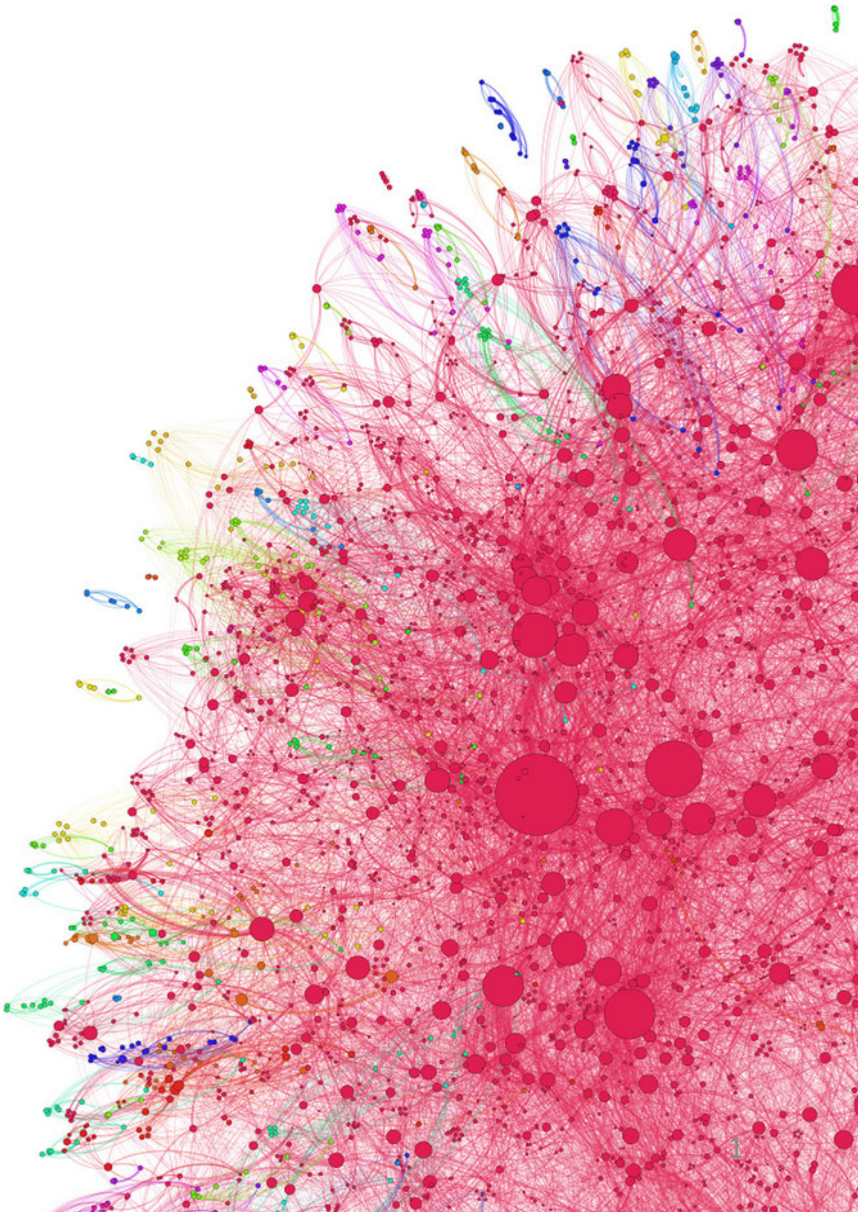
Inductive Representation Learning on Large Graphs

