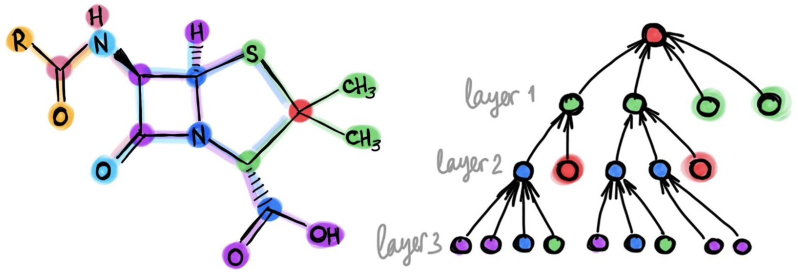
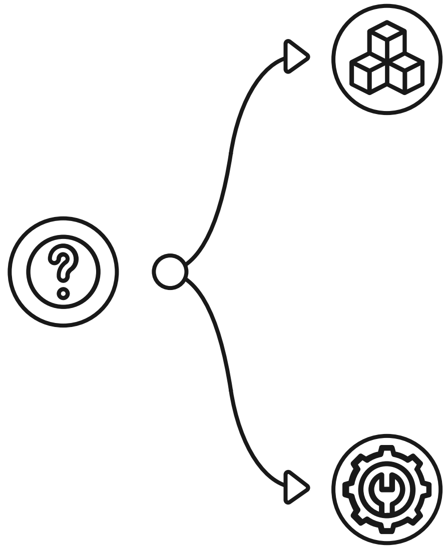


GRAPH ML

Ildus Sadrtdinov
Lecturer, intern researcher, HSE University

MACHINE LEARNING TASKS

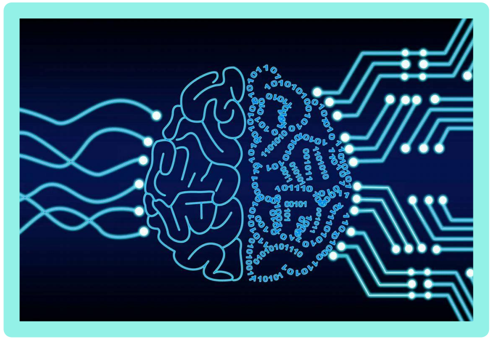


DATA

- Tabular
- Images
- Texts
- Audios
- Graphs
- ...

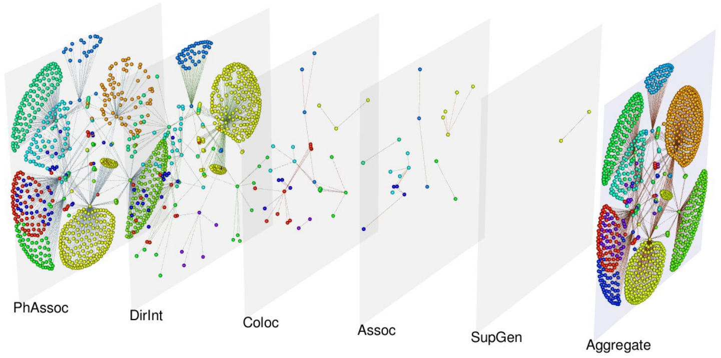
ML ALGORITHM

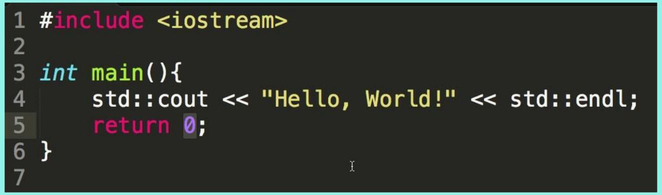
- Linear models
- Decision trees
- Gradient boosting
- Neural networks



EXAMPLES OF NON-TRIVIAL DATA

- Medical datasets
- Interaction networks (e.g. proteins)
- Molecules datasets (e.g. drugs)
- DNA chains
- Source code datasets



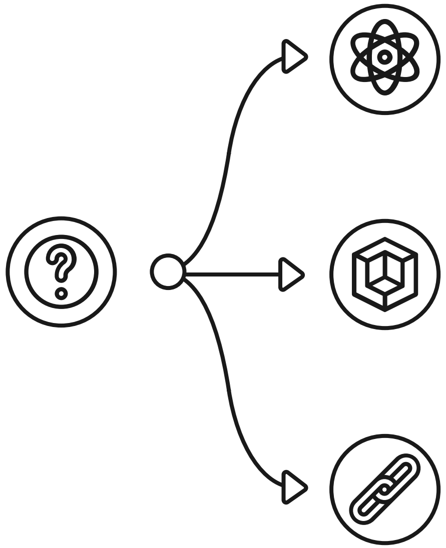


```cpp
#include <iostream>

int main(){
    std::cout << "Hello, World!" << std::endl;
    return 0;
}

```

HOW TO PROCESS?



CLASSICAL SOLUTIONS

- Come from earlier works in particular field
- Usually heuristic

DATA-SPECIFIC ML ALGORITHMS

- Often are state-of-the-art solutions
- May be rather complicated

FEATURE EXTRACTION + CLASSICAL ML

- Relatively easy to implement
- Most likely perform great (good baselines)

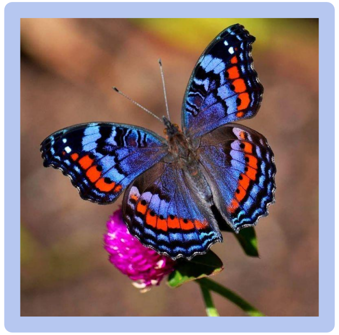
DATA STRUCTURE

SEQUENCES (1D)

- Texts
- Audios
- Time series

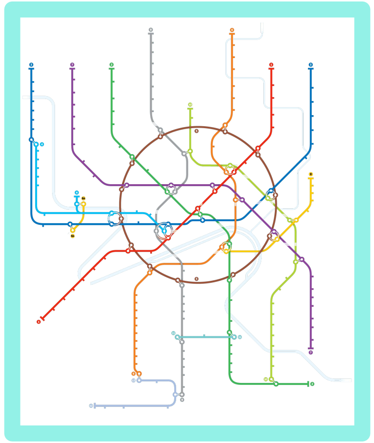
GRID (2D)

- Images
- Spectrograms



ARBITRARY (?D)

- Graphs

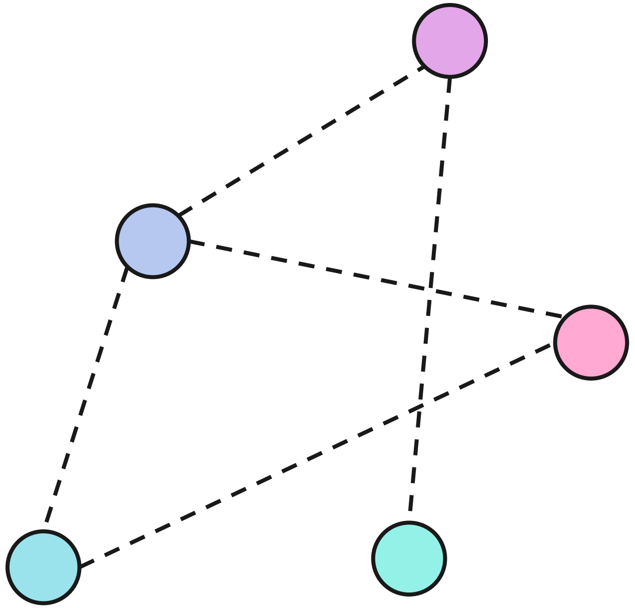


WHAT IS A GRAPH?

