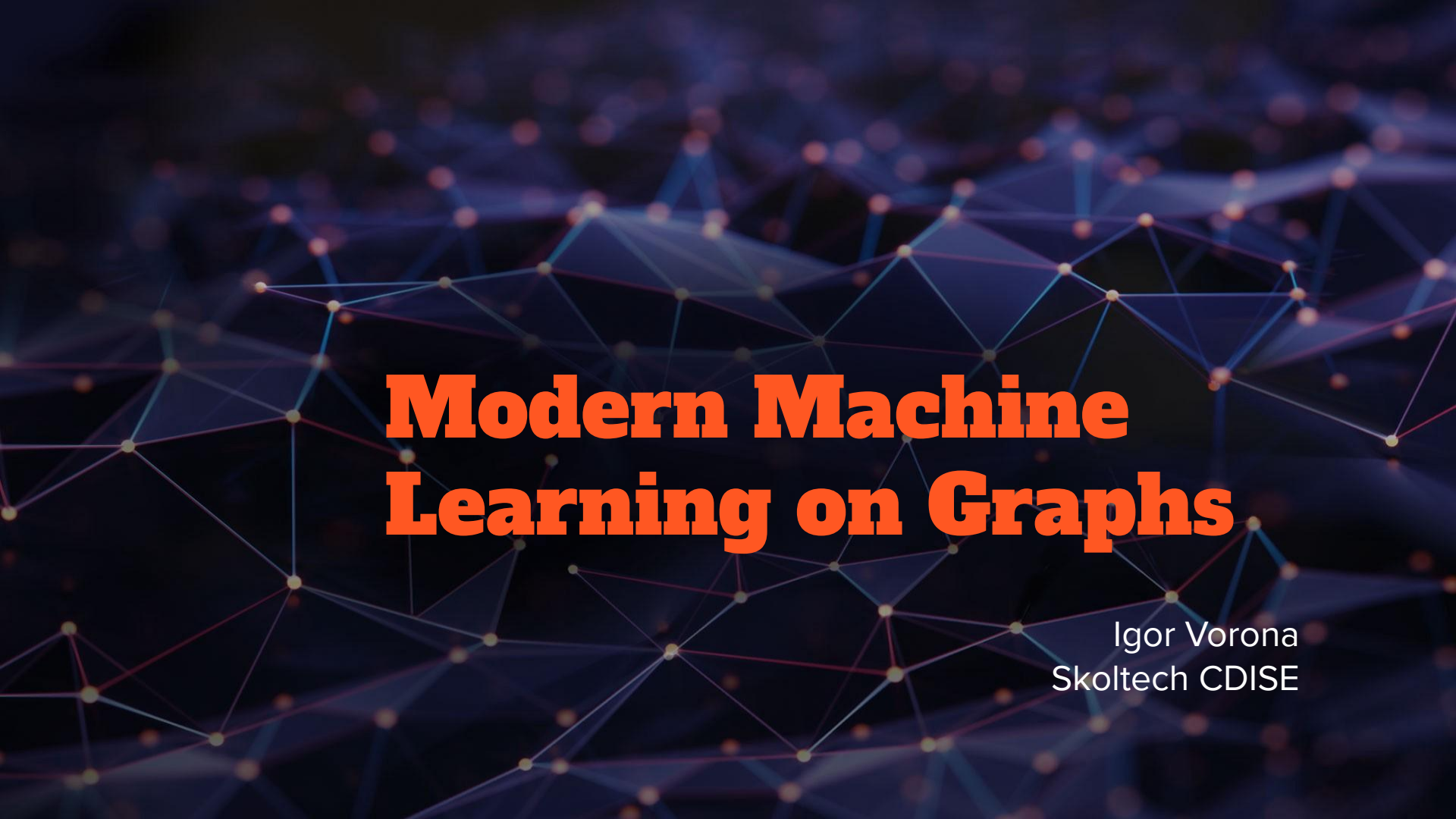


Igor Vorona
Skoltech CDISE

About me

From 

- BSc (2011 - 2015) and MSc (2015 - 2017) at the Lomonosov MSU (CMC faculty)
- Working (2017-2018) on the Matrix Factorization methods in the NLP

To Skoltech

Skolkovo Institute of Science and Technology

- PhD student (first year) at the CDISE Skoltech.
- Supervisors: Andrzej Cichocki, Anh-Huy Phan
- Topic of research about generalized tensor models in the different ML application.

About subject of report

Main reason.

Answer on question:

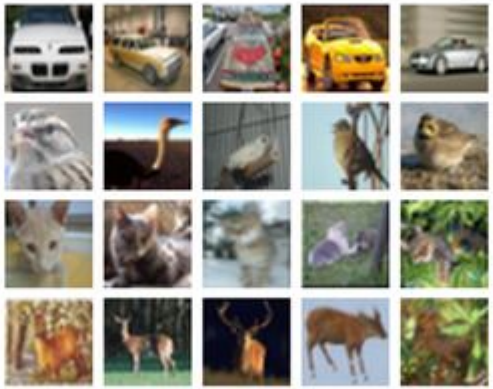
How to work with structured irregular domains in modern DL paradigm?

Entry point

The most efficient DL methods focused so far on data defined on Euclidean domains (i.e. grids), particularly ConvNets for images.

MNIST

CIFAR-10

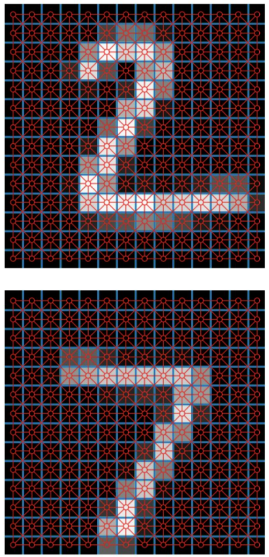


ImageNet

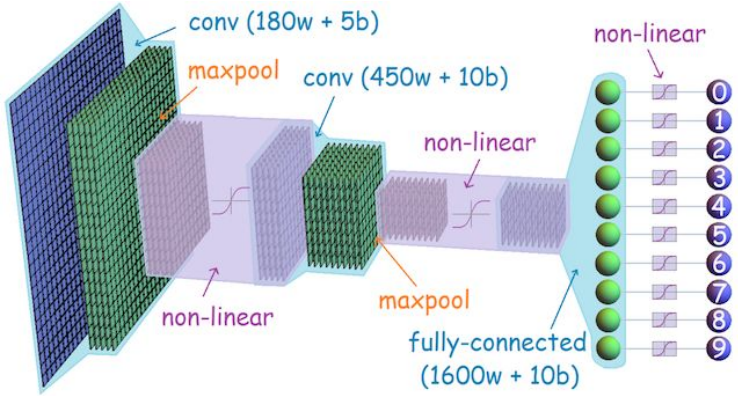
ConvNet Ideas

In this model:

- We work with input data as data which lay on Euclidean domain.
- We include in our model special kind of prior knowledge (***inductive bias***), which based on concrete structure of images. We want to focus on small invariant to some transformation parts of images using convolution filters and pooling.