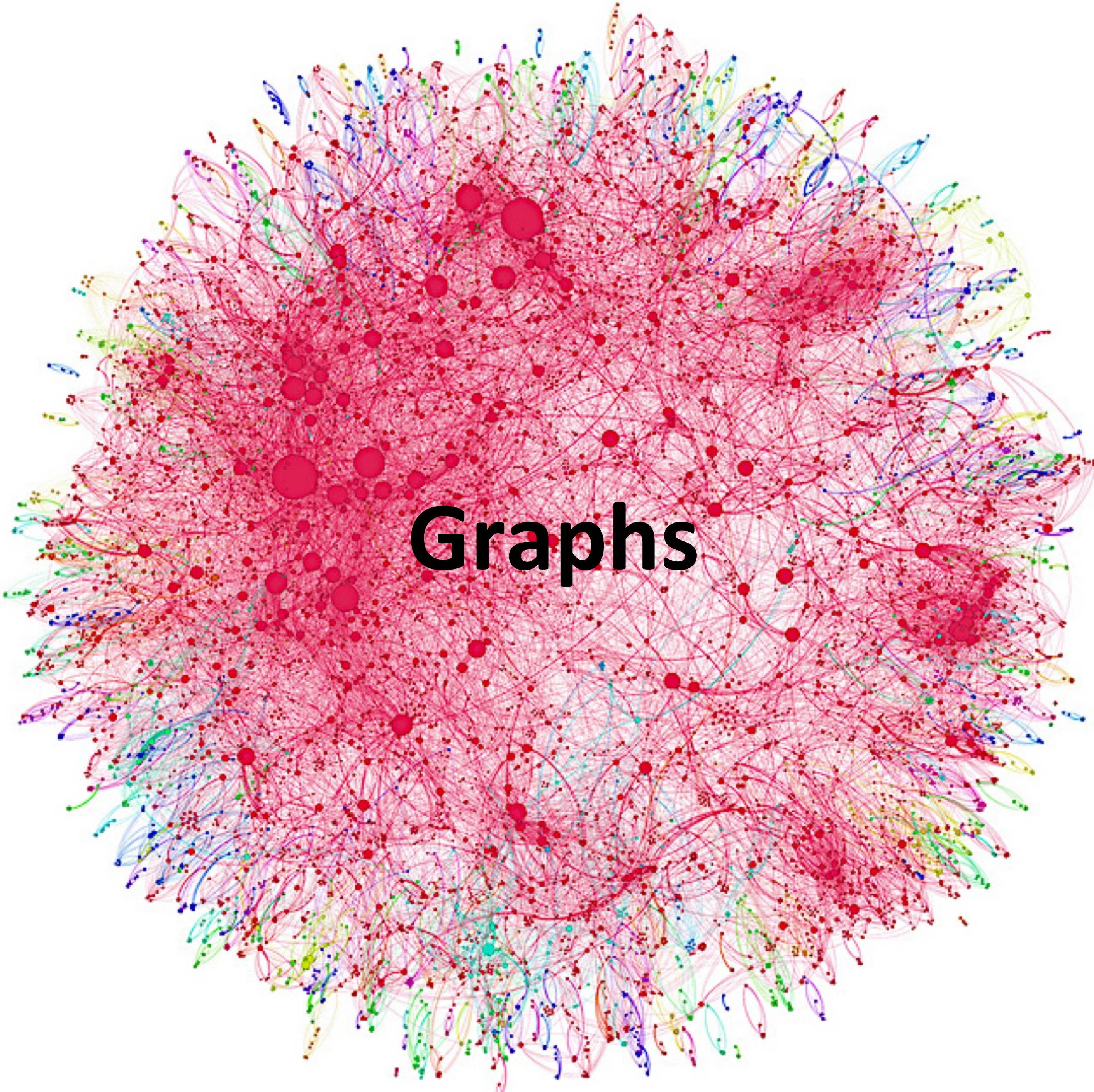


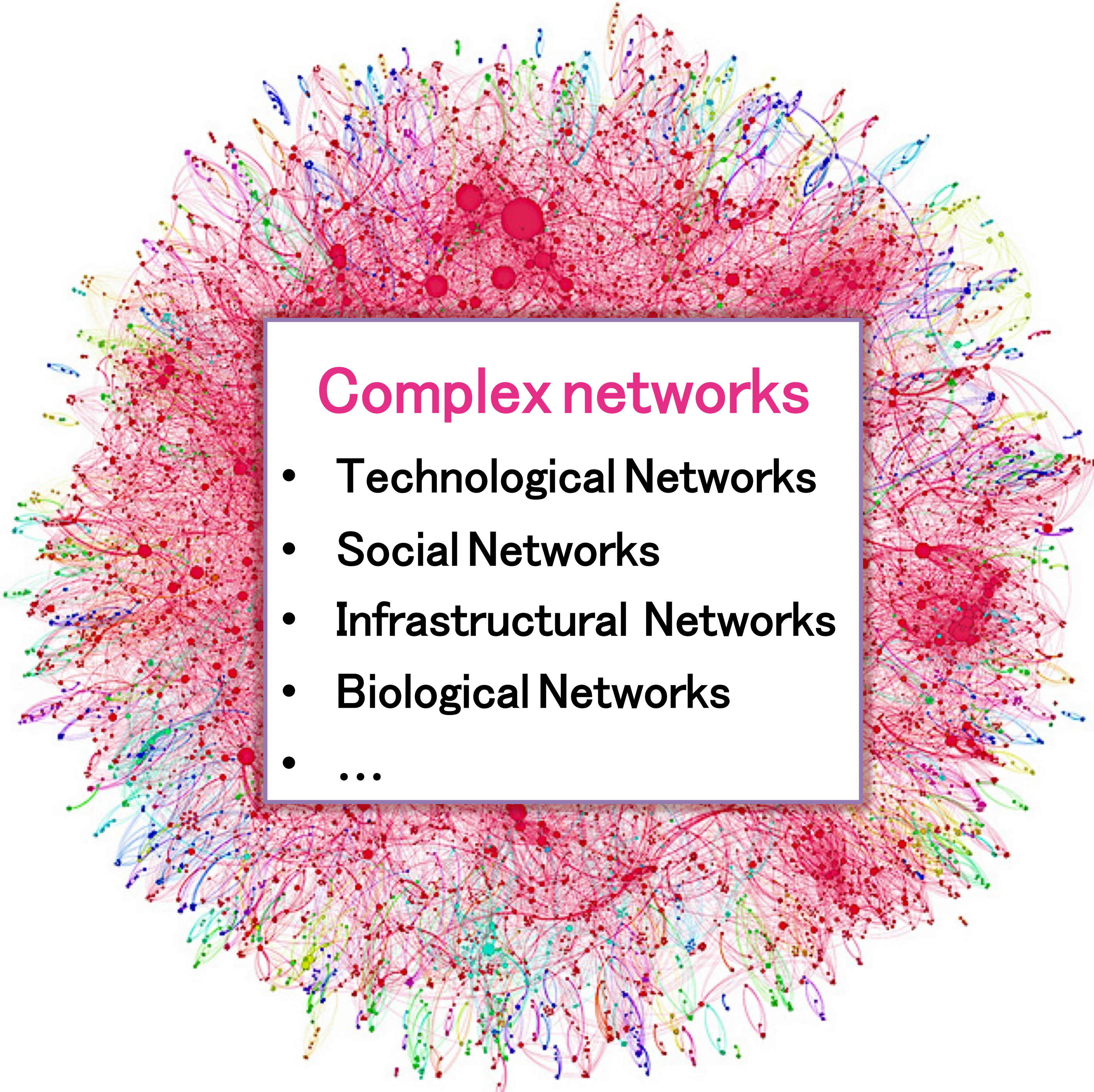
By William Hamilton, Rex Ying, and Jure Leskovec – NIPS'17

