Advanced machine learning

Graphs and node embeddings

Mikkel N. Schmidt

Technical University of Denmark,
DTU Compute, Department of Applied Mathematics and Computer Science.

Overview

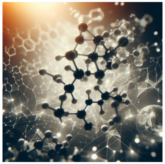
Background

Background: Motivation

Motivation

Humanities and social science
Friendship. Collaboration. Disease spread.

Biology
Neural cells. Protein interactions. Metabolic networks.



Chemistry
Molecular structure. Reaction networks. Material properties.

Motivation

Technical science
Communication networks. Road networks. Flight planning.

Political science
Case law. International relations. Influence analysis.

Computer science
Call graph analysis. Control flow.

Background: Graphs

Simple graph

Definition: A simple graph is a tuple

$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$

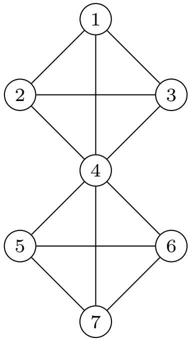
where

$\mathcal{V}$: A set of nodes, representing entities in the graph.

$\mathcal{E}$: A set of edges, representing relationships between nodes.

Example

$$\begin{aligned}
\mathcal{V} =& \{1, 2, \ldots, 6\} \\
\mathcal{E} =& \{(1,2), (1,3), (1,4), \ldots, (7,6)\} \\
\boldsymbol{A} =& \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \\
\boldsymbol{D} =& \text{diag}(3, 3, 3, 6, 3, 3, 3)
\end{aligned}$$



Heterogenous graph

Definition: A heterogeneous graph is a tuple

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}_{\mathrm{V}}, \mathcal{T}_{\mathrm{E}}, f_{\mathrm{V}}, f_{\mathrm{E}})$$

where

$\mathcal{V}$: A set of nodes, representing entities in the graph.

$\mathcal{E}$: A set of edges, representing relationships between nodes.

$\mathcal{T}_{\mathrm{V}}$: A set of node types; each node $v \in \mathcal{V}$ has a specific type $t_v \in \mathcal{T}_{\mathrm{V}}$.

$\mathcal{T}_{\mathrm{E}}$: A set of edge types; each edge $e \in \mathcal{E}$ has a specific type $t_e \in \mathcal{T}_{\mathrm{E}}$.

f_{V}: A node type mapping function, $f_{\mathrm{V}} : \mathcal{V} \to \mathcal{T}_{\mathrm{V}}$
that assigns each node v to its corresponding type, $t_v = f_{\mathrm{V}}(v)$.

f_{E}: An edge type mapping function, $f_{\mathrm{E}} : \mathcal{E} \to \mathcal{T}_{\mathrm{E}}$
that assigns each edge e to its corresponding type, $t_e = f_{\mathrm{E}}(e)$.

Special graphs

Directed	Edges have a direction.
Undirected	Edges have no direction (sometimes represented with duplicate opposed directed edges).
Simple	No self edges and no multiple edges.
Bipartite	Nodes can be divided into to disjoint sets such that all edges are between sets.
Complete	Any two nodes are connected by an edge.
Connected	There exists a path between any two distinct nodes.
Tree	Acyclic graph where each pair of nodes are connected by exactly one path.

Graph Laplacians

Simple undirected graphs

1. Unnormalized Laplacian
$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$$
2. Symmetric normalized Laplacian
$$L_{\text{sym}} = \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{L} \boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}$$
3. Random walk Laplacian
$$L_{\text{sym}} = \boldsymbol{D}^{-1} \boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1} \boldsymbol{A}$$

All non-negative eigenvalues.

Graph Laplacians

Laplacians offer a way to understand graphs through a matrix representation.

- Links a discrete graph with a continuous mathematical structure.
- Eigenvalues/vectors relate to structural properties of the graph.

