A Short Journey through Graph Embedding Techniques

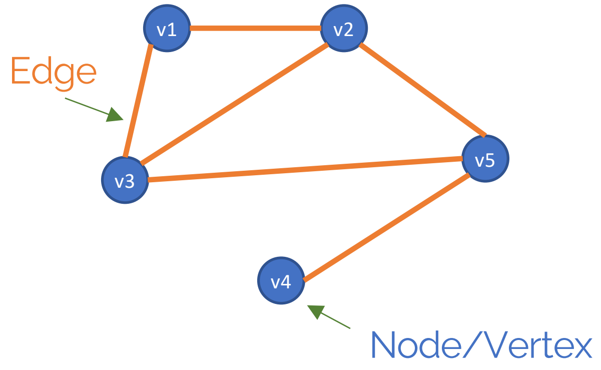
XVI Summer School on Operational Research, Data and Decision Making

Mario R. Guarracino

Version 2.1 2024 05 14 1900

What is a network?

- A collection of points joined together in pairs by lines
  - Points joined together depend on the context
    - *Points -> Vertices, nodes, actors …*
    - *Lines -> Edges*
- There are many real problems that can be modeled as networks
  - Individual parts linked in some way
    - *Internet*
      - A collection of computers linked together by data connections
    - *Human societies*
      - A collection of people linked by acquaintance or social interaction



Why Networks?

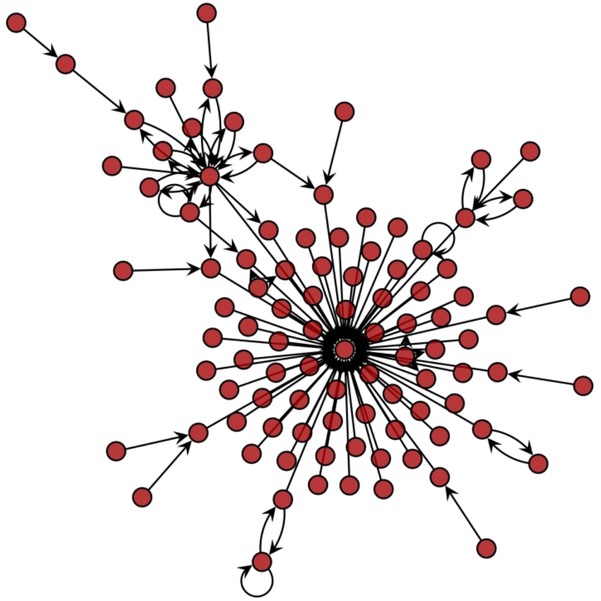
- Universal language for describing complex data
  - Networks from science, nature, and technology are more similar than one would expect
- Shared vocabulary between fields
  - Computer Science, Social science, Physics, Economics, Statistics, Biology,…
- Data availability (+computational challenges)
  - Web/mobile, bio, health, and medical
- Impact!
  - Social networking, Social media, Drug design

Why networks?

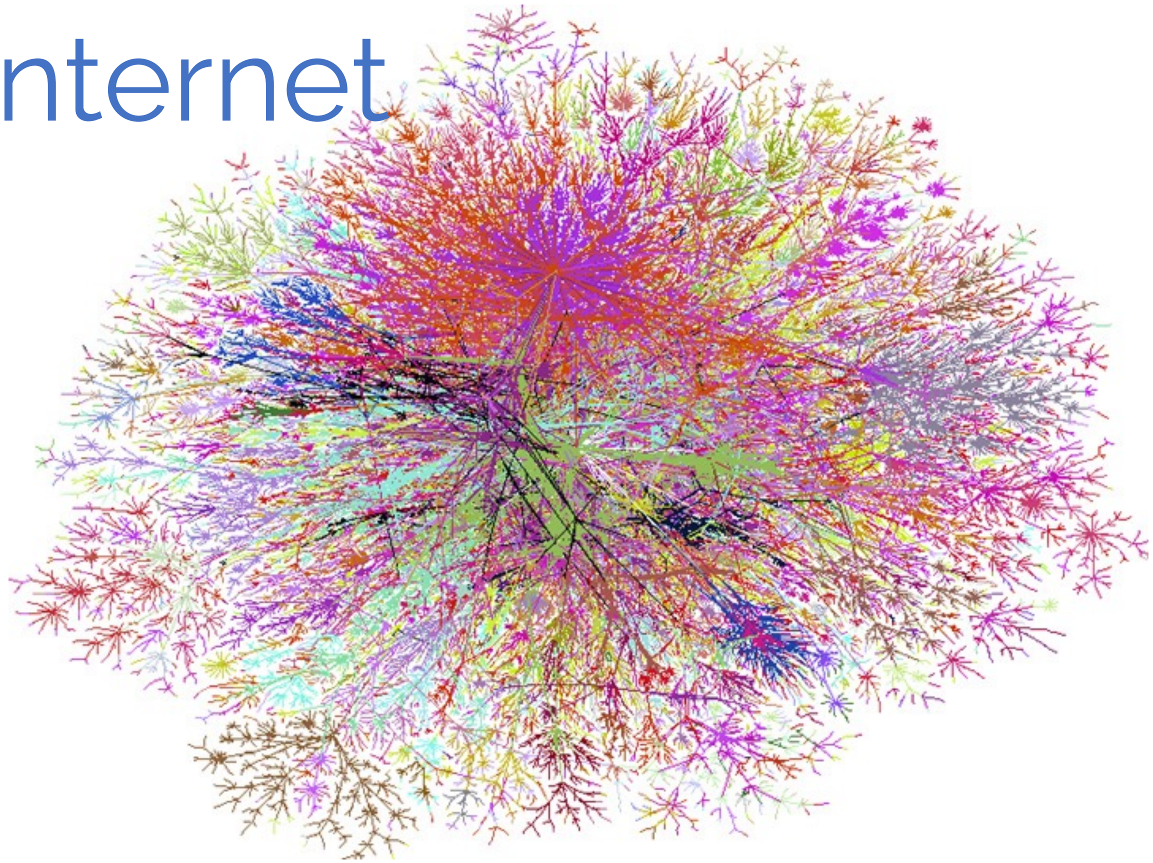
- Both the individual components of a system (e.g., computer machines, people etc.) and the nature of their interaction are important.
- Equally important is the pattern of connections between these components.
    - These patterns significantly affect the performance of the underlying system.
- Patterns in a social network affect the way people obtain information, form opinions etc.
- Patterns in a network of financially connected companies provides the evidence of casual behavior among financial assets.