Presented by **Aida Sheshbolouki**
CS-848 Winter'19

UNIVERSITY OF WATERLOO

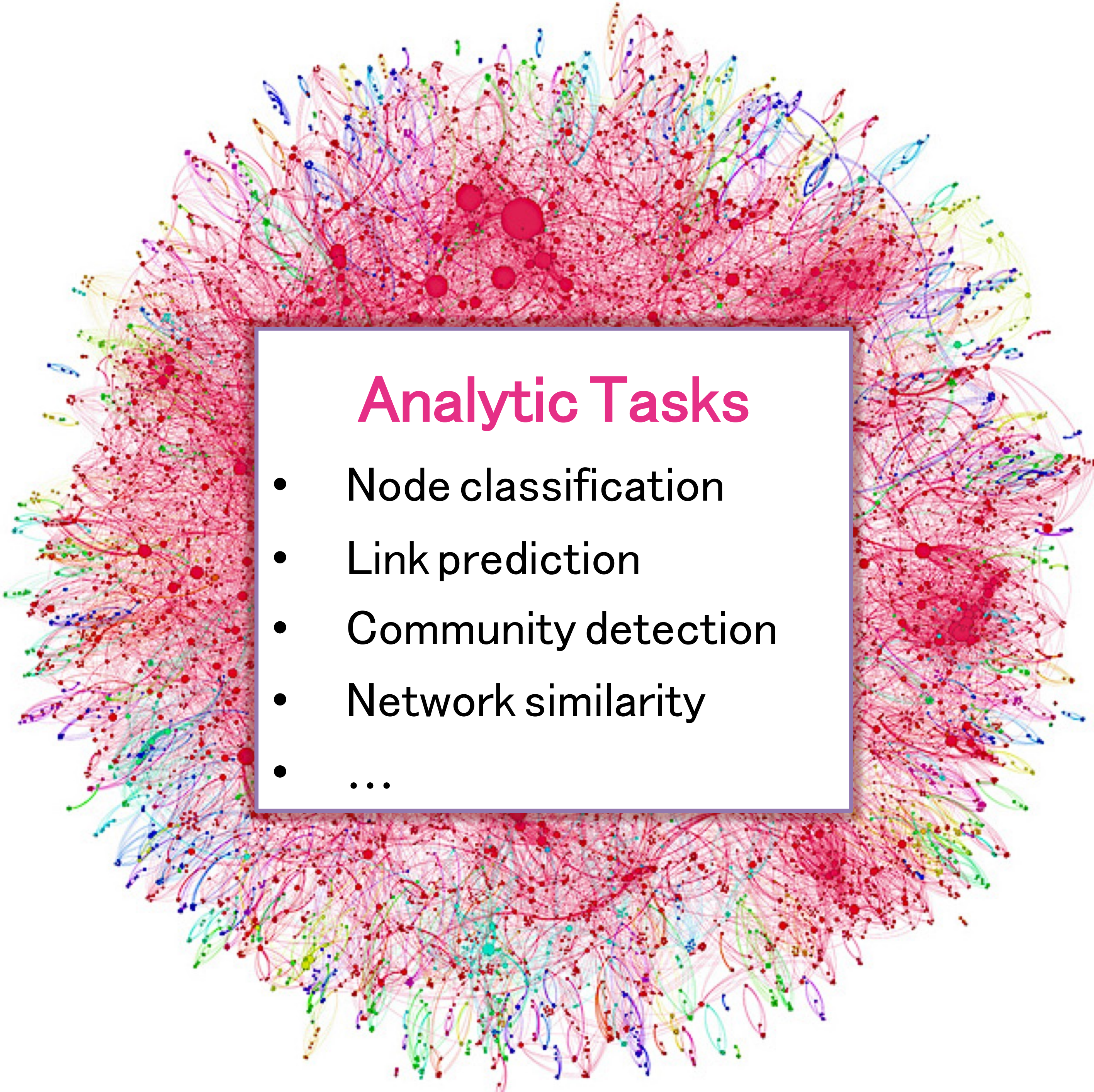
Graphs





Complex networks

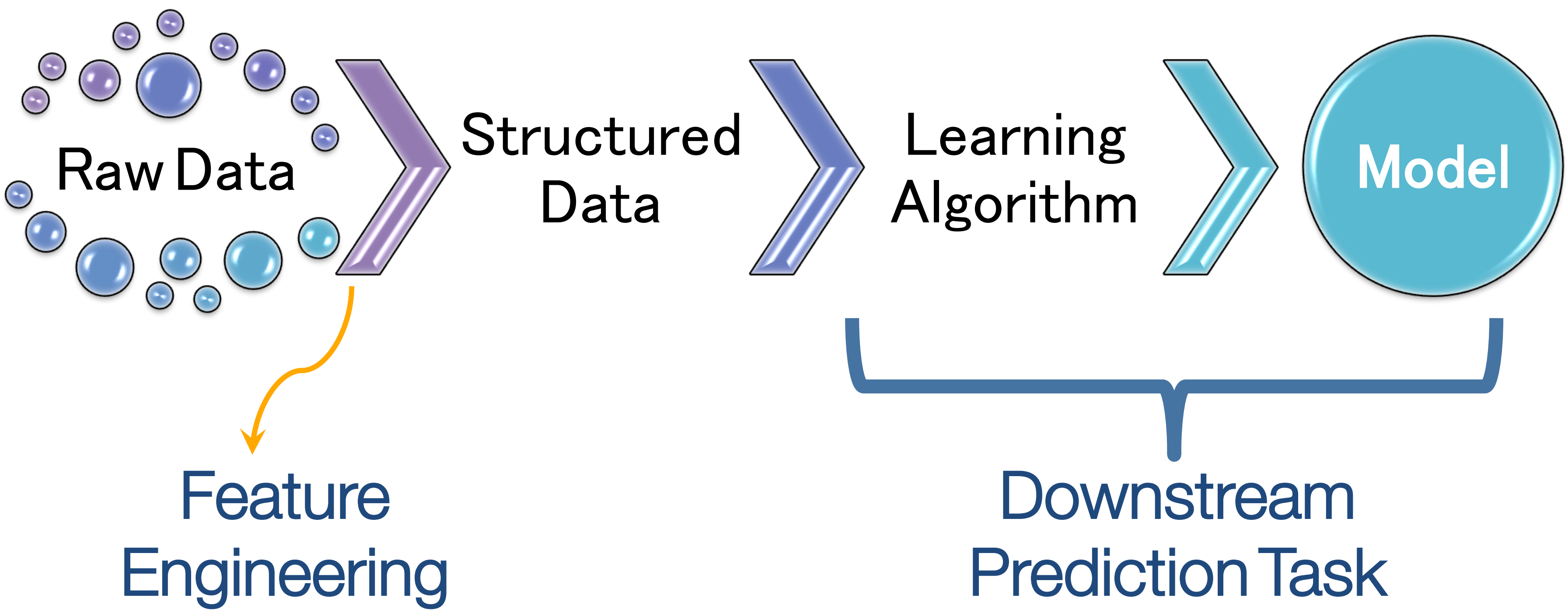
- Technological Networks
