POLITECNICO
MILANO 1863

Advances in Deep Learning with Applications in Text and Image Processing

- Learning on Graphs -

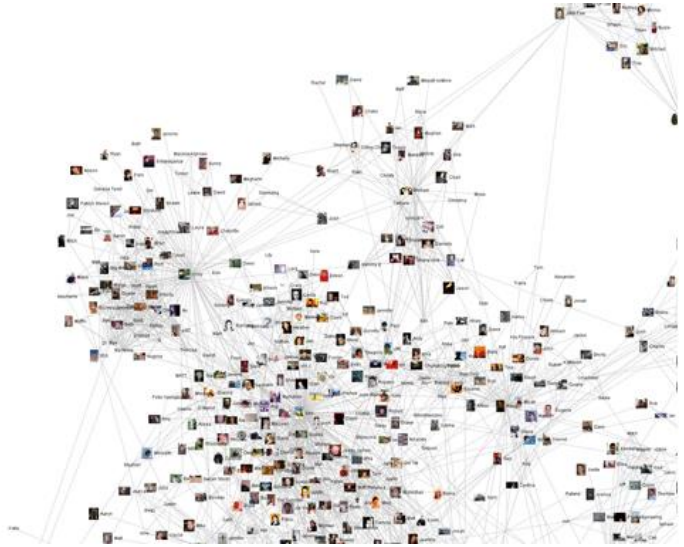
Prof. Matteo Matteucci – *matteo.matteucci@polimi.it*

Department of Electronics, Information and Bioengineering
Artificial Intelligence and Robotics Lab - Politecnico di Milano

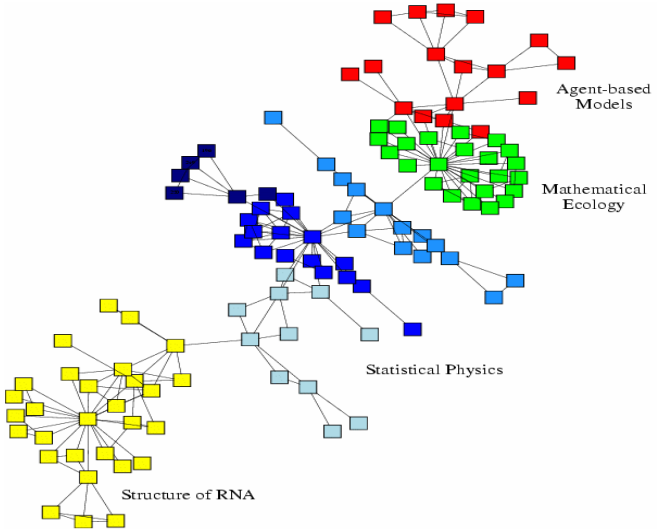
Disclaimer!

This is the 1st time I give this lecture, expect obscure passages, glitches, and typos. Hope you will get the big picture and get curious about the topics as I did! Keep up with me until the end of the day and give me feedbacks on improving these slides so next edition will be marvelous and unforgettable!

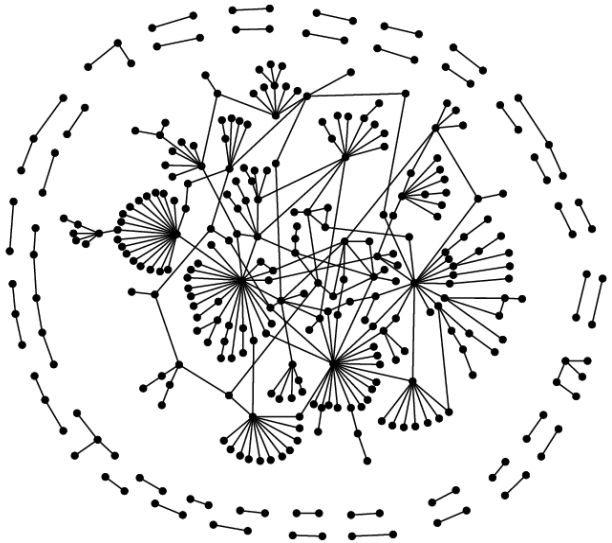
Many data have the form of a graph/network



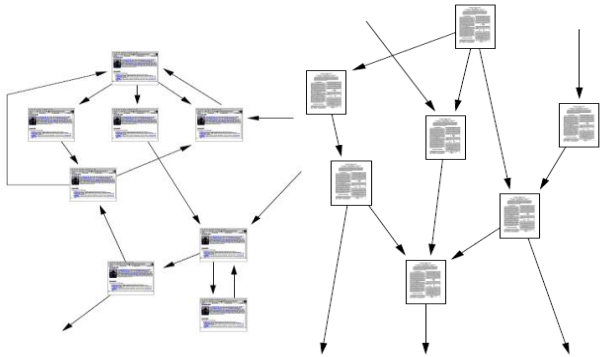
Social networks



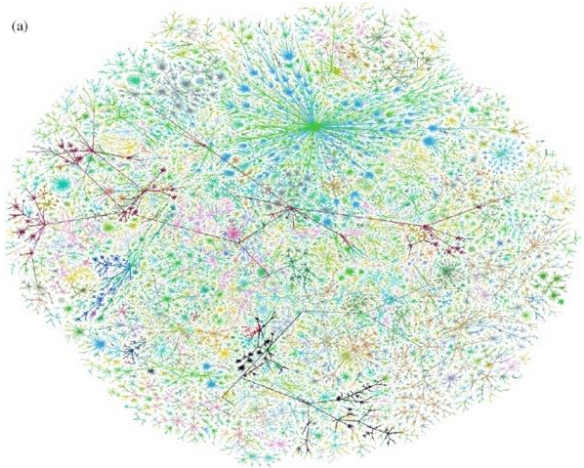
Citation networks



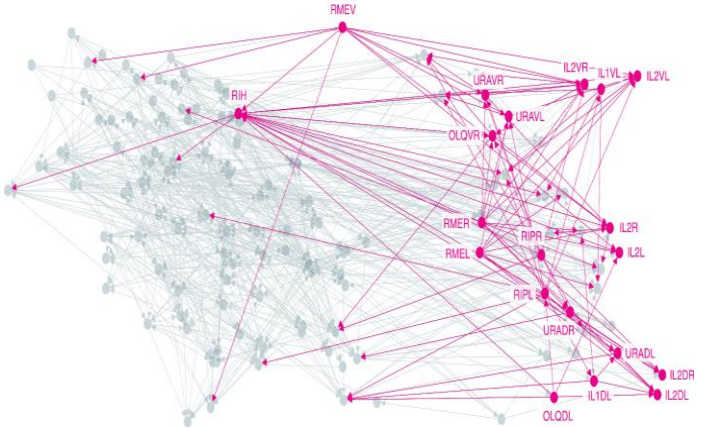
Biomedical networks



Information networks & Web



Internet

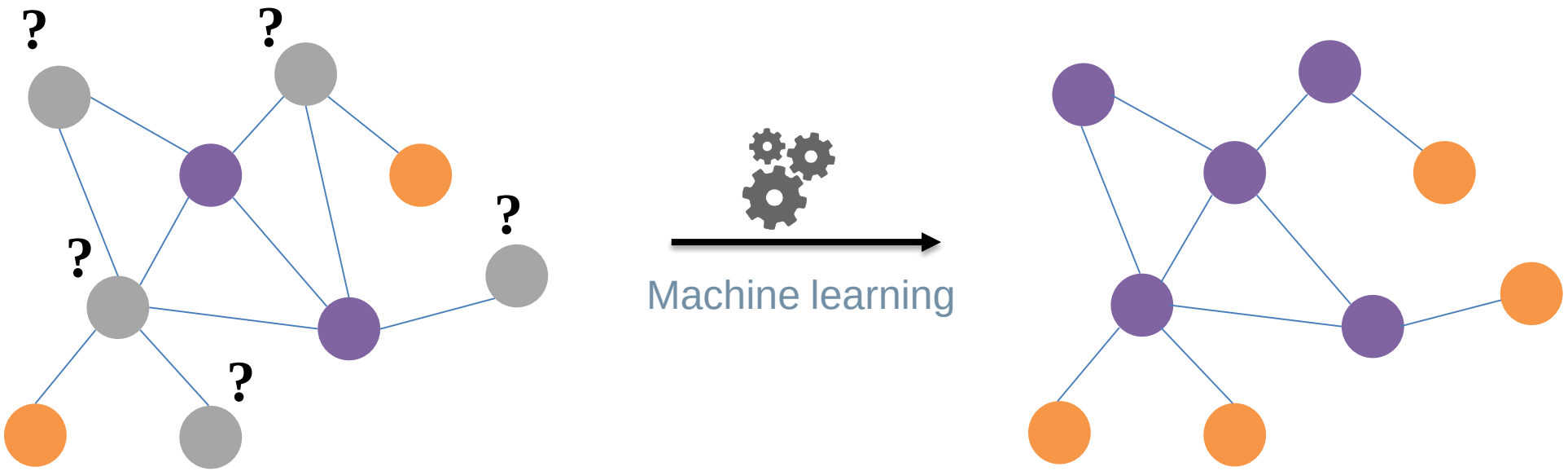


Networks of neurons

Learning on networks

Classical ML tasks can be applied to network structures too:

- Graph/Node classification, i.e., predict the type of a given graph/node



Example: Node Classification

Classifying the function of proteins in the interactome!