Used for task such as

- Estimate node connectedness
- Graph partitioning/clustering
- Graph signal processing (signals that propagate on a graph)

Background: Machine learning tasks on graphs

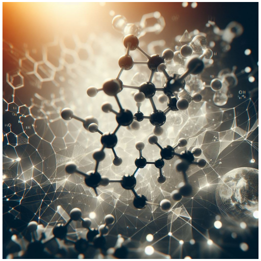
Machine learning tasks

Unsupervised	
Community detection	Identify groups of connected nodes.
Link prediction	Predict formation of new edges.
Node embedding	Learn low-dimensional representations of nodes.
Supervised	
Node/edge classification	Predict node/edge labels
Graph property prediction	Predict properties of the entire graph.
Generative	
Graph generation	Generate new graphs that resemble real-world data.
Graph completion	Predict missing nodes/edges in incomplete graph.

Task levels

Examples in the context of molecular graphs

Node level	Atomic forces and charge.
Edge level	Bond formation. Binding sites.
Path level	Electron flow.
Subgraph level	Functional groups. Reaction prediction.
Graph level	Molecule toxicity, energy, stability, solubility.



Message passing on graphs

Why message passing

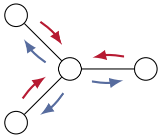
- There is no intrinsic notion of node order. The adjacecy matrices

$$\boldsymbol{A} \quad \text{and} \quad \boldsymbol{A}^* = \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^T$$

  describe the same graph for any permutation matrix $\boldsymbol{P}$.
- The way we represent and process a graph must not depend on any particular permutation.

$$f(\boldsymbol{A}) = f(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\top)$$

Message passing



1. *Initialization:* Each node has an initial state representation.
2. *Messages:* Each node sends a messages to its neighbors based on its current state.
3. *Aggregate:* Each node aggregates its received messages in combination with its previous state.
4. *Update:* Nodes update their state based on aggregated messages.
5. *Iteration:* Steps 2-4 are are repeated to let information propagate through the graph.
6. *Readout:* The final node states are used to make node level predictions, or aggregated to make graph level predictions.

Message passing

Generalized message passing

$$\boldsymbol{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\boldsymbol{h}_v^{(k)} : v \in \mathcal{N}(u)\})\right)$$

Graph level readout

$$\boldsymbol{y} = \text{READOUT}\left(\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N\}\right)$$

Graph statistics

Graph statistics: Eigenvector centrality

Eigenvector centrality

- Measures “influence” in networks: Scores nodes based on connections to other influential nodes.
- Connections matter, but not equally: Links from high-scoring nodes count more than those from low-scoring ones.
- High score = connected to many other high-scorers.
- Similar to PageRank (a website importance score based on quality backlinks)

Eigenvector centrality definition

- Recursive definition: Centrality is a scaled sum of neighbors' centrality.

$$e_u = \overbrace{\frac{1}{\lambda}}^{\text{A constant that defines the scale}} \underbrace{\sum_{v \in \mathcal{N}(u)}}_{\text{Sum over all neighbors of } u} e_v$$

  This can be seen as "message passing" on the graph.
- In matrix notation, this is an eigenvalue problem

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

  and the solution[1] is the eigenvector corresponding to the largest eigenvalue.

[1] The Perron-Frobenius theorem guarantees that the largest eigenvalue is unique with an eigenvector than can be chosen to have non-negative entries: So that solution is the most interesting.

Graph statistics: Clustering coefficient

Clustering coefficient

- Measures ”clumpiness” of connections around a single node.
- High score = neighbors are connected to each other.
- Calculated as ratio of triangles to possible triangles around the node.
- Values between 0 (no triangles) and 1 (all possible triangles formed).
- Density of local neighborhood subgraph.

Clustering coefficient definition

- Ratio of triangles to possible triangles around the node.

$$c_u = \frac{\overbrace{|(v_1, v_2) \in \mathcal{E} : v_1, v_2 \in \mathcal{N}(u)|}^{\text{Number of edges between neighbors of } u}}{\underbrace{\binom{d_u}{2}}_{\text{Number of possible edges between neighbors of } u}}$$

  Not directly computable by message passing.
- In matrix notation

$$\boldsymbol{c} = \big(\boldsymbol{D}(\boldsymbol{D} - \boldsymbol{I})\big)^{-1} \mathrm{diag}(\boldsymbol{A}^3)$$