- Social Networks
- Infrastructural Networks
- Biological Networks
- …



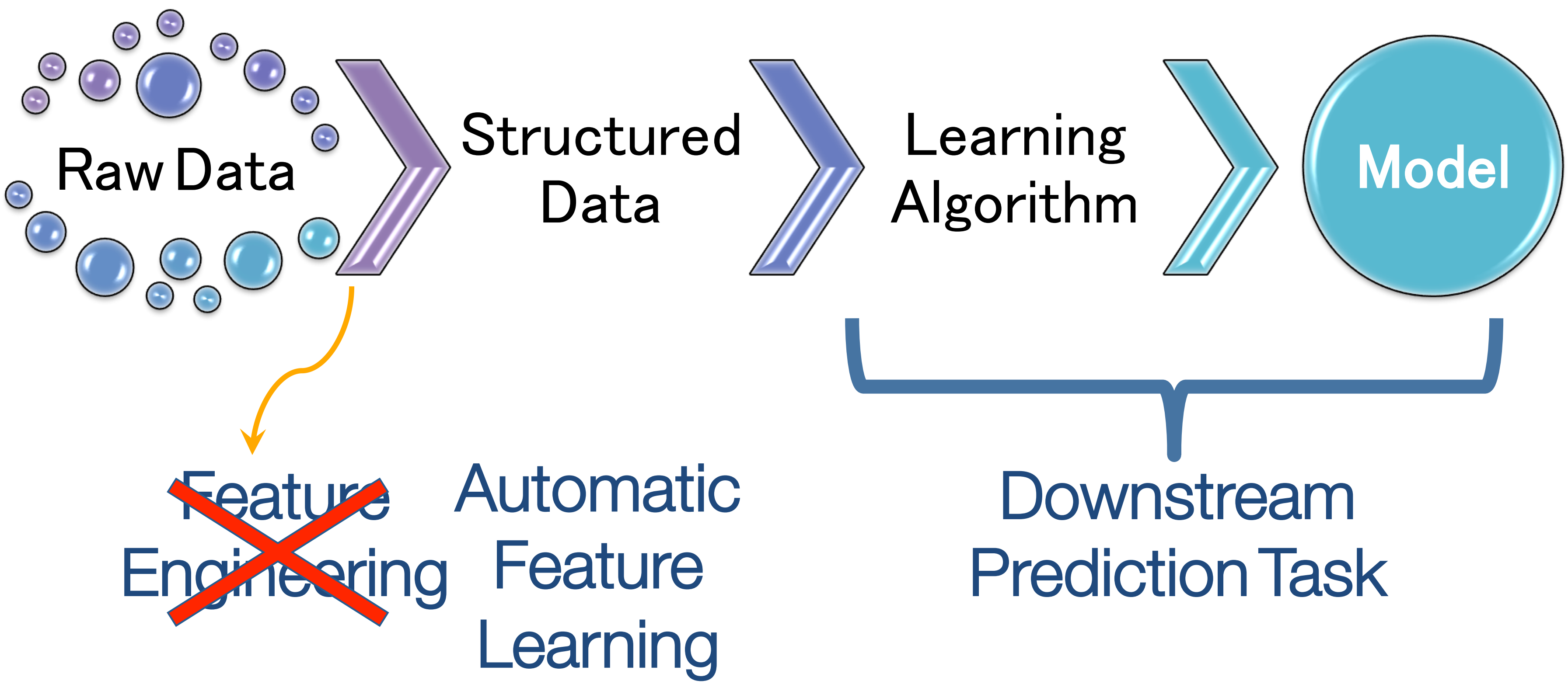
Analytic Tasks

- Node classification
- Link prediction
- Community detection
- Network similarity
- …

