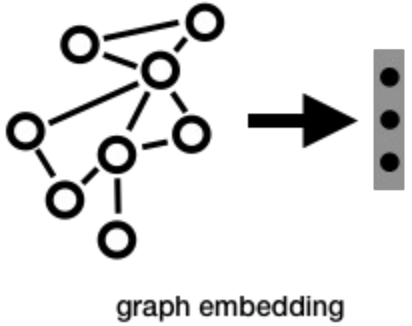

Graph Transformer
Encodings [Sec. 2.2]
Input Features [Sec. 2.3]
Tokens [Sec. 2.4]
Propagation [Sec. 2.5]
Node-level Features
Positional
Structural
Edge-level
Graph-level
Non-geometric
Geometric
Nodes
Nodes + Edges
Subgraphs
Fully connected w/ standard attention
Fully connected w/ modified attention
Sparse
Hybrid
Local Global Relative
Local Global Relative
Shortest path 3D distances Random walks
Ego-graph Subgraph
GT SAN GraphiT SAT TokenGT GPS MLP-Mixer
SE(3)-Transformer TorchMD-Net Equiformer Graphormer Transformer-M GPS++
GT SAN GraphiT GraphTrans SAT GPS SE(3)-Transformer Equiformer
EGT TokenGT
GMT Coarformer MLP-Mixer NAGphormer
GT TokenGT
Graphormer GraphiT SAN SAT EGT Transformer-M GRIT
GKAT Exphormer
GraphTrans GPS GPS++ Specformer GOAT

L. Müller et al. Attending to Graph Transformers, Trans. on Machine Learning Research (2024)

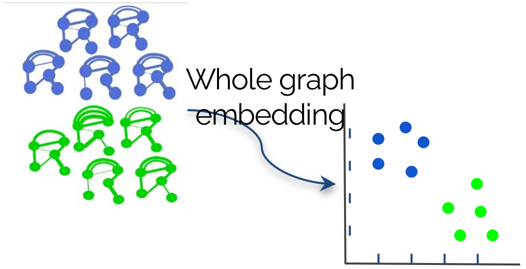
Whole-graph embedding

Given a set of graphs $\mathcal{G} = \{G_1, \dots, G_m\}$ with the same set of vertices V, a whole-graph embedding of dimension d is an encoding ENC

$$\mathrm{ENC}: G_i \in \mathcal{G} \longrightarrow y_i \in \mathbb{R}^d, i \in 1, \dots, |\mathcal{G}|$$



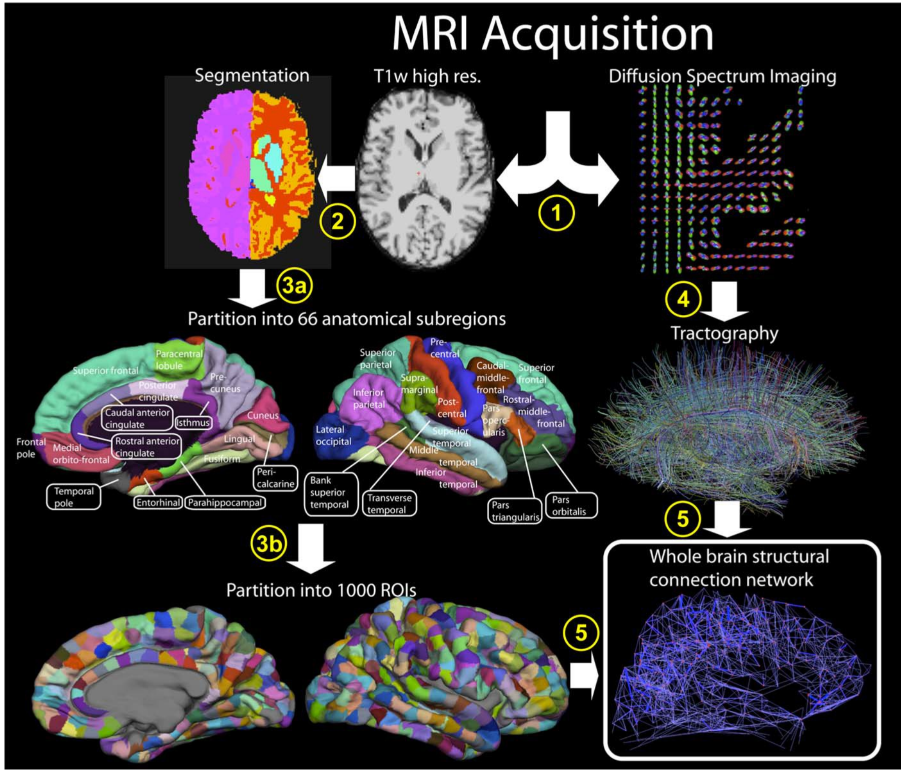
such that ENC preserves some proximity measure defined on $\mathcal{G}$



→ *Applications*
- Graphs classification
- Graphs clustering
- Visualization

Functional brain networks

- One network for each patient
- All networks the same nodes
- Nodes represent neural regions of interest



Hagmann et al., *Mapping the Structural Core of Human Cerebral Cortex*, PLoS Biology, 2008