● Arbitrary set of nodes **(vertices)**

● Connections between them **(edges)**

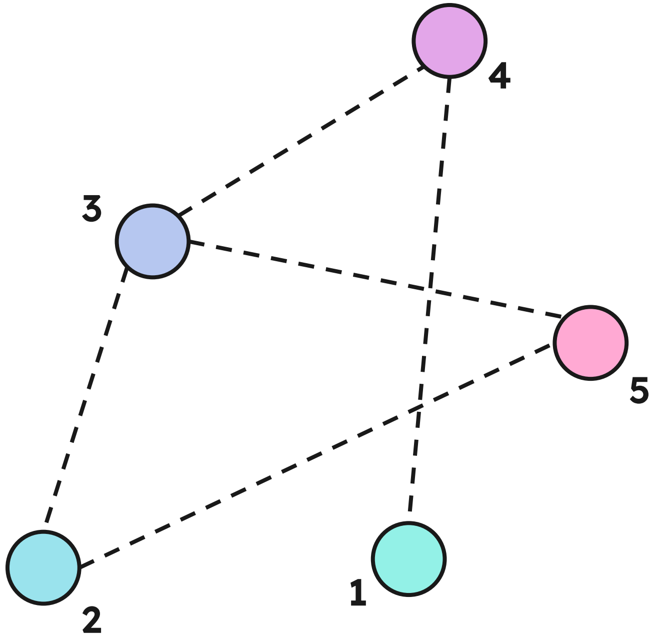
$$\mathcal{G} = (\mathcal{V}, \mathcal{E}) \text{ – graph}$$
$$\mathcal{V} \text{ – set of vertices}$$
$$\mathcal{E} \subseteq V \times V \text{ – set of edges}$$



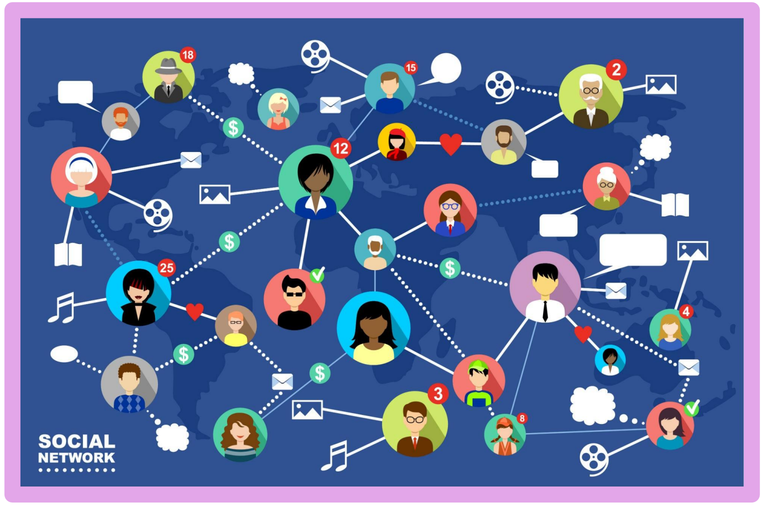
ADJACENCY MATRIX

$$A = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 \\ 3 & 0 & 1 & 0 & 1 & 1 \\ 4 & 1 & 0 & 1 & 0 & 0 \\ 5 & 0 & 1 & 1 & 0 & 0 \end{array}$$

$$\boxed{\begin{array}{c} \mathcal{G} = (\mathcal{V}, \mathcal{E}) \text{ – graph} \\ A \text{ – adjacency matrix} \end{array}}$$

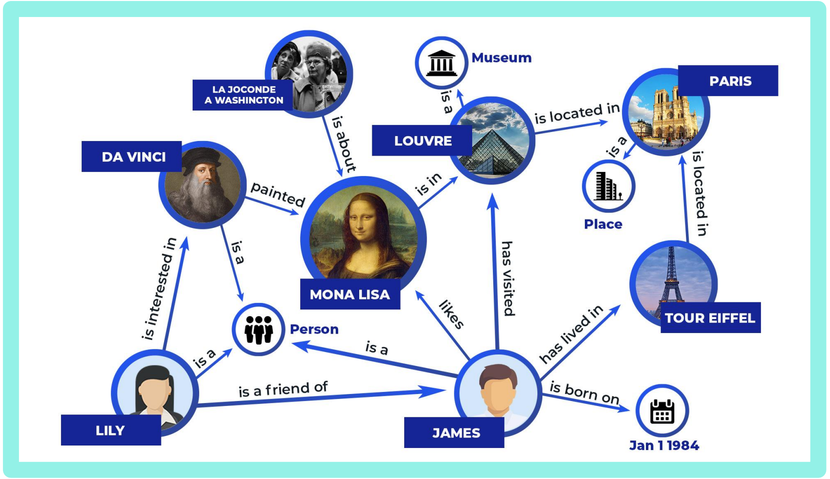


EXAMPLES OF GRAPHS

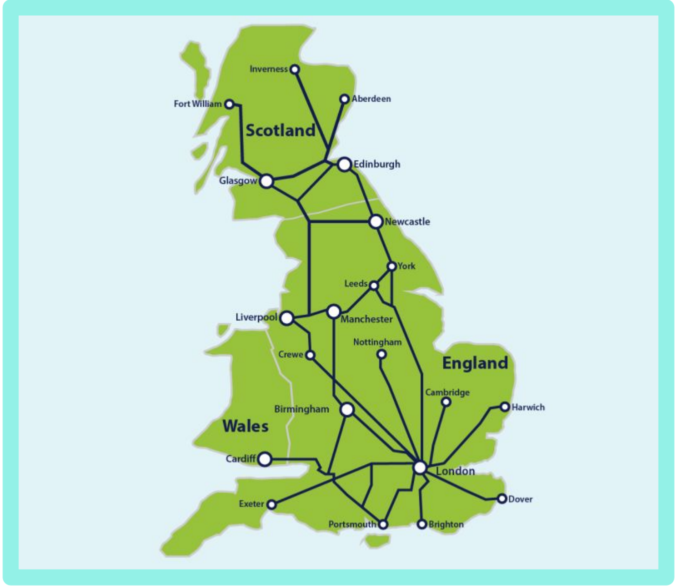


SOCIAL NETWORKS

KNOWLEDGE GRAPH

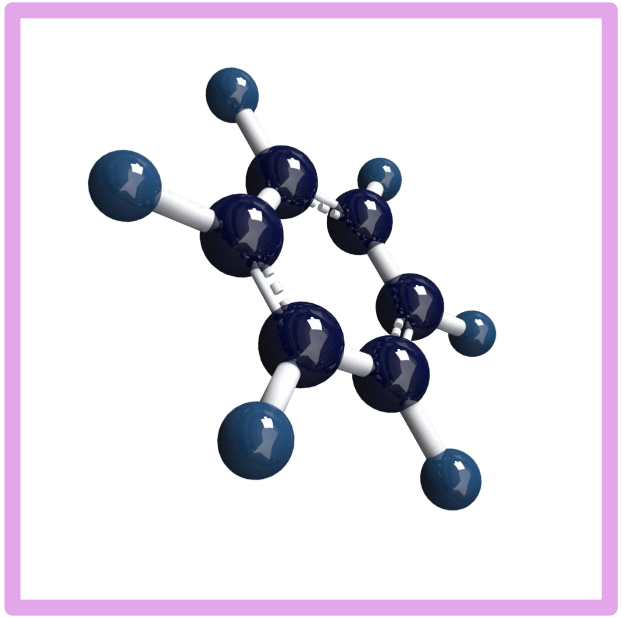


EXAMPLES OF GRAPHS



TRANSPORT NETWORKS

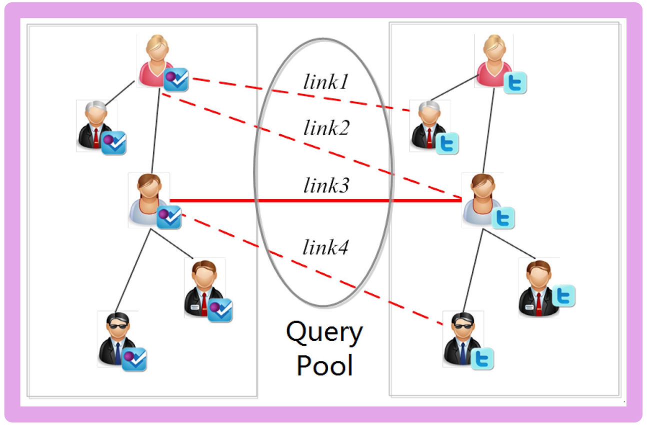
MOLECULES



GRAPH TASKS

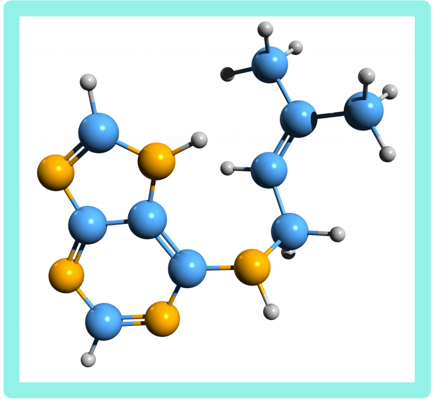
NODE-FOCUSED

- Node classification
- Structural/relationship role determination
- Link prediction
- Node recommendation