Regular grid



- **Ok! It works for images.**

- **But can we use this approach for other data?**

- **Depends on REPRESENTATION of this data.**

Useful Representations

We interested to embed our data in **real vector space** with **preserving as much as possible semantically meaningful features** of our data (this increase area of possible applications of our vectors) and which allow us to **easy solve downstream (i.e. target) tasks**.

Some features of “good” representations (Bengio et al.):

- **Smoothness**: assumes that function to be learned f is s.t. $x \approx y \Rightarrow \mathrm{f}(x) \approx \mathrm{f}(y)$
- **Sparsity**: for any given observation only a small fraction of possible factors are relevant. (0 or sensitivity loss)
- **Natural clustering**: different values of categorical variables such as object classes are associated with separate manifolds. It means that linear interpolation between examples of different classes in general involves going through a low density region.
- **Simple and disentangled factor dependencies** as in natural science.

Good cases

For this data we know how to represent it in the real vector space for solve current scope of tasks enough well.

The quick brown fox jumps over the lazy dog

Text

Audio

Word Embeddings / Pre-trained LM

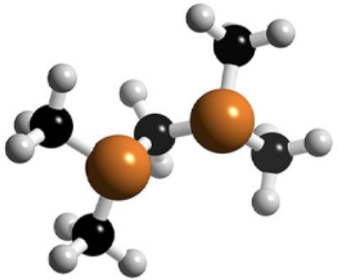
GMM + MFCC / SincNet

Still bad cases

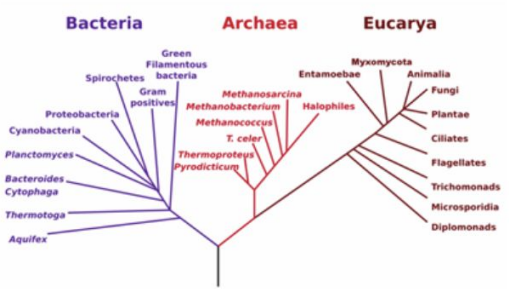
Non-euclidean data represent more complex items and concepts via different specific formats.

It means that data already have sufficient dense representation.

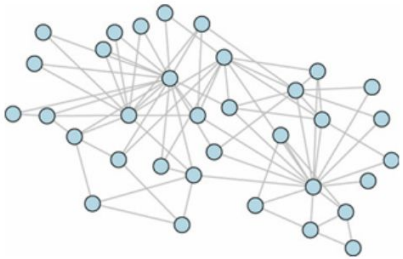
This property increase hardness of efficient embedding this data in euclidean space.



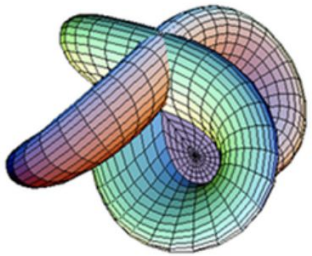
Molecules



Trees



Networks



Manifolds

Map to other data

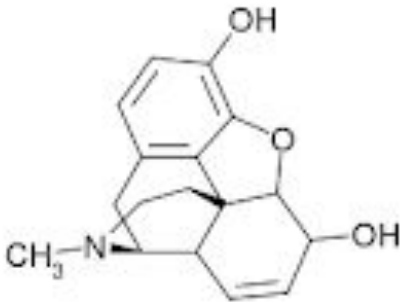
Mapping from our data to another data which can efficiently represented in euclidean space: images, text, audio.

Example:

simplified-molecular-input-line-entry-system (SMILE) string to represent molecules

Drawbacks:

- In transfer function we already lose some structural information about data
- “New” data can unpreserve text, image, audio ordinary features.



Morphine

SMILES: CN1CCC23C4C1CC5=C2C(=C(C=C5)O)OC3C(C=C4)O

Geometric Deep Learning

- It’s umbrella term for collection of techniques to generalize DL methods for non-Euclidean domain.
- Most popular representation of manifolds in GDL:
  - point-based (PointNet, PointNet++)
  - graph-based (Graph ConvNets)
- Use cases:
  - Molecular learning
  - Network learning
  - 3D modeling

Graph Is All You Need

Machine Learning on Graphs

WHY GRAPHS?

- Most general way to work with discrete data. In practical tasks we don't know exact features of object (black-box nature). We only know relation between them. In this case **graphs are a powerful medium for representing diverse knowledge**.

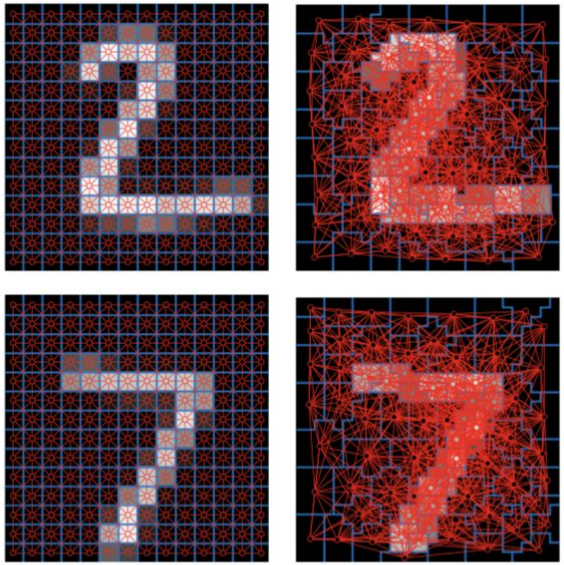
- Easy interpretable and human-focused.

- Huge amount of available data, which already in graph format. Other data we can easily map on graph space without any information loss. Most of old-fashion AI based on specific type of knowledge graphs, which allow to work with weak computational powers and low-amount of data. This correspond to new **zero-shot learning**. Why we don't use those experience?

- Already existed concrete success stories

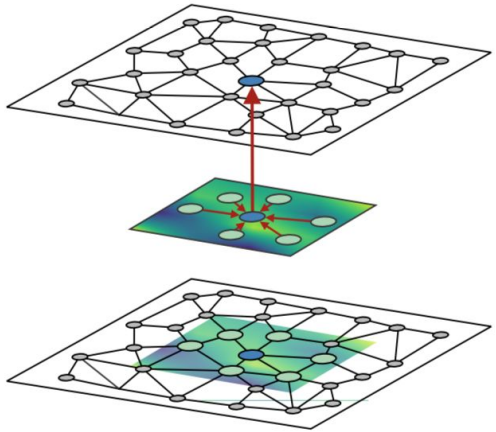
Graphs in Computer Vision

From pixel to superpixel!