Approaches to WGE

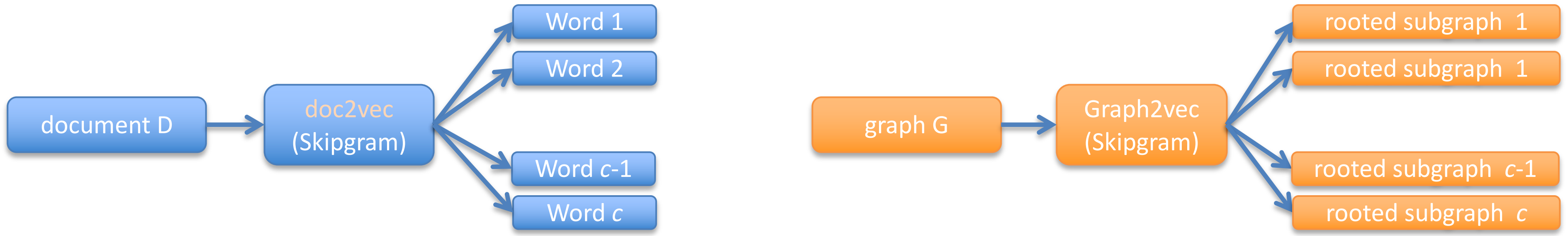
Matrix Factorization
- ❑ Laplacian Eigenmaps
- ❑ Adjacency Spectral Embedding
- ❑ Joint Embedding
- ❑ ...

Graph Kernels
- ❑ Weisfeiler-Lehman (WL)
- ❑ Shortest Path (SP)
- ❑ Random Walk (RW)
- ❑ DeepWL
- ❑ ...

Neural Network-based
- ❑ Patchy-San
- ❑ Graph2vec
- ❑ sub2vec
- ❑ Anonymous walk embeddings
- ❑ Autoencoders
- ❑ GraphSAGE
- ❑ DGCNN
- ❑ U2GNN
- ❑ ...

Graph2vec

Inspired by the neural document embedding model doc2vec.



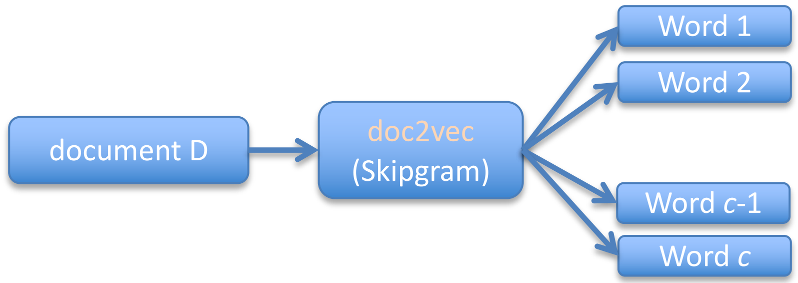
- Given a labeled graph *G*, sample *c* *rooted subgraphs* of degree d* from *G* to learn a representation of *G*
- Graph2vec = feed-forward NN to learn distributed representations of graphs

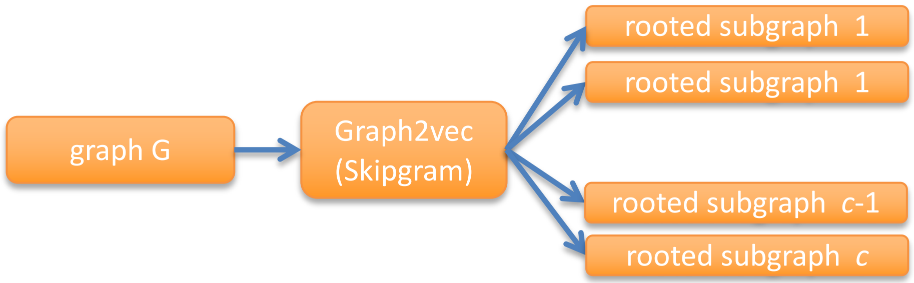
* subgraph including all nodes reachable in *d* hops from the node

Narayanan et al., graph2vec: Learning Distributed Representations of Graphs, ArXiv, 2017
Le and Mikolov, *Distributed Representations of Sentences and Documents*, Proc. ICML, 2014

Graph2vec

Inspired by the neural document embedding model doc2vec.





- A graph is viewed as a document
- The *subgraphs of degree d rooted in* each node are the words composing the document
  - Subgraph extraction and relabeling follow the WL refinement

Narayanan et al., graph2vec: Learning Distributed Representations of Graphs, ArXiv, 2017
Le and Mikolov, *Distributed Representations of Sentences and Documents*, Proc. ICML, 2014