Social networks

- Network of people
  - Edges can represent friendships, relative relations, co-locations, replies to a given tweet.
- Traditionally social network studies were based on small scale networks
- Online social media have provided network data on previously unreachable scale



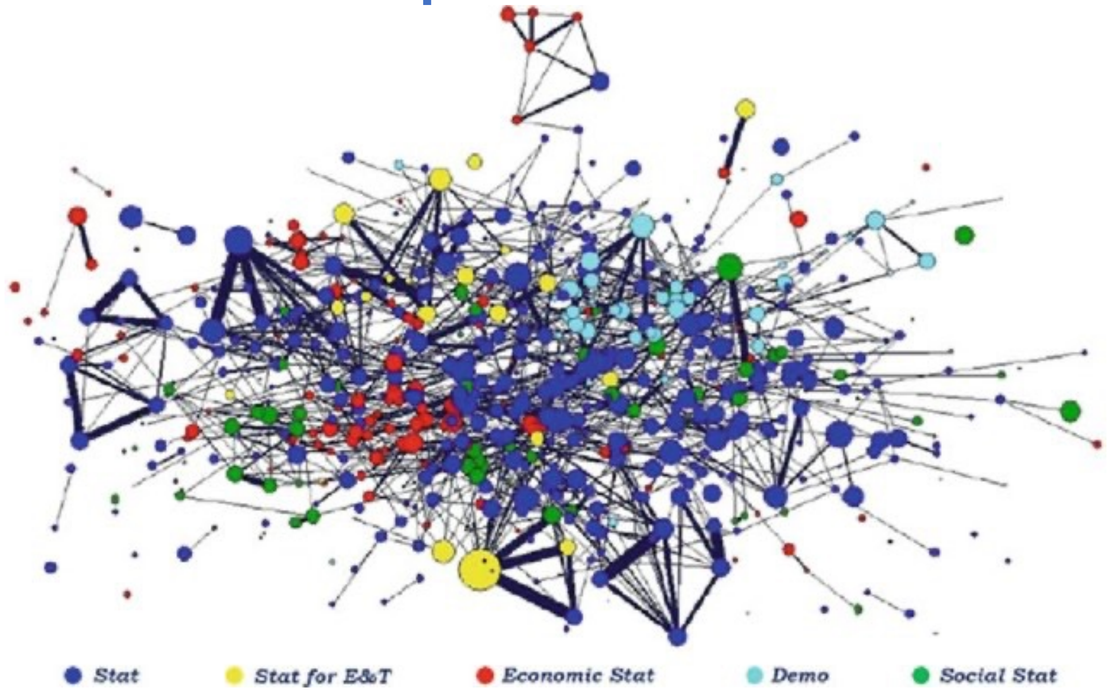
The Internet



Studying the Internet structure can help understand and improve the performance

http://www.jeffkennedyassociates.com:16080/connections/concept/image.html

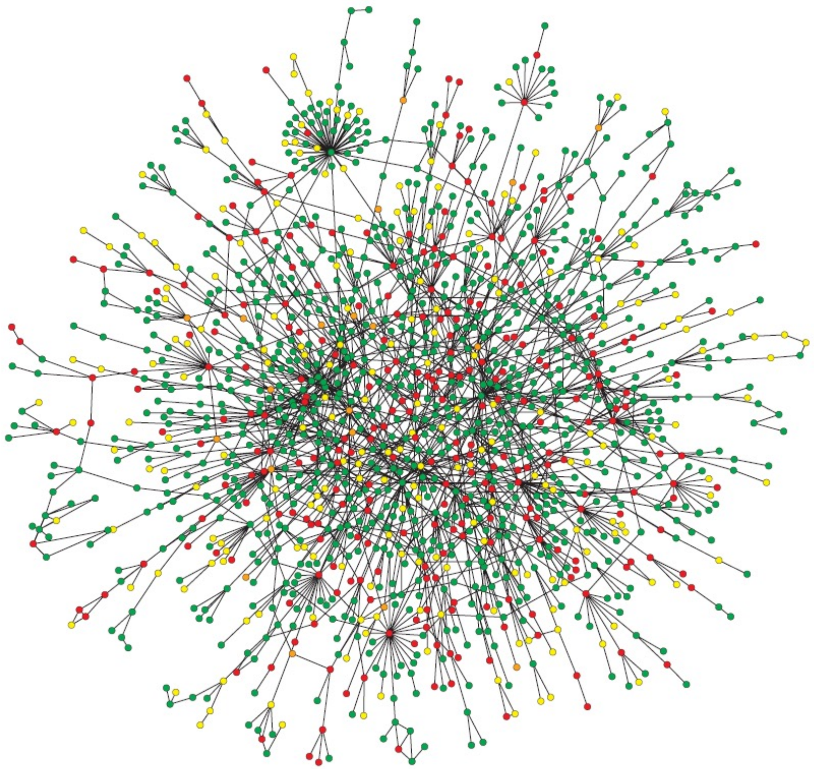
Co-authorship for statisticians



Node size: # publications per author. Edge size: # pubs shared by pairs of authors