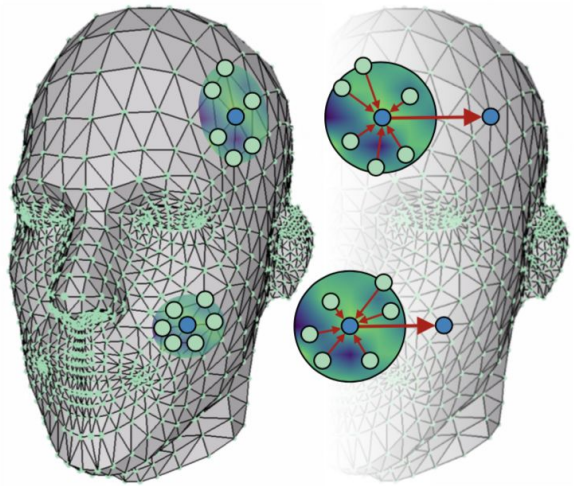


Regular grid Superpixels

Figure 2. Representation of images as graphs. Left: regular grid (the graph is fixed for all images). Right: graph of superpixel adjacency (different for each image). Vertices are shown as red circles, edges as red lines.



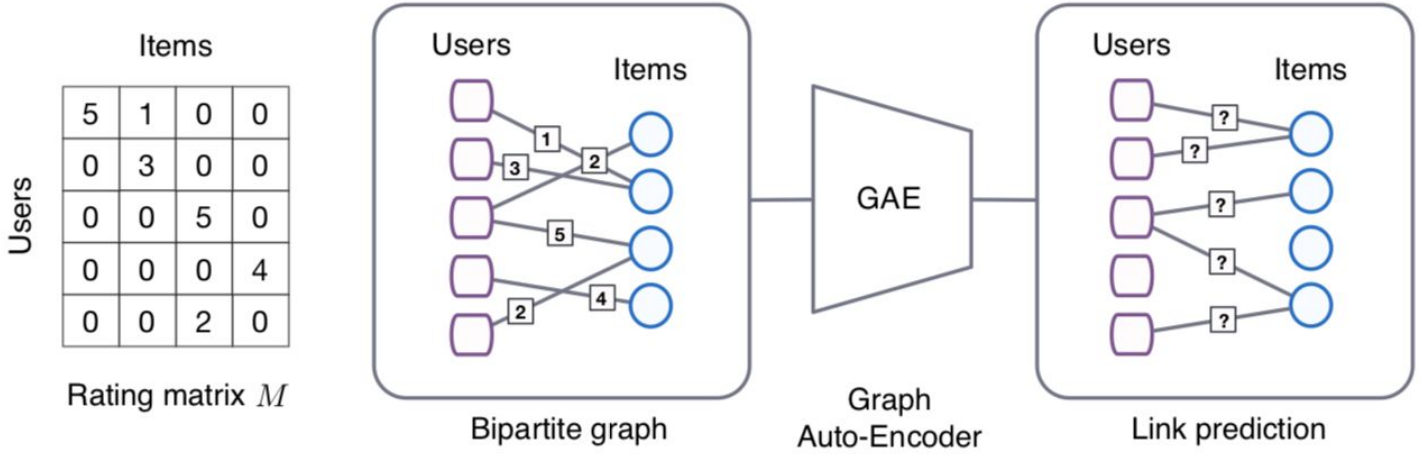
(a) Filtering of graphs



(b) Filtering of meshes

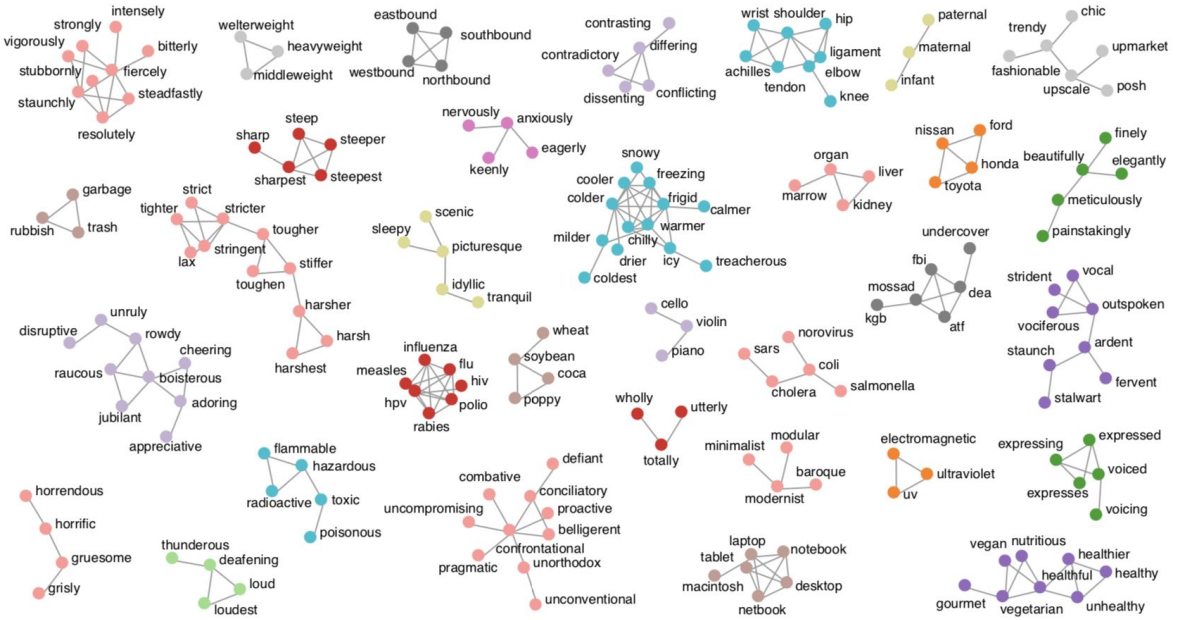
Graphs in Recommender Systems

Representing collaborative filtering rating matrix via bipartite graph (MATRIX COMPLETION)



Graphs in NLP

- Text Summarization
- Knowledge Graphs
- Question Answering
- Word Sense Disambiguation
- Word similarity matrices
- Relation Extraction
- Textual Entailment