Graph2vec

Algorithm 1: GRAPH2VEC $(\mathbb{G}, D, \delta, \mathfrak{e}, \alpha)$

```
input  : $\mathbb{G} = \{G_1, G_2, ..., G_n\}$: Set of graphs such that each
         graph $G_i = (N_i, E_i, \lambda_i)$ for which embeddings have to
         be learnt
         $D$: Maximum degree of rooted subgraphs to be
         considered for learning embeddings. This will produce
         a vocabulary of subgraphs, $SG_{vocab} = \{sg_1, sg_2, ...\}$
         from all the graphs in $\mathbb{G}$
         $\delta$: number of dimensions (embedding size)
         $\mathfrak{e}$: number of epochs
         $\alpha$: Learning rate
output: Matrix of vector representations of graphs $\Phi \in \mathbb{R}^{|\mathbb{G}| \times \delta}$
begin
    Initialization: Sample $\Phi$ from $R^{|\mathbb{G}| \times \delta}$
    for $e = 1$ to $\mathfrak{e}$ do
        $\mathfrak{G}$ = SHUFFLE ($\mathbb{G}$)
        for each $G_i \in \mathfrak{G}$ do
            for each $n \in N_i$ do
                for $d = 0$ to $D$ do
                    $sg_n^{(d)} :=$ GETWLSUBGRAPH$(n, G_i, d)$
                    $J(\Phi) = -\log \Pr(sg_n^{(d)} | \Phi(G))$
                    $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$
    return $\Phi$
```

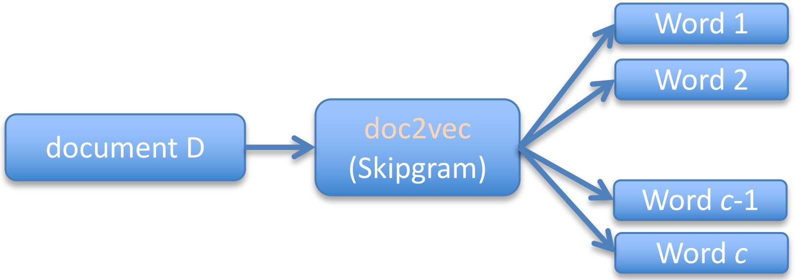
Algorithm 2: GETWLSUBGRAPH (n, G, d)

```
input  : $n$: Node which acts as the root of the subgraph
         $G = (N, E, \lambda)$: Graph from which subgraph has to be
         extracted
         $d$: Degree of neighbours to be considered for extracting
         subgraph
output: $sg_n^{(d)}$: Rooted subgraph of degree $d$ around node $n$
begin
    $sg_n^{(d)} = \{\}$
    if $d = 0$ then
        $sg_n^{(d)} := \lambda(n)$
    else
        $\mathcal{N}_n := \{n' \mid (n, n') \in E\}$
        $M_n^{(d)} := \{$GETWLSUBGRAPH$(n', G, d-1) \mid n' \in \mathcal{N}_n\}$
        $sg_n^{(d)} := sg_n^{(d)} \cup$ GETWLSUBGRAPH
          $(n, G, d-1) \oplus sort(M_n^{(d)})$
    return $sg_n^{(d)}$
```

WL = Weisefeiler-Lehman

Netpro2vec

Inspired by a neural document embedding model doc2vec.

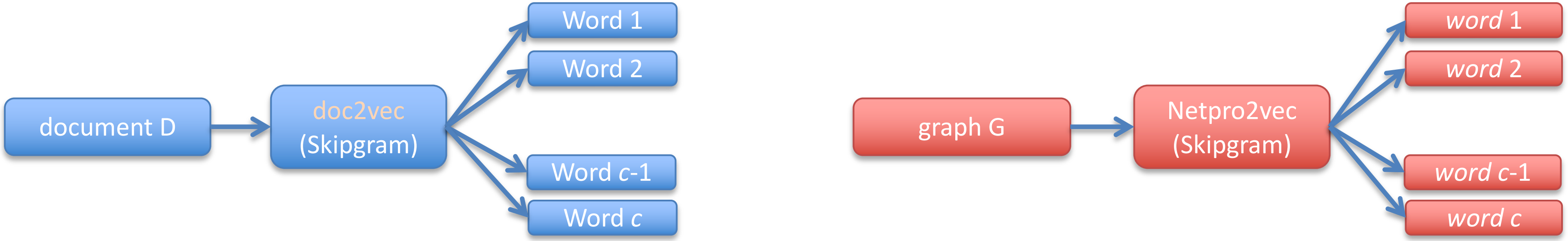


- Given a document *D*, sample *c* words from *D* and use them to learn a representation of *D*
- Doc2vec = feed-forward NN to learn distributed representations of document sequences

I. Manipur et al., *Netpro2vec: a Graph Embedding Framework for Biomedical Applications*, IEEE/TCBB 2022

Netpro2vec

Inspired by a neural document embedding model doc2vec.

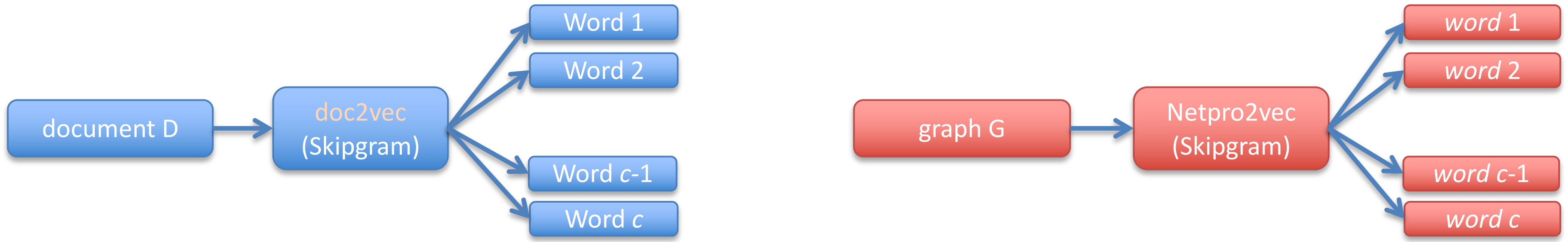


- Given a labeled graph *G*, *extract c words* from *G* and use them to learn a representation of *G*
- Doc2vec = feed-forward NN to learn distributed representations of graphs