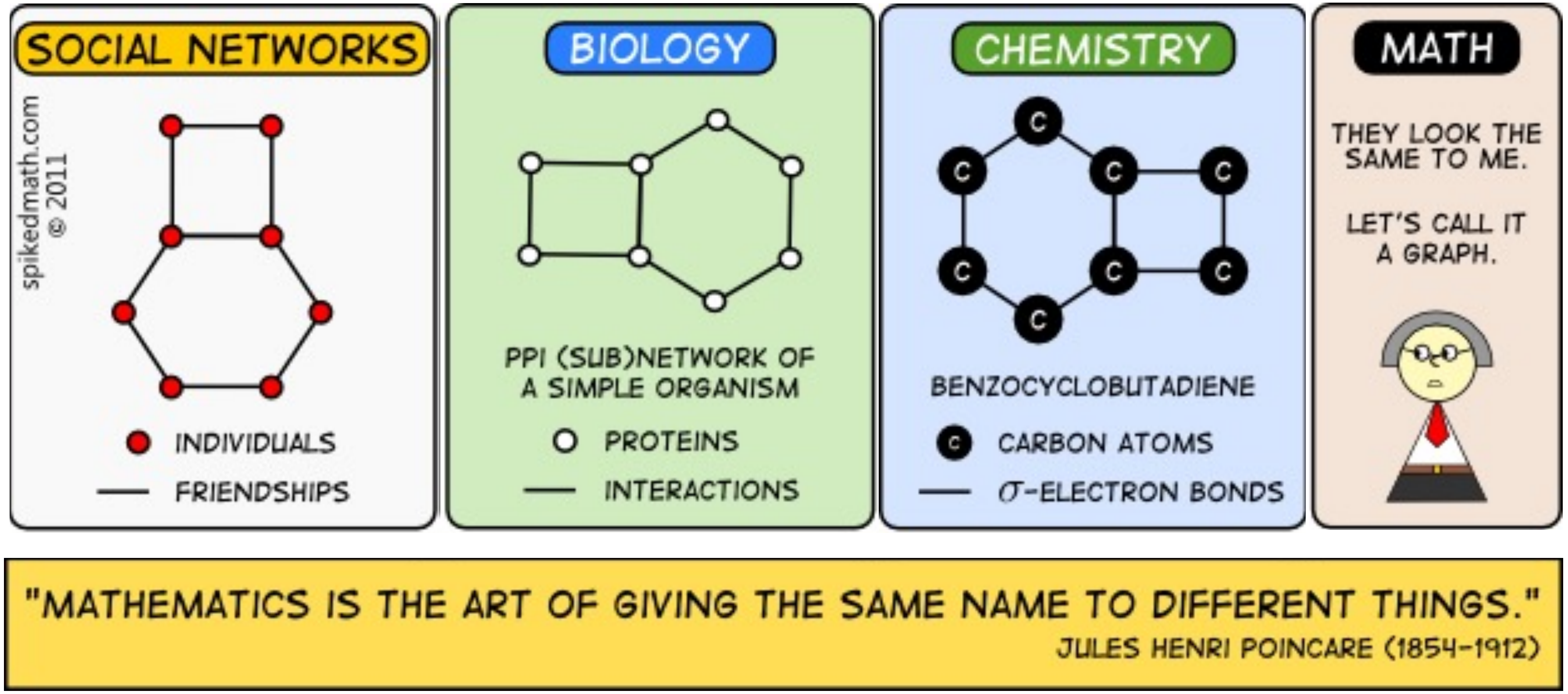
DOI: 10.1007/978-3-319-44093-4_11

Networks in Molecular Biology



Barabasi & Oltvai, Nature Reviews, 2004

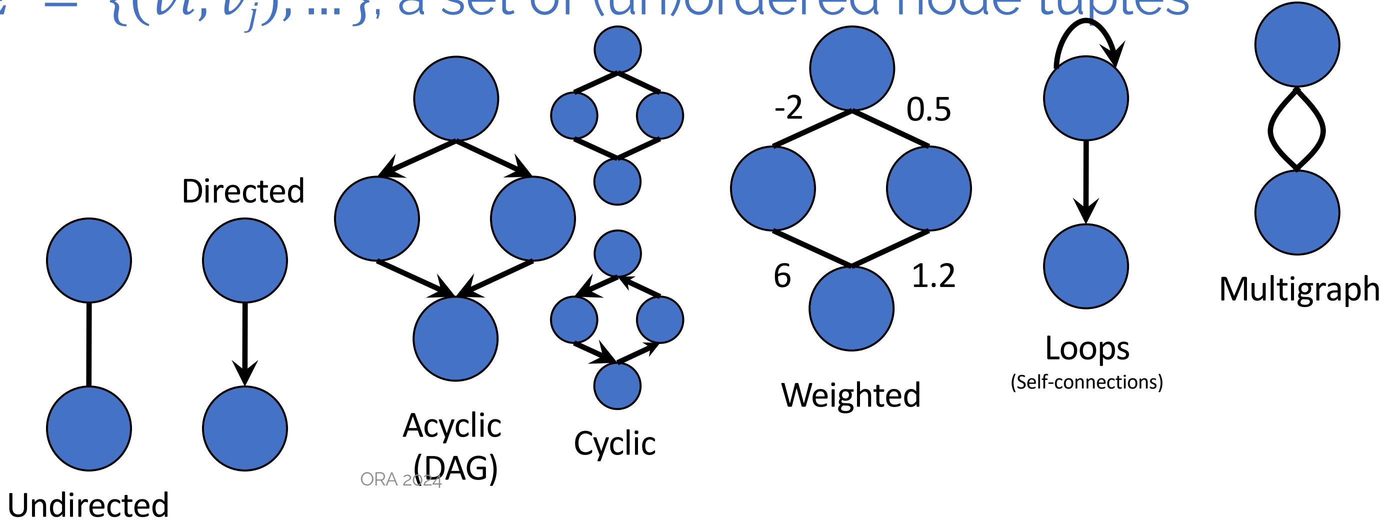
- Protein-Protein interactions
- Protein-DNA interactions
- Genetic interactions
- Metabolic reactions
- Co-expression interactions
- Text mining interactions
- Association Networks
- Etc.



http://spikedmath.com/382.html

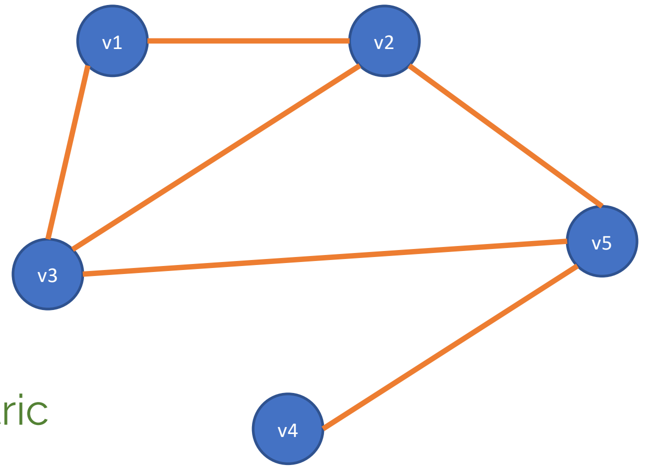
Networks: definitions

- Formally, a network is (a graph is)…
  - $G = (V, E)$, an ordered tuple of two sets
  - $V = \{v_1, \dots, v_n\}$, a set of unique nodes, and
  - $E = \{(vi, v_j), \dots\}$, a set of (un)ordered node tuples