Graphs in Social Science

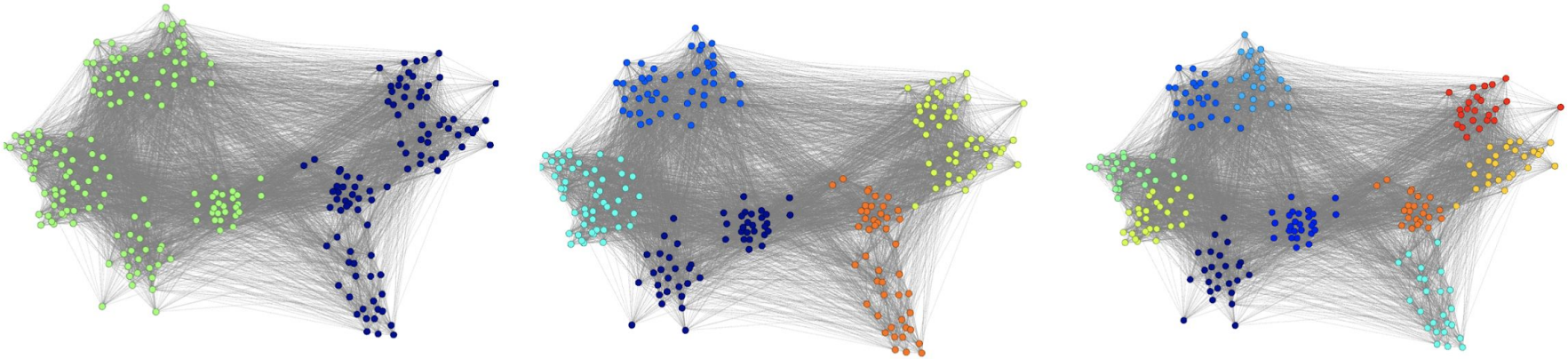
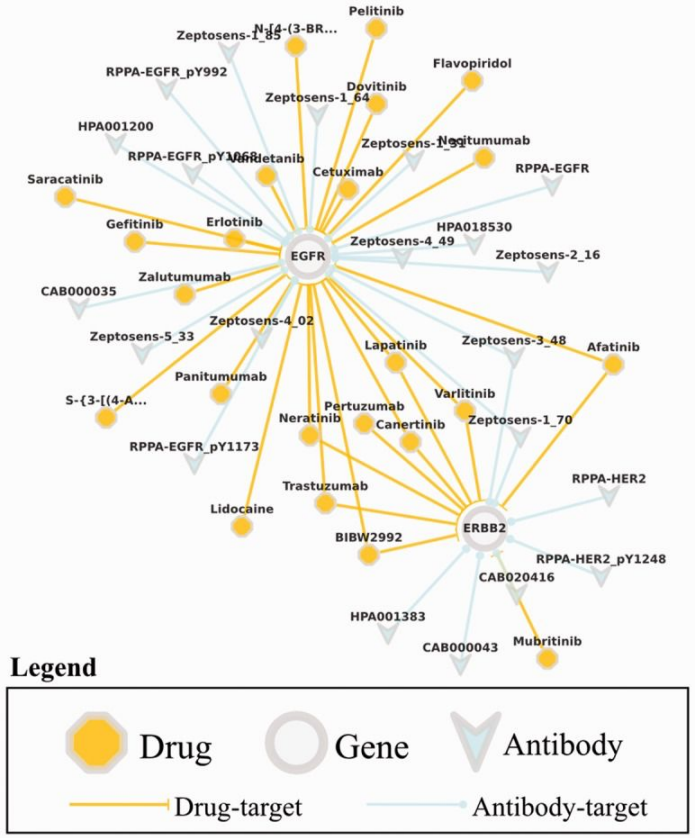


Fig. 6. Multiscale community structures in a graph of social interactions between children in a primary school. The different figures show the partition of the original social network in 2, 5 and 10 communities, respectively. From [199], with permission.

Graphs in Chemistry & Biology



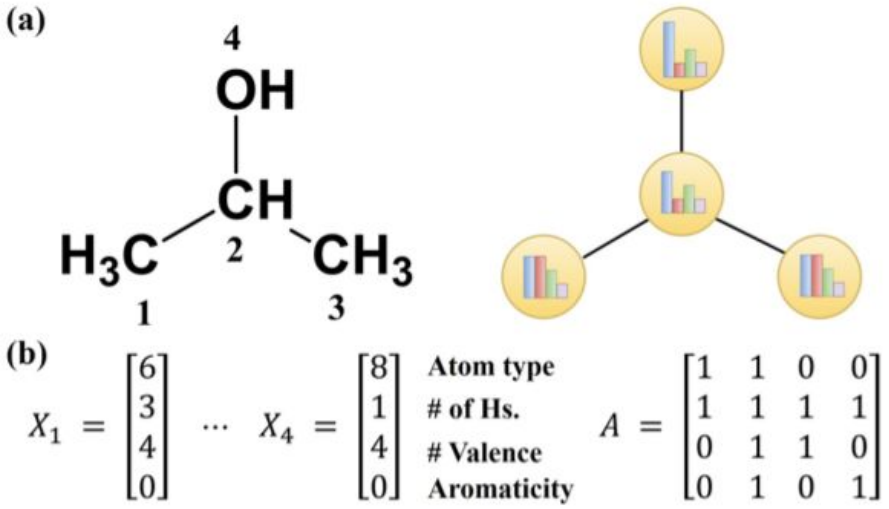


Fig. 1 (a) Graph representation of 2-propanol, $G(A,X)$. The colored columns in each node represent atom descriptors, X_i. (b) The i-th atom descriptors X_i contains initial atom features (atom type, number of hydrogens attached, number of valence electrons, and aromaticity) and the adjacency matrix A represents the connectivity between atom pairs including self-connections.

Additional use cases of Graph Learning

- Use case in Sensor Networks
- Discrete Optimization via Continuous Optimization
- In all described areas graph can be used as format for data augmentation and knowledge transfer

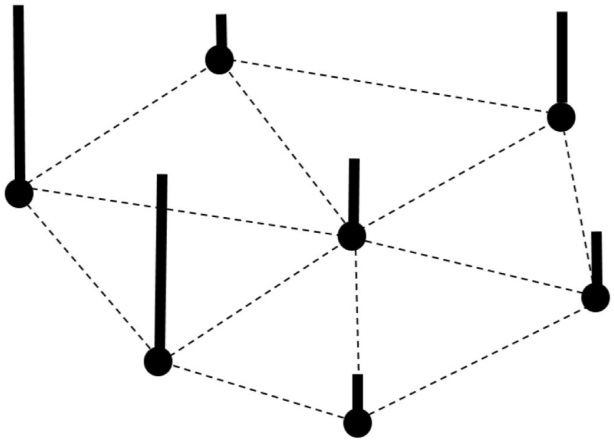
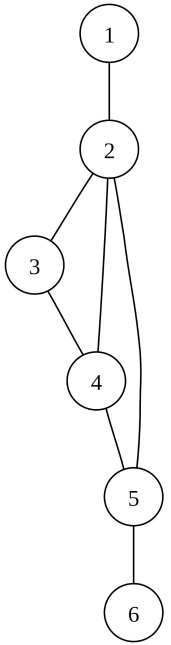


Fig. 1. Signal expanded on exemplary graph.