Machine Learning Life Cycle

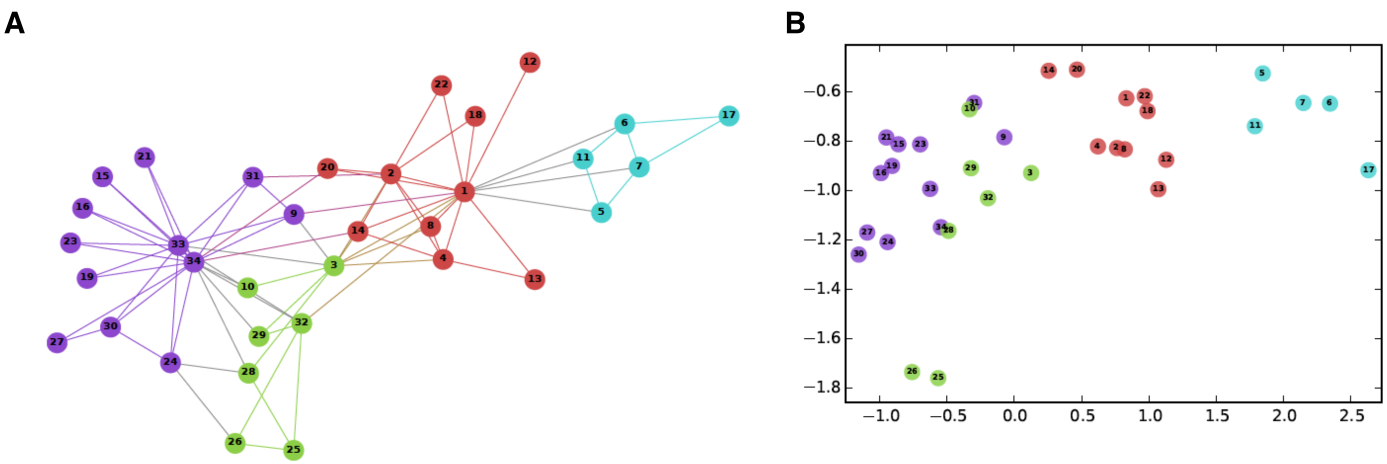


Machine Learning Life Cycle



Problem

Node/Subgraph → A point in low dimensional vector space



Encoder-Decoder Perspective

Encoder Function

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

Decoder Function

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

Similarity Function

$$\text{DEC}(\text{ENC}(v_i), \text{ENC}(v_j)) = \text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \approx s_{\mathcal{G}}(v_i, v_j)$$

Loss Function

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} \ell\left(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j), s_{\mathcal{G}}(v_i, v_j)\right)$$

Shallow Embedding

$$\mathrm{ENC}(v_i) = \mathbf{Z}\mathbf{v}_i$$

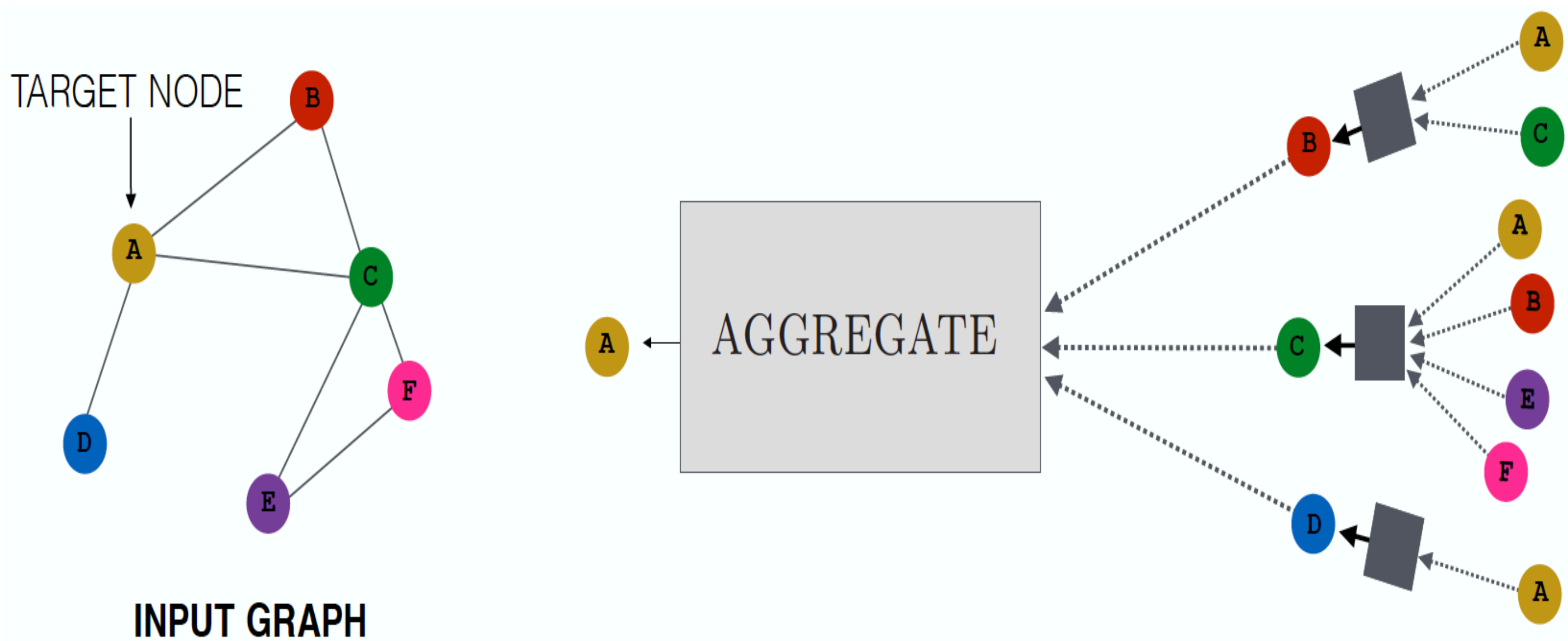
Type	Method	Decoder	Similarity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps	$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2$	general	$\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\|\mathrm{DEC}(\mathbf{z}_i,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