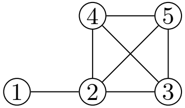
Clustering coefficient example

Clustering coefficient of node 1:

$$c_1 = \frac{0}{\binom{1}{2}} = \frac{0}{0}$$

Clustering coefficient of node 2:

$$c_2 = \frac{3}{\binom{4}{2}} = \frac{3}{6} = 0.5$$



$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{A}^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 4 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 2 \\ 1 & 2 & 2 & 3 & 2 \\ 1 & 2 & 2 & 2 & 3 \end{bmatrix}$$

$$\boldsymbol{A}^3 = \begin{bmatrix} 0 & 4 & 2 & 2 & 2 \\ 4 & 6 & 8 & 8 & 8 \\ 2 & 8 & 6 & 7 & 7 \\ 2 & 8 & 7 & 6 & 7 \\ 2 & 8 & 7 & 7 & 6 \end{bmatrix}$$

$$\boldsymbol{D} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Graph statistics: Weisfieler-Lehman test

Weisfieler-Lehman

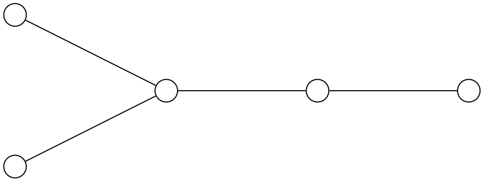
- Determine if two graphs are isomorphic, meaning they have the same underlying structure even if labeled differently.
- Effective but not guaranteed to always determine if two graphs are truly isomorphic.

Weisfieler-Lehman algorithm

1. Assign initial labels
$$l_v^{(0)} = d_v$$
2. Iteratively assign new labels by hasing the multi-set within neighborhood
$$l_v^{(i)} = \textsc{hash}(\{\{l_u^{(i-1)} \forall u \in \mathcal{N}(v)\}\})$$
3. Summarize the labels
$$\textsc{hash}(\{\{l_u^{(i-1)} \forall u \in \mathcal{V}\}\})$$
4. If summary for two graphs do not agree, they cannot be isomorphic.