I. Manipur et al., *Netpro2vec: a Graph Embedding Framework for Biomedical Applications*, IEEE/TCBB 2022

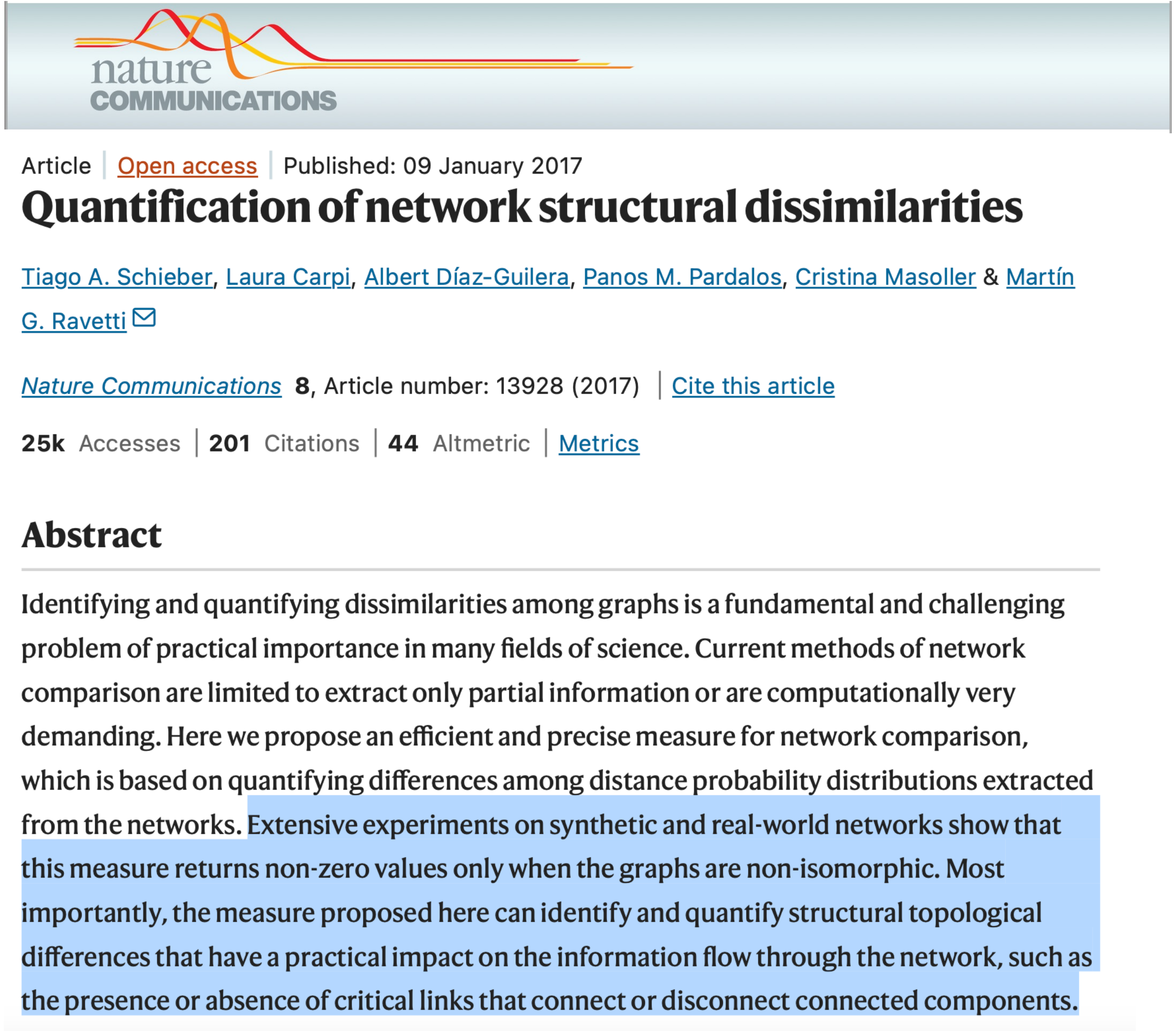
Netpro2vec

Inspired by a neural document embedding model doc2vec.



- A graph is viewed as a document
- But how can *words* be extracted by graphs?

