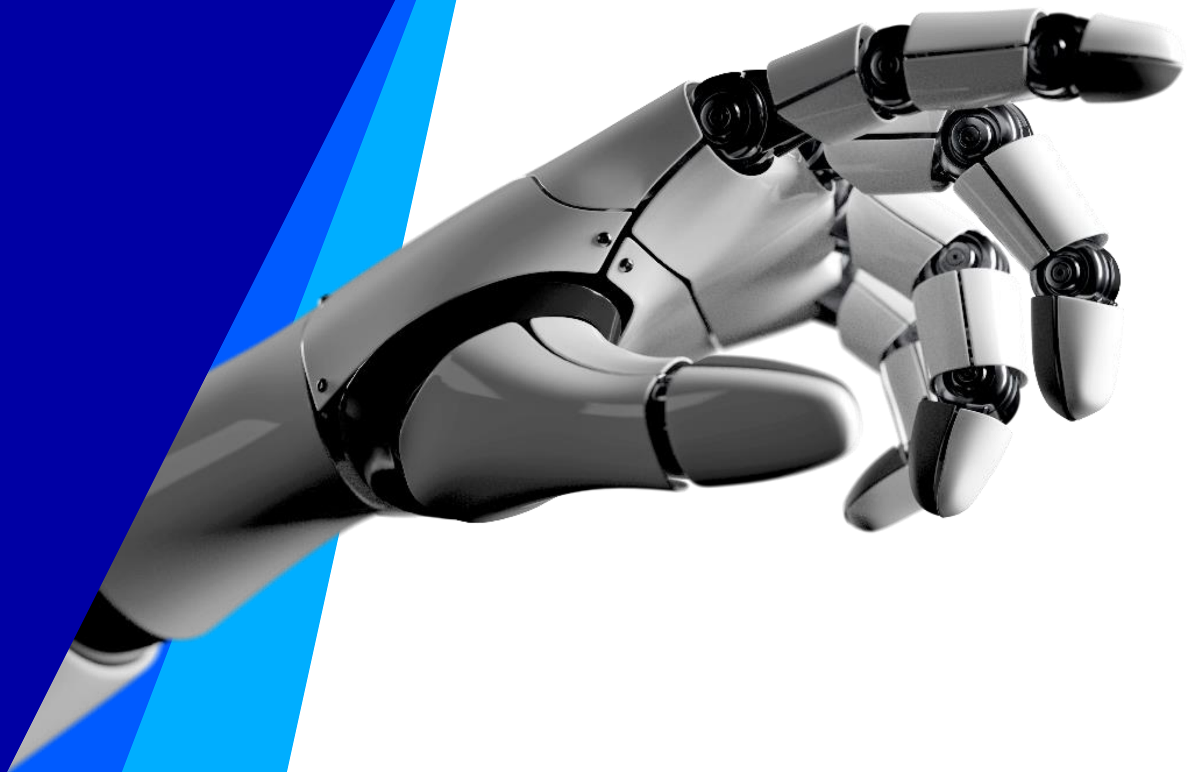
fi group

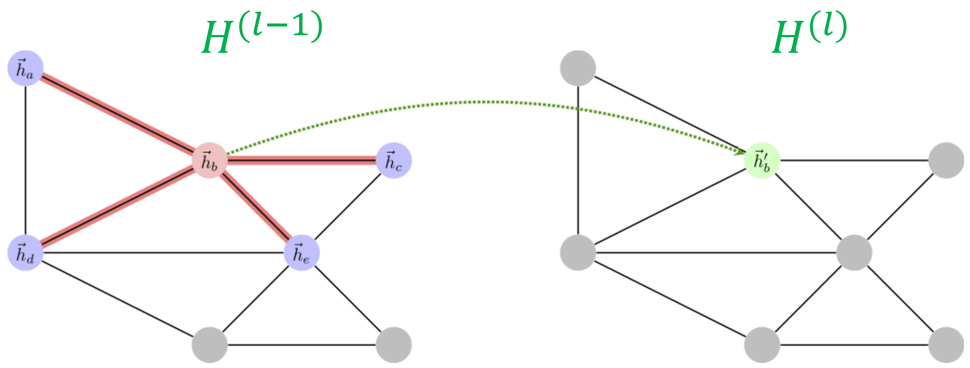


Session 15

PERFORMANCE
EN FINANCEMENT
DE L'INNOVATION

GNN layers

- $\tilde{A} = A + I$ add self-loop
- Sum pooling $H^{(l)} = \sigma(\tilde{A}H^{(l-1)}W^{(l)})$
- Mean pooling $H^{(l)} = \sigma(\tilde{D}^{-1}\tilde{A}H^{(l-1)}W^{(l)})$
- GCN $H^{(l)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l-1)}W^{(l)})$
- GAT $H^{(l)} = \sigma(\alpha H^{(l-1)}W^{(l)})$ where $\alpha_{ij} = \frac{\exp(f_{att}(h_i,h_j))}{\sum_{k\in N_i}\exp(f_{att}(h_i,h_k))}$
- GraphSage $h_{N_i}^{(l)} \leftarrow agg^{(l)}\left(\left\{h_j^{(l-1)}, \forall j \in N_i\right\}\right)$ Neigborhood aggregation (Sum, Mean, MAX, LSTM)
  $h_i^{(l)} \leftarrow \sigma(W^{(l)}\ concat\left(h_i^{(l-1)}, h_{N_i}^{(l)}\right))$ Node embedding update



On pytorch geometric

```
from torch_geometric.nn import GCNConv, GATConv, SAGEConv

gcn = GCNConv(in_channels, out_channels) # graph conv
gat = GCNConv(in_channels, out_channels) # graph attention
sage = SAGEConv(in_channels, out_channels, aggr='max') # graphSage
```

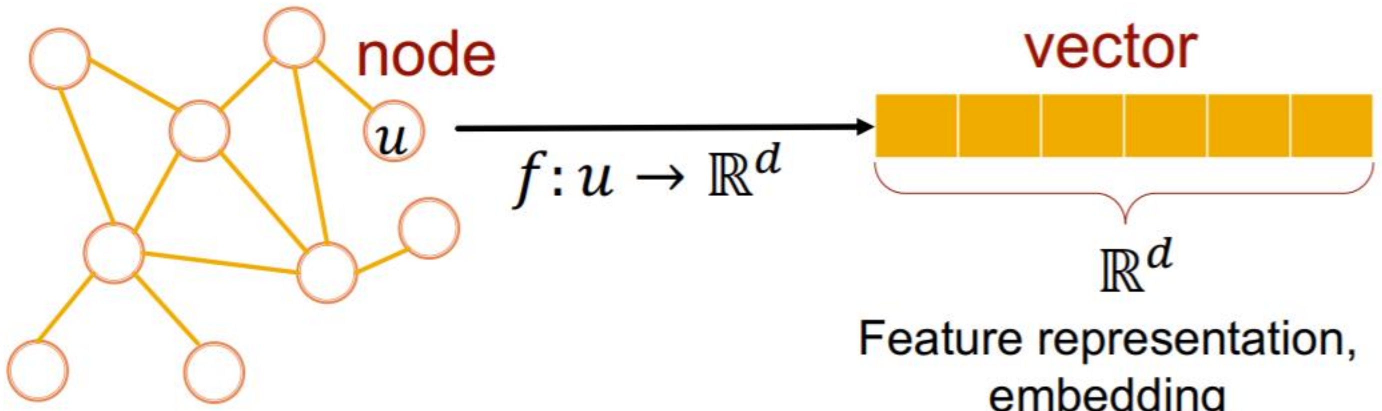
Node embeddings