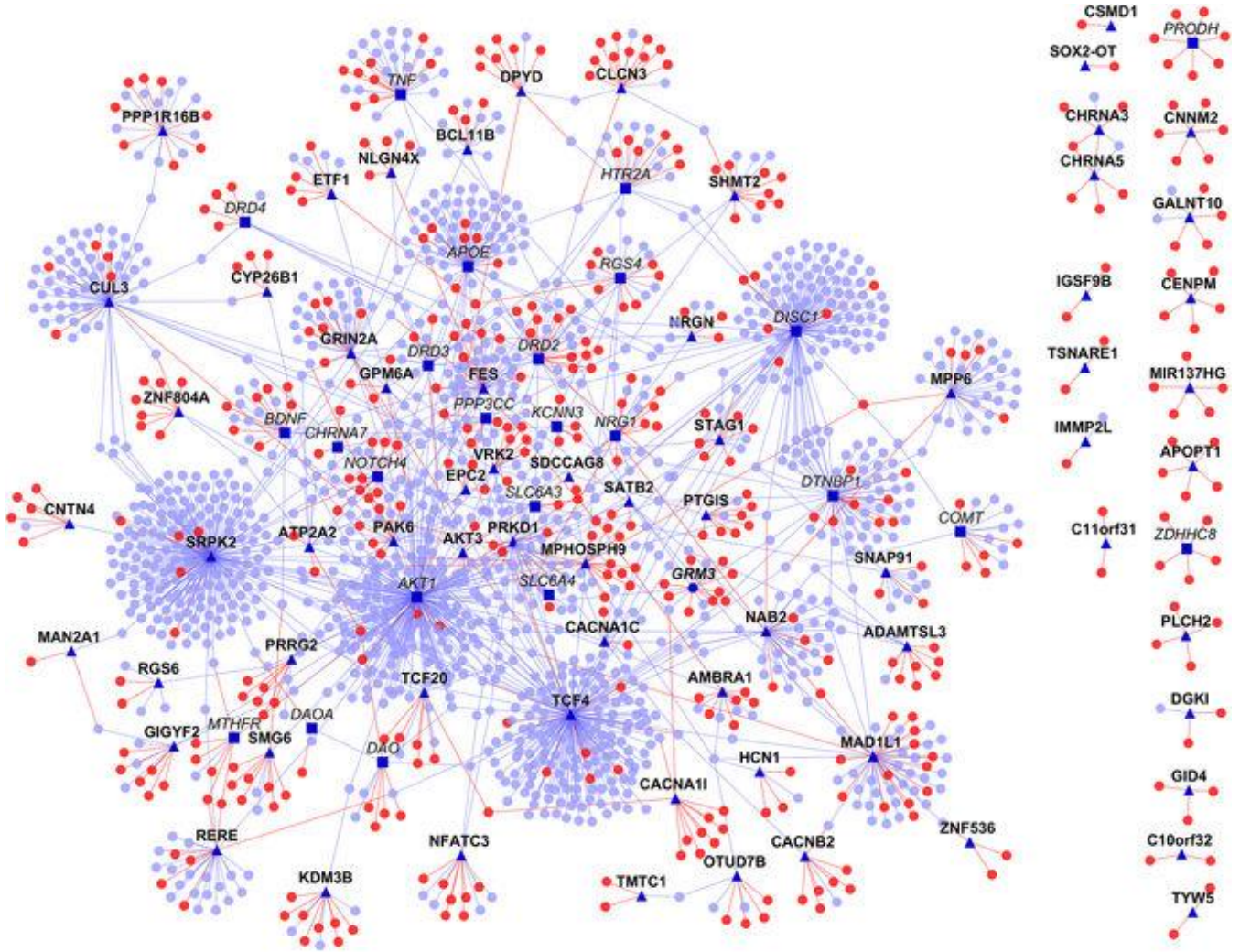
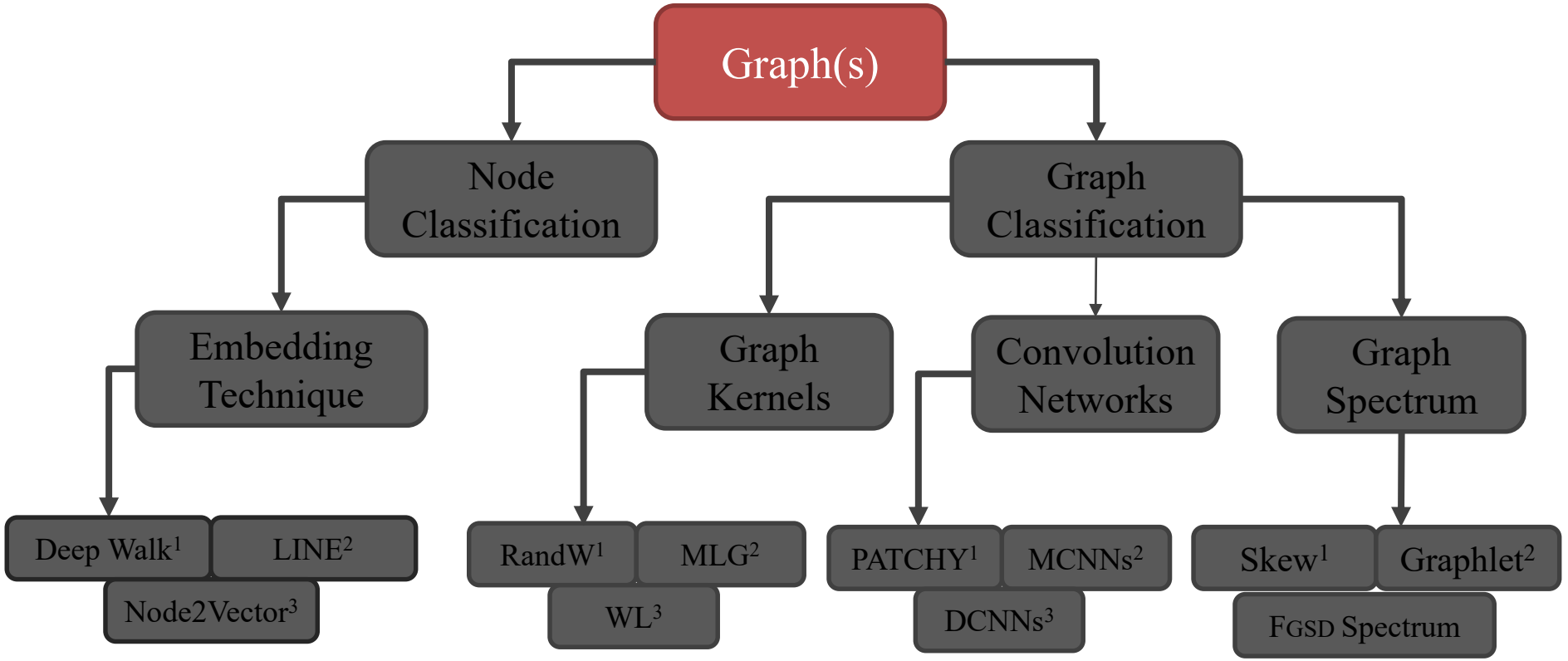
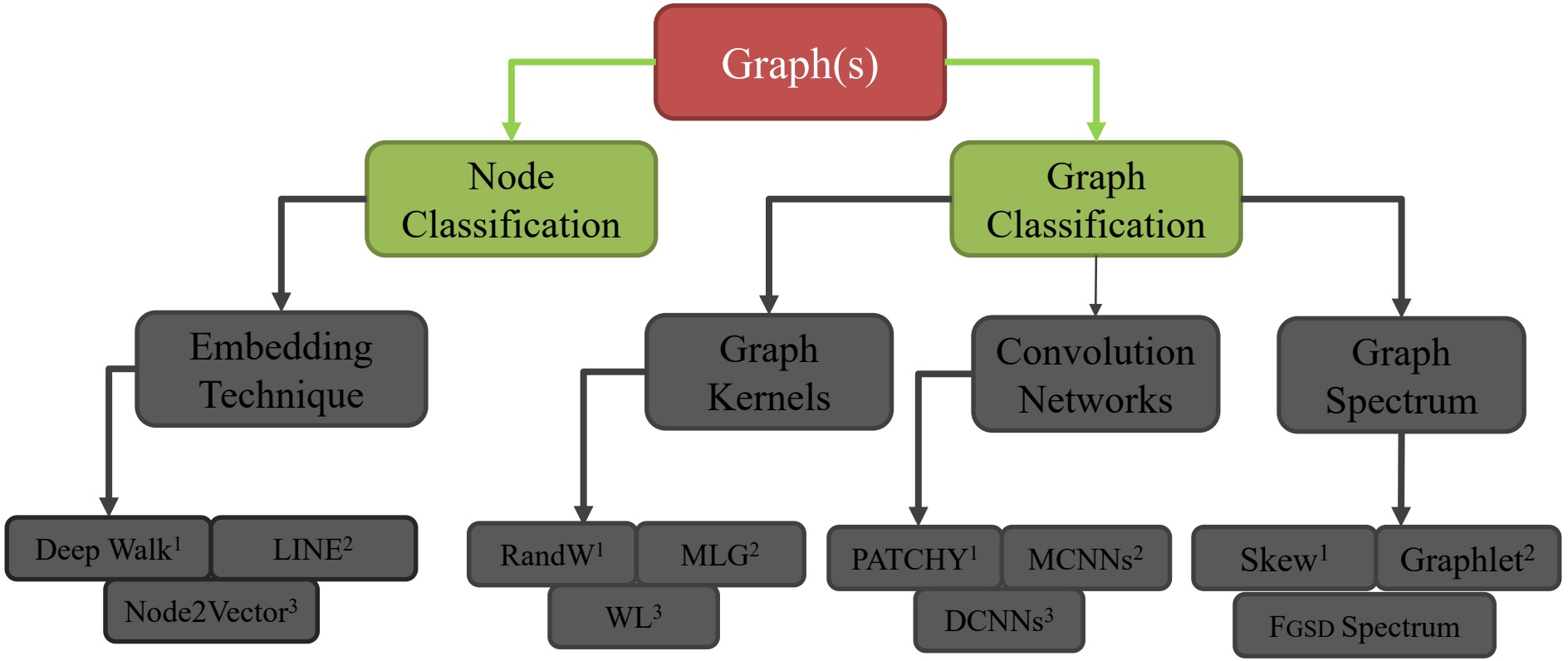


Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

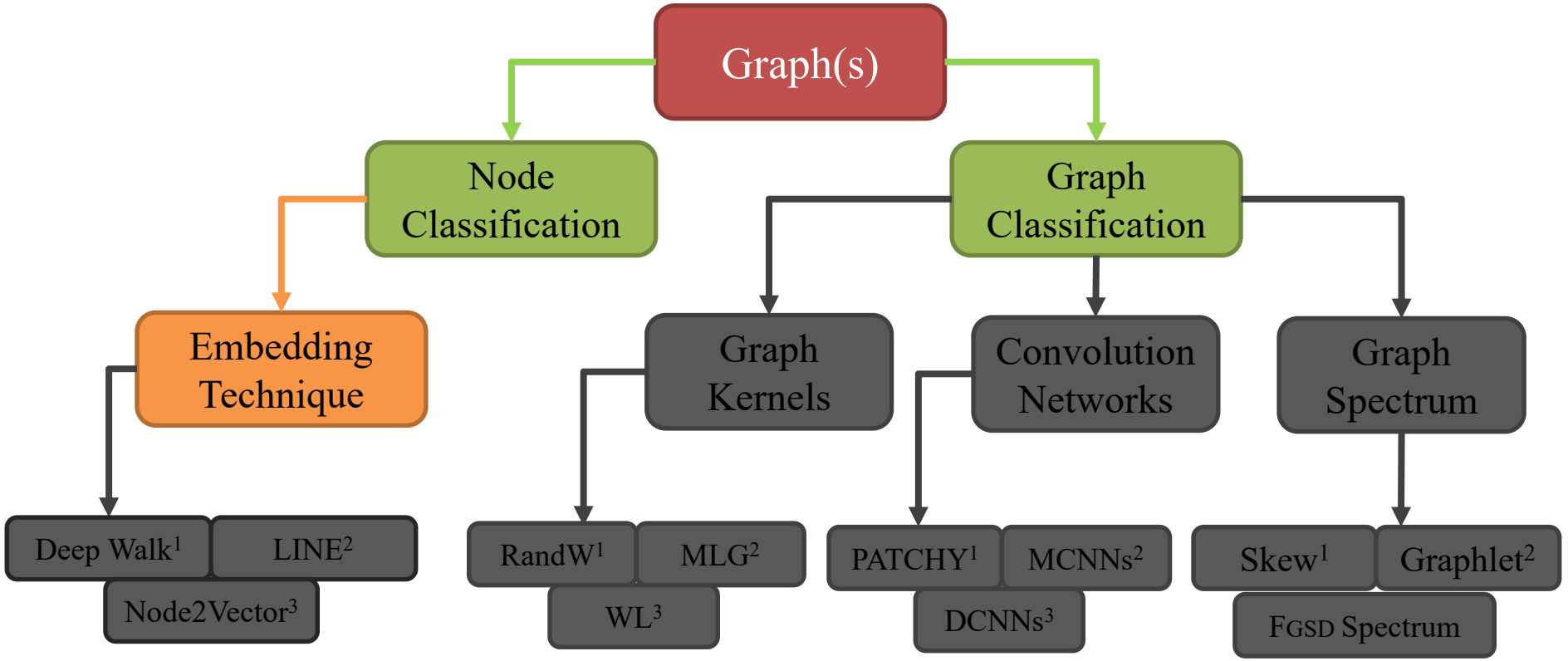
Taxonomy of Graph Learning



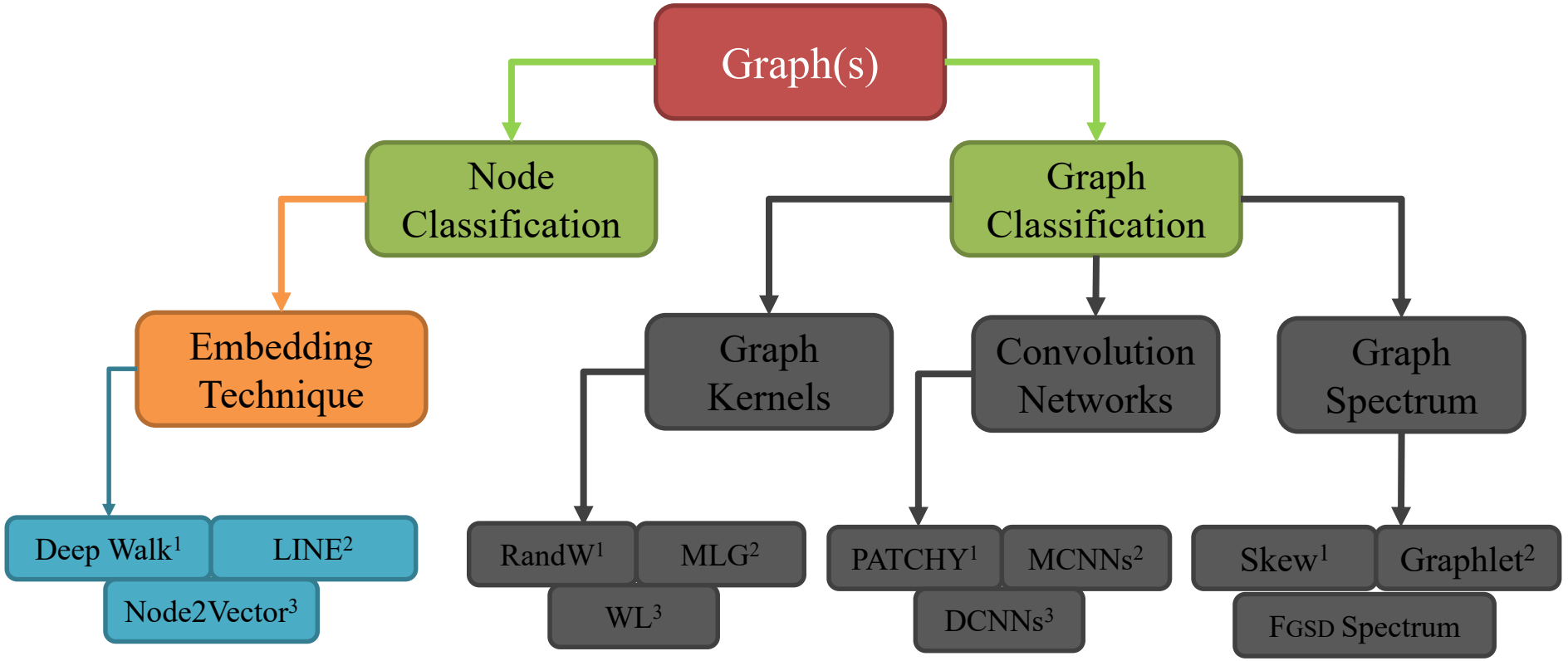
Taxonomy of Graph Learning



Taxonomy of Graph Learning

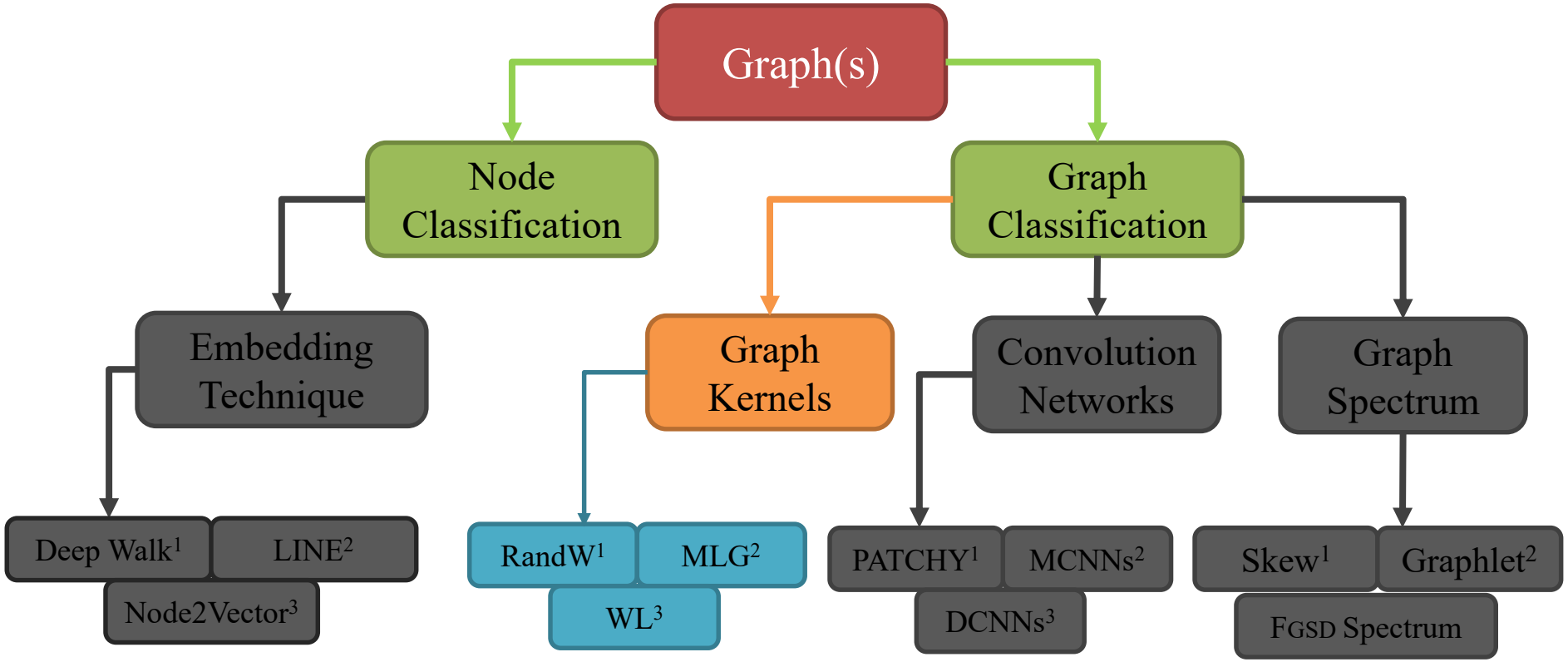


Taxonomy of Graph Learning



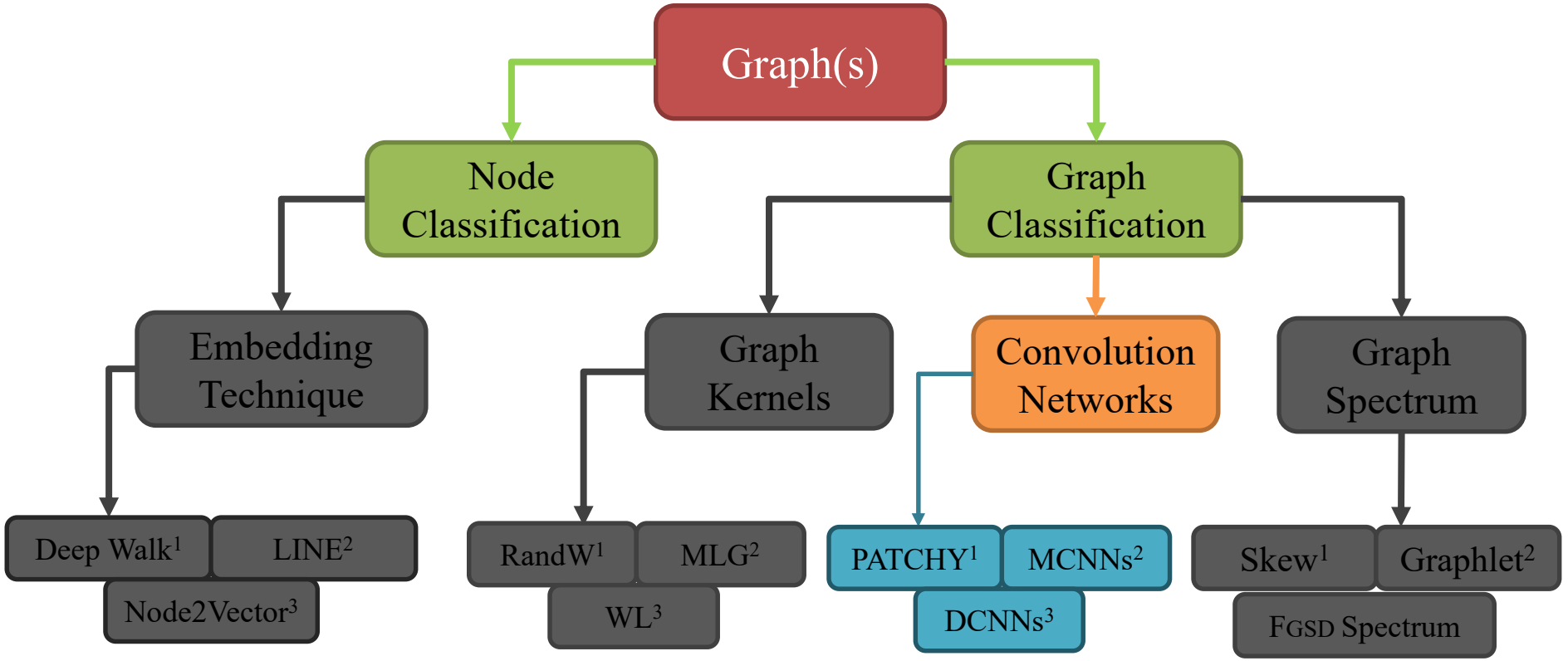
[1] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proc. of the 20th ACM SIGKDD*. ACM, 2014.
[2] Tang, Jian, et al. "Line: Large-scale information network embedding." *Proc. of the 24th International Conference on World Wide Web.*, 2015.
[3] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proc. of the 22nd ACM SIGKDD*. ACM, 2016.

Taxonomy of Graph Learning



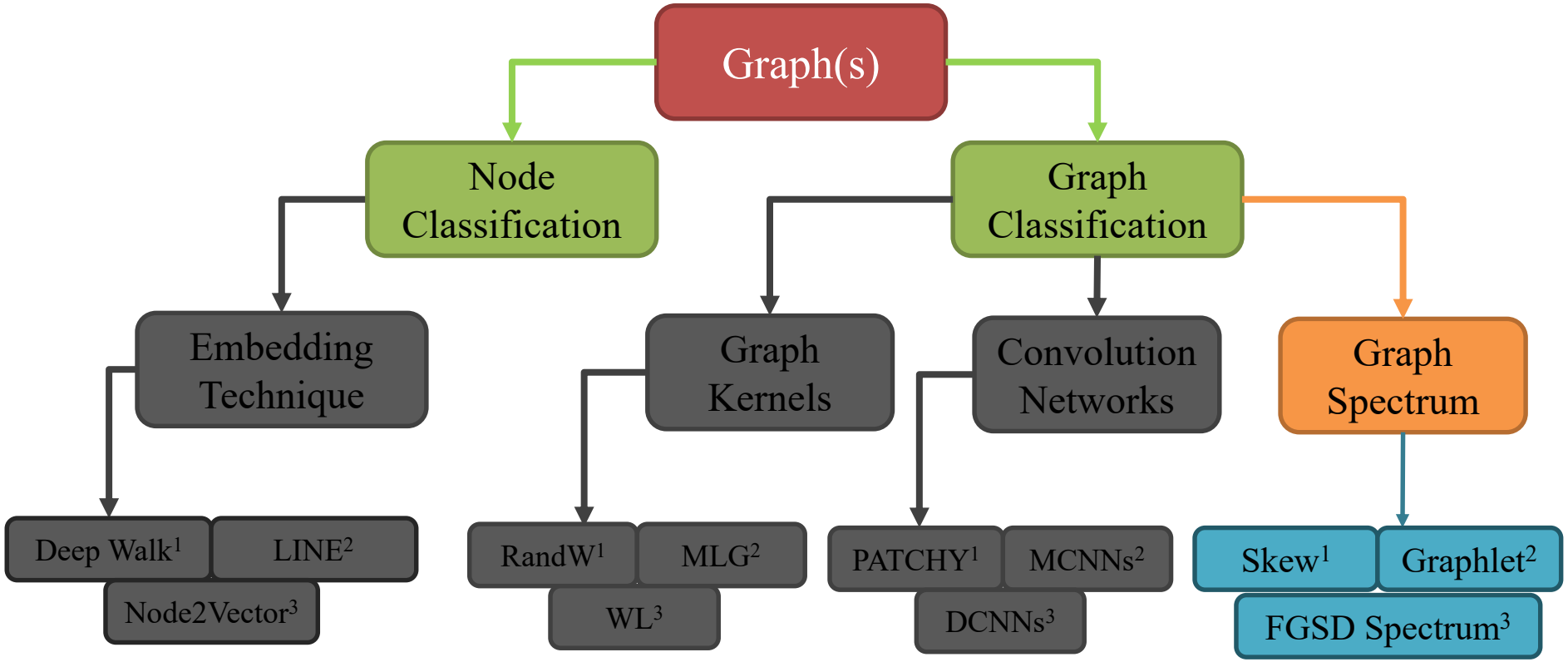
[1] T. Gärtner, P. Flach, and S. Wrobel. "On graph kernels: Hardness results and efficient alternatives." Learning Theory and Kernel Machines (2003): 129-143.
[2] Kondor, Risi, and Horace Pan. "The multiscale Laplacian graph kernel." Advances in Neural Information Processing Systems. 2016.
[3] Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." Journal of Machine Learning Research 12.Sep (2011): 2539-2561.

Taxonomy of Graph Learning



[1] M. Niepert, M. Ahmed, and K. Kutzkov. "Learning convolutional neural networks for graphs." International Conference on Machine Learning. 2016.
[2] Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." Advances in neural information processing systems. 2015.
[3] Atwood, James, and Don Towsley. "Diffusion-convolutional neural networks." Advances in Neural Information Processing Systems. 2016.

Taxonomy of Graph Learning

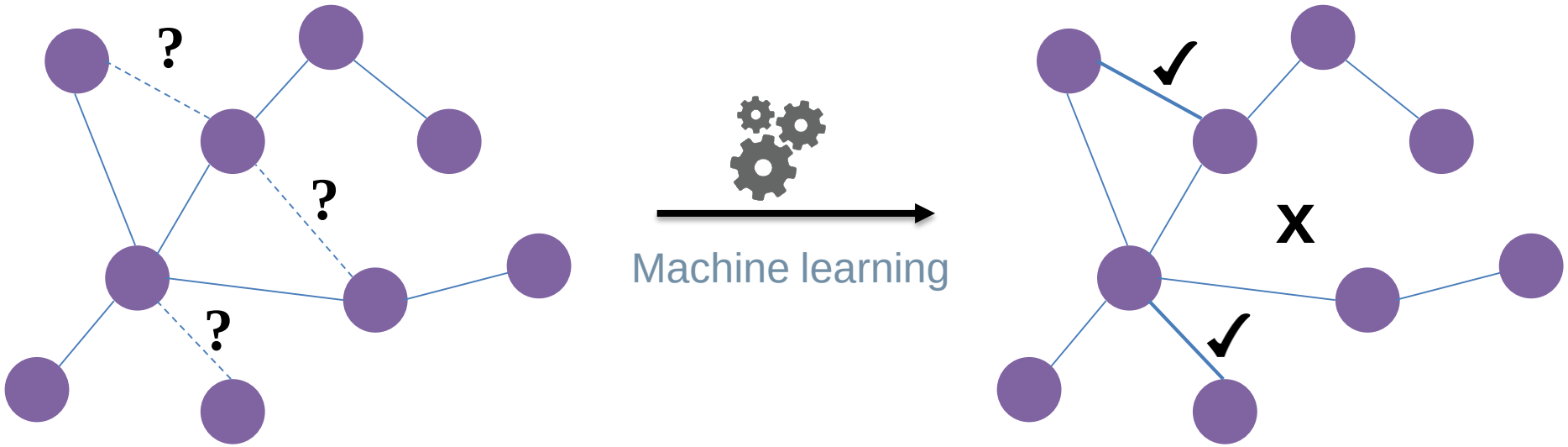


[1] Kondor, Risi, and Karsten M. Borgwardt. "The skew spectrum of graphs." Proc. of 25th International Conference on Machine learning. ACM, 2008.
[2] Kondor, Risi, Nino Shervashidze, and Karsten M. Borgwardt. "The graphlet spectrum." Proc. of 26th International Conference on Machine Learning. ACM, 2009.
[3] Saurabh Verma and Zhi-li Zhang. ``Hunting For a Unique, Stable, Sparse and Fast Feature Algorithm on Graphs". In 31st NIPS, 2017.