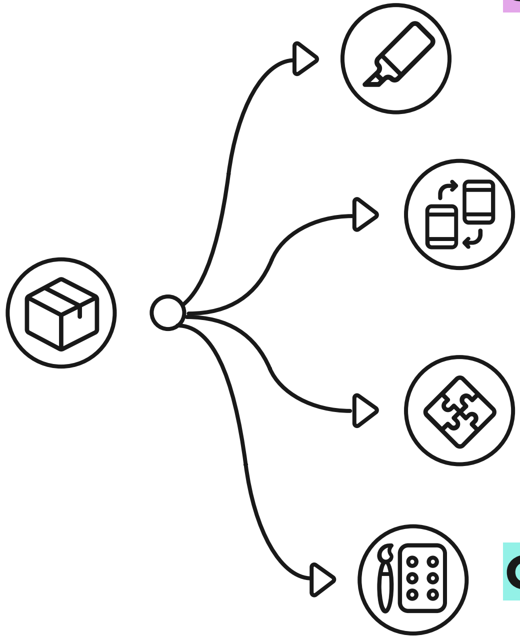


GRAPH-FOCUSED

- Graph classification
- Graph generation
- Estimating global graph properties



GRAPH MACHINE LEARNING



SUPERVISED LEARNING

- Predict particular labels for graphs or nodes

REINFORCEMENT LEARNING

- Interpret graph as an interaction environment

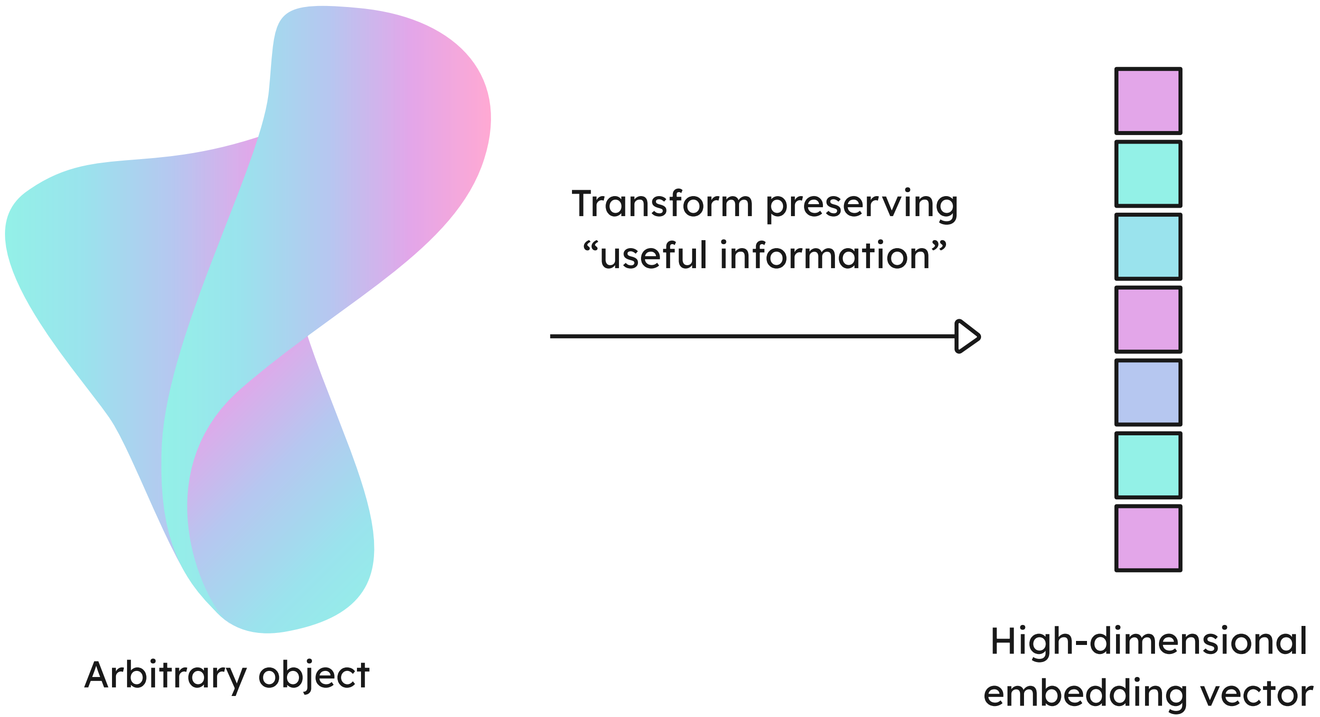
REPRESENTATION LEARNING

- Extract informative features describing graphs or nodes

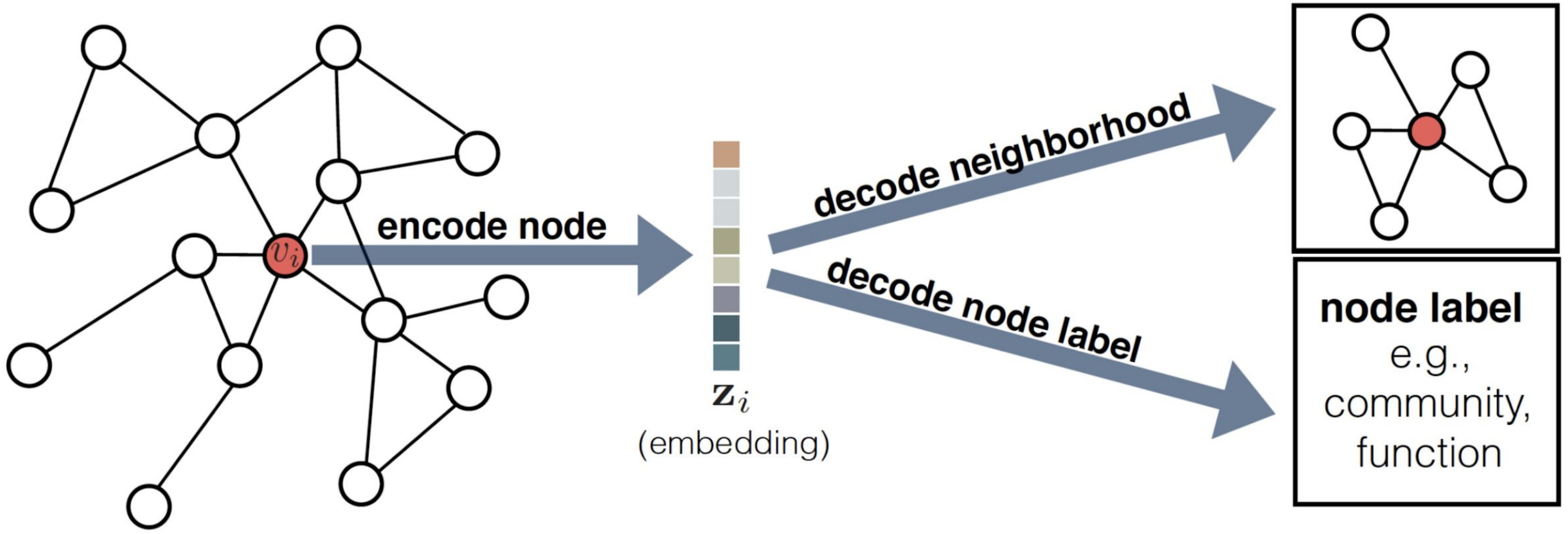
GENERATIVE MODELS

- Generate new graphs with specific features

REPRESENTATION LEARNING



EMBEDDING NODES



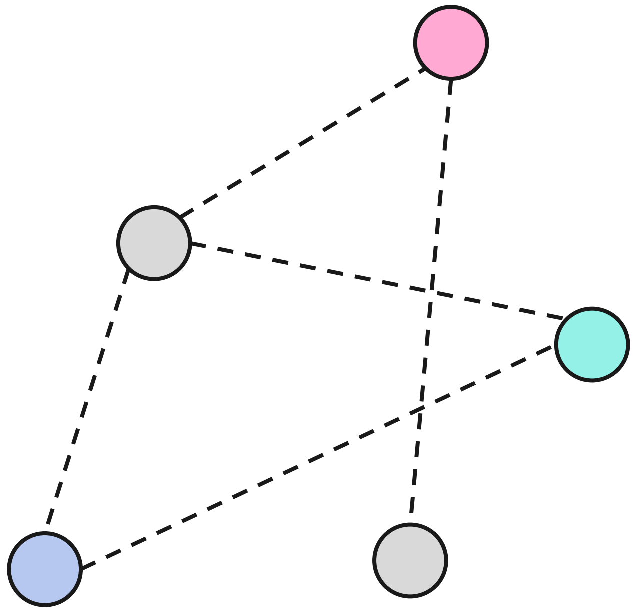
PROXIMITY MEASURE

We can define some distance function between the nodes in our graph

$$s_{\mathcal{G}}\left(\bigcirc, \bigcirc\right) = 1$$

$$s_{\mathcal{G}}\left(\bigcirc, \bigcirc\right) = 2$$

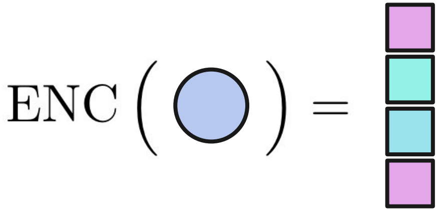
$$\boxed{\begin{array}{c} \mathcal{G} = (\mathcal{V}, \mathcal{E}) - \text{graph} \\ s_{\mathcal{G}} - \text{proximity measure} \end{array}}$$



ENCODER-DECODER APPROACH

- **Encoder** — map a node to some highly dimensional embedding vector
- **Decoder** — take node embeddings and approximate the proximity between these nodes

$$\boxed{\begin{array}{c}\mathcal{G} = (\mathcal{V}, \mathcal{E}) - \text{graph} \\ s_{\mathcal{G}} - \text{proximity measure} \\ v_i, v_j - \text{graph nodes} \\ z_i, z_j - \text{node embeddings}\end{array}}$$



$$\mathrm{ENC}(v_i) = z_i$$

$$\mathrm{DEC}\Big(\mathrm{ENC}(v_i), \mathrm{ENC}(v_j)\Big) =$$
$$= \mathrm{DEC}\Big(z_i, z_j\Big) \approx s_{\mathcal{G}}(v_i, v_j)$$

HOW TO TRAIN?