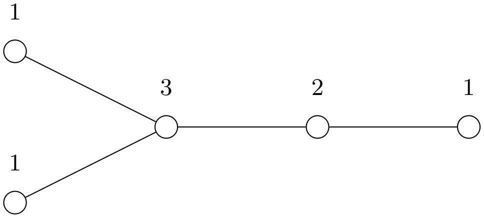
Weisfieler-Lehman example

1. Label nodes by their degree



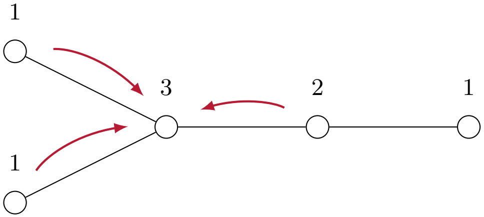
Weisfieler-Lehman example

1. Label nodes by their degree



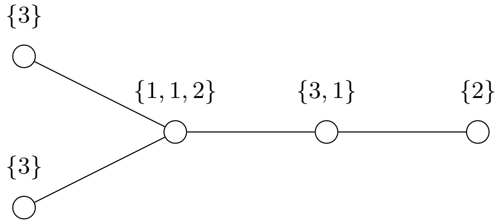
Weisfieler-Lehman example

1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels



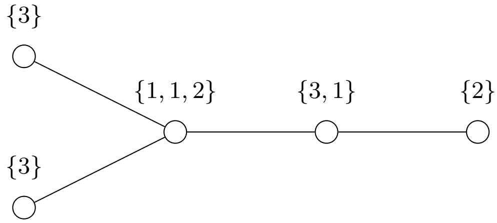
Weisfieler-Lehman example

1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels



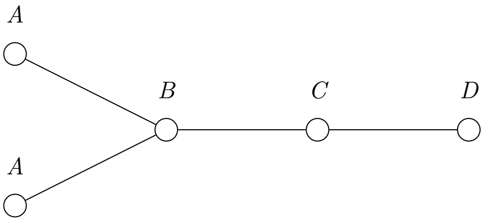
Weisfieler-Lehman example

1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels
3. Hash the labels



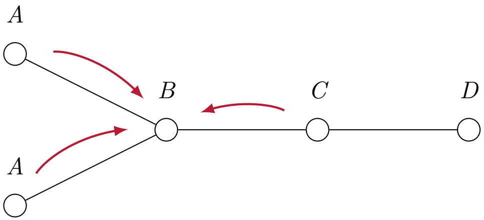
Weisfieler-Lehman example

1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels
3. Hash the labels



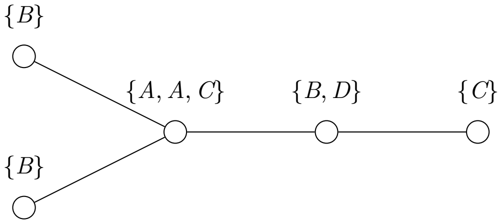
Weisfieler-Lehman example

1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels
3. Hash the labels



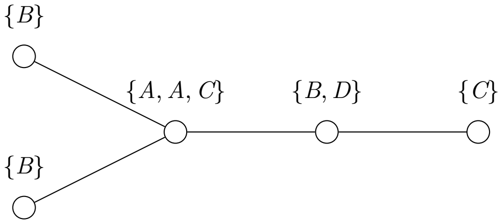
Weisfieler-Lehman example

1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels
3. Hash the labels



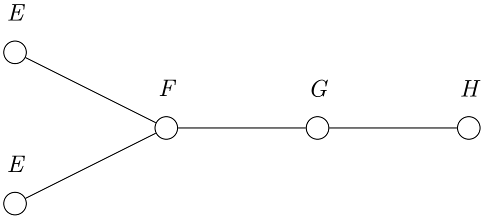
Weisfieler-Lehman example

1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels
3. Hash the labels



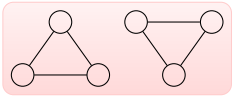
Weisfieler-Lehman example

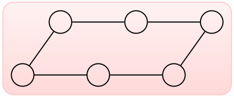
1. Label nodes by their degree
2. Update labels by forming the multi-set of neighbors' labels
3. Hash the labels
4. Summarize the labels



Weisfieler-Lehman failure case

- Cannot always detect that two graphs are isomporphic.
  For example, Weisfieler-Lehman cannot distinguish these two graphs.





- More advanced isomorphism test exist, but it is a hard problem.
- A general problem with message passing: Difficult to detect global structure.

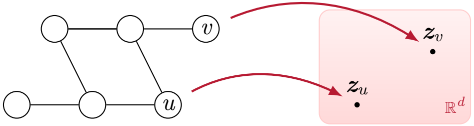
Node embeddings

Node embeddings: Encoder-decoder perspective

Encoder-decoder perspective

Encoder: Maps a vectex into a latent representation.

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



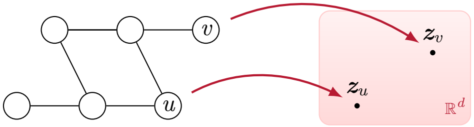
Example 1: Learned embedding (lookup table).

Example 2: Message passing graph neural network.

Encoder-decoder perspective

Encoder: Maps a vectex into a latent representation.

$$\text{ENC}:\mathcal{V}\rightarrow\mathbb{R}^d$$



Example 1: Learned embedding (lookup table).

Example 2: Message passing graph neural network.

Pairwise decoder: Maps a pair of latent variables into a node pair statistic.

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

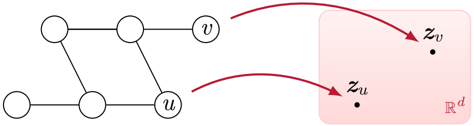
Example 1: Predict link or non-link, $\mathbb{S}=\{0,1\}$.

Example 2: Predict a graph-based similarity measure, $\mathbb{S}=\mathbb{R}_+=\{x\in\mathbb{R}|x\geq0\}$.

Encoder-decoder perspective

Encoder: Maps a vectex into a latent representation.

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



Example 1: Learned embedding (lookup table).

Example 2: Message passing graph neural network.

Pairwise decoder: Maps a pair of latent variables into a node pair statistic.

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{S}$$

Example 1: Predict link or non-link, $\mathbb{S} = \{0, 1\}$.