Limitations:

1. No shared Parameters → O(|V|) parameters
2. Transductive
3. Not leveraging nodes' features

Neighborhood Encoders

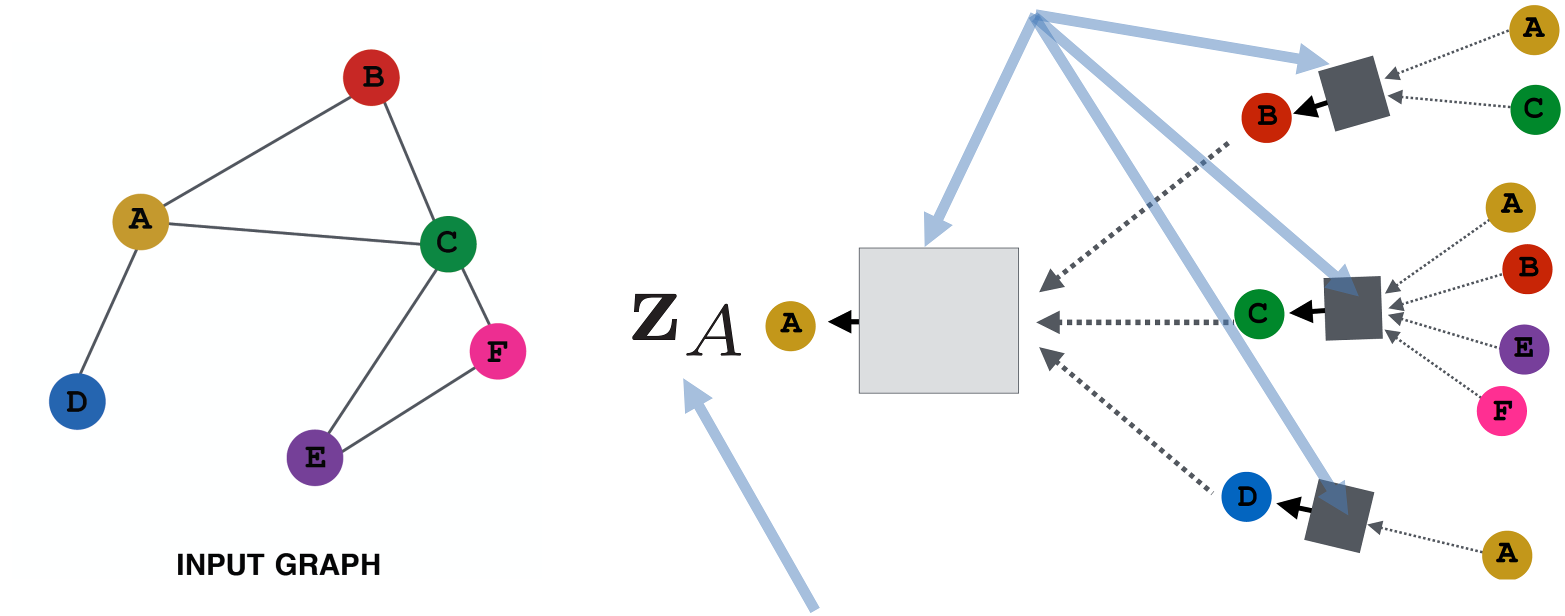
Key idea: Generate node embeddings based on local neighborhoods.



- **Intuition:** Nodes aggregate information from their neighbours using neural networks

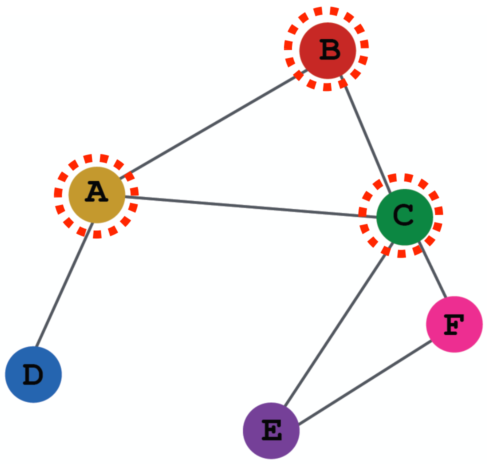
Neighborhood Encoders

1) Define a neighborhood aggregation function.