We need a loss function, e.g. MSE:

$$\Big(\mathrm{DEC}(z_i, z_j) - s_{\mathcal{G}}(v_i, v_j)\Big)^2$$

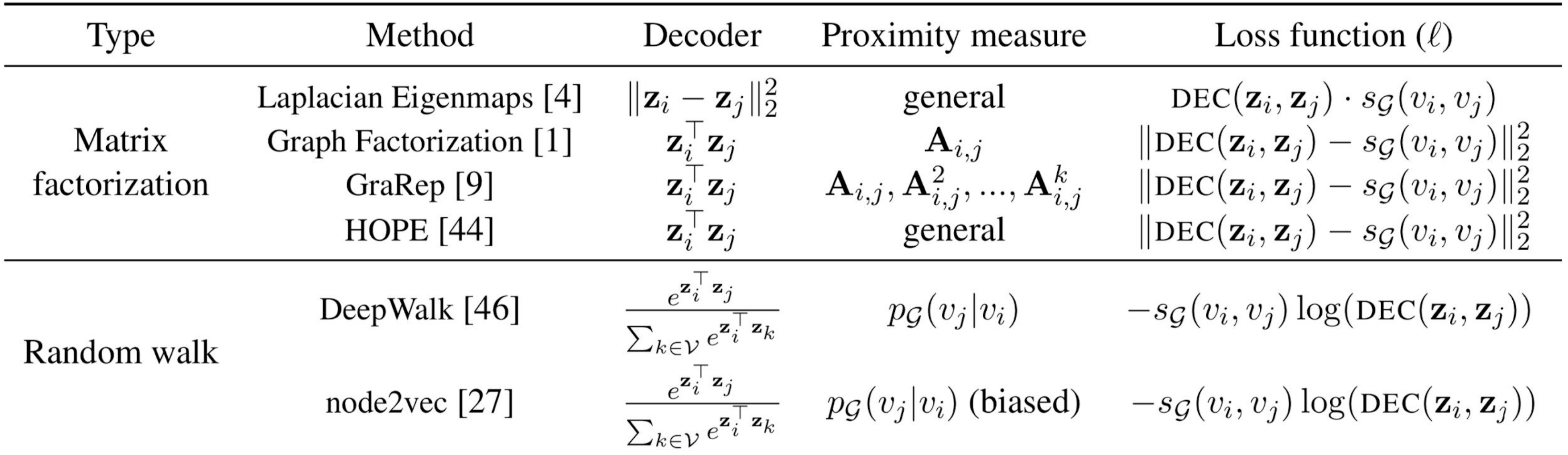
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ – graph
$s_{\mathcal{G}}$ – proximity measure
v_i, v_j – graph nodes
z_i, z_j – node embeddings

Considering all nodes of the graph:

$$\mathcal{L} = \sum_{i,j} \Big(\mathrm{DEC}(z_i, z_j) - s_{\mathcal{G}}(v_i, v_j)\Big)^2 =$$

$$= \sum_{i,j} \Big(\mathrm{DEC}\left(\mathrm{ENC}(v_i), \mathrm{ENC}(v_j)\right) - s_{\mathcal{G}}(v_i, v_j)\Big)^2 \to \min_{\mathrm{ENC,DEC}}$$

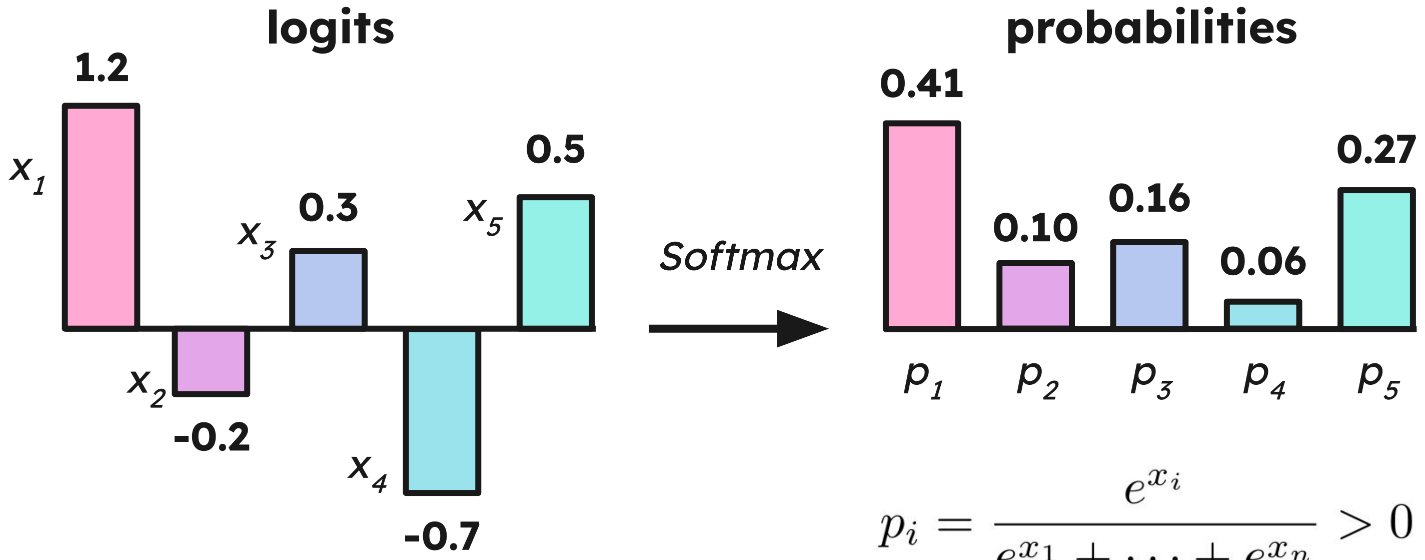
METHODS



Type	Method	Decoder	Proximity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps [4]	$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2$	general	$\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization [1]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	GraRep [9]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}, \mathbf{A}_{i,j}^2, ..., \mathbf{A}_{i,j}^k$	$\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
	HOPE [44]	$\mathbf{z}_i^\top \mathbf{z}_j$	general	$\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2$
Random walk	DeepWalk [46]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \| v_i)$	$-s_{\mathcal{G}}(v_i, v_j) \log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j))$
	node2vec [27]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \| v_i)$ (biased)	$-s_{\mathcal{G}}(v_i, v_j) \log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j))$

v_i – graph nodes
z_i – node embeddings
A – adjacency matrix
$p_{\mathcal{G}}(v_j|v_i)$ – random walk probability

SOFTMAX OPERATOR

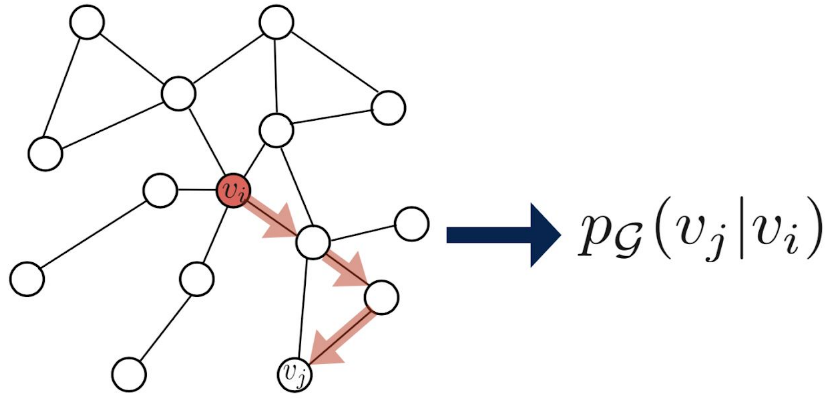


$$p_i = \frac{e^{x_i}}{e^{x_1} + \cdots + e^{x_n}} > 0$$

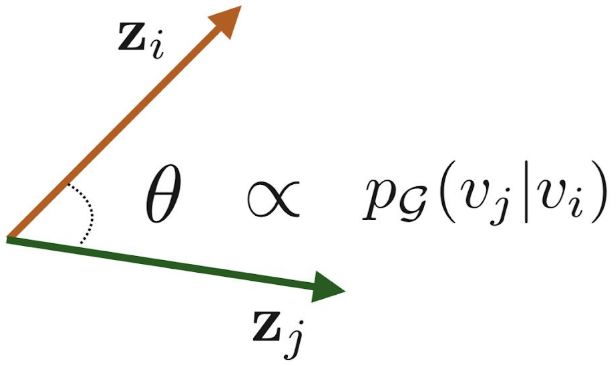
$$p_1 + \cdots + p_n = 1$$

NODE2VEC

$$p_{\mathcal{G}}(v_j|v_i) \approx \text{Softmax}(z_1^T z_i, \dots, z_n^T z_i)_j =$$
$$= \frac{e^{z_i^T z_j}}{e^{z_1^T z_i} + \cdots + e^{z_n^T z_i}}$$