Learning on networks

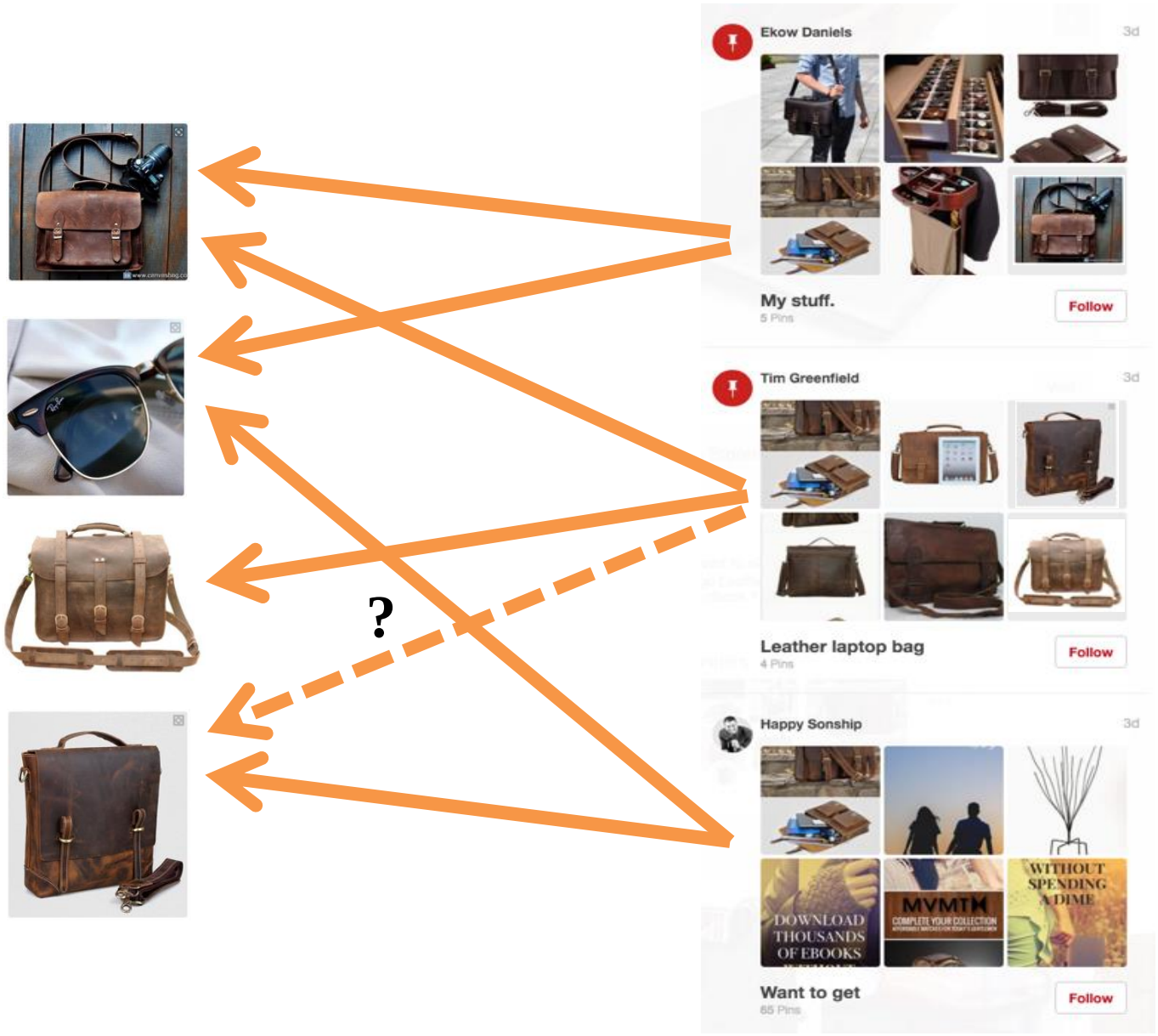
Classical ML tasks can be applied to network structures too:

- Graph/Node classification, i.e., predict the type of a given graph/node
- Link prediction, i.e., predict whether two nodes are linked



Example: Link Prediction

Content recommendation is link prediction!



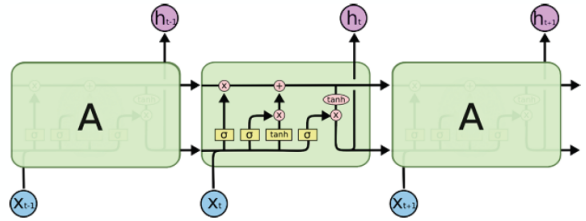
Learning on networks

Classical ML tasks can be applied to network structures too:

- Graph/Node classification, i.e., predict the type of a given graph/node
- Link prediction, i.e., predict whether two nodes are linked
- Community (cluster) detection, i.e., identify densely linked clusters of nodes
- Network similarity, i.e., how similar are two (sub)networks

The main difficulty stays in the non Euclidean geometry of the space

- Sound and text are 1D



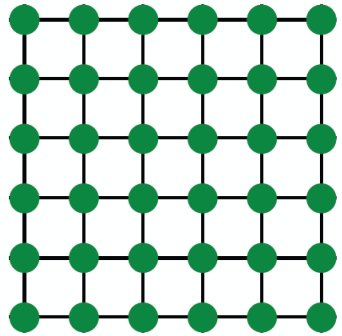
Learning on networks

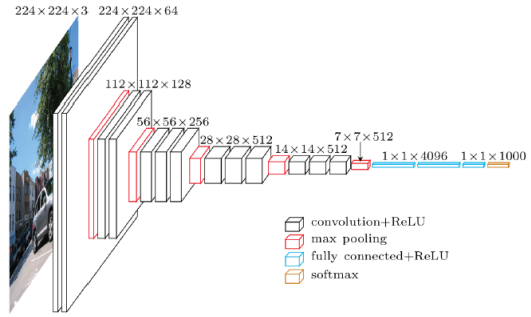
Classical ML tasks can be applied to network structures too:

- Node classification, i.e., predict the type of a given node
- Link prediction, i.e., predict whether two nodes are linked
- Community (cluster) detection, i.e., identify densely linked clusters of nodes
- Network similarity, i.e., how similar are two (sub)networks

The main difficulty stays in the non Euclidean geometry of the space

- Sound and text are 1D
- Images are 2D





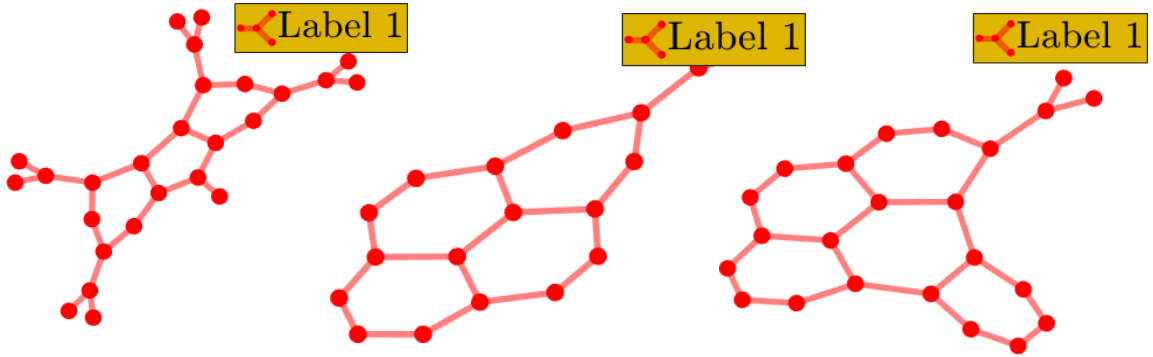
Learning on networks

Classical ML tasks can be applied to network structures too:

- Node classification, i.e., predict the type of a given node
- Link prediction, i.e., predict whether two nodes are linked
- Community (cluster) detection, i.e., identify densely linked clusters of nodes
- Network similarity, i.e., how similar are two (sub)networks

The main difficulty stays in the non Euclidean geometry of the space

- Sound and text are 1D
- Images are 2D
- Graphs are …



POLITECNICO
MILANO 1863

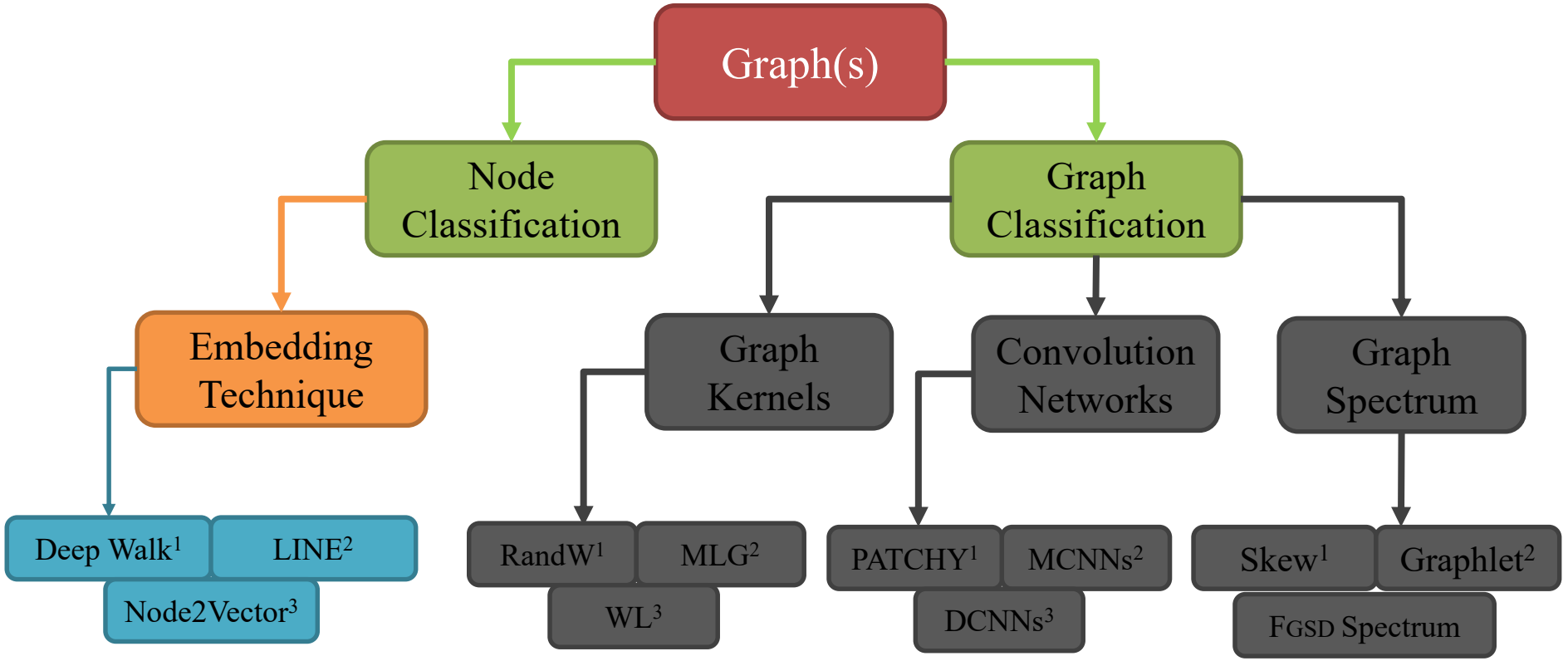
Advances in Deep Learning with Applications in Text and Image Processing

- Node Embedding -

Prof. Matteo Matteucci – *matteo.matteucci@polimi.it*

Department of Electronics, Information and Bioengineering
Artificial Intelligence and Robotics Lab - Politecnico di Milano