Graph ML Algorithms

Graph definition and useful properties

- Ordinary graph - G = {V, E}, where V - set of vertices (enumerated objects), E - set of edges (pair of vertices)
- In ML settings researchers work with attributed and highly non-regular graphs
    - Regularity means that degree (#of incident edges) of each vertex is the same
- Classical graph representation:
    - Incidence (vertex - edge) matrix
    - Adjacency (vertex - edge) matrix
- Also graph data is network data, so, it's inherently higher-order, which allow to extract multi-hop features



Classical Graph ML

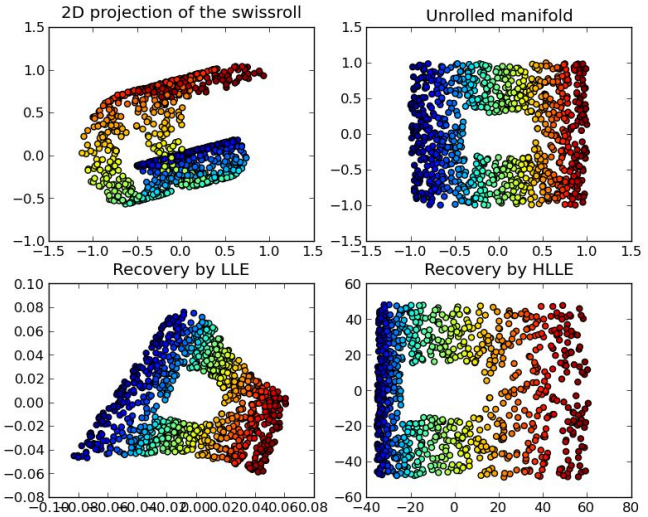
- Graph clustering for unsupervised learning
  - Graph cuts
  - Message passing
  - Chinese Whispers
- Nonlinear dimensionality reduction
  - Locally Linear Embedding (LLE)

$$L(Z) = \|Z - AZ\|_2^2, \text{s.t} \frac{1}{N} Z^T Z = I$$

  - Laplacian eigenmaps:

$$L(Z) = tr(Z^T L Z)$$

- Graph kernels for supervised learning (kernel under string of vertices (=characters/numbers))



Spectral / Spatial Graph Deep Learning

- There are two main approach to formulate Graph Convolution for Deep Learning
  - Spectral-based on eigenvalues of Graph Laplacian

$$\mathbf{L} = \mathbf{I_n} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

$$\mathbf{D}_{ii} = \sum_j (\mathbf{A}_{i,j})$$

  - Spatial-based

Convolutional Graph Neural Networks

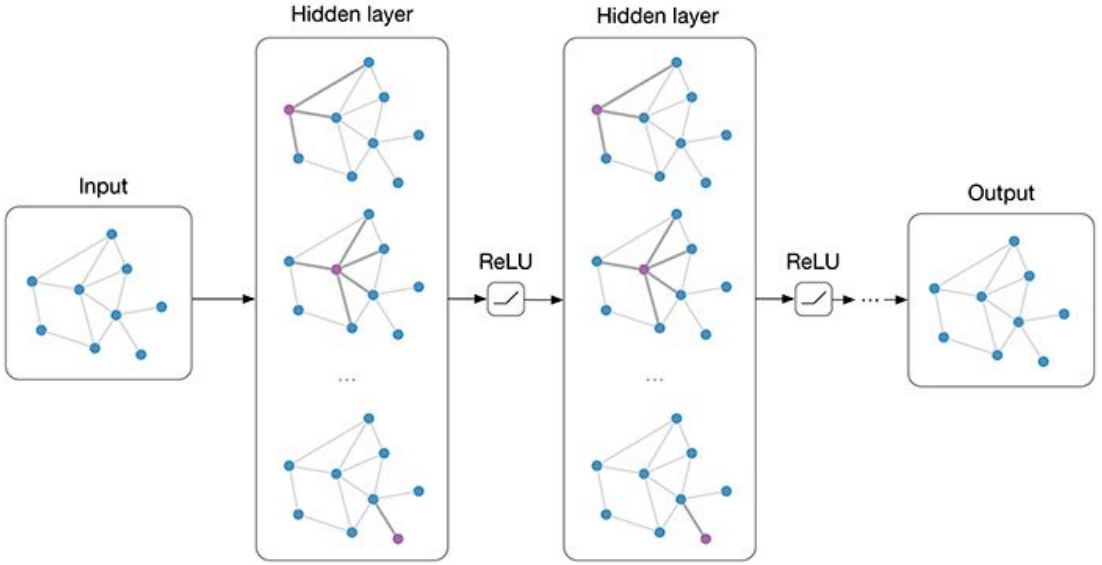
Instead of iterating node states with contractive constraints, ConvGNNs address the cyclic mutual dependencies architecturally using a fixed number of layers with different weights in each layer. ConvGNNs fall into two categories:

- **Spectral-based**. Example, GCN:

$$H = X * \Theta_g = f(\overline{A}X\Theta),$$

- **Spatial-based**. Example, GCN:

$$\mathbf{h}_v = f((\sum_{u \in \{N(v) \cup v\}} \bar{A}_{v,u}\mathbf{x}_u)\mathbf{\Theta} \quad \forall v \in V$$



$$\overline{A} = I_n + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

Graph Representation via Random Walks

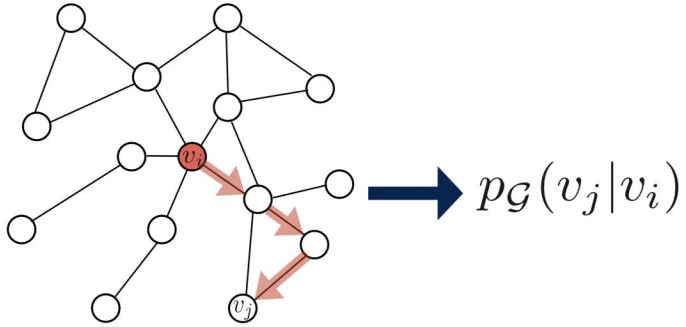
1. Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R.

2. For each node u collect NR(u), the multiset*of nodes visited on random walks starting from u.

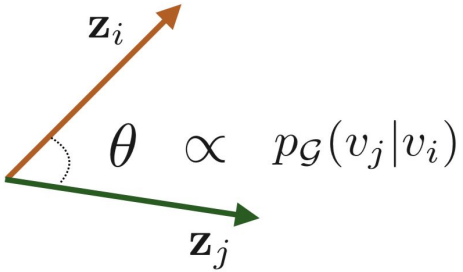
3. Optimize embeddings to encode these random walk statistics.