Adjacency matrix

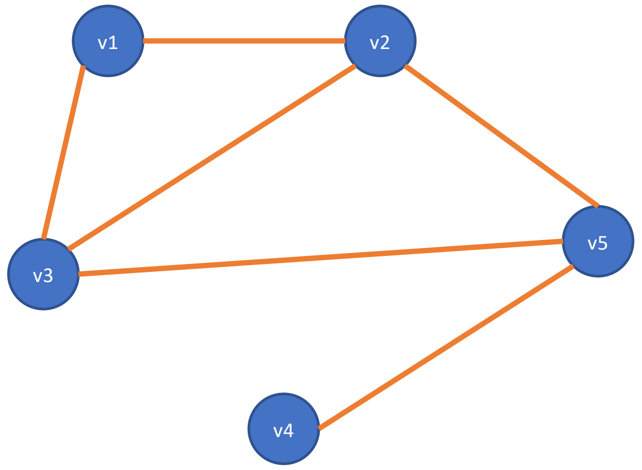
- A is $n \times n$ matrix (n = # of nodes)
  - Unweighted graph:
    $A_{ij} = 1$ if $(i, j) \in E$, and 0 otherwise
  - Weighted graph:
    $A_{ij} =$ weight of edge (i, j)
  - A is symmetric for undirected graphs, and asymmetric for directed



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

- A can be sparse for real networks (very few non-zero entries)
  - Facebook friendship network:
    - $|V| = n = 2.23e9$
    - $|E| = \#edges = 173e9$,
    - *fraction of non-zero entries* $\sim 7 \approx 10e-8$

Other network representations



Edge list	**Adjacency list**
$(1,2)$	$1:\ 2,\ 3$
$(1,3)$	$2:\ 1,\ 3,\ 5$
$(2,3)$	$3:\ 1,\ 2,\ 5$
$(2,5)$	$4:\ 5$
$(3,5)$	$5:\ 2,\ 3,\ 4$
$(4,5)$	

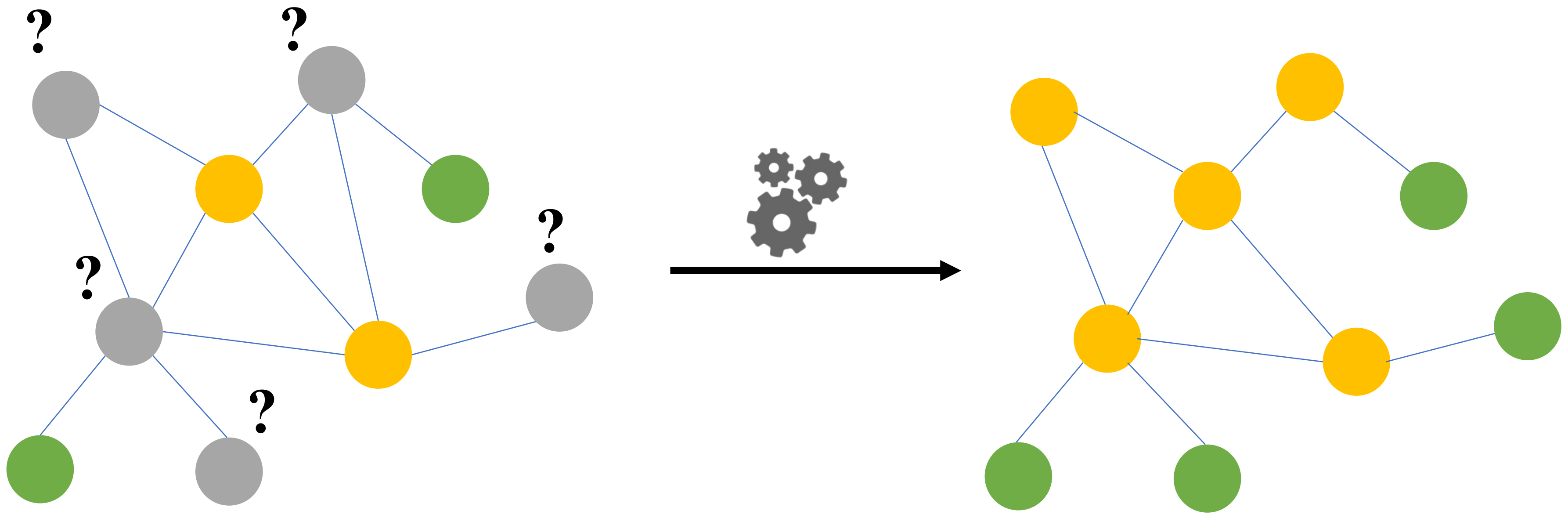
- Memory and computationally efficient for large, sparse graphs
- Edge list: Popular format for storing graphs
- Adjacency list: Fast retrieval of neighbours of node
- Adjacency matrix/list, edge list can be defined for directed graphs

ML tasks on networks

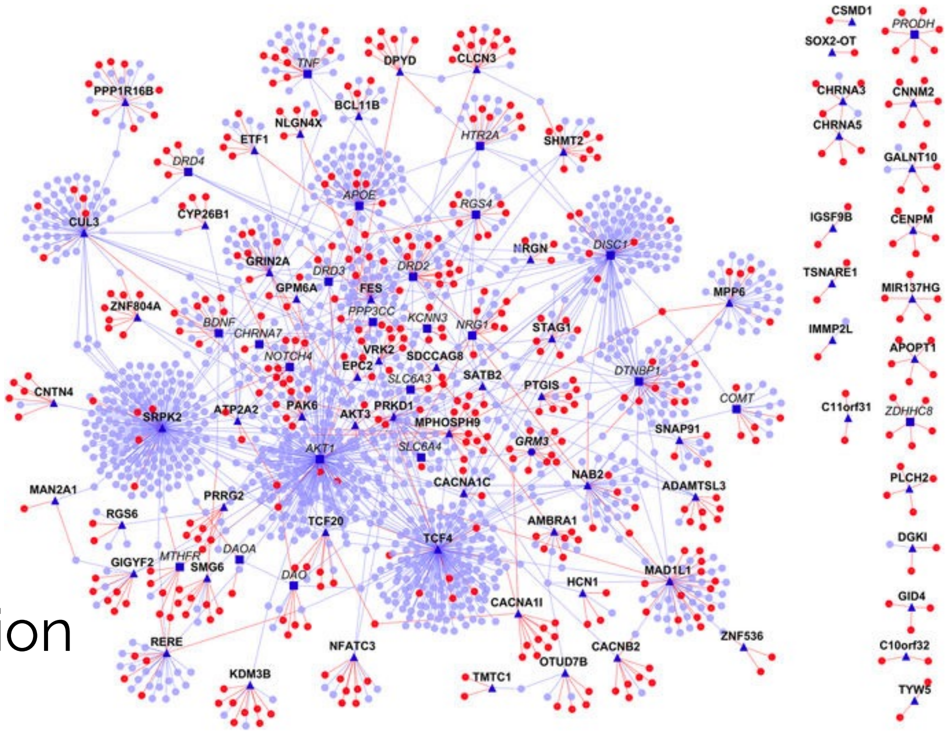
- Node classification/regression
  - Predict a type/value of a given node
- Link prediction
  - Predict whether two nodes are linked
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks

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

Example: Node Classification



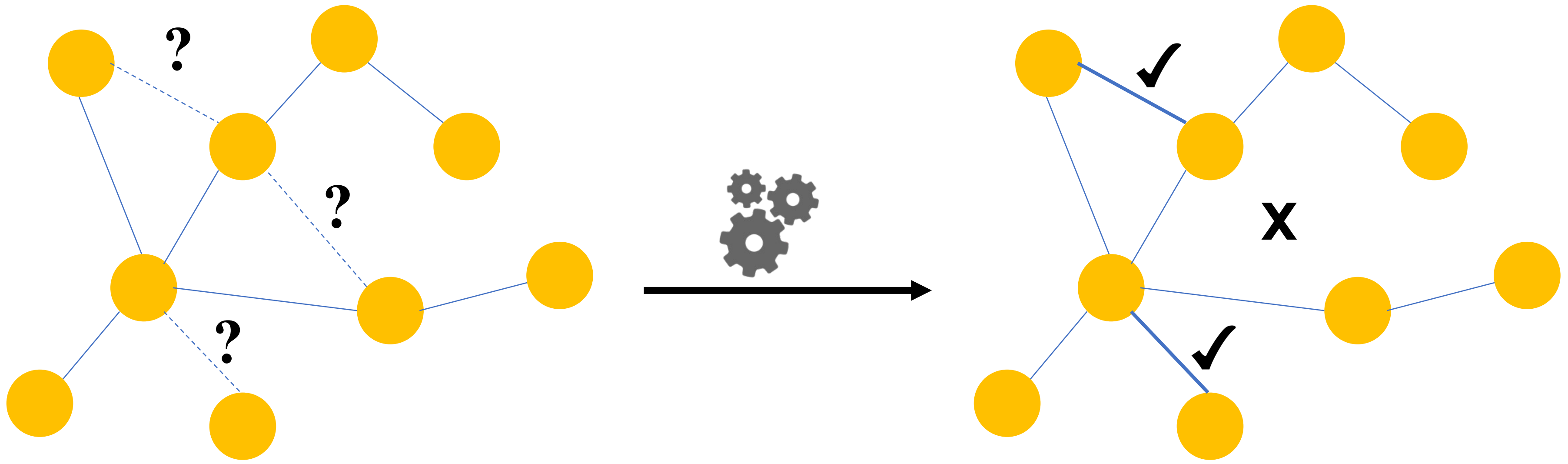
Example: Node Classification



Proteins function classification

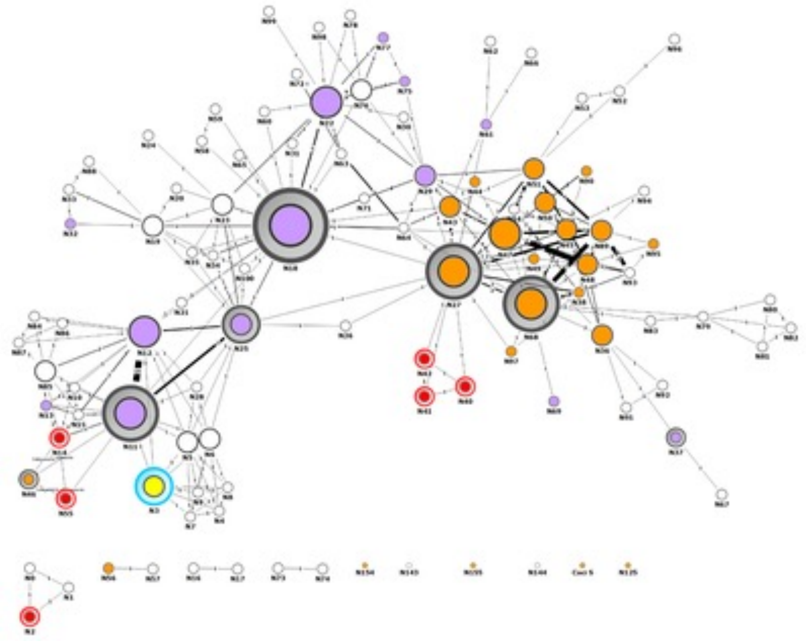
Ganapathiraju et al. *Nature* 2016.

Example: Link Prediction



Example: mafia meetings

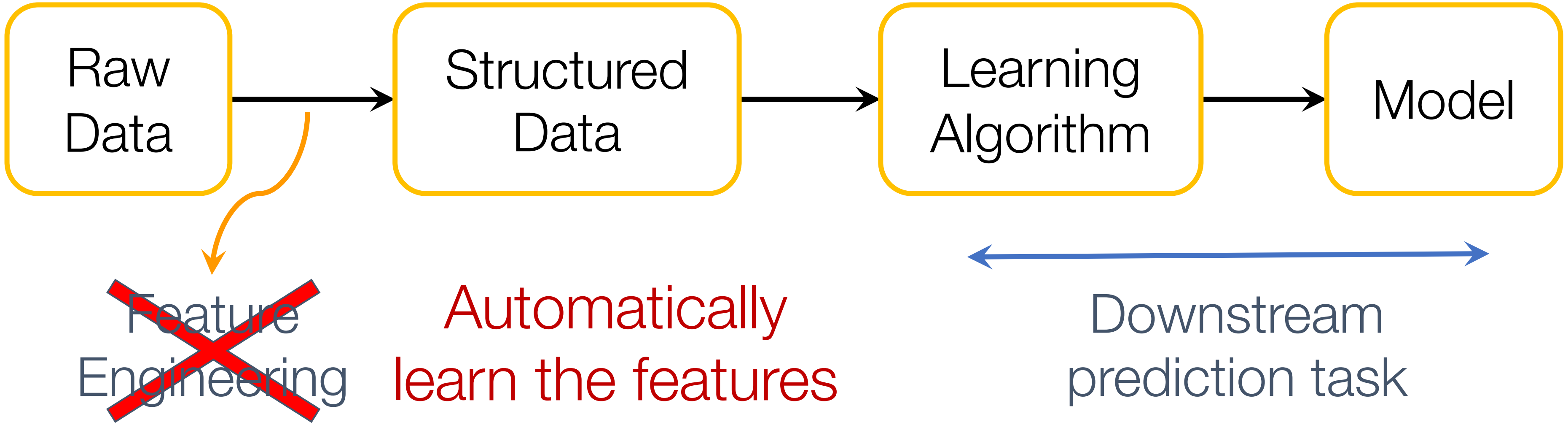
- Nodes represent the members of the "Mistretta" and "Batanesi" family.
- Circled nodes represent the subjects investigated for being association leaders.
- The red and yellow circled nodes refer to bosses of other districts.
- The white knots represent the other subjects close to the association or useful for the purposes of the association.
- The width of the edges is proportional to the number of meetings and the size of the nodes to their degree.