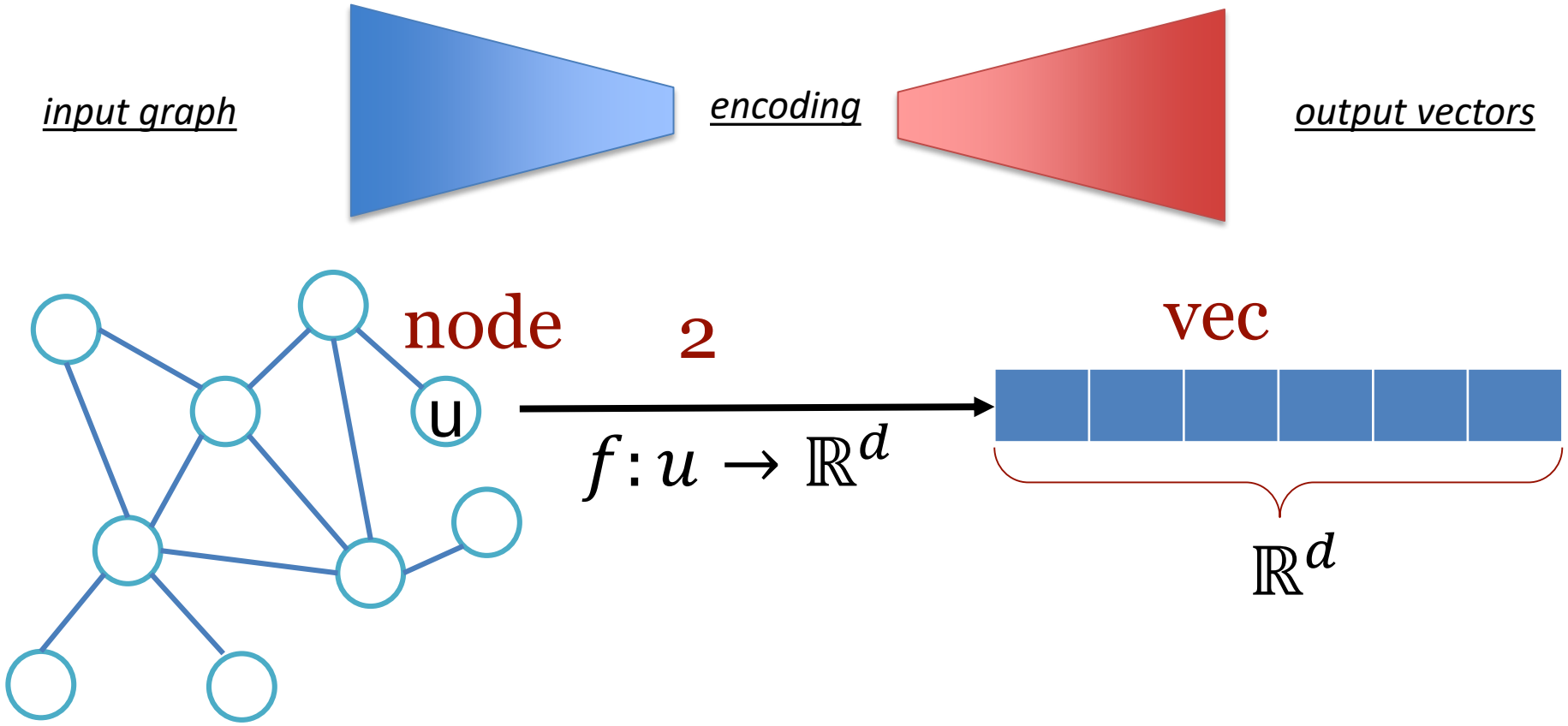
Taxonomy of Graph Learning



[1] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proc. of the 20th ACM SIGKDD*. ACM, 2014.
[2] Tang, Jian, et al. "Line: Large-scale information network embedding." *Proc. of the 24th International Conference on World Wide Web*., 2015.
[3] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proc. of the 22nd ACM SIGKDD*. ACM, 2016.

Feature Learning in Graphs

Goal: Efficient task-independent feature learning for machine learning in networks!



Example: Zachary's Karate Club Network

input graph → *encoding* → *output vectors*

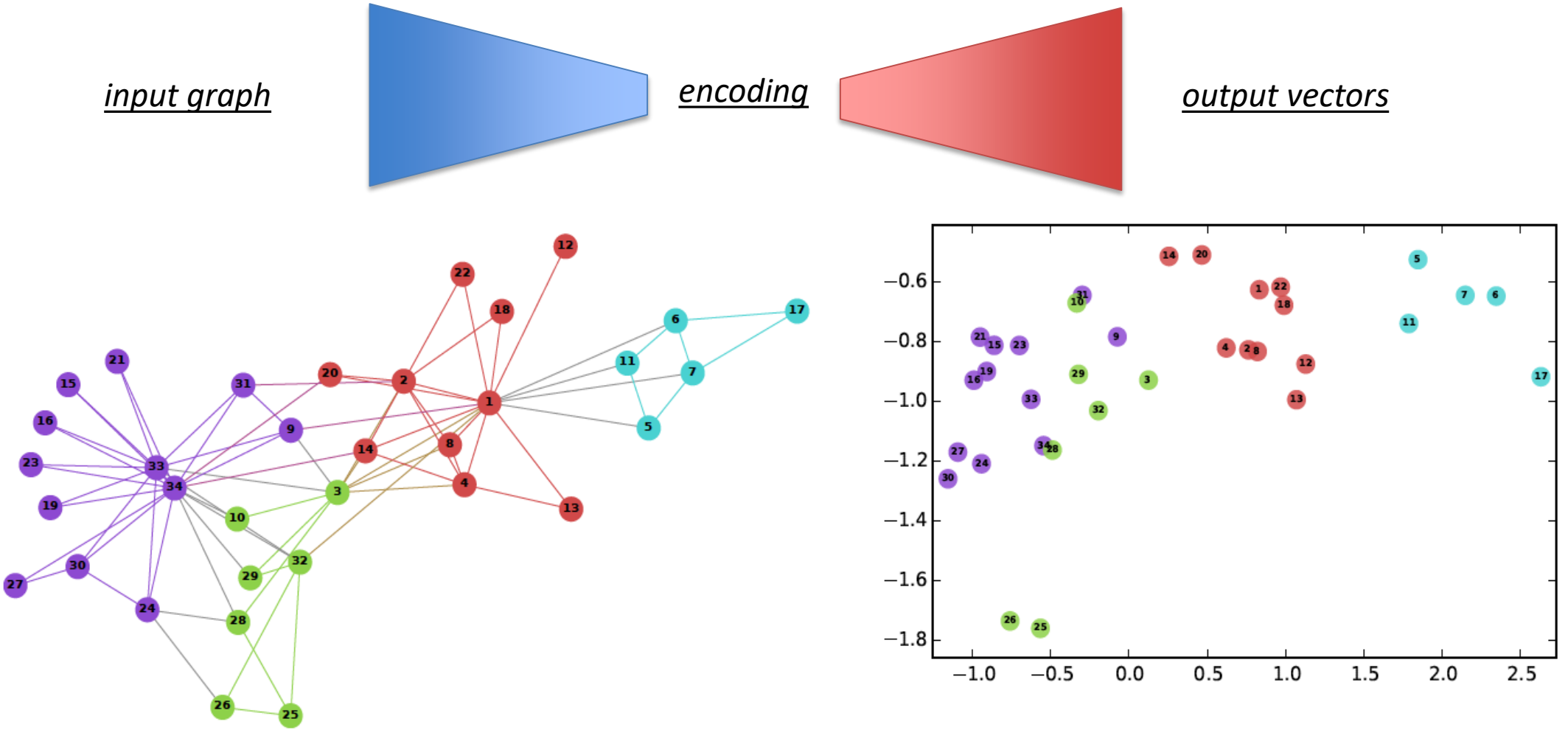


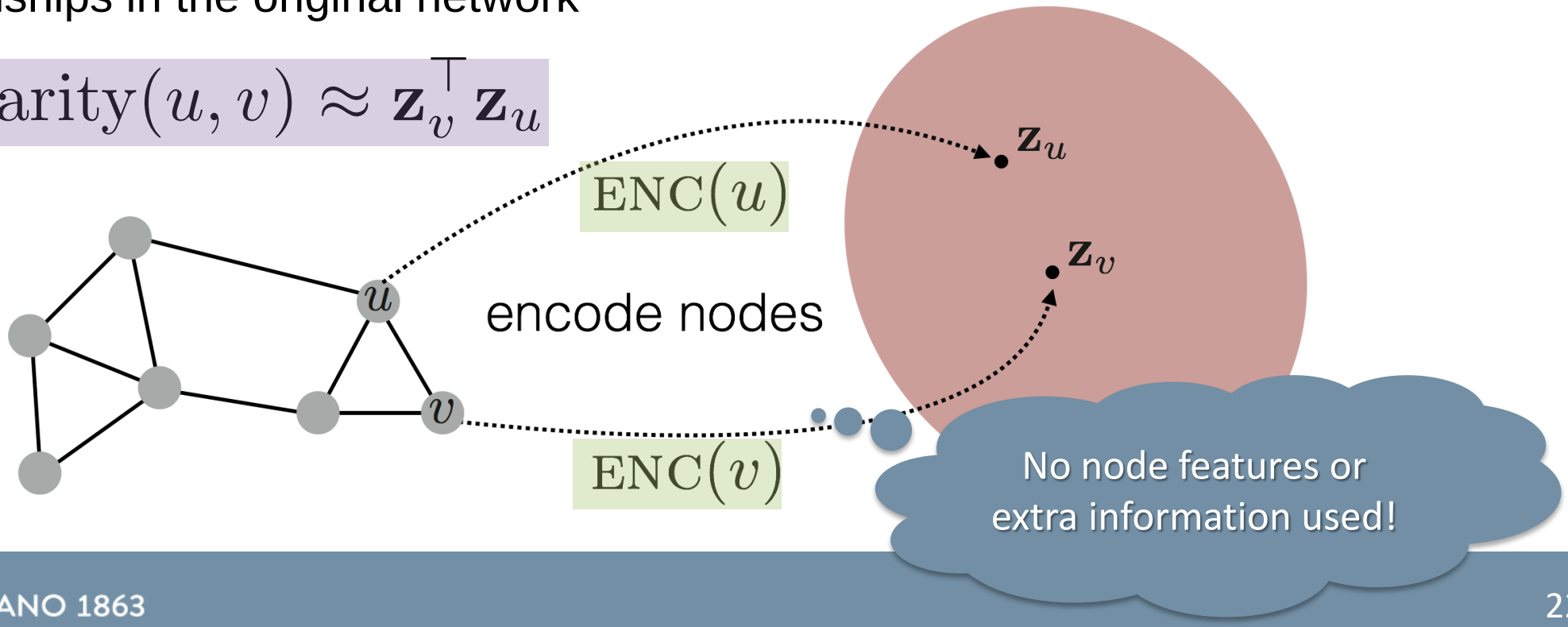
Image from: Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD.*

Node Embedding

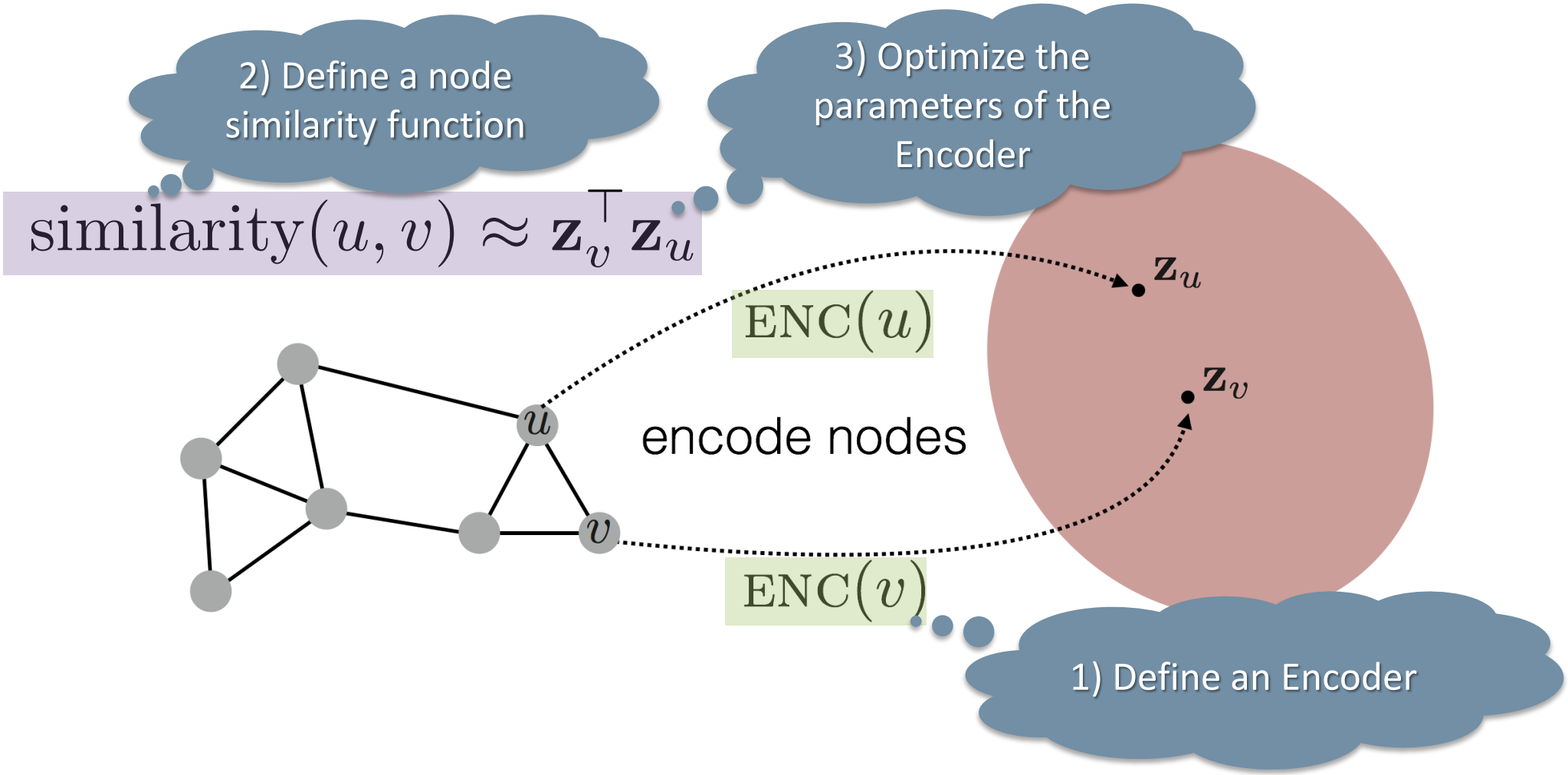
Embedding such that “similar” nodes in the graph have encodings which are close together in a d-dimensional space.