4. Expressivity: local and high-order neigh info

5. Efficiency: only need to consider pairs that co-occur on random walks



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



2. Optimize embeddings based on co-occurrence statistics.

Neural Graph Representation via RW

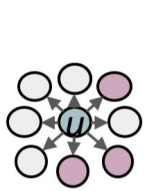
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

formulate via Negative Sampling:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

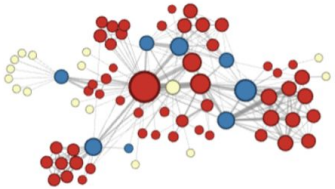
1. DeepWalk (Perozzi et al., 2013)- just fixed-length, unbiased random-walks starting from each node

2. Node2vec (Grover and Leskovec, 2016) used flexible random walks that can trade off between local and global views of the network: p - return back to previous node; q - DFS/BFS;

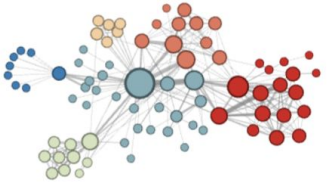


BFS:
Micro-view of neighbourhood

DFS:
Macro-view of neighbourhood

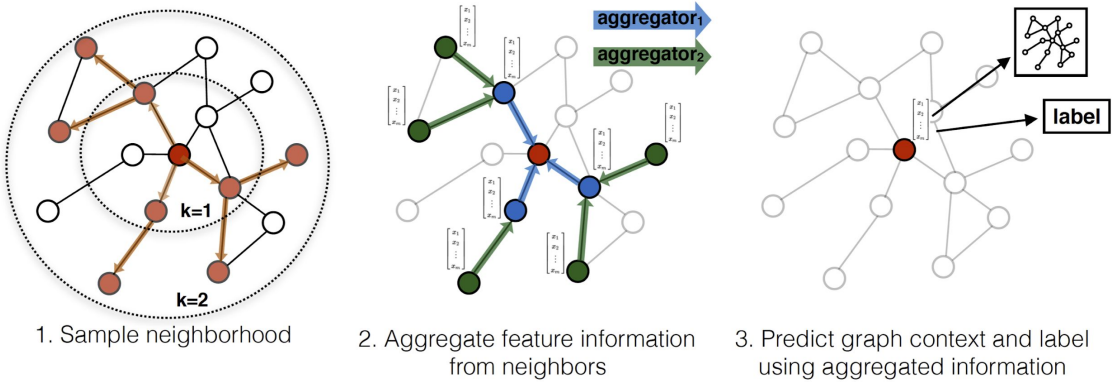


p=1, q=2
Microscopic view of the network neighbourhood



p=1, q=0.5
Macroscopic view of the network neighbourhood

GraphSAGE (SAmple and aggreGatE)



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

GraphSAGE is general graph inductive framework which can represent unseen nodes (vertices).

Based on two general parameters:

- Neighborhood definitions
- Aggregator architecture

GraphSAGE (SAmple and aggreGatE)

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

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**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

GraphSAGE (SAmple and aggreGatE)

- Neighborhood definition: Uniform sample from adjacency vector
- Aggregator architectures ideally should be symmetric (i.e., invariant to permutations of it's inputs):
    - Mean aggregator

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$

    - LSTM aggregator: LSTMs are not inherently symmetric (i.e., they are not permutation invariant), since they process their inputs in a sequential manner. Authors adapt LSTMs to operate on an unordered set by simply applying the LSTMs to a random permutation of the node's neighbors.
    - Pooling aggregator

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\}),$$

Graph Attention Networks (GAT)

GAT also aggregate information of neighborhood nodes like GCN, but adopts attention mechanisms to learn the relative weights between two connected nodes

$$\mathbf{h}_v^{(k)} = \sigma\Big(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu} \mathbf{W}^{(k-1)} \mathbf{h}_u^{(k-1)}\Big)$$

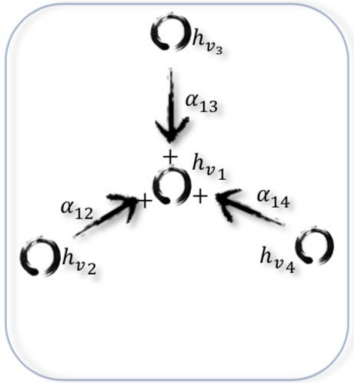
Where

$h_v^{(0)} = x_v$ The attention weight α_{vu}

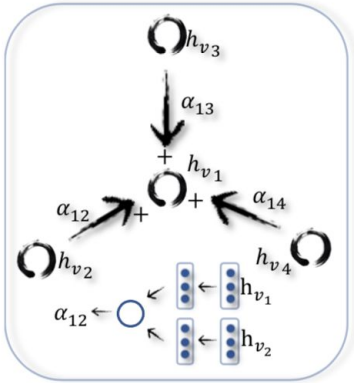
Measures the connection strength from node v to its neighbor u

$$\alpha_{vu} = softmax(g(\mathbf{a}^T[\mathbf{W}^{(k-1)}\mathbf{h}_v || \mathbf{W}^{(k-1)}\mathbf{h}_u]))$$

$g(\cdot)$ is LeakyReLU activation function and **a** is a vector of learnable parameters.



(a) GCN [22] explicitly assign a non-parametric weight $a_{ij} = \frac{1}{\sqrt{deg(v_i)deg(v_j)}}$ to the neighbor v_j of v_i during the aggregation process.

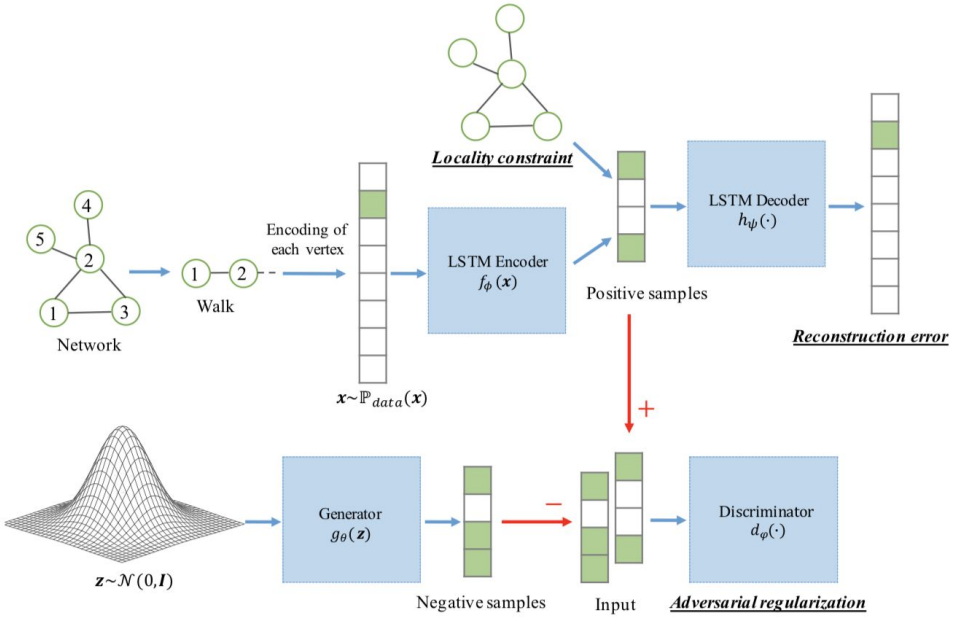


(b) GAT [43] implicitly capture the weight a_{ij} via an end-to-end neural network architecture, so that more important nodes receive larger weights.