Some slides adapted from Jure Leskovec's lecture,2021
http://web.stanford.edu/class/cs224w/slides/03-nodeemb.pdf

Motivation

Task: map nodes into an embedding space

- Similarity of embeddings between nodes indicates their similarity in the graph
- Encode graph information
- Use for many downstream tasks: Node classification, Link prediction, Visualization/clustering



Shallow encoding

- Encoder-decoder approach
  - Encoder is an embedding lookup (pytorch encoder = nn.Embedding(num_nodes, emb_dim))
  - Decoder maps from embeddings to similarity scores (ex: dot product)
    - z_u and z_v are respectively the embedding for node u and v
    - $decoder(z_u, z_v) = z_u^T z_v$ (In practice $decoder(z_u, z_v) = \sigma(z_u^T z_v)$ to keep in the range 0 and 1)
  - Goal: optimize encoder s.t $decoder(z_u, z_v) \approx sim(u, v)$
  - How to define similarity ?
    - Use adjacency matrix
      - $sim(u, v) = A_{uv}$ (or $sim(u, v) = 1$ if $(u, v) \in E$, 0 otherwise)
    - Instead of simply using the direct neighborhood, use k-hop neighborhood (use A^k instead of A)
    - Random Walk to define similarity (DeepWalk, Node2Vec)