I. Manipur et al., *Netpro2vec: a Graph Embedding Framework for Biomedical Applications*, IEEE/TCBB 2022



Article | Open access | Published: 09 January 2017

Quantification of network structural dissimilarities

Tiago A. Schieber, Laura Carpi, Albert Díaz-Guilera, Panos M. Pardalos, Cristina Masoller & Martín G. Ravetti

Nature Communications **8**, Article number: 13928 (2017) | Cite this article

25k Accesses | **201** Citations | **44** Altmetric | Metrics

Abstract

Identifying and quantifying dissimilarities among graphs is a fundamental and challenging problem of practical importance in many fields of science. Current methods of network comparison are limited to extract only partial information or are computationally very demanding. Here we propose an efficient and precise measure for network comparison, which is based on quantifying differences among distance probability distributions extracted from the networks. Extensive experiments on synthetic and real-world networks show that this measure returns non-zero values only when the graphs are non-isomorphic. Most importantly, the measure proposed here can identify and quantify structural topological differences that have a practical impact on the information flow through the network, such as the presence or absence of critical links that connect or disconnect connected components.

Empirical distributions

1. Node Distance Distribution (NDD) of node u_i in graph $\mathbf{G}$ is the fraction of nodes reachable with a shortest path of length d.
   - Nodes in disconnected components will be considered at ∞ distance.
2. Transition Matrix (TM_s) of order s: probability of a node u_i to be reached in s steps by a random walker located in node u_j.
   - TM_1 is the adjacency matrix of the graph rescaled by the degree of each node.

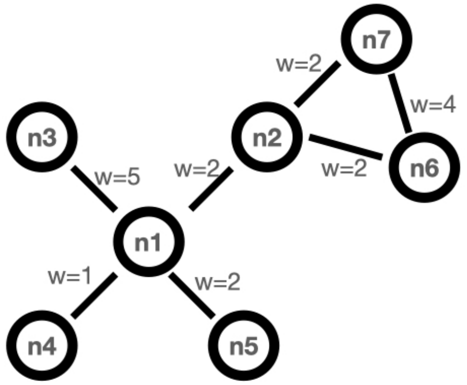
While NDD provides information about the global topology of the graph, the TM_s contains local information about its connectivity.

Schieber, T. *et al. Quantification of network structural dissimilarities*. *Nat Commun* **8**, 13928 (2017).

Netpro2vec: *words* extraction

Words are concatenation of node labels with PDs exceed a threshold *p*

Input graph —> **Node distributions** —> **Words extraction** —> **Graph embedding (doc2vec)**



TM1

	n1	n2	n3	n4	n5	n6	n7
n1	0	0.2	0.5	0.1	0.2	0	0
n2	0.33	0	0	0	0	0.33	0.33
...	...	...	...	...	...	...	...

vi=n1, *p*=0.1
seqp(vi)=tm1_n1_n3_n2_n5;
vi=n2 *p*=0.1
seqp(vi)=tm1_n2_n1_n6_n7

TM2

	n1	n2	n3	n4	n5	n6	n7
n1	0	0	0	0	0	0.5	0.5
n2	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...

vi=n1, *p*=0.1
seqp(vi)=tm2_n1_n6_n7;
vi=n2 *p*=0.1
seqp(vi)=tm2_n2_n4

h
—>

NDD

	1	2	3	4	5	6	7	8	9
n1	0.143	0.286	0	0.286	0.143	0	0	0	0
...	...	...	...	...	...	...	...	...	...

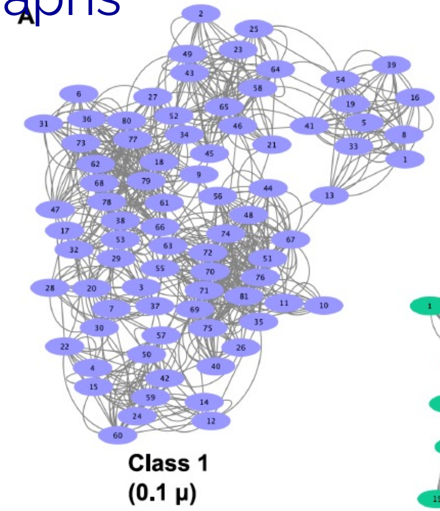
vi=n1, *p*=0
seqp(vi)=ndd_n1_2_4_1_5;
vi=n1 *p*=0.15
seqp(vi)=ndd_n1_2_4

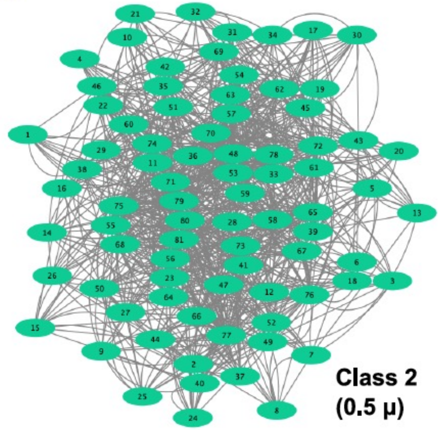
LFR Dataset

Generated using

Lancichinetti–Fortunato–Radicchi (LFR) software

- 1600 undirected and unweighted graphs
  A. 600 with μ=0.1
  B. 1000 with μ=0.5
- 81 nodes for each graph