doi.org/10.1016/j.eswa.2020.113666

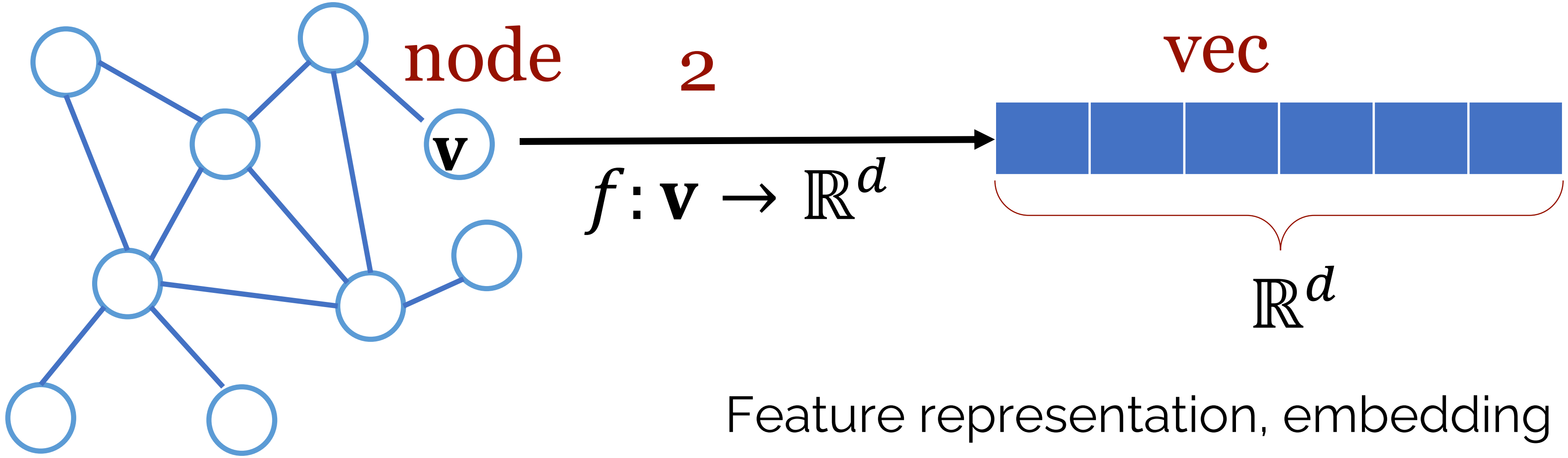
ML Lifecycle

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



Feature Learning in Graphs

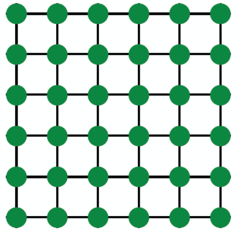
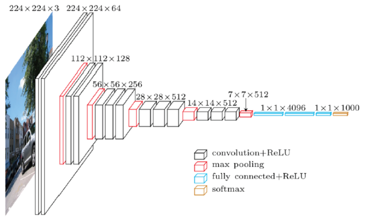
Goal: Efficient task-independent feature learning for machine learning in networks!



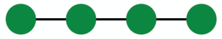
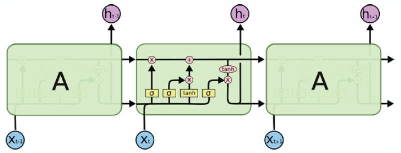
Feature representation, embedding

Why Is It Hard?

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

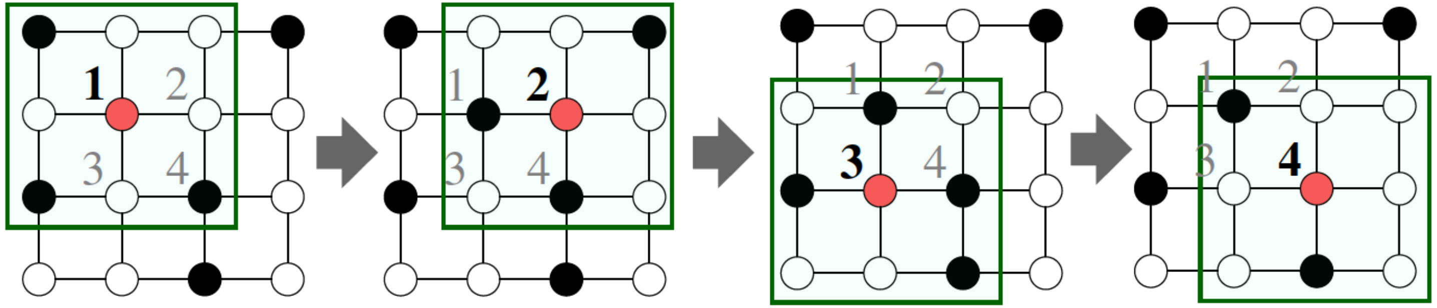
  - RNNs or word2vec for text/sequences…

Why Is It Hard?

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.

Background and traditional approaches

Graph Statistics and Kernel Methods

- Traditional approaches to ML using graph data follow the standard machine learning paradigm that was popular prior to the advent of deep learning.
- We begin by extracting some statistics or features—based on heuristic functions or domain knowledge—and then use these features as input to a standard machine learning algorithm (e.g., logistic regression).

Node degree

- The most straightforward node feature is *degree*, which is usually denoted d_u for a node $u \in V$ and simply counts the number of edges incident to a node:

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

- In cases of directed and weighted graphs, one can differentiate between different notions of degree.
  - corresponding to outgoing/incoming edges by summing over rows or columns
- In general, the node degree is an essential statistic, and it is often one of the most informative features in traditional ML models for node-level tasks.

Node centrality

- More powerful are the node centrality measures, which can form useful features in a wide variety of node classification tasks.
- One popular measure of centrality is the eigenvector centrality, which takes into account the importance of node's neighbors.
- In particular, we define a node's eigenvector centrality e_u via a recurrence relation in which the node's centrality is proportional to the average centrality of its neighbors:

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

- where λ is a constant.