Pytorch session 1

- Shallow encoder with pytorch
- Shallow encoder with pytorch geometric

Limitation of shallow embeddings

- No parameter sharing between nodes in the encoder, since the encoder directly optimizes a unique embedding vector for each node.
- Do not leverage node features in the encoder.
- Shallow embedding methods are inherently "transductive" (do not work for unseen nodes)
- Solution: Use GNN layers as encoder

William Hamilton Book, Graph Representation Learning

Graph Auto-encoder with GCNs

- Same as before but use a GCN layer as encoder (GAT, GraphSage can work aswell)
- Dot-product as decoder
- Objective·
  - reconstruct the adjacency matrix i.e optimize node embeddings such that $decoder(z_u, z_v) \approx A_{uv}$
- "A randomly initialized graph convolutional network may already extract highly useful features and represents a strong baseline" => strong inductive bias (Velickovic et al., 2018)
- Can also use Variational auto-encoder to improve performance to downstream tasks

Method	Cora AUC	Cora AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00
DW [6]	83.1 ± 0.01	85.0 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01
VGAE*	84.0 ± 0.02	87.7 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$

Kipf, Thomas and M. Welling. "Variational Graph Auto-Encoders." ArXiv abs/1611.07308 (2016)

Pytorch session 2

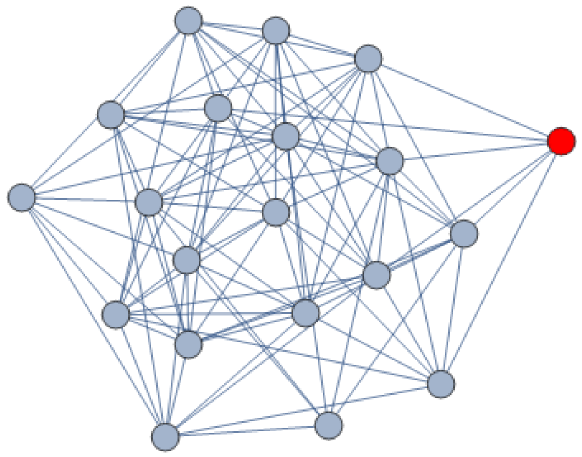
- Graph Autoencoder
- Graph Variational Autoencoder

Random walk

- Given a starting node u in a graph G, we randomly walk to one of its neighbors. We repeat this process from the node until T (walk length) nodes are visited.

Next node probability $\mathrm{p(v|u)} = \begin{cases} \frac{1}{d(u)} \text{ if } v \in N(u) \\ 0 \text{ otherwise} \end{cases}$

- Random walk generation· $RandomWalk\left(G, u^{(0)}, T\right) = \left(u^{(0)}, \dots, u^{(T-1)}\right)$