Lancichinetti et al, *Physical review E*, 2008

Dataset available at https://github.com/leoguti85/GraphEmbs

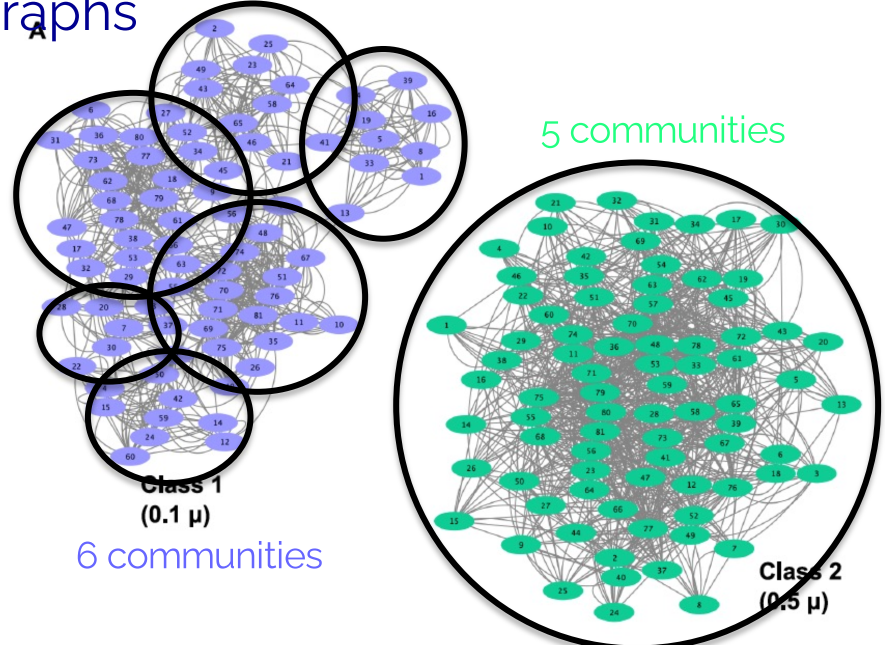
LFR Dataset

Generated using

Lancichinetti–Fortunato–Radicchi (LFR) software

- 1600 undirected and unweighted graphs
  A. 600 with μ=0.1
  B. 1000 with μ=0.5
- 81 nodes for each graph

μ controls the strength of the community arrangements



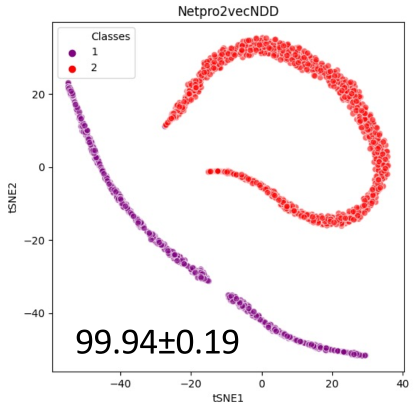
5 communities

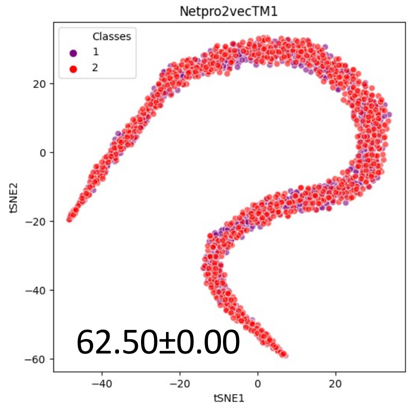
6 communities

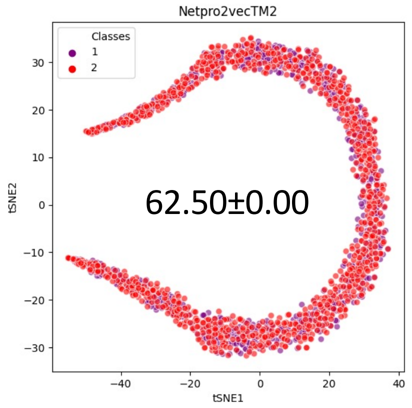
Lancichinetti et al, *Physical review E*, 2008

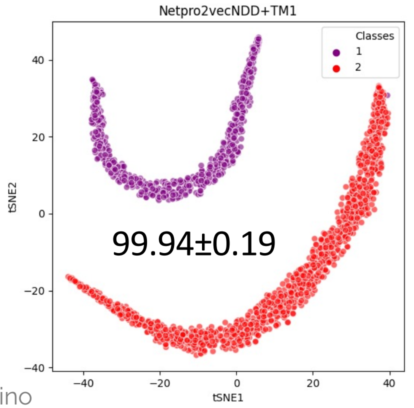
Dataset available at https://github.com/leoguti85/GraphEmbs