Node centrality

- Rewriting this equation in vector notation with e as the vector of node centralities, it defines the standard eigenvector equation for the adjacency matrix:

$$\lambda \mathbf{e} = \mathbf{A}\mathbf{e}$$

- the centrality measure that satisfies the above equation corresponds to the eigenvector of the adjacency matrix corresponding to the largest eigenvalue.
- One view of eigenvector centrality is that it ranks the likelihood that a node is visited on a random walk of infinite length on the graph.

The clustering coefficient

- The popular local variant of the clustering coefficient is computed as follows:

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

  - The numerator counts the number of edges between neighbours of node u in $\mathcal{N}(u) = \{v \in \mathcal{V} : (u, v) \in \mathcal{E}\}$.
  - The denominator calculates how many pairs of nodes there are in u's neighborhood.

The clustering coefficient

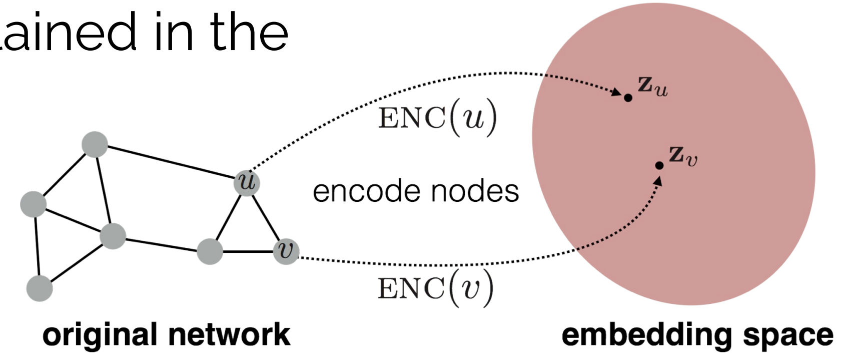
- The clustering coefficient measures how tightly clustered a node's neighborhood is.
- A clustering coefficient of 1 would imply that all of u's neighbors are also neighbors of each other.
- As with centrality, there are numerous variations of the clustering coefficient (e.g., to account for directed graphs).
- An important property of real-world networks is that they tend to have higher clustering coefficients than one would expect if edges were sampled randomly.

Node embeddings

William L. Hamilton, *Graph Representation Learning*, 2020

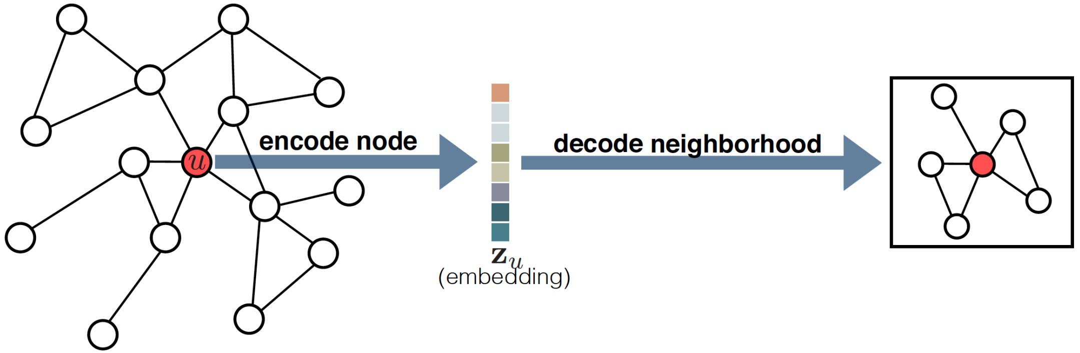
Node embeddings

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



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 embeddings $z_v \in \mathbb{R}^d$, where z_v corresponds to the embedding for node $v \in V$.
- In the simplest case, the encoder has the following signature:
$$\text{ENC} : V \rightarrow \mathbb{R}^d$$
- 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:
$$\text{ENC}(v) = \mathbf{Z}[v]$$
- 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.

Beyond shallow embedding

- The encoder can also be generalized beyond the shallow embedding approach.
- For instance, the encoder can use node features or the local graph structure around each node as input to generate an embedding.
- These generalized encoder architectures are often called graph neural networks (GNNs)

The decoder

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

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

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

The decoder

- A simple pairwise decoder could predict whether two nodes are neighbors in the graph.
- Applying the pairwise decoder to a pair of embeddings $(\mathbf{z}_u, \mathbf{z}_v)$ results in the reconstruction of the relationship between u and v.
- The goal is optimizing the encoder and decoder to minimize the reconstruction loss, so that:

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

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

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

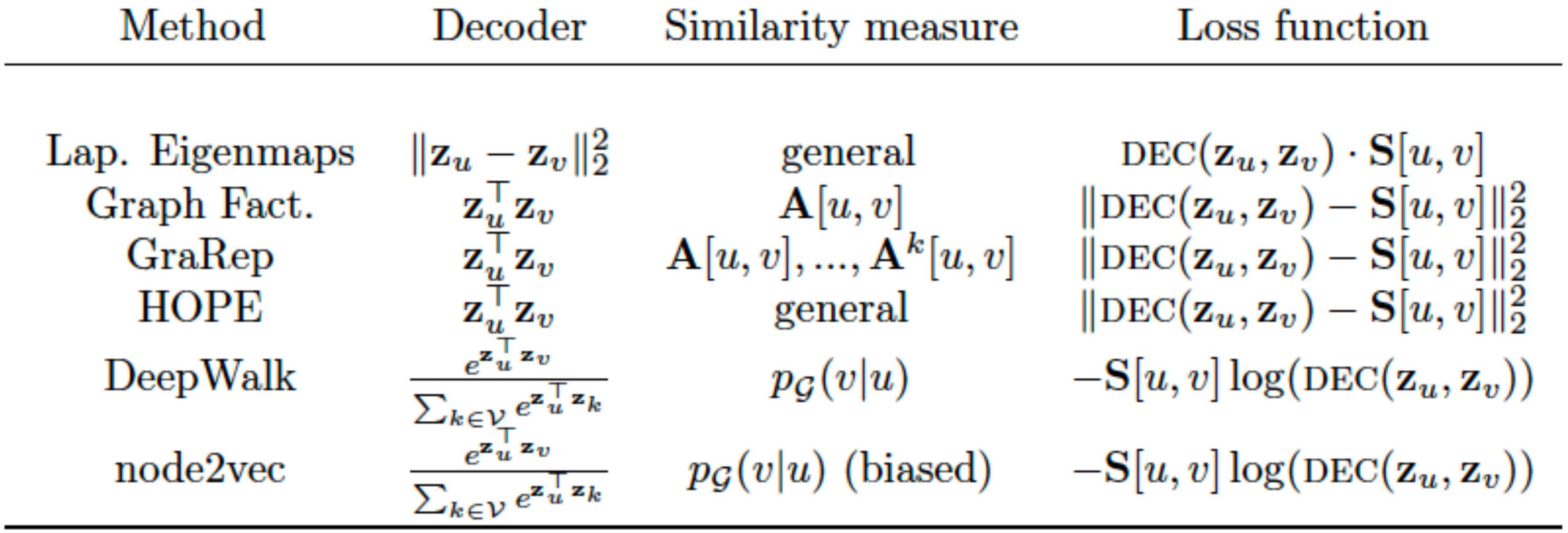
Optimizing an Encoder-Decoder

- 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)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