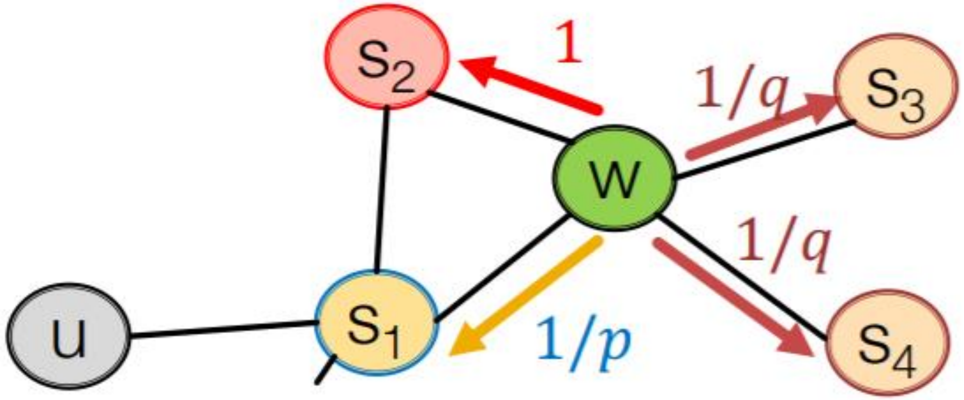
DeepWalk

- Idea: Nodes are considered similar to each other if they tend to co-occur in random walks.
  - Generate γ random walks of length T per nodes
  - Train skipgram + Negative samplings
    - Use generated walks as positive walks
    - Generate negative walks by random sampling
  - Loss function for one walk
    - $\mathcal{L}(u, P, N) = -\sum_{v \in P} \log\left(\sigma({z_u}^T z_v)\right) - \sum_{w \in N} \log(1 - \sigma({z_u}^T z_w))$
    - Intuition: maximize score of positive pairs while minimizing score of negative pairs (exactly the same of optimization as Skipgram-Negative sampling)

Node2Vec

- Same training as DeepWalk but use a “biased” random walk

Walker came over edge $(S_1, W)$and is at W.
Where to go next?



W →

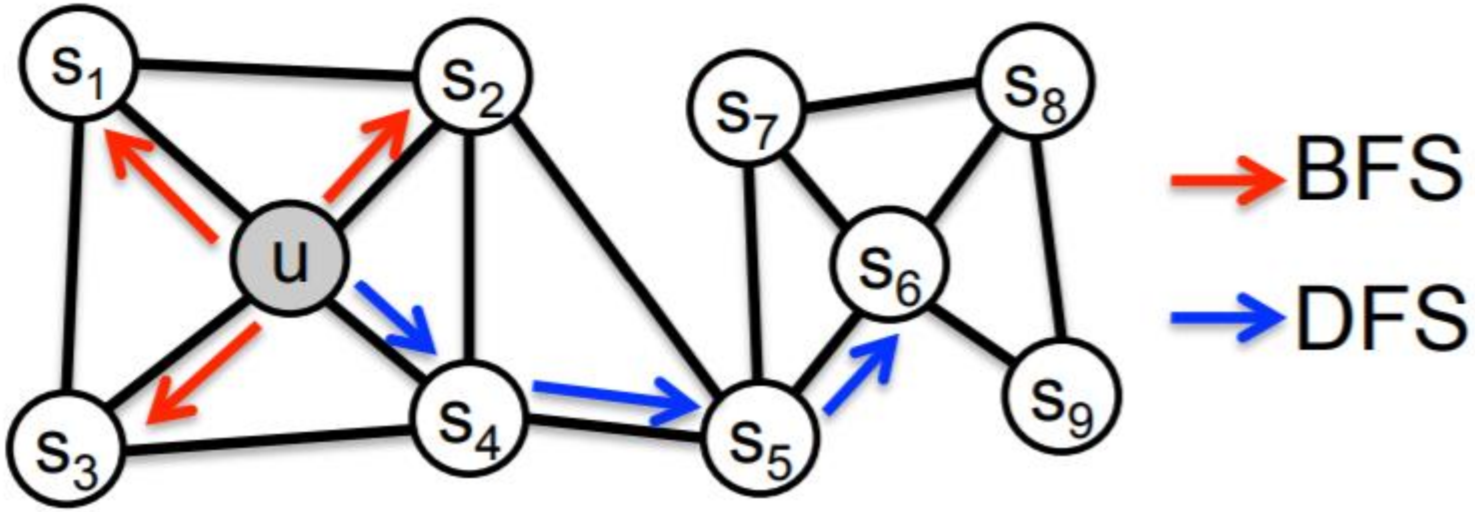
Target t	Prob.	Dist. (s_1, t)
S_1	$1/p$	0
S_2	1	1
S_3	$1/q$	2
S_4	$1/q$	2

Unnormalized transition prob. segmented based on distance from S_1

- BFS-like walk: Low value of p (“return” parameter)
- DFS-like walk: Low value of q (“walk away” parameter)

Grover & Leskovec. “node2vec: Scalable Feature Learning for Networks.” Proceedings of the 22nd ACM SIGKDD 2016

Node2Vec



- BFS-like walk: Low value of p (“return” parameter)
- DFS-like walk: Low value of q (“walk away” parameter)

Pytorch session 3

- DeepWalk and Node2Vec

Algorithm	Dataset		
	BlogCatalog	PPI	Wikipedia
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p,q)	0.25, 0.25	4, 1	4, 0.5
Gain of *node2vec* [%]	**22.3**	**1.3**	**21.8**

Table 2: Macro-F_1 scores for multilabel classification on BlogCatalog, PPI (Homo sapiens) and Wikipedia word cooccurrence networks with 50% of the nodes labeled for training.