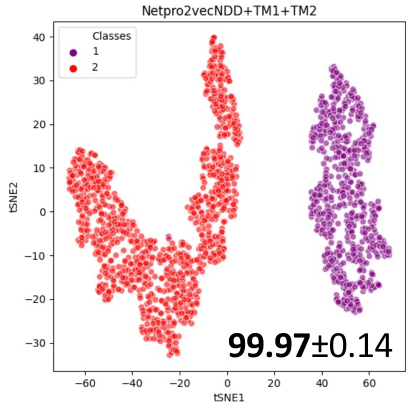
t-SNE visualization of LFR

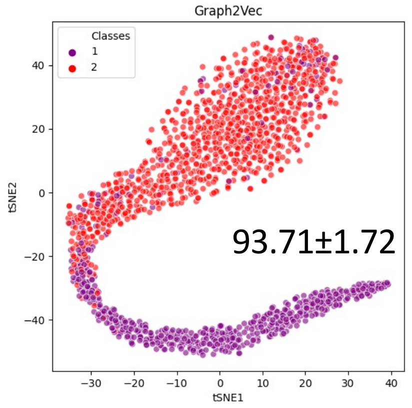












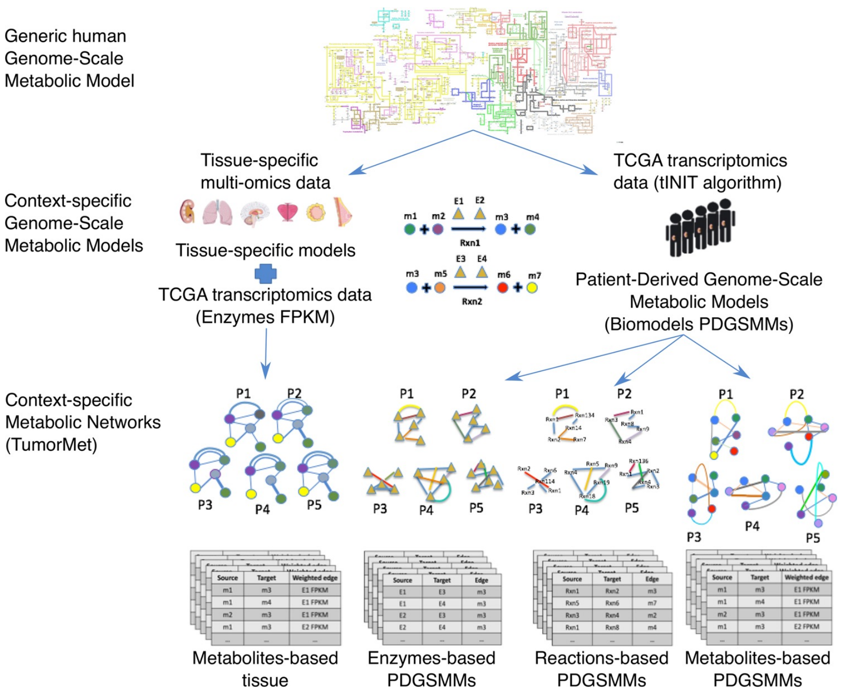
scientific **data**

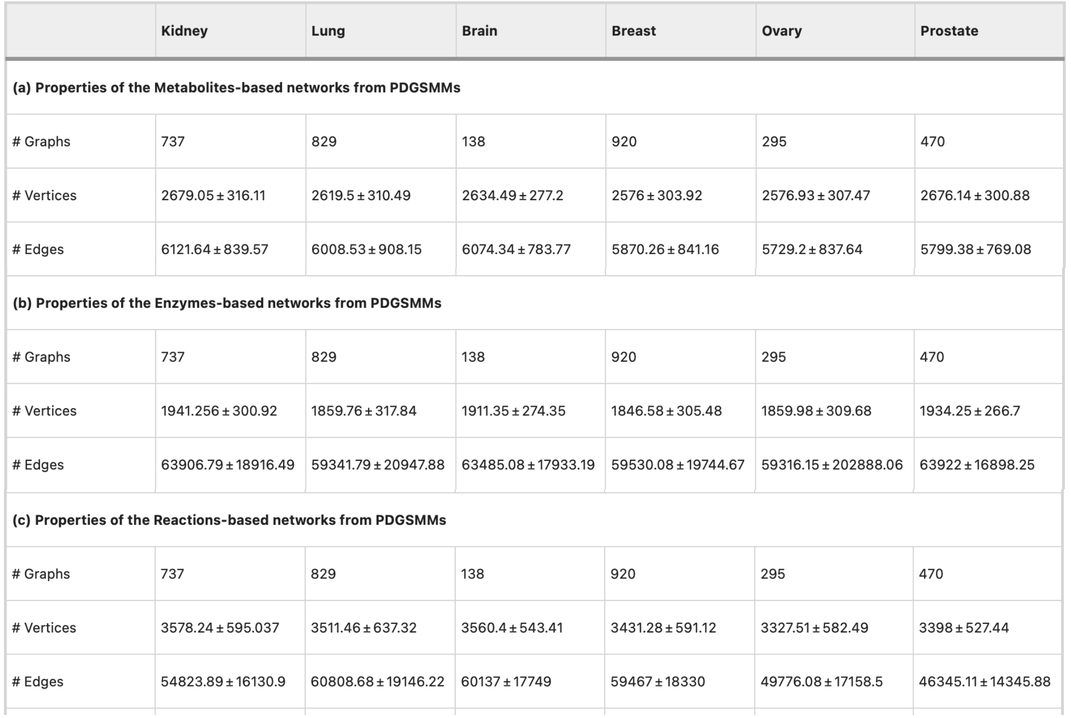
OPEN

DATA DESCRIPTOR

TumorMet: A repository of tumor metabolic networks derived from context-specific Genome-Scale Metabolic Models