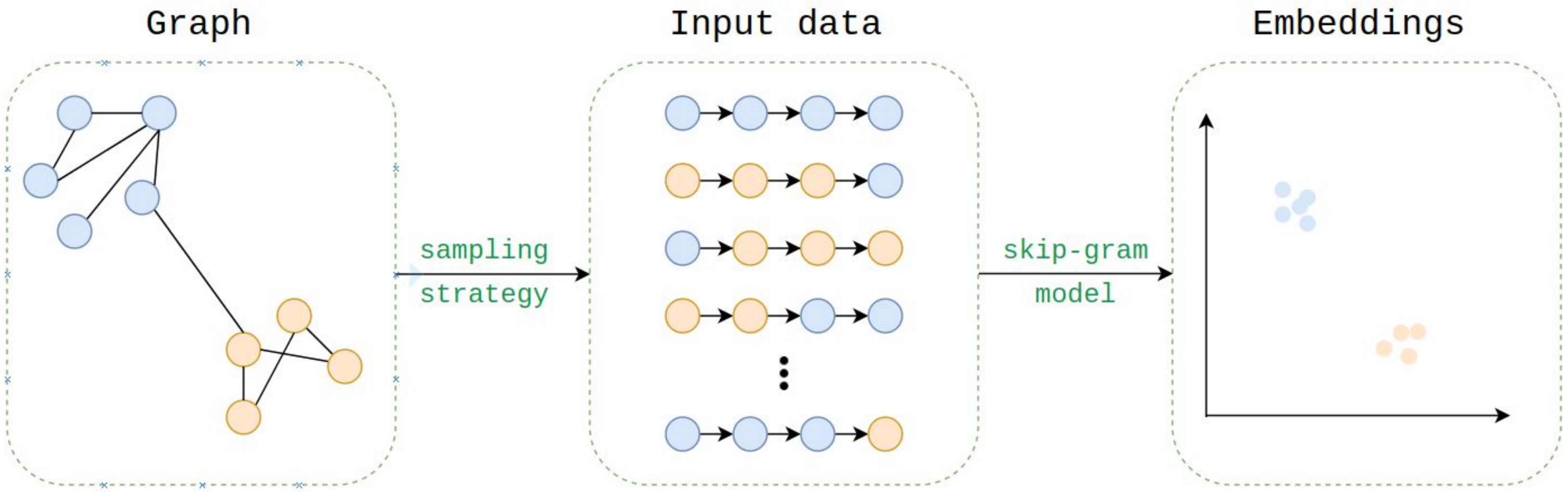


1. Run random walks to obtain co-occurrence statistics.



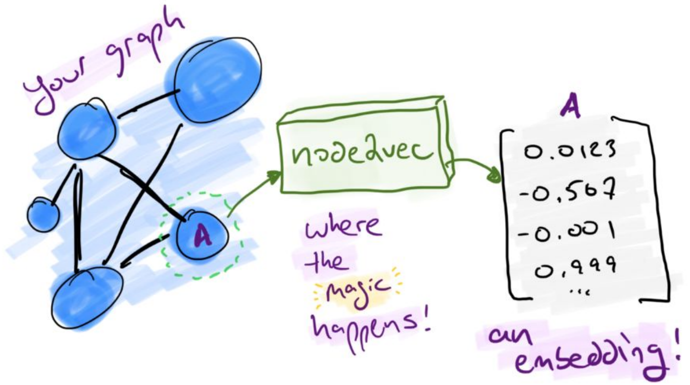
2. Optimize embeddings based on co-occurrence statistics.

NODE2VEC

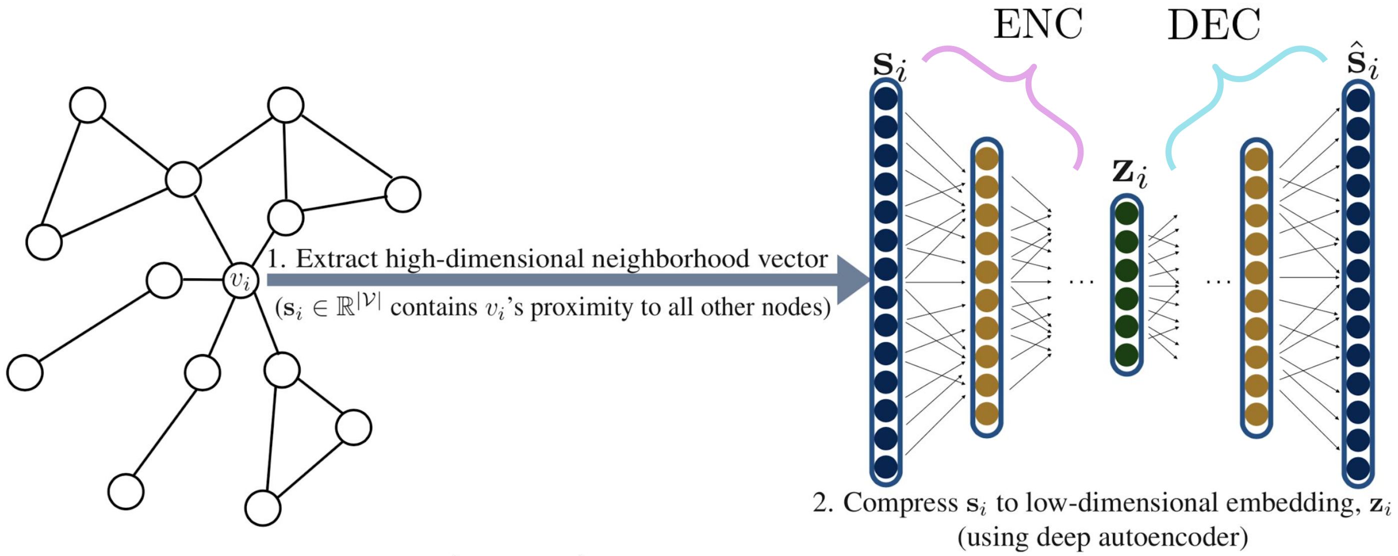

Graph
Input data
Embeddings
sampling strategy
skip-gram model

NODE2VEC

+ Encode structural information about the nodes (random walk statistics)
+ Relatively easy to train
− No shared parameters: individual embedding for each node
− Node metadata is not used
− No embeddings for new nodes



NEIGHBORHOOD AUTOENCODERS



ENC
DEC
$\mathbf{s}_i$
$\hat{\mathbf{s}}_i$
$\mathbf{z}_i$
v_i
1. Extract high-dimensional neighborhood vector
($\mathbf{s}_i \in \mathbb{R}^{|\mathcal{V}|}$ contains v_i's proximity to all other nodes)
2. Compress $\mathbf{s}_i$ to low-dimensional embedding, $\mathbf{z}_i$
(using deep autoencoder)

$$s_i : s_{ij} = s_{\mathcal{G}}(v_i, v_j)$$

$$\text{DEC}\Big(\text{ENC}(s_i)\Big) = \text{DEC}(z_i) \approx s_i$$

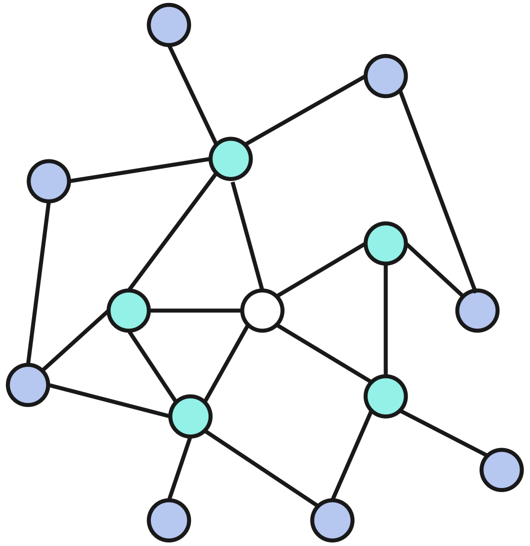
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ – graph
v_i – graph nodes
s_i – neighborhood vectors
z_i – node embeddings