How to use embedding z_u of nodes ?

- Clustering/community detection: cluster points z_u (ex: Kmeans, ...)
- Node classification: predict label of node i based on z_u
- Link prediction: Predict edge (u, v) based on (z_u, z_v)
  - Concat: $f(z_u, z_v) = g([z_u, z_v])$
  - Element-wise (Hadamard) product: $f(z_u, z_v) = g(z_u \odot z_v)$ (best result in Node2vec paper)
  - Sum/Mean: $f(z_u, z_v) = g(z_u + z_v)$
  - Distance: $f(z_u, z_v) = g\left(\|z_u - z_v\|_2\right)$
- Graph classification: compute graph embedding z_G via aggregating node embeddings (Sum, AVG, MAX). Use z_G to predict label (ex: predict with MLP)

Complex graphs

Complex graphs

- Heterogeneous graph: each node and each edge are associated with a type

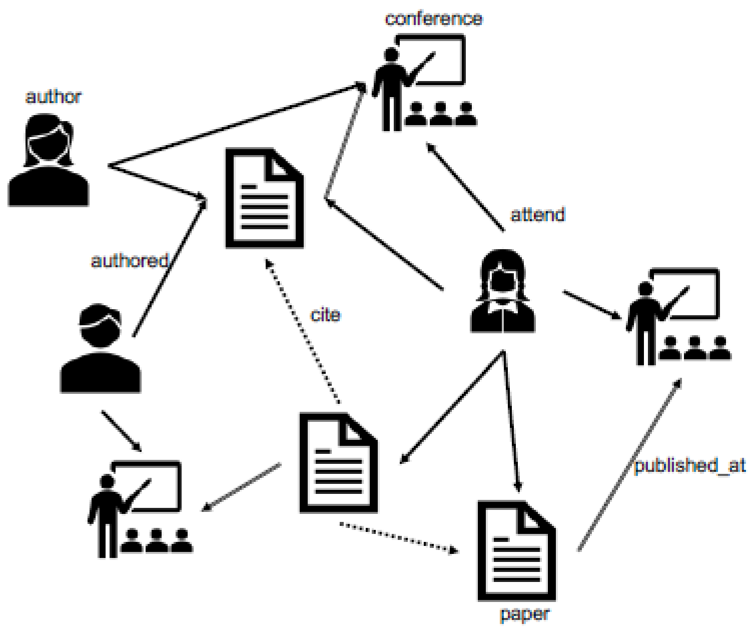


Figure 2.6 A heterogeneous academic graph

Graph Deep leaning book, Ma & Tang chapiter 2.6

Complex graphs

- Bipartite graph: its nodes V can be divided in two disjoint subsets V_1 and V_2 where every edges in E connects a node in V_1 and a node in V_2

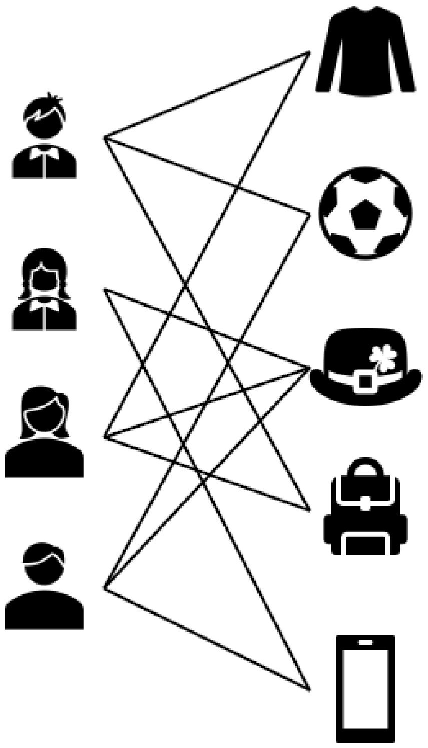


Figure 2.7 An e-commerce bipartite graph

Definition 2.36 (Bipartite Graph) Given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, it is bipartite if and only if $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ and $v_e^1 \in \mathcal{V}_1$ while $v_e^2 \in \mathcal{V}_2$ for all $e = (v_e^1, v_e^2) \in \mathcal{E}$.