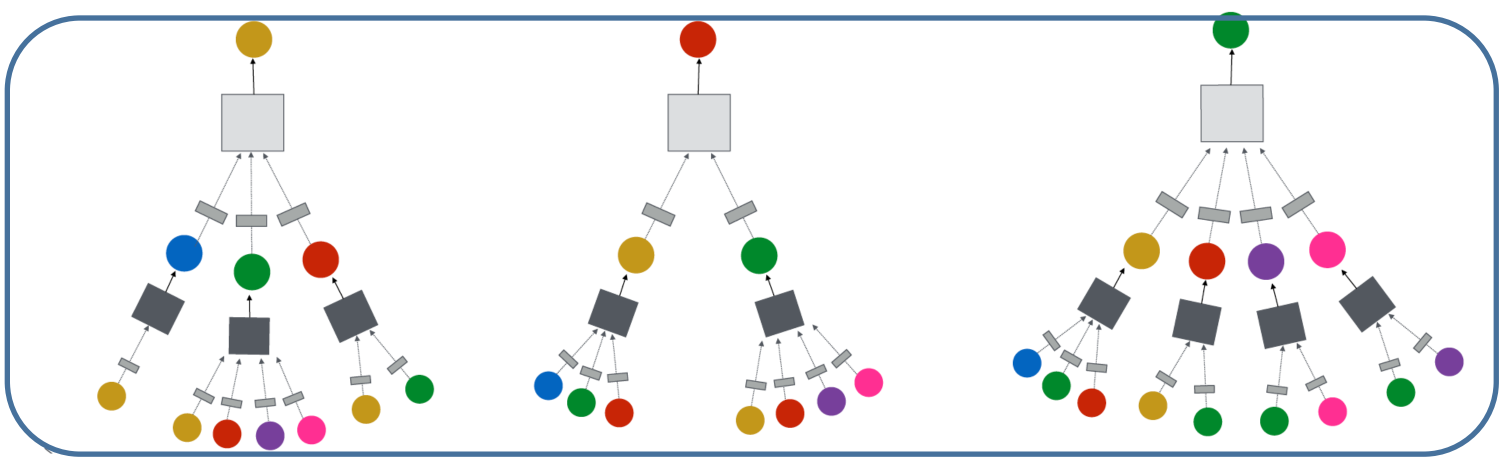
2) Define a loss function on the embeddings, $\mathcal{L}(\mathbf{z}_u)$

Neighborhood Encoders

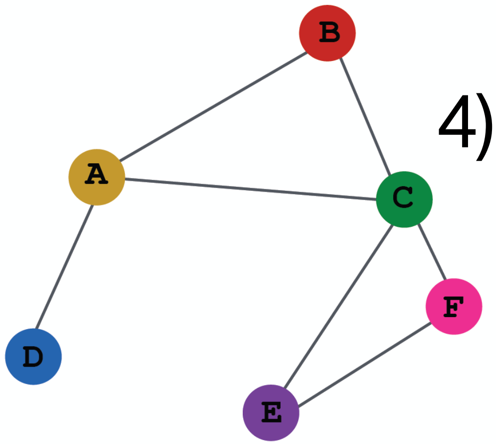


INPUT GRAPH

3) Train on a set of nodes, i.e., a batch of compute graphs



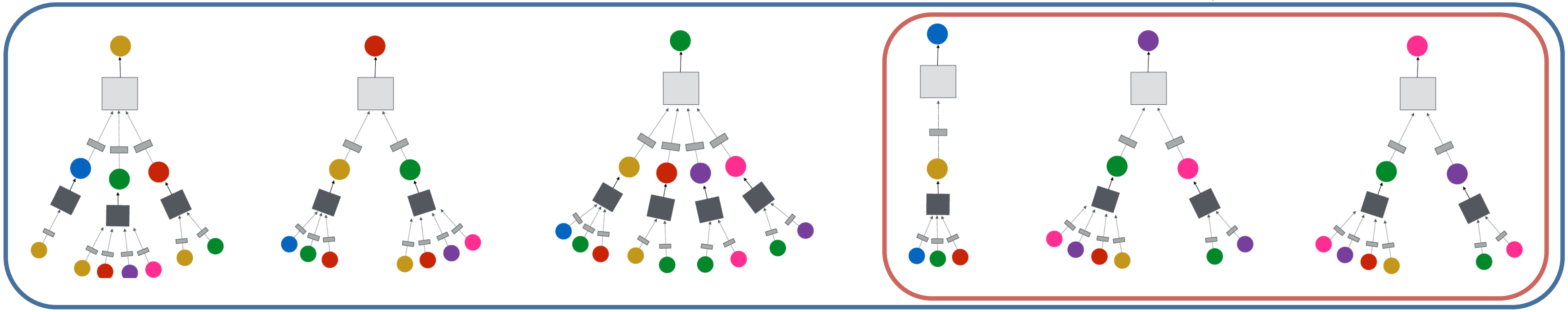
Neighborhood Encoders



INPUT GRAPH

4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!



Neighborhood Aggregation

Basic Neighborhood Aggregation

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

vs.