NEIGHBORHOOD AUTOENCODERS

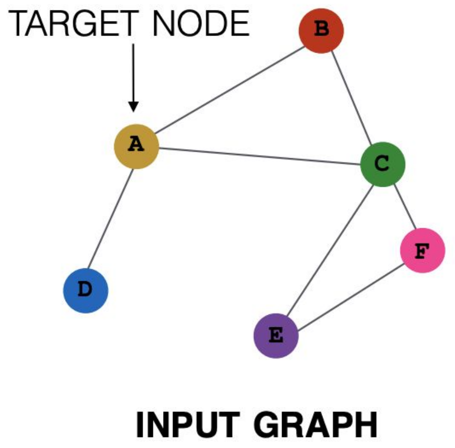
+ The parameters are shared between nodes, consistent learning of embeddings

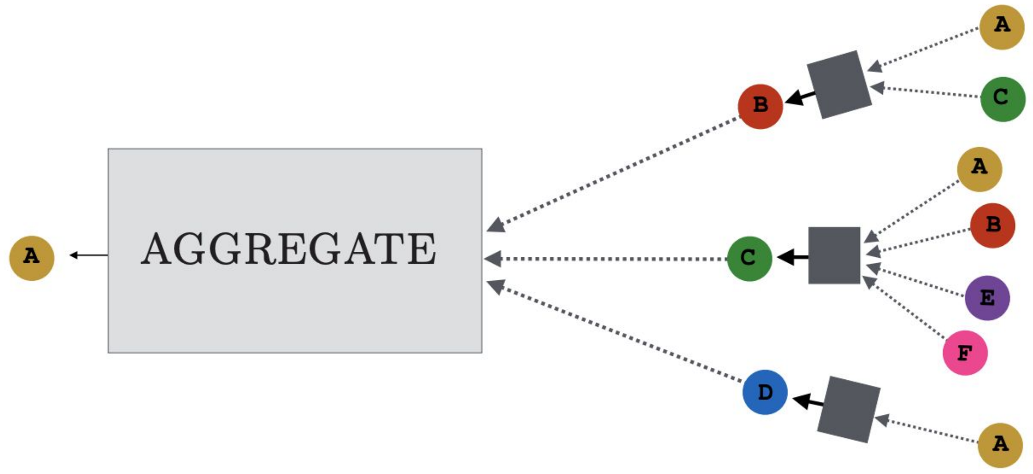
− Node metadata still not in use

− Still no embeddings for new nodes

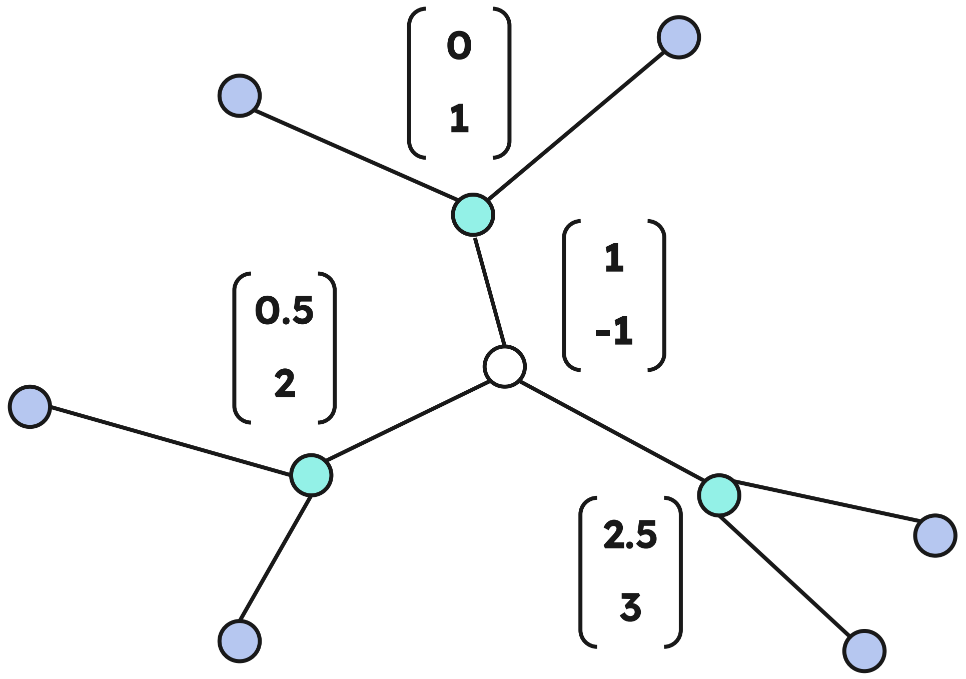


NEIGHBORHOOD AGGREGATION



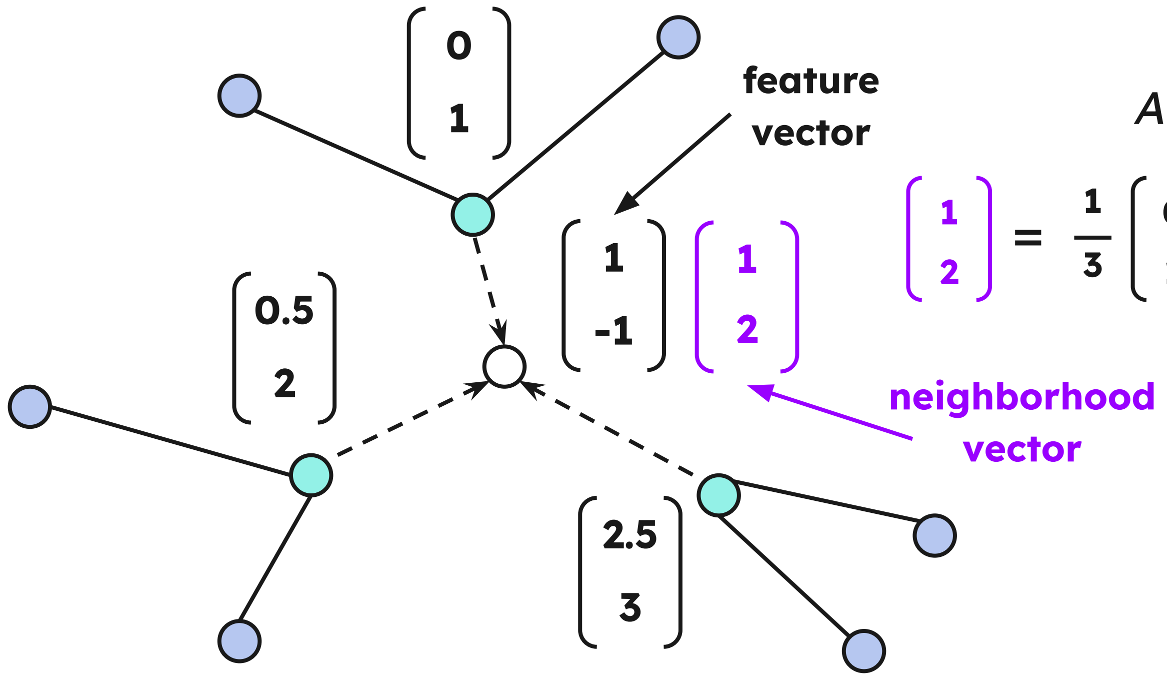


NEIGHBORHOOD AGGREGATION



$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
$\begin{pmatrix} 1 \\ -1 \end{pmatrix}$
$\begin{pmatrix} 0.5 \\ 2 \end{pmatrix}$
$\begin{pmatrix} 2.5 \\ 3 \end{pmatrix}$