Generative Graph Autoencoders

Possible solution of two main problems:

1. Preservation of complex structure property
   both global and local
   a. Using LSTM as complex aggregation function that map rw to vec
   b. Locality-preserving loss and reconstruction loss
2. Sparsity of network sampling
   Sampled data represent only a small proportion of all the vertex sequences
   a. More prior info and more complex regularization via generative adversarial learning



Matrix and Tensor Decomposition Models in Graph Learning

Matrix Factorization Models for Graphs

- Graph can represented via matrix. It can be:
  - Adjacency matrix
  - Incidence matrix
  - Laplacian matrix
  - Similarity matrix between graph elements (nodes/vertices or some more complex structure)

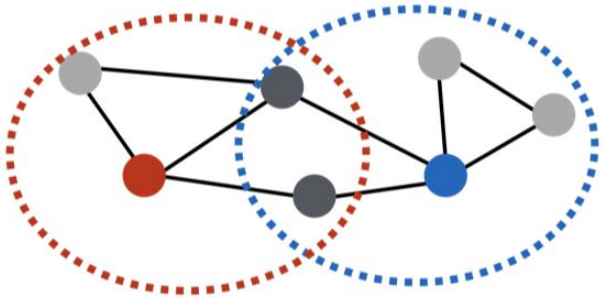
- Let’s factorize this matrix to represent graph elements via low-dimensional vectors. In this case dot products between node embeddings approximate edge existence.

Hope

Idea: Measure overlap between node neighborhoods

$S_{u,v}$ is the neighborhood overlap between u and v

Example: Jaccard overlap or Adamic-Adar score



$$\mathcal{L} = \sum_{(u,v)\in V\times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{S}_{u,v}\|^2$$

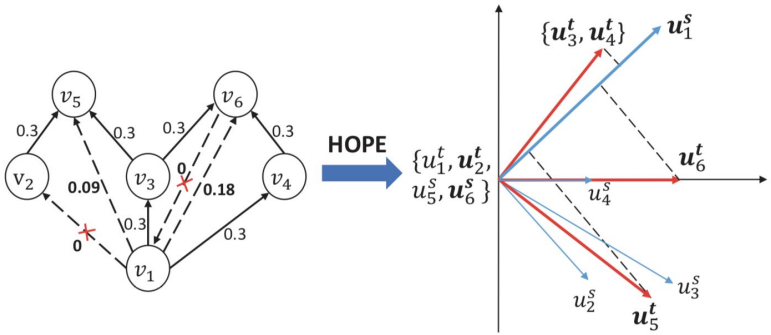
embedding similarity

multi-hop network similarity (i.e., any neighborhood overlap measure)

Hope

Table 1: General Formulation for High-order Proximity Measurements

Proximity Measurement	$\mathbf{M}_g$	$\mathbf{M}_l$
Katz	$\mathbf{I} - \beta \cdot \mathbf{A}$	$\beta \cdot \mathbf{A}$
Personalized Pagerank	$\mathbf{I} - \alpha\mathbf{P}$	$(1-\alpha) \cdot \mathbf{I}$
Common neighbors	$\mathbf{I}$	$\mathbf{A}^2$
Adamic-Adar	$\mathbf{I}$	$\mathbf{A} \cdot \mathbf{D} \cdot \mathbf{A}$



Algorithm 1 High-order Proximity preserved Embedding

Require: adjacency matrix $\mathbf{A}$, embedding dimension K, parameters of high-order proximity measurementθ.

Ensure: embedding source vectors $\mathbf{U}^s$ and target vectors $\mathbf{U}^t$.

1: calculate $\mathbf{M}_g$ and $\mathbf{M}_l$.
2: perform JDGSVD with $\mathbf{M}_g$ and $\mathbf{M}_l$, and obtain the generalized singular values $\{\sigma_1^l, \cdots, \sigma_K^l\}$ and $\{\sigma_1^g, \cdots, \sigma_K^g\}$, and the corresponding singular vectors, $\{\mathbf{v}_1^s, \cdots, \mathbf{v}_K^s\}$ and $\{\mathbf{v}_1^t, \cdots, \mathbf{v}_K^t\}$.
3: calculate singular values $\{\sigma_1, \cdots, \sigma_K\}$ according to Equation (21).
4: calculate embedding matrices $\mathbf{U}^s$ and $\mathbf{U}^t$ according to Equation (19) and (20).

$$\sigma_i = \frac{\sigma_i^l}{\sigma_i^g} \tag{21}$$

$$\mathbf{U}^s = [\sqrt{\sigma_1} \cdot \mathbf{v}_1^s, \cdots, \sqrt{\sigma_K} \cdot \mathbf{v}_K^s] \tag{19}$$

$$\mathbf{U}^t = [\sqrt{\sigma_1} \cdot \mathbf{v}_1^t, \cdots, \sqrt{\sigma_K} \cdot \mathbf{v}_K^t] \tag{20}$$

JDGSVD - Jacobi-Davidson type GSVD

M-NMF

M-NMF - Modularized Nonnegative Matrix Factorization to incorporate community structure into network embedding.

Preserve:

- Microscopic structure:
  - First-order proximities based on adjacency matrix: $S^{(1)} = A$
  - Second-order proximities: Define $S_i = (S_{i,1}^{(1)}, \ldots, S_{i,n}^{(1)})$ then $S_{ij}^{(2)} = \frac{S_i^{(1)} S_j^{(1)}}{\|S_i^{(1)}\| \|S_j^{(1)}\|}$
  - $S = S^{(1)} + \eta S^{(2)}$, where $\eta$is some positive hyperparameter
- Mesoscopic structure
  - Community structure: where k - degree of node, e - #edges, first community indicator

    $Q = \frac{1}{4e} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2e}) h_i h_j$ $h \in \{1, -1\}$

M-NMF

For community structure define following matrix: $B_{ij} = A_{ij} - \frac{k_i k_j}{2e}$

Finally we have following problem:

$$\min_{M,U,H,C} \|S - MU^T\|_F^2 + \alpha\|H - UC^T\|_F^2 - \beta tr(H^T BH)$$

Subject to

$$M \geq 0, U \geq 0, H \geq 0, C \geq 0, tr(H^T H) = n$$

Problem solved via **Multiplicative Updates** (MU) which derived from KKT conditions.