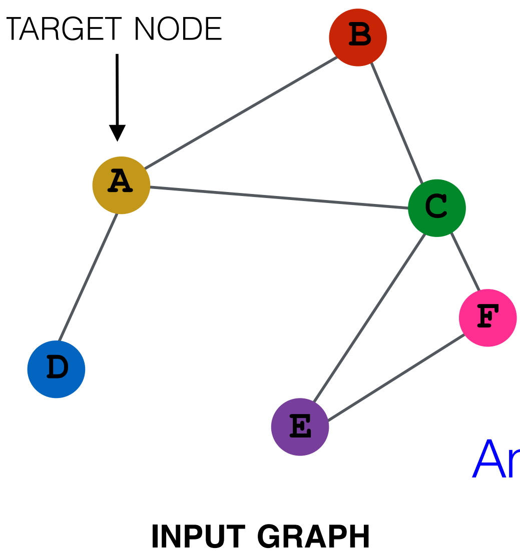
GCN Neighborhood Aggregation

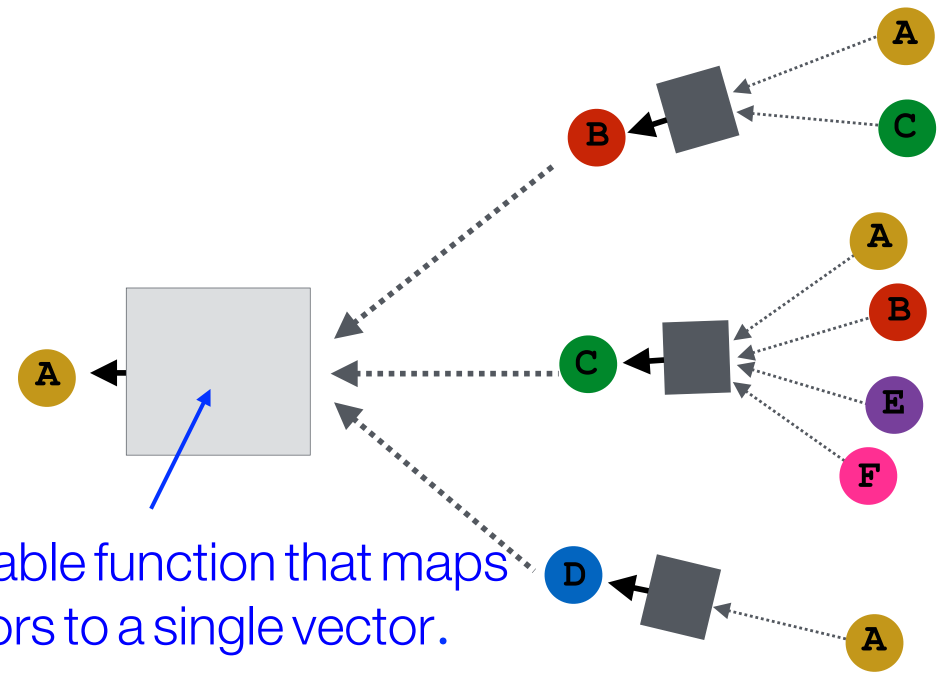
$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v) \cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

same matrix for self and neighbor embeddings

per-neighbor normalization

GraphSAGE





Any differentiable function that maps set of vectors to a single vector.

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{A}_k \cdot \boxed{\text{AGG}(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\})}, \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

GraphSAGE Differences

- Simple neighborhood aggregation:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

- GraphSAGE:

concatenate self embedding and
neighbor embedding

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{W}_k \cdot \boxed{\mathrm{AGG}\left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)}, \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

generalized aggregation

Neighborhood sampling

GraphSAGE Functions

- **Mean:** $\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$
- **Pool** $\text{AGG} = \gamma\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$
- **LSTM** $\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$

Loss function

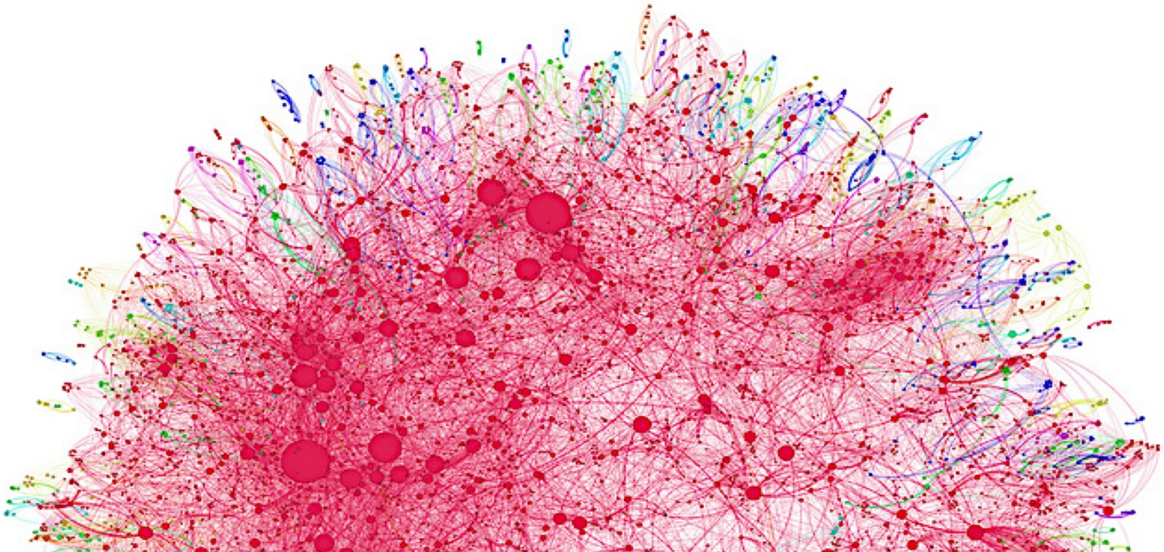
$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

GraphSAGE

An inductive encoder

Parameter sharing

Integrating the graph structure and node feature



References

- Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

- Hamilton, William L., Rex Ying, and Jure Leskovec. "Representation learning on graphs: Methods and applications." *arXiv preprint arXiv:1709.05584* (2017).

Discussion

- GraphSAGE is optimized for two or three layers, what if we want to go deeper? What are the challenges?

- How can we extend GraphSAge to support multi-layer networks?

- GraphSAGE generates embedding for nodes, what about subgraph embedding?