Graph Deep leaning book, Ma & Tang chapter 2.6

Complex graphs

- Signed graph: contain both positive and negative edges

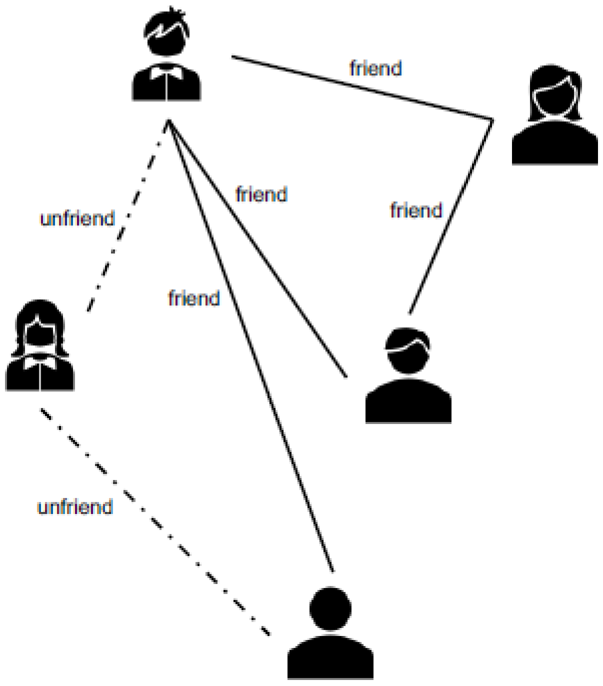
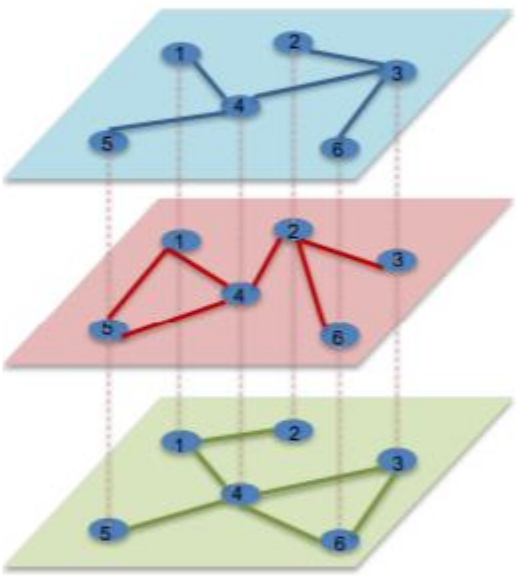


Figure 2.8 An illustrative signed graph

Graph Deep leaning book, Ma & Tang chapiter 2.6

Complex graphs

- Multi-dimensional graph: many edge types
  - A Multi-dimensional graph can be represented by D adjacency matrix $A^{(1)} \dots A^{(D)}$ where each $A^{(d)} \in \mathbb{R}^{N \times N}$



(c) Multi-dimensional graph

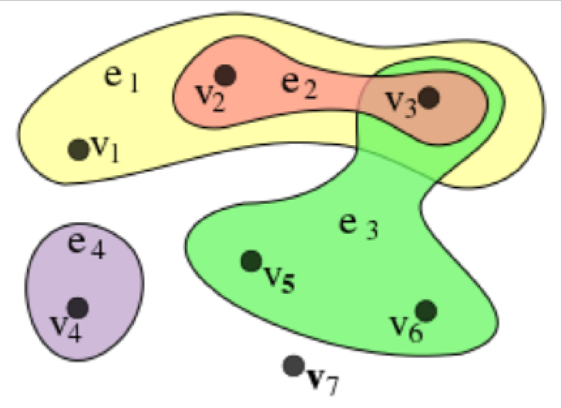
Multi-dimensional Graph Convolutional Networks, Ma et al. 2018

Complex graphs

- Dynamic graph: each node and/or each edge is associated with a timestamp indicating the time it emerged => two mapping functions ϕ_v and ϕ_e mapping each node and each edge to their emerging timestamps.