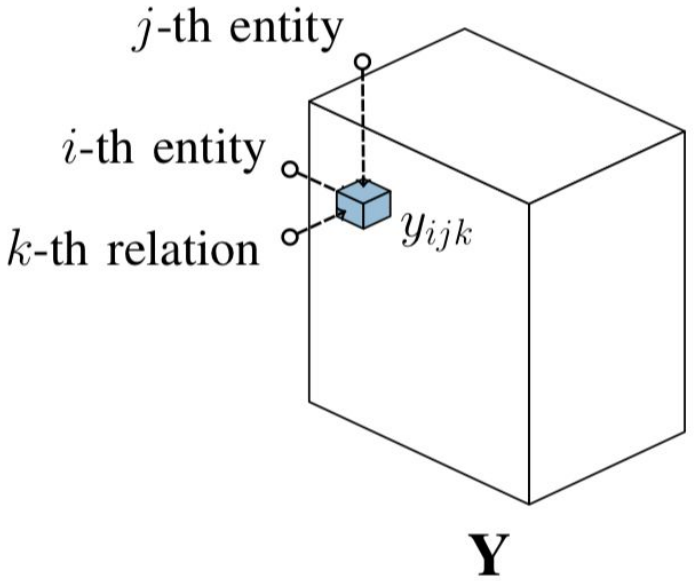
M - basis matrix
U - node representation matrix
H - community indicator
C - community representation matrix

Tensor Decomposition Models for Graphs

- Intuition
    - As we have already seen in M-NMF methods **graphs internally have higher-order structure** which can not capture via simple node similarity matrix
    - Graph Convolution (in neural approaches) work with graph often via **graph coarsening**
- Higher-order structure of graphs
    - Connected components and subgraphs
    - Analyzed via Motifs, GraphLets
- Concrete methods particularly for Knowledge Graph (already 3-order tensor) for general graph needs tensorizing

Knowledge Graphs

- KB/KG is huge amount of facts about the world which can be useful for downstream tasks (example: NLP)
- KG contain relational data via triples of two entities (head and tail) and relation between them: (e_i, r_k, e_j)
  Example: (“Barack Obama”, “place of birth”, “Honolulu”)
- Task is to to predict new facts about the world => which is equivalent to predicting new edges in the graph => predict missing values in associated tensor => Tensor Completion
- Underlying problems: tensor is binary and contain only information of “true” facts.



$$y_{ijk} = \begin{cases} 1, & \text{if the triple } (e_i, r_k, e_j) \text{ exists} \\ 0, & \text{otherwise.} \end{cases}$$

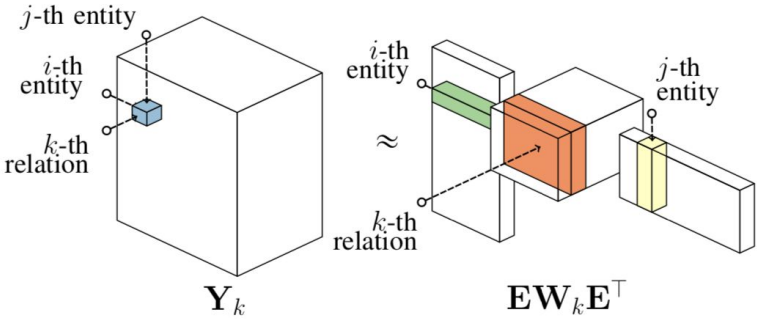
Rescal: a bilinear model

- Use Tucker2 decomposition of adjacency tensors

$$f_{ijk}^{\text{RESCAL}} := \mathbf{e}_i^\top \mathbf{W}_k \mathbf{e}_j = \sum_{a=1}^{H_e} \sum_{b=1}^{H_e} w_{abk} e_{ia} e_{jb}$$

- Problem formulate:
  - In euclidean space (optimized via ALS)

$$\min_{\mathbf{E},\{\mathbf{W}_k\}} \sum_k \|\mathbf{Y}_k - \mathbf{E}\mathbf{W}_k\mathbf{E}^\top\|_F^2 + \lambda_1 \|\mathbf{E}\|_F^2 + \lambda_2 \sum_k \|\mathbf{W}_k\|_F^2$$



  - In logit space (optimized via SGD)

$$\max_{\Theta} \sum_{n=1}^{N_d} \log \text{Ber}(y^n \mid \sigma(f(x^n; \Theta))) + \log p(\Theta \mid \lambda)$$

Other models for KG

- Tatec model $f_{ijk}^{TATEC} = e_i^T R_k e_j + e_i^T r + e_j^T r + e_i^T D e_j$
- DistMult: $f_{ijk}^{DISTMULT} = \mathrm{diag}(e_i)\mathrm{diag}(r_k)\mathrm{diag}(e_j)$
- HolE: $f_{ijk}^{HOLE} = r_k^T (e_i * e_j)$
- ComplEx: $f_{ijk}^{COMPLEX} = \mathrm{Re}(\mathrm{diag}(e_i)\mathrm{diag}(r)\mathrm{diag}(e_j))$
- Also used loss function based on hinge loss with Corrupted Triples:

$$\sum_{(h,r,t)\in Y} \sum_{(h',r',t')\in C} \max(0, 1 - f_{h,r,t} + f_{h',r',t'})$$

Other graph embedding methods

- Also existed some additional methods based on:
  - Hyperbolic space as part of new ML field called Hyperbolic Deep Learning
  - Based on graph Dihedral Group (Relation Embedding with Dihedral Group in Knowledge Graph)

Graph Learning Resources

- Blogs of companies, organizations and already famous researchers which interesting in Graphs:
  - Octavian (https://www.octavian.ai)
  - Deepmind (https://arxiv.org/abs/1806.01261)
  - Thomas Kipf (https://tkipf.github.io)
  - Maximilian Nickel (https://mnick.github.io/)

- Concrete ML paradigm (corresponding to graphs):
  - Geometric deep learning
    - Famous researcher: Michael Bronstein
    - GDL Site: (http://geometricdeeplearning.com)
  - Hyperbolic deep learning
    - Famous researchers: Maximilian Nickel
    - HDL Site: (http://hyperbolicdeeplearning.com)

- Paper collection:
  - https://github.com/DeepGraphLearning/LiteratureDL4Graph
  - THUNLP lab at Tsinghua University (https://github.com/thunlp)

Programming Toolboxes

- Python Frameworks:
  - PyTorch Geometric: (https://pytorch-geometric.readthedocs.io/en/latest/)
  - Deep Graph Library (https://www.dgl.ai)
  - Deepmind Sonnet: (https://sonnet.dev)
  - PyTorch-BIGGRAPH (https://torchbiggraph.readthedocs.io/en/latest/)
  - GraphVite Graph Embedding Library (https://graphvite.io)
  - GEM Graph Embedding Libraries (https://github.com/palash1992/GEM)
  - TorchKGE KG Embedding Library (https://torchkge.readthedocs.io)
  - AmpliGraph KG Embedding Library (https://docs.ampligraph.org/en/1.1.0/install.html)
  - Pykg2vec Embedding Library (https://github.com/Sujit-O/pykg2vec)

- General Graph Analysis:
  - NetworkX (https://networkx.github.io)
  - Graph-tool (https://graph-tool.skewed.de)

- Program for graph visualization:
  - Gephi (https://gephi.org)

Thank you for your attention!