Example 2: Predict a graph-based similarity measure, $\mathbb{S} = \mathbb{R}_+ = \{x \in \mathbb{R} | x \geq 0\}$.

Loss: Measure discrepancy between observed and estimated graph statistics.

Optimizing an encoder-decoder model

$$\mathcal{L} = \sum_{\underbrace{(u,v)\in\mathcal{V}^2}_{\text{Sum over all node pairs}}} \overset{\text{Element-wise loss}}{\ell}\Big(\textsc{dec}(\underset{\textsc{enc}(u)}{\boldsymbol{z}_u}, \underset{\textsc{enc}(v)}{\boldsymbol{z}_v}), \overset{\text{Graph similarity measure}}{\boldsymbol{S}_{u,v}}\Big)$$

Node embeddings: Encoder: Shallow embedding

Shallow embeddings

- Each node has an embedding vector which is a learned parameter.

$$\boldsymbol{Z} = \begin{bmatrix} \overbrace{-\boldsymbol{z}_1-}^{\text{Embedding vector for each node}} \\ -\boldsymbol{z}_2- \\ \vdots \\ -\boldsymbol{z}_N- \end{bmatrix}$$

- Equivalently, we can think of this as a single linear layer

$$\text{ENC}(u) = \boldsymbol{z}_u = \boldsymbol{s}_u^\top \boldsymbol{Z}$$

where $\boldsymbol{s}_u$ is a one-hot encoding of the node index $\boldsymbol{Z}$ is a weight matrix.

Decoders

Choice of decoder depends on which graph statistic to model.

- *Dot product:* Decodes to a real number, $(-\infty, \infty)$.

$$\textsc{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \boldsymbol{z}_u^\top \boldsymbol{z}_v$$

Decoders

Choice of decoder depends on which graph statistic to model.

- *Dot product:* Decodes to a real number, $(-\infty, \infty)$.

$$\textsc{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \boldsymbol{z}_u^\top \boldsymbol{z}_v$$

- *Squared distance:* Distances decode to a non-negative number, $[0, \infty)$.

$$\textsc{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \|\boldsymbol{z}_u - \boldsymbol{z}_v\|_2^2$$

Decoders

Choice of decoder depends on which graph statistic to model.

- *Dot product:* Decodes to a real number, $(-\infty, \infty)$.

$$\textsc{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \boldsymbol{z}_u^\top \boldsymbol{z}_v$$

- *Squared distance:* Distances decode to a non-negative number, $[0, \infty)$.

$$\textsc{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \|\boldsymbol{z}_u - \boldsymbol{z}_v\|_2^2$$

- *Sigmoid:* Decodes to a binary probability, $[0, 1]$.

$$\textsc{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)$$

Decoders

Choice of decoder depends on which graph statistic to model.

- *Dot product:* Decodes to a real number, $(-\infty, \infty)$.

$$\text{DEC}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \boldsymbol{z}_u^\top \boldsymbol{z}_v$$

- *Squared distance:* Distances decode to a non-negative number, $[0, \infty)$.

$$\text{DEC}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \|\boldsymbol{z}_u - \boldsymbol{z}_v\|_2^2$$

- *Sigmoid:* Decodes to a binary probability, $[0, 1]$.

$$\text{DEC}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)$$

- *Softmax:* Decodes to a discrete probability distribution (not symmetric).

$$\text{DEC}(\boldsymbol{z}_u, \boldsymbol{z}_v) = \frac{e^{\boldsymbol{z}_u^\top \boldsymbol{z}_v}}{\sum_{w \in \mathcal{E}} e^{\boldsymbol{z}_u^\top \boldsymbol{z}_w}}$$

Loss function

- *Squared error*
  In the case when the node similarity is a real number.

$$\mathcal{L}_{\text{mse}} = \underbrace{\sum_{(u,v)\in\mathcal{D}}}_{\text{Sum over all node pairs in training set}} \left(S_{u,v} - \boldsymbol{z}_u^\top \boldsymbol{z}_v\right)^2$$

Loss function

- *Squared error*
  In the case when the node similarity is a real number.