NEIGHBORHOOD AGGREGATION

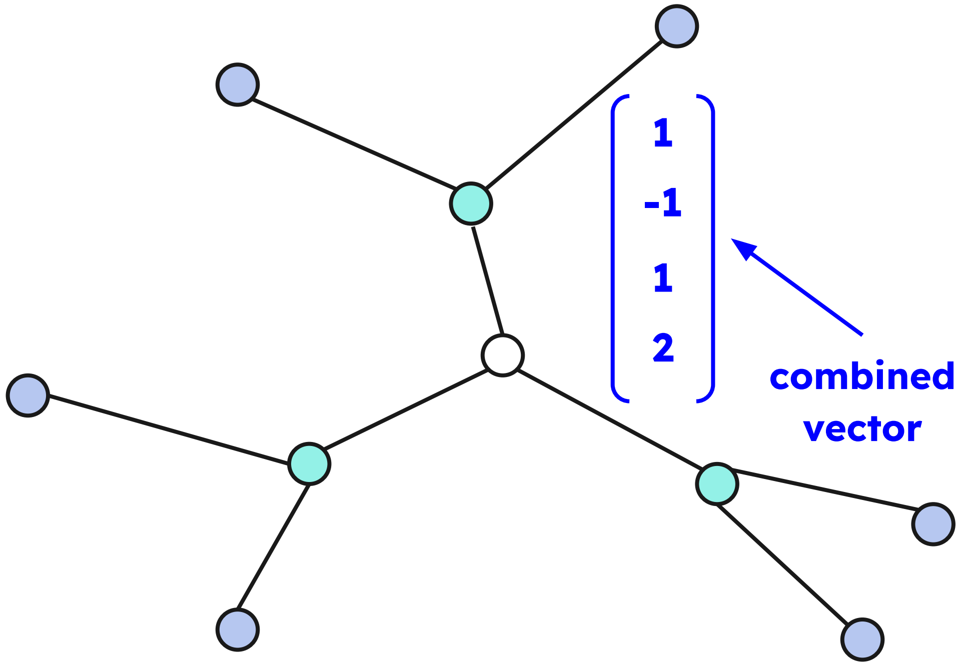


$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
feature vector
$\begin{pmatrix} 1 \\ -1 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$
$\begin{pmatrix} 0.5 \\ 2 \end{pmatrix}$
neighborhood vector
$\begin{pmatrix} 2.5 \\ 3 \end{pmatrix}$

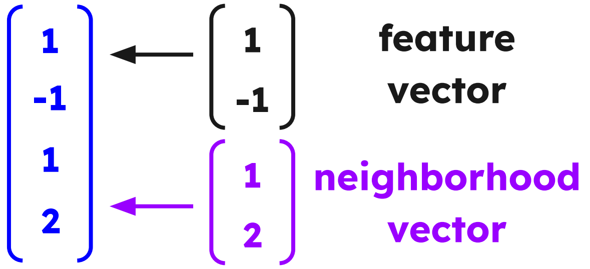
Aggregate:

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \frac{1}{3}\begin{pmatrix} 0 \\ 1 \end{pmatrix} + \frac{1}{3}\begin{pmatrix} 0.5 \\ 2 \end{pmatrix} + \frac{1}{3}\begin{pmatrix} 2.5 \\ 3 \end{pmatrix}$$

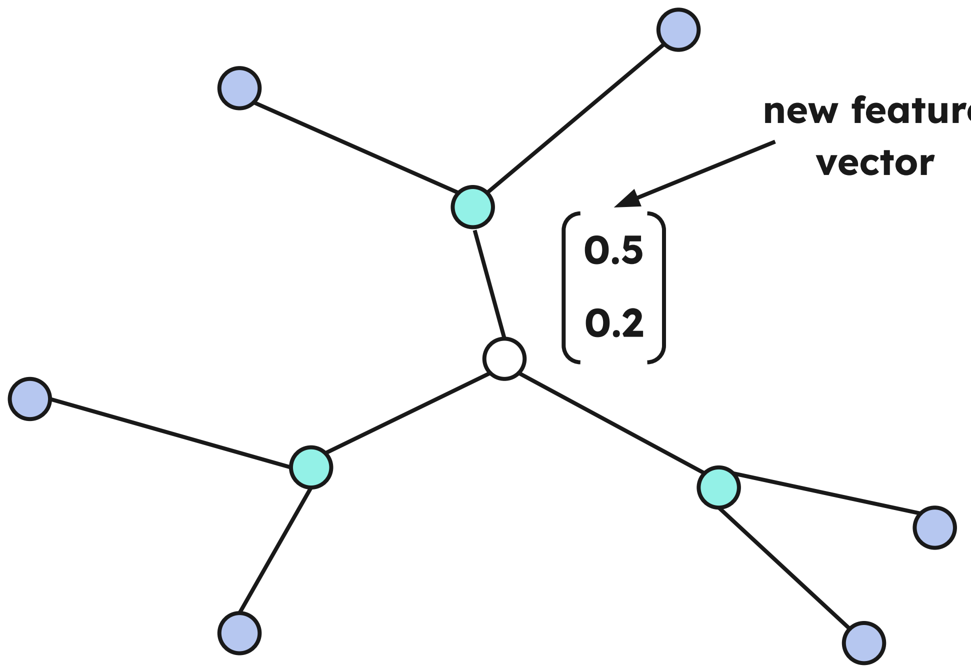
NEIGHBORHOOD AGGREGATION



Combine:



NEIGHBORHOOD AGGREGATION



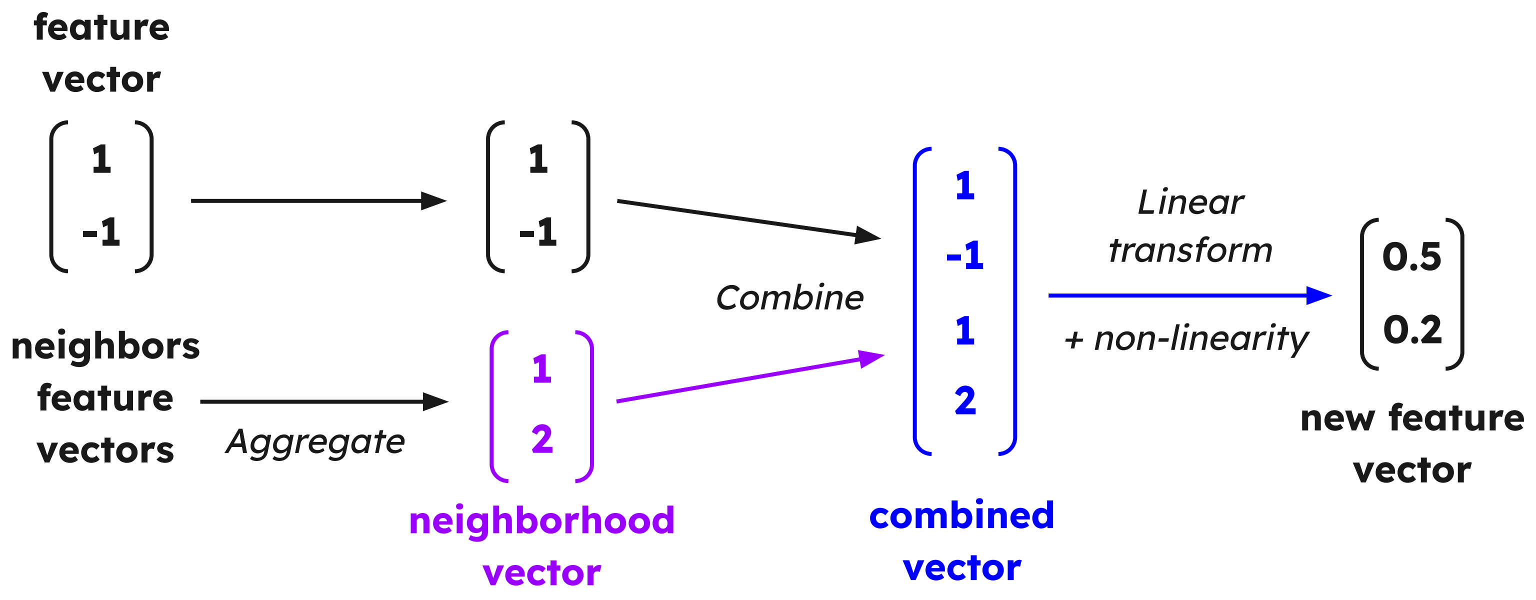
Apply linear transformation and non-linearity:

$$\begin{bmatrix} 0.5 \\ 0.2 \end{bmatrix} = \sigma\left(W \begin{bmatrix} 1 \\ -1 \\ 1 \\ 2 \end{bmatrix} \right)$$