Ilaria Granata[1]✉, Ichcha Manipur[1], Maurizio Giordano[1], Lucia Maddalena[1] & Mario Rosario Guarracino[2]


Generic human Genome-Scale Metabolic Model
Tissue-specific multi-omics data
TCGA transcriptomics data (tINIT algorithm)
Context-specific Genome-Scale Metabolic Models
Tissue-specific models
TCGA transcriptomics data (Enzymes FPKM)
m1 + m2 → m3 + m4
E1 E2
Rxn1
m3 + m5 → m6 + m7
E3 E4
Rxn2
Patient-Derived Genome-Scale Metabolic Models (Biomodels PDGSMMs)
Context-specific Metabolic Networks (TumorMet)
P1
P2
P3
P4
P5
Source
Target
Weighted edge
Edge
Metabolites-based tissue
Enzymes-based PDGSMMs
Reactions-based PDGSMMs
Metabolites-based PDGSMMs



	Kidney	Lung	Brain	Breast	Ovary	Prostate
(a) Properties of the Metabolites-based networks from PDGSMMs						
# Graphs	737	829	138	920	295	470
# Vertices	2679.05 ± 316.11	2619.5 ± 310.49	2634.49 ± 277.2	2576 ± 303.92	2576.93 ± 307.47	2676.14 ± 300.88
# Edges	6121.64 ± 839.57	6008.53 ± 908.15	6074.34 ± 783.77	5870.26 ± 841.16	5729.2 ± 837.64	5799.38 ± 769.08
(b) Properties of the Enzymes-based networks from PDGSMMs						
# Graphs	737	829	138	920	295	470
# Vertices	1941.256 ± 300.92	1859.76 ± 317.84	1911.35 ± 274.35	1846.58 ± 305.48	1859.98 ± 309.68	1934.25 ± 266.7
# Edges	63906.79 ± 18916.49	59341.79 ± 20947.88	63485.08 ± 17933.19	59530.08 ± 19744.67	59316.15 ± 202888.06	63922 ± 16898.25
(c) Properties of the Reactions-based networks from PDGSMMs						
# Graphs	737	829	138	920	295	470
# Vertices	3578.24 ± 595.037	3511.46 ± 637.32	3560.4 ± 543.41	3431.28 ± 591.12	3327.51 ± 582.49	3398 ± 527.44
# Edges	54823.89 ± 16130.9	60808.68 ± 19146.22	60137 ± 17749	59467 ± 18330	49776.08 ± 17158.5	46345.11 ± 14345.88

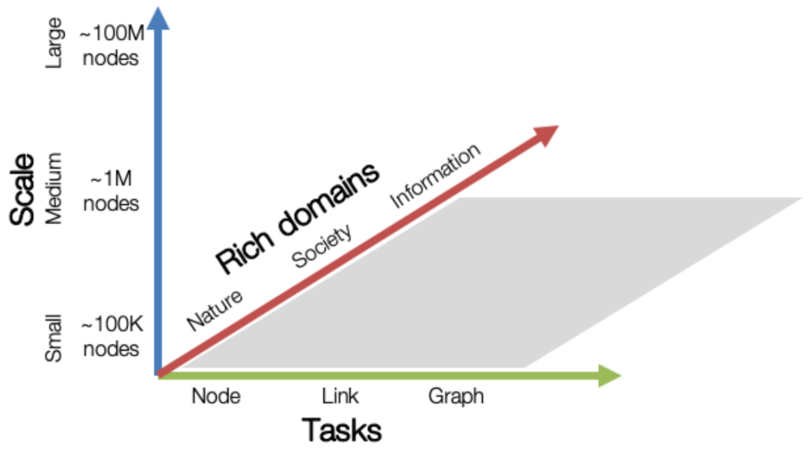
OGB Dataset Overview

The Open Graph Benchmark (OGB) aims to provide graph datasets that cover important graph machine learning tasks, diverse dataset scale, and rich domains.

Multiple task categories: We cover three fundamental graph machine learning task categories: predicting the properties of nodes, links, and graphs.

Diverse scale: Small-scale graph datasets can be processed within a single GPU, while medium- and large-scale graphs might require multiple GPUs and/or sophisticated mini-batching techniques.

Rich domains: Graph datasets come from diverse domains and include biological networks, molecular graphs, academic networks, and knowledge graphs.



Research directions in GML

- Expressivity
  - Are we really interested in whether two graph are the same?
    - Expressiveness of architectures, uniform results for graph classes, expressiveness vs computational complexity.
- Generalization
  - From transductive to inductive methods
    - Structure vs generalization, data augmentation, out of sample generalization.
- Optimization
  - When are significant solutions also relevant?
    - Rate of convergence, architectural choices vs convergence, defeating randomness, attention mechanisms.
- Applications
  - Are theoretical results aligned with practical applications?
    - Domain knowledge vs architectures, learning paradigms, LMM/GPT token style training.

C. Morris et al *Future Directions in the Theory of Graph Machine Learning* PMLR (2024)

Conclusions

- Network representations capture interactions and dependencies among variables or observations.
- Many open problems and possibilities to contribute to the field of network data analysis.

Learning from networked data unlocks hidden structure and relationships, enabling richer insights and better decisions than isolated-data analysis.