$$\mathcal{L}_{\mathrm{mse}} = \underbrace{\sum_{(u,v)\in\mathcal{D}}}_{\text{Sum over all node pairs in training set}} \left(S_{u,v} - \boldsymbol{z}_u^\top \boldsymbol{z}_v\right)^2$$

- *Binary cross-entropy*
  When the node similarity is binary (e.g. the adjacency matrix).

$$\mathcal{L}_{\mathrm{bce}} = \sum_{(u,v)\in\mathcal{D}} -S_{u,v} \log\left(\underbrace{\sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)}\right) - (1 - S_{u,v}) \log\left(1 - \underbrace{\sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)}_{\text{Logistic sigmoid}}\right)$$

Loss function

- *Squared error*
  In the case when the node similarity is a real number.

$$\mathcal{L}_{\text{mse}} = \underbrace{\sum_{(u,v)\in\mathcal{D}}}_{\text{Sum over all node pairs in training set}} \left(S_{u,v} - \boldsymbol{z}_u^\top \boldsymbol{z}_v\right)^2$$

- *Binary cross-entropy*
  When the node similarity is binary (e.g. the adjacency matrix).

$$\mathcal{L}_{\text{bce}} = \sum_{(u,v)\in\mathcal{D}} -S_{u,v} \log\left(\underbrace{\sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)}\right) - (1 - S_{u,v}) \log\left(1 - \underbrace{\sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)}_{\text{Logistic sigmoid}}\right)$$

- *Random walk*

$$\mathcal{L}_{\text{rw}} = \underbrace{\sum_{(u,v)\in\mathcal{W}}}_{\text{Sum over random walks}} - \log \underbrace{\frac{e^{\boldsymbol{z}_u^\top \boldsymbol{z}_v}}{\sum_{w\in\mathcal{E}} e^{\boldsymbol{z}_u^\top \boldsymbol{z}_w}}}_{\text{Expensive to compute}}$$

Squared error

Minimizing squared error with a dot product decoder is a *matrix factorization*

$$\begin{aligned}\mathcal{L}_{\text{mse}} &= \sum_{(u,v)\in\mathcal{D}} \left(S_{u,v} - \boldsymbol{z}_u^\top \boldsymbol{z}_v\right)^2 \\ &= \|\boldsymbol{S} - \boldsymbol{Z}\boldsymbol{Z}^\top\|_F^2\end{aligned}$$

Binary cross entropy loss

- When predicting the adjacency matrix and using the whole graph as training data we have

$$\begin{aligned}\mathcal{L}_{\text{bce}} &= \overbrace{\sum_{(u,v)\in\mathcal{E}} -\log\sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b)}^{\text{All edges}} + \overbrace{\sum_{(u,v)\notin\mathcal{E}} -\log\sigma(-\boldsymbol{z}_u^\top \boldsymbol{z}_v - b)}^{\text{All non-edges}^2} \\ &= -\sum_{u\in\mathcal{V}}\left(\underbrace{\sum_{v\in\mathcal{N}_u}}_{\text{Neighbors of } u} \log\sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b) + \underbrace{\sum_{v\notin\mathcal{N}_u}}_{\text{Non-neighbors of } u} \log\sigma(-\boldsymbol{z}_u^\top \boldsymbol{z}_v - b)\right)\end{aligned}$$

- *Idea:*
  1. Randomly sample a set of neighbors from a (flexible) distribution in the neighborhood around u.
  2. Sample a set of non-neighbors uniformly.

[2] Note that $1 - \sigma(z) = \sigma(-z)$.

Random walk embeddings

A stochastic approximation of the loss can be computed efficiently using a *random walk* combined with *negative sampling*.

$$\mathcal{L}_{\text{bce-rw}} = \underbrace{\sum_{(u,v)\in\mathcal{W}}}_{\text{Sum over random walks}} - \log \sigma(\boldsymbol{z}_u^\top \boldsymbol{z}_v + b) - \gamma \underbrace{\mathbb{E}_{w\sim P(w)}\left[\log \sigma(-\boldsymbol{z}_u^\top \boldsymbol{z}_w - b)\right]}_{\text{Negative sampling}}$$

Exercises

Exercises

A Node level statistics

B Random walks

C Shallow embeddings

D Programming exercise (shallow embedding)