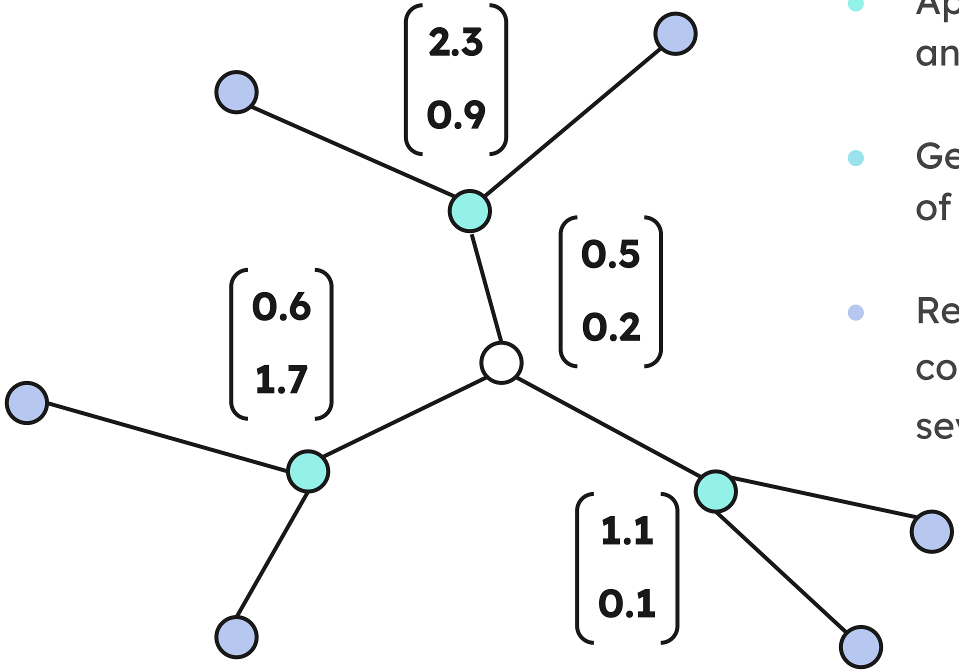
non-linear function

matrix

NEIGHBORHOOD AGGREGATION



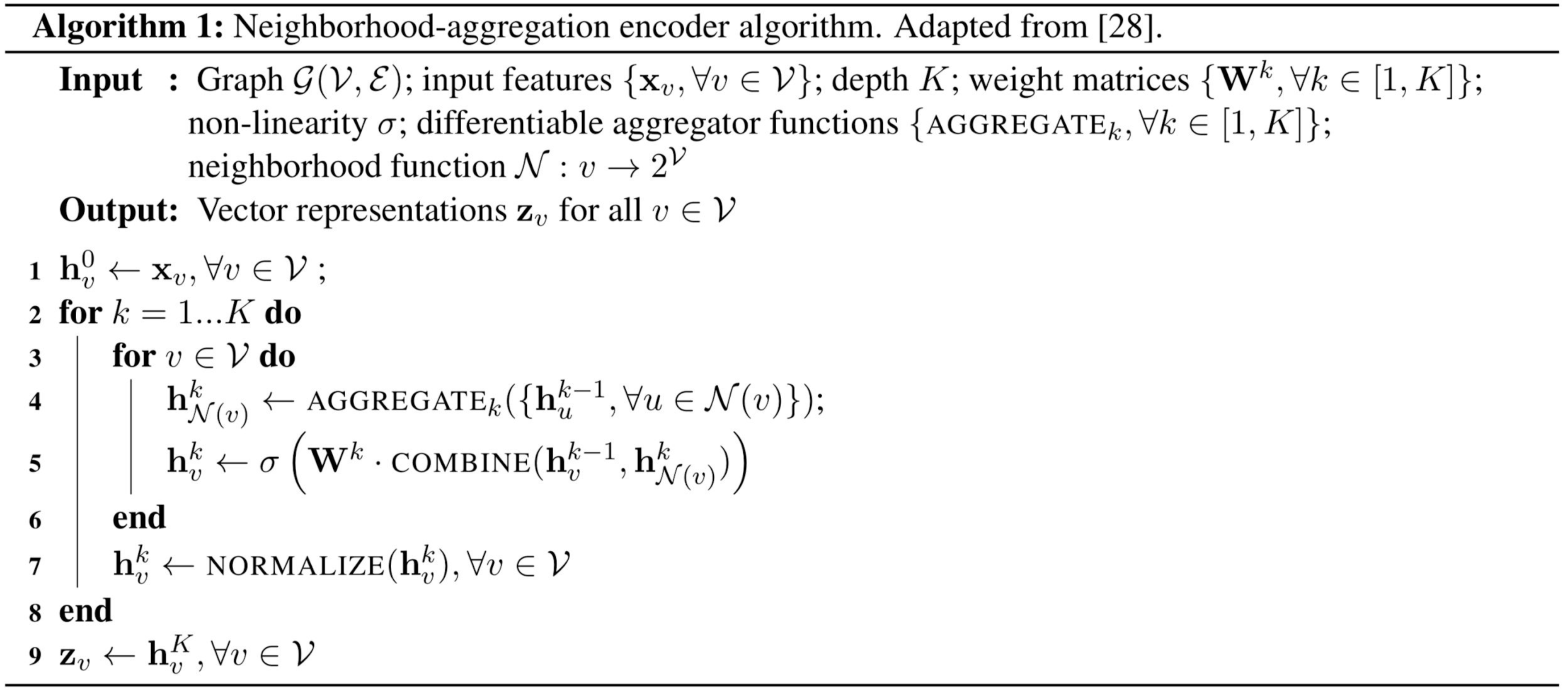
NEIGHBORHOOD AGGREGATION



$\begin{bmatrix} 2.3 \\ 0.9 \end{bmatrix}$
$\begin{bmatrix} 0.5 \\ 0.2 \end{bmatrix}$
$\begin{bmatrix} 0.6 \\ 1.7 \end{bmatrix}$
$\begin{bmatrix} 1.1 \\ 0.1 \end{bmatrix}$

- Apply the aggregation for each node and its neighborhood
- Get new feature vectors for each node of the graph
- Repeat the same procedure consequently for different matrices **W** several times **K**

NEIGHBORHOOD AGGREGATION



Algorithm 1: Neighborhood-aggregation encoder algorithm. Adapted from [28].

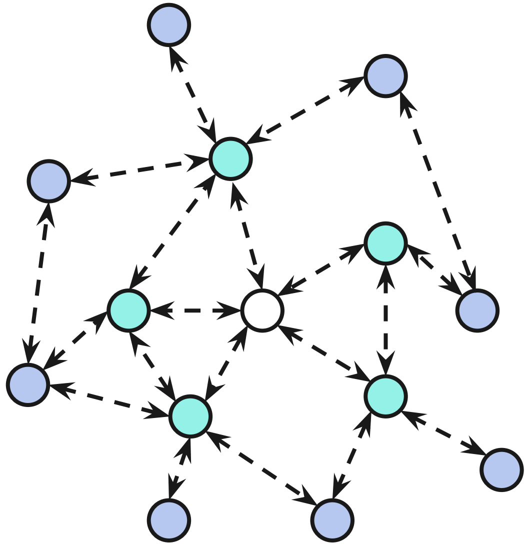
Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\{\mathbf{W}^k, \forall k \in [1, K]\}$; non-linearity σ; differentiable aggregator functions $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

```
$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
for $k = 1 ... K$ do
    for $v \in \mathcal{V}$ do
        $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
        $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{COMBINE}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
    end
    $\mathbf{h}_v^k \leftarrow \text{NORMALIZE}(\mathbf{h}_v^k), \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

NEIGHBORHOOD AGGREGATION

- The parameter matrices $\mathbf{W}_k$ are shared for different nodes
- Node embeddings are build upon node metadata
- It is possible to handle new nodes



FROM NODES TO GRAPHS

- Aggregate node embeddings, e.g. take the mean vector:

$$z_{\mathcal{G}} = \frac{1}{|\mathcal{V}|} \sum_i z_i$$

- Express graph embedding as a sequence of node embeddings:

$$z_{\mathcal{G}} = \left(z_1, \cdots, z_{|\mathcal{V}|}\right)$$

- More complicated techniques using Graph Neural Networks

$$\boxed{\begin{array}{c} \mathcal{G} = (\mathcal{V}, \mathcal{E}) - \text{graph} \\ z_{\mathcal{G}} - \text{graph embedding} \\ z_i - \text{node embeddings} \end{array}}$$

SUMMARY

- Many real-world data examples can be represented as a graph
- We can solve non-trivial tasks with feature extraction and classical ML methods
- **Node2Vec:** embeddings preserve random walk information
- **Neighborhood autoencoder:** embedding is a compressed neighborhood vector
- **Neighborhood aggregation:** combine features iteratively over the neighborhood of each node

LITERATURE

- **Deep Learning on Graphs: A Survey** — Ziwei Zhang, Peng Cui and Wenwu Zhu, https://arxiv.org/pdf/1812.04202.pdf
- **Representation Learning on Graphs: Methods and Applications** — William L. Hamilton, Rex Ying and Jure Leskovec, https://arxiv.org/pdf/1709.05584.pdf
- **node2vec: Scalable Feature Learning for Networks** — Aditya Grover and Jure Leskovec, https://arxiv.org/pdf/1607.00653.pdf
- **Inductive Representation Learning on Large Graphs** — William L. Hamilton, Rex Ying and Jure Leskovec, https://arxiv.org/pdf/1706.02216.pdf