- Encoder maps each node to a low-dimensional vector.
- Similarity function specifies how relationships in vector space map to relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$



Learning Nodes Embeddings



«Shallow» Encoding

The encoder is just an embedding-lookup matrix, each column is the embedding, i.e., what we learn, nodes are encoded via “one hot” encoding.

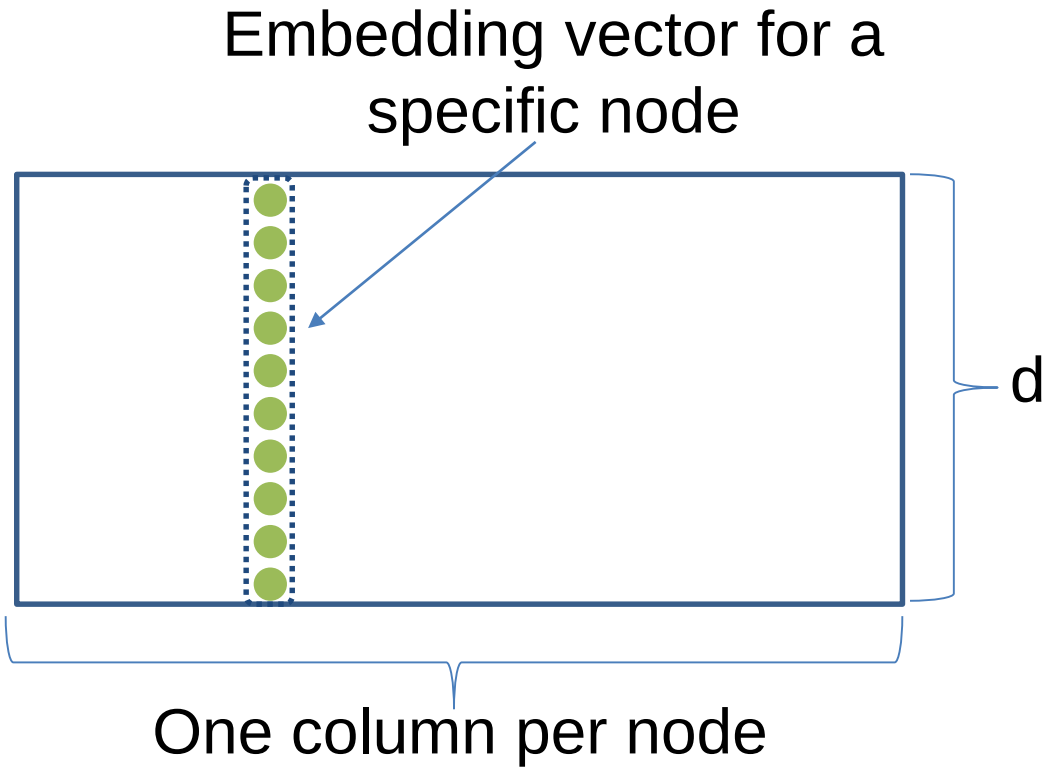
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$$

$$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$$

Embedding matrix

$\mathbf{Z} =$

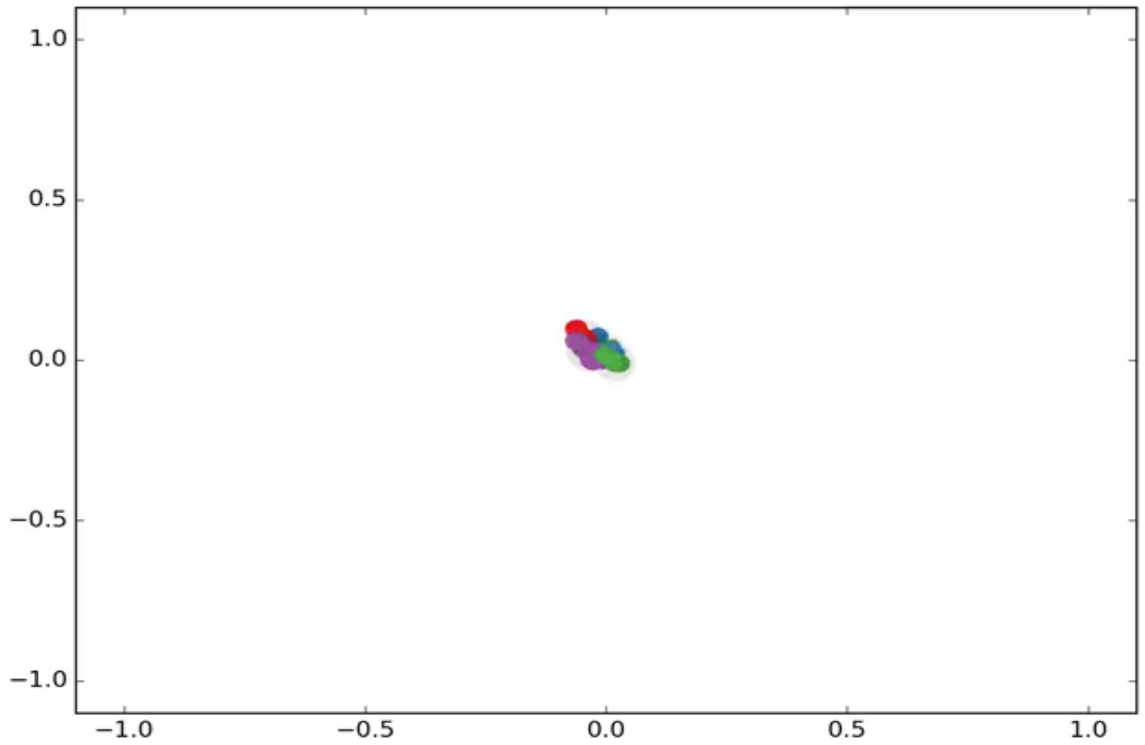


Each node is assigned a unique embedding vector, e.g., node2vec, DeepWalk, LINE, etc.

Node Similarity

The key distinction between “shallow” methods is how they define node similarity, i.e., two nodes have similar embeddings if they:

- are connected (?)
- share neighbors (?)
- have similar “structural roles” (?)



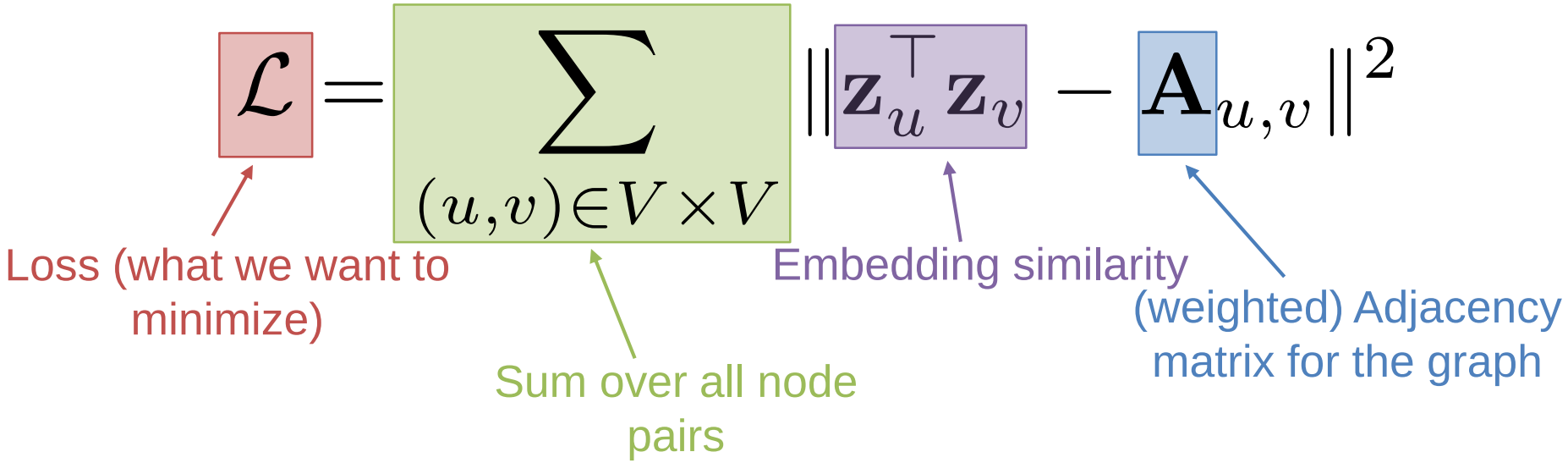
Possible choices are:

- Adjacency-based similarity
- Multi-hop similarity
- Random walk approaches

Material from: Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.

Adjacency-based Similarity

Similarity function is just the edge weight between u and v in the original network, i.e., dot products between node embeddings approximate edge existence.



Ahmed et al. 2013. Distributed Natural Large Scale Graph Factorization. *WWW.*

Adjacency-based Similarity

Similarity function is just the edge weight between u and v in the original network, i.e., dot products between node embeddings approximate edge existence.

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

Find embedding matrix $\mathbf{Z} \in \mathbb{R}^{(d \times |V|)}$ which minimizes the loss $\mathcal{L}$

1. Use stochastic gradient descent (SGD) as a general optimization method (Highly scalable, general approach)

or

2. Solve matrix decomposition solvers, e.g., SVD or QR decomposition routines, but works in limited cases.

Adjacency-based Similarity

Similarity function is just the edge weight between u and v in the original network, i.e., dot products between node embeddings approximate edge existence.

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

Has some drawbacks:

- O(|V|^2) runtime, since it must consider all node pairs (can make O([E]) by only summing over non-zero edges and using regularization (Ahmed et al., 2013)
- O(|V|) parameters (one vector per node)
- Only considers direct, local connections

Adjacency-based Similarity

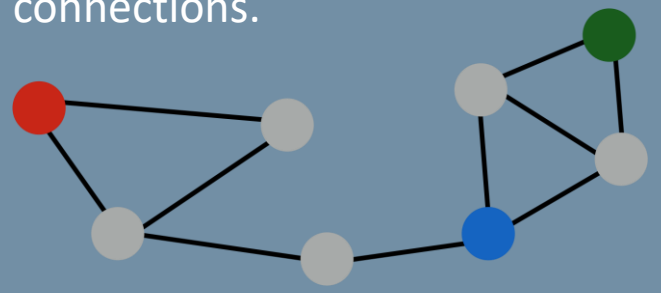
Similarity function is just the edge weight between u and v in the original network, i.e., dot products between node embeddings approximate edge existence.

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

Has some drawbacks:

- O(|V|^2) runtime, since it must consider all nod
  summing over non-zero edges and using regu
- O(|V|) parameters (one vector per node)
- Only considers direct, local connections

The blue node is more similar to green compared to red node, despite none having direct connections.