Factorization-based approaches

- One way of viewing the encoder-decoder idea is from the perspective of matrix factorization.
- Indeed, decoding local neighborhood structure from a node's embedding is closely related to reconstructing entries in the graph adjacency matrix.
- We can view this as a matrix factorization task to learn a low-dimensional approximation of a node-node similarity matrix S, where S generalizes the adjacency matrix and captures some user defined notion of node-node similarity

Laplacian eigenmaps

- One of the earliest factorization-based approaches is the Laplacian eigenmaps (LE) technique, which builds upon the spectral clustering.
- In this approach, the decoder based on the $\mathrm{L2}$-distance between the node embeddings is:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \|\mathbf{z}_{\boldsymbol{u}} - \mathbf{z}_{\boldsymbol{v}}\|_{\mathbf{2}}^{\mathbf{2}}.$$

- The loss function then weighs pairs of nodes according to their similarity in the graph:

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

Laplacian eigenmaps

- The intuition behind this approach is that we penalize the model when very similar nodes have embeddings that are far apart.
- If $\boldsymbol{S}$ satisfies the properties of a Laplacian matrix, then the node embeddings that minimize the loss are identical to the solution for spectral clustering.
- If we assume the embeddings $\mathbf{z}_u$ are d-dimensional, then the optimal solution is given by the d smallest eigenvectors of the Laplacian (excluding zero eigenvalues and the eigenvector of all ones).

Inner-product methods

- Following on the Laplacian eigenmaps technique, we can use an inner-product based decoder, defined as follows:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^\top \mathbf{z}_v$$

- Here, we assume that the similarity between two nodes – e.g., the overlap between their local neighborhoods – is proportional to the dot product of their embeddings.
- Some examples of this style of node embedding algorithms include the Graph Factorization (GF) approach [*Ahmed et al., 2013*], GraRep [*Cao et al., 2015*], and HOPE [*Ou et al., 2016*].

Inner-product methods

- These three methods combine the inner-product decoder with the following mean-squared error:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \|\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u;v]\|_2^2.$$

- They differ primarily in how they define $\mathbf{S}[u;v]$, i.e., the notion of node-node similarity or neighborhood overlap that they use.
- Whereas the GF approach directly uses the adjacency matrix and sets $\mathbf{S} \triangleq \mathbf{A}$, the GraRep defines $\mathbf{S}$ based on powers of the adjacency matrix, while the HOPE uses neighborhood overlap measures.

Random walk embeddings

- The inner-product methods discussed so far all employ deterministic measures of node similarity.
- They often define $\boldsymbol{S}$ as a polynomial function of the adjacency matrix, and the node embeddings are optimized so that $\mathbf{z}_u^\top \mathbf{z}_v \approx \mathbf{S}[u, v]$.
- Building on these, many methods have adapted the inner-product approach to use stochastic measures of neighborhood overlap.
- Key innovation: two nodes have similar embeddings if they tend to co-occur on short random walks over the graph.

DeepWalk and node2vec

- Similar to the matrix factorization approaches described so far, DeepWalk and node2vec use a shallow embedding approach and an inner-product decoder.
- The key distinction in these methods is in how they define the notions of node similarity and neighborhood reconstruction.
- Instead of directly reconstructing the adjacency matrix $\mathbf{A}$ – or some deterministic function of $\mathbf{A}$ – these approaches optimize embeddings to encode the statistics of random walks.

DeepWalk and node2vec

- The goal is to learn embeddings so that the following holds:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) \equiv \frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{v_k \in V} \mathbf{z}_u^\top \mathbf{z}_v} \approx p_{\mathcal{G},T}(u|v)$$

- where $p_{\mathcal{G},T}(u|v)$ is the probability of visiting v on a length-T random walk starting at u, with T usually defined to be in the range $T \in \{2, \dots, 10\}$.
- Again, a key difference with the factorization-based approaches is that here the similarity measure is both stochastic and asymmetric.

DeepWalk and node2vec

- To train random walk embeddings, the general strategy is to use the above decoder and minimize the cross-entropy loss:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} -\log(\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v)).$$

- Here, we use $\mathcal{D}$ to denote the training set of random walks, which is generated by sampling random walks starting from each node.
- For example, we can assume that N pairs of co-occurring nodes for each node u are sampled from the distribution $(u, v) \sim p_{\mathcal{G},T}(u|v)$.

DeepWalk and node2vec

- Evaluating that loss function can be computationally expensive.
- node2vec employs a *noise contrastive* approach, where the normalizing factor is approximated using negative samples [Grover and Leskovec, 2016]:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - \gamma \mathbb{E}_{v_n\sim P_n(\mathbf{V})}[\log(-\sigma\left(\mathbf{z}_u^\top \mathbf{z}_{v_n}\right))].$$

- Here, σ denotes the logistic function, $P_n(\mathbf{V})$ to denote a distribution over the set of nodes $\mathbf{V}$, and $\gamma > \mathbf{0}$ is a hyperparameter.

Limits: parameter sharing

- 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 $\mathcal{O}(|\mathbf{V}|)$, which can be intractable in massive graphs.

Limits: leveraging

- A second key issue with shallow embedding approaches is that they do not leverage node features in the encoder.
- Many graph datasets have rich feature information, which could potentially be informative in the encoding process.

Limits: transductivity

- Shallow embedding methods are transductive: they can generate embeddings only for training nodes.
- Generating embeddings for new nodes is sometimes possible with additional optimizations to learn their embeddings.
- This restriction prevents shallow embedding methods from being used on inductive applications, which involve generalizing to unseen nodes after training.