- Discrete Dynamic graph: consists of T graph snapshots $\{G_1, \dots, G_T\}$

- Hypergraph:

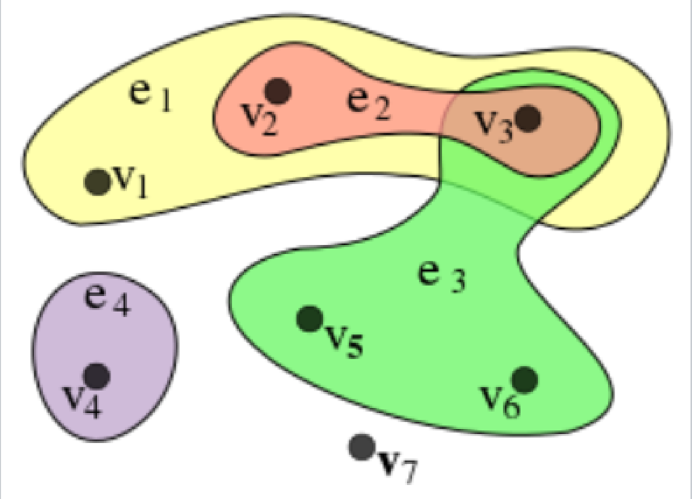


An example of an undirected hypergraph, with $X = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ and $E = \{e_1, e_2, e_3, e_4\} = \{\{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_5, v_6\}, \{v_4\}\}$. This hypergraph has order 7 and size 4. Here, edges do not just connect two vertices but several, and are represented by colors.

Graph Deep leaning book, Ma & Tang chapter 2.6

Complex graphs

- Hypergraph: a graph in which an edge can join any number of nodes.

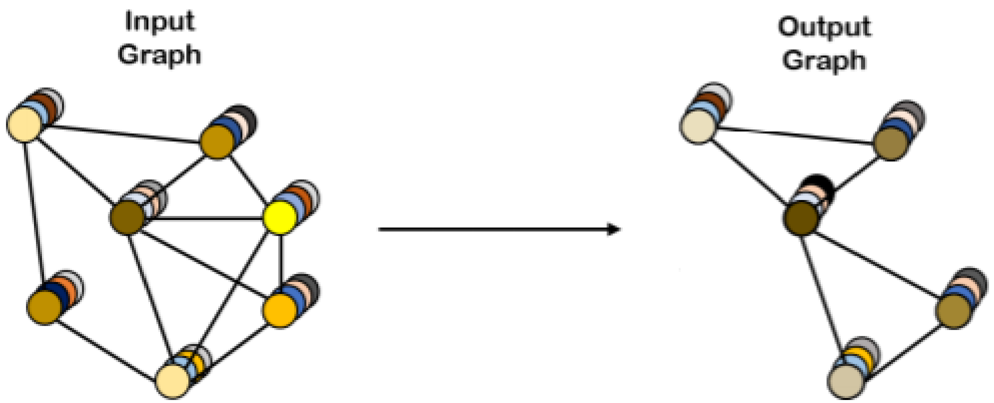


An example of an undirected hypergraph, with $X = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ and $E = \{e_1, e_2, e_3, e_4\} = \{\{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_5, v_6\}, \{v_4\}\}$. This hypergraph has order 7 and size 4. Here, edges do not just connect two vertices but several, and are represented by colors.

https://en.wikipedia.org/wiki/Hypergraph

Next session

- Graph-level classification
  - Mini-batching
  - Global pooling (Global average pooling, Global max pooling)
  - Hierarchical pooling:
    - diffPool https://arxiv.org/abs/1806.08804
    - topkPool https://openreview.net/forum?id=HJePRoAct7
    - sagPool https://arxiv.org/abs/1904.08082



Graph ML useful links

- Graph Representation Learning, William Hamilton 2020 https://www.cs.mcgill.ca/~wlh/grl_book/
- Deep Learning on Graphs, Ma & Tang 2021 https://cse.msu.edu/~mayao4/dlg_book/
- cs224W stanford lecture 2020 and 2021,
- Michael Bronstein Blogposts
- Michael Bronstein keynote @ ICLR 2021 « Geometric Deep Learning »
- Pytorch Geometric tutorial series, Antonio Longa et al., 2021 on Youtube
- Graph Ml workshops: GRL@ICML, GNNSys21@MLSys, DiffGeo4DL@NIPS, GNN&Beyond@ICML