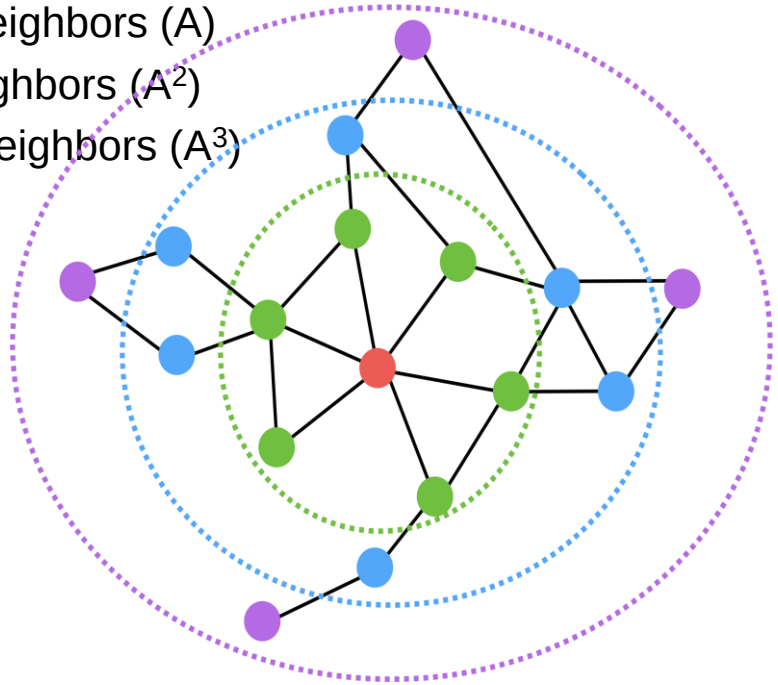


Multi-hop Similarity

Extend the Adiacency matrix to consider k-hop node neighbors and train embeddings to predict k-hop neighbors.

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

Red: Target node
Green: 1-hop neighbors (A)
Blue: 2-hop neighbors (A^2)
Purple: 3-hop neighbors (A^3)



In practice log-transformed, probabilistic adjacency matrix

$$\tilde{\mathbf{A}}^k_{i,j} = \max\left(\log\left(\frac{(\mathbf{A}_{i,j}/d_i)}{\sum_{l\in V}(\mathbf{A}_{l,j}/d_l)^k}\right)^k - \alpha, 0\right)$$

node degree

constant shift

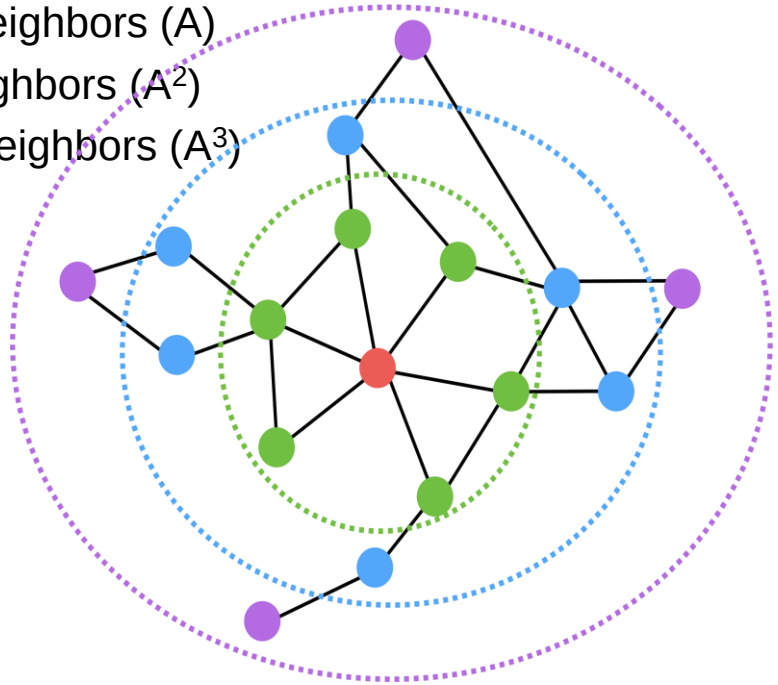
Cao et al. 2015. GraRep: Learning Graph Representations with Global Structural Information. *CIKM.*
Ou et al. 2016. Asymmetric Transitivity Preserving Graph Embedding. *KDD.*

Multi-hop Similarity

Extend the Adiacency matrix to consider k-hop node neighbors and train embeddings to predict k-hop neighbors.

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

Red: Target node
Green: 1-hop neighbors (A)
Blue: 2-hop neighbors (A^2)
Purple: 3-hop neighbors (A^3)



In practice log-transformed, probabilistic adjacency matrix

$$\tilde{\mathbf{A}}_{i,j}^k = \max \left(\log \left(\frac{(\mathbf{A}_{i,j}/d_i)}{\sum_{l \in V} (\mathbf{A}_{l,j}/d_l)^k} \right)^k - \alpha, 0 \right)$$

node degree

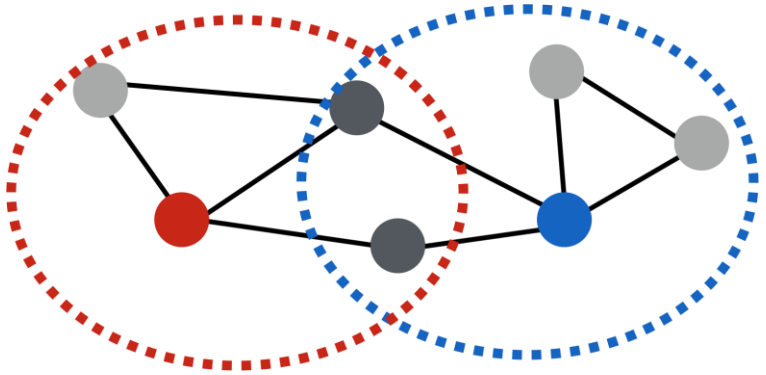
constant shift

Concatenate the output of different hop lengths.

Multi-hop Similarity

An alternative option is to measure the overlap between node neighborhoods, e.g., via Jaccard similarity or Adamic-Adar score

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



Where $\mathbf{S}_{u,v}$ is the neighborhood overlap between u and v (HOPE (Yan et al., 2016).)

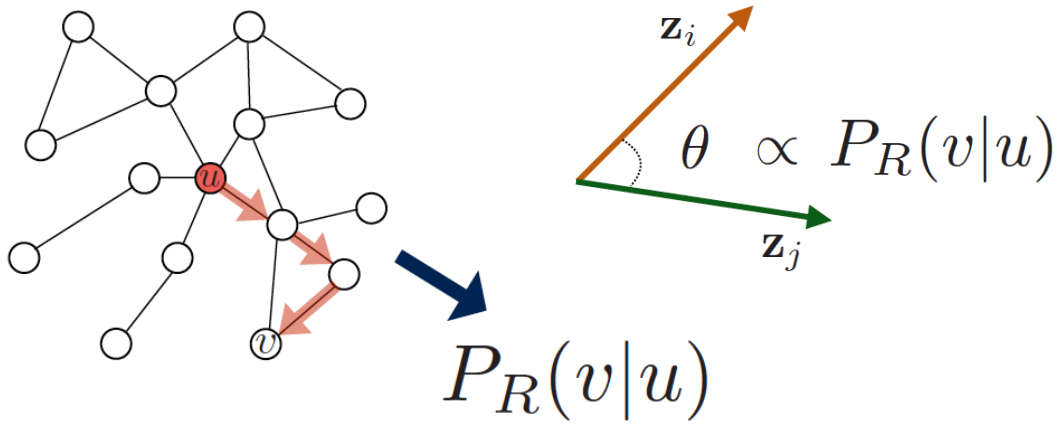
Usual drawbacks are still there:

- Expensive: Generally $O(|V|^2)$, since we need to iterate over all pairs of nodes.
- Brittle: Must hand-design deterministic node similarity measures.
- Massive parameter space: $O(|V|)$ parameters

Random Walk Approaches

Interpret the dot product in feature space as the probability u and v co-occur on a random walk over the network

1. Estimate probability of visiting node v on a random walk starting from node u using random walk strategy R.
2. Optimize embeddings to encode these random walk statistics

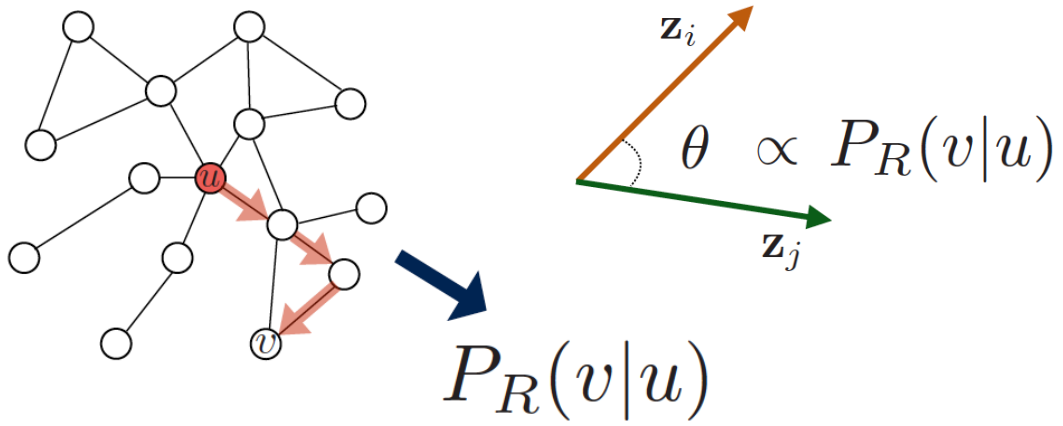


Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD.*
Grover et al. 2016. node2vec: Scalable Feature Learning for Networks. *KDD.*

Random Walk Approaches

Interpret the dot product in feature space as the probability u and v co-occur on a random walk over the network

1. Estimate probability of visiting node v on a random walk starting from node u using random walk strategy R.
2. Optimize embeddings to encode these random walk statistics



Introduces sone advantages in terms of:

- Expressivity: Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information.
- Efficiency: Does not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks.

Random Walk Optimization

Optimize embeddings to maximize likelihood of random walk co-occurrences estimating the probability of a node v to be visited starting from node u

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

1. Run short random walks from each node on the graph using some strategy R
2. For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u
3. Optimize embeddings to according to:

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

Random Walk Optimization

Optimize embeddings to maximize likelihood of random walk co-occurrences

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Random Walk Optimization

Optimize embeddings to maximize likelihood of random walk co-occurrences

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

Nested sum over nodes gives
$O(|V|^2)$ complexity!!

Random Walk Optimization

Optimize embeddings to maximize likelihood of random walk co-occurrences

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

We use Negative Sampling, i.e., just normalize against k random "negative" examples

Random distribution over all nodes

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right) \approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k} \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

Sample negative nodes proportional to degree of nodes

Sigmoid function

Higher k gives more robust estimates as it corresponds to higher prior on negative events

node2vect

You can just run fixed-length, unbiased random walks starting from each node (i.e., DeepWalk from Perozzi et al., 2013), but biased random walks can trade off local and global views of the network (i.e., node2vect Grover and Leskovec, 2016).

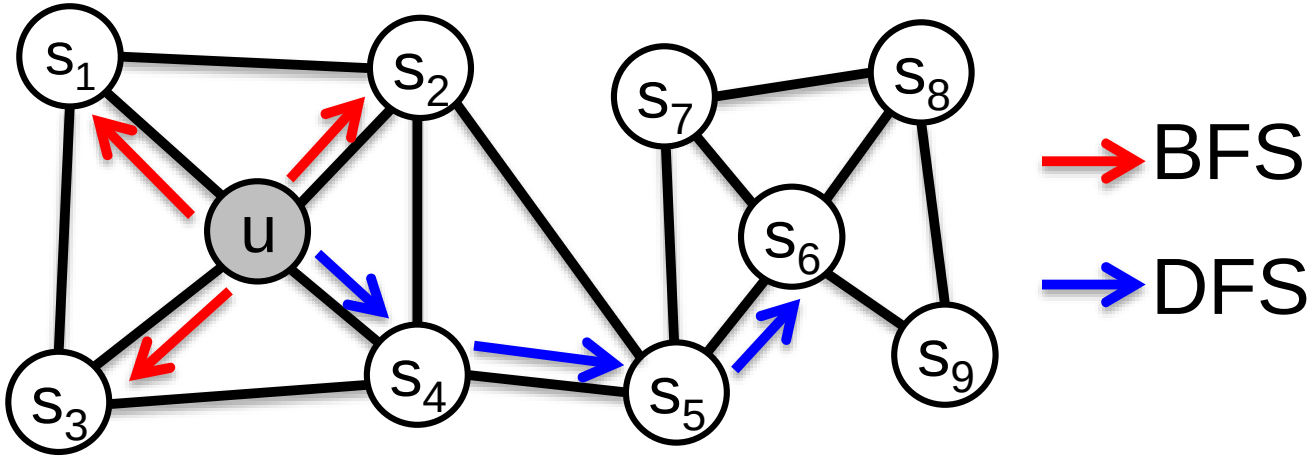
- Return parameter p: return back to the previous node
- In-out parameter q: moving outwards (DFS) vs. inwards (BFS)

$$N_{BFS}(u) = \{ s_1, s_2, s_3 \}$$

Local microscopic view

$$N_{DFS}(u) = \{ s_4, s_5, s_6 \}$$

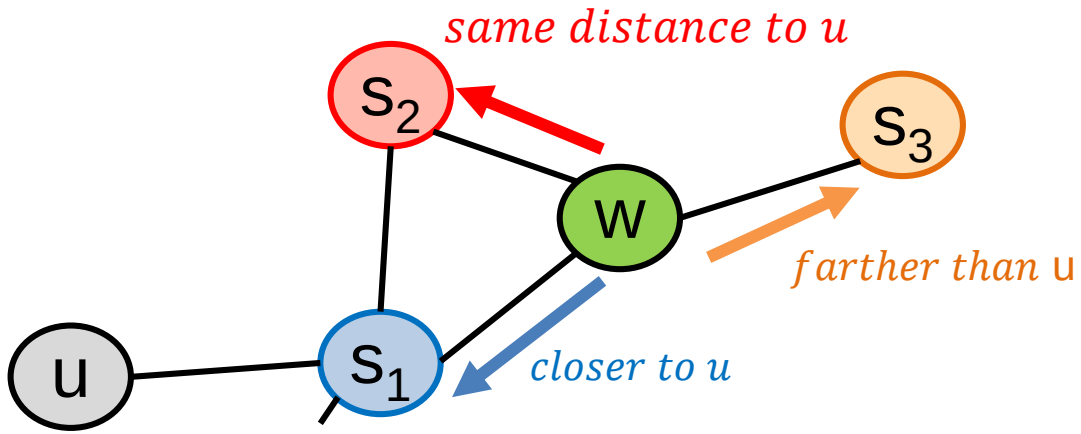
Global macroscopic view



Biased Random Walk

Biased 2nd order random walks explore network neighborhoods:

- Assume the random walk started at u and is now at w
- Neighbors of w can only be (remember where you came from)



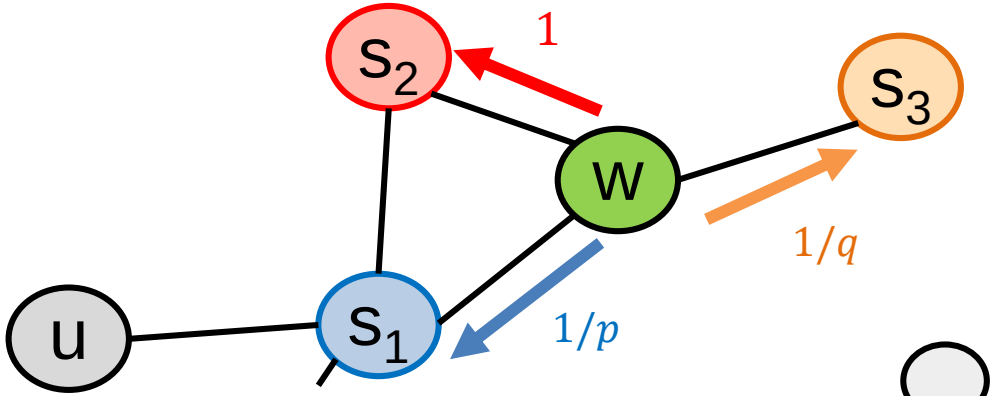
Walker is at W. Where to go next?

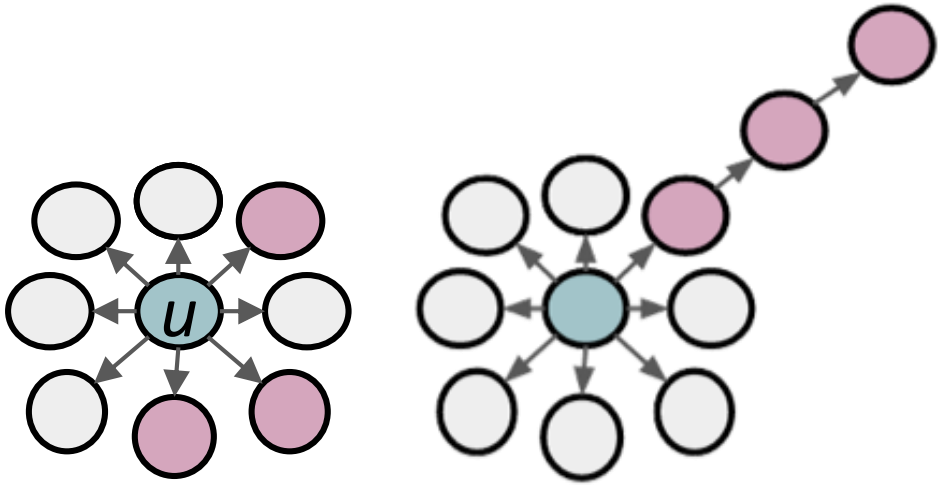
- p … return parameter
- q … "walk away" parameter

Biased Random Walk

Biased 2nd order random walks explore network neighborhoods:

- Assume the random walk started at u and is now at w
- Neighbors of w can only be (remember where you came from)

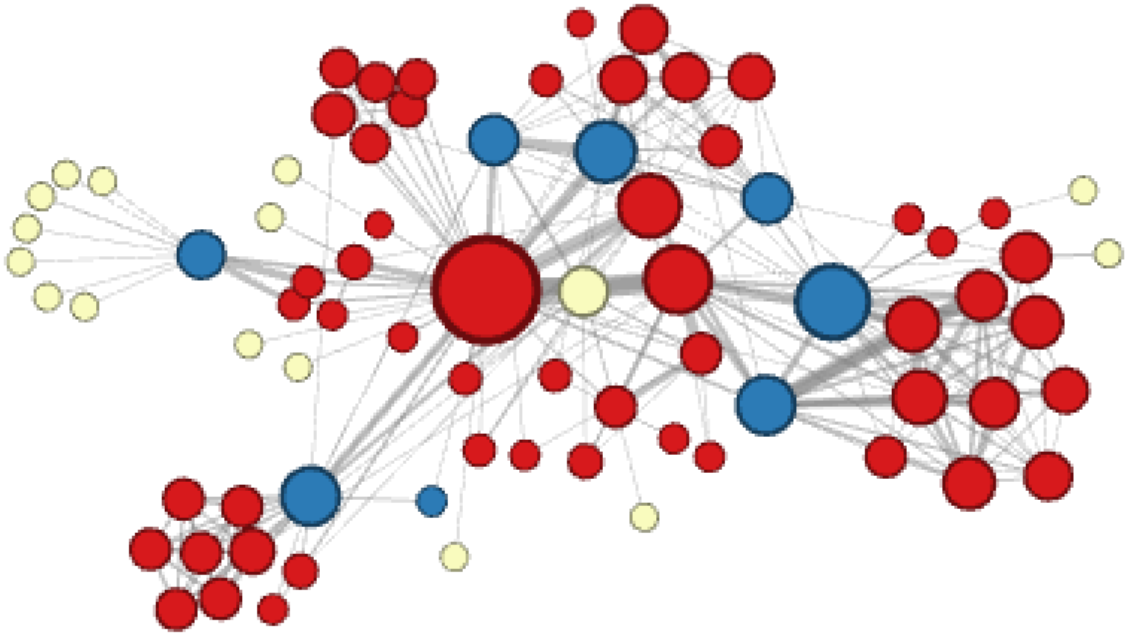




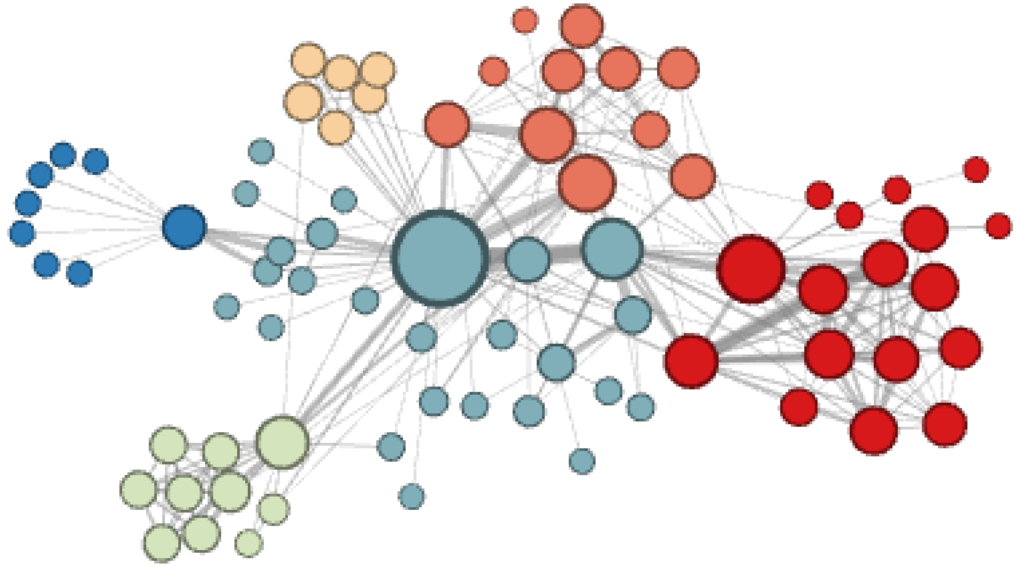
Walker is at W. Where to go next?

- BFS-like walk: Low value of p
- DFS-like walk: Low value of q

Example: Interaction of characters in a novel



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



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

From «Shallow» to «Deep» Encodings

Shallow encoding, i.e., $\text{ENC}(v) = \mathbf{Z}\mathbf{v}$ has some drawbacks:

- O(|V|) parameters are needed, i.e., there is no parameter sharing and every node has its own unique embedding vector of size d
- Inherently “transductive”, i.e., it is impossible to generate embeddings for nodes that were not seen during training
- Does not incorporate node features, i.e., graphs may have features that we can and should leverage upon

POLITECNICO
MILANO 1863

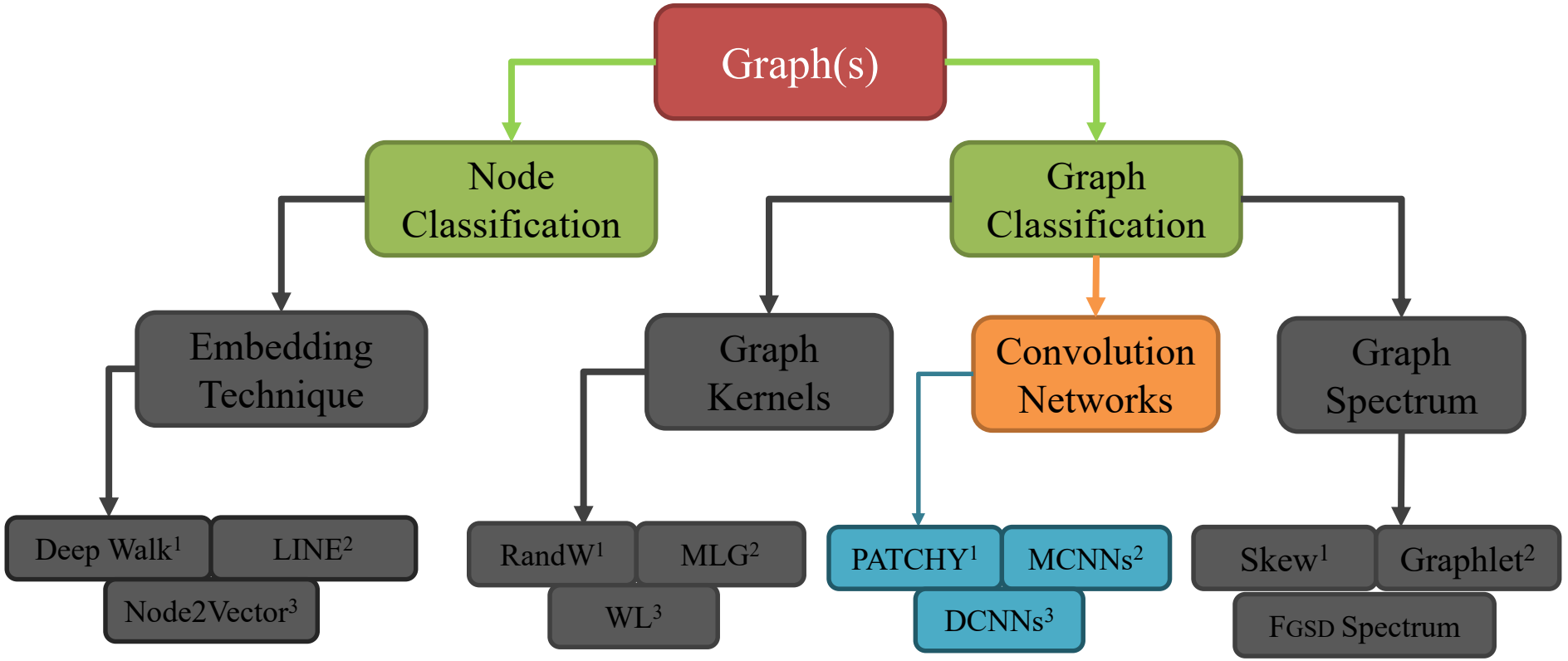
Advances in Deep Learning with Applications in Text and Image Processing

- Graph Neural Networks -

Prof. Matteo Matteucci – *matteo.matteucci@polimi.it*

Department of Electronics, Information and Bioengineering
Artificial Intelligence and Robotics Lab - Politecnico di Milano

Taxonomy of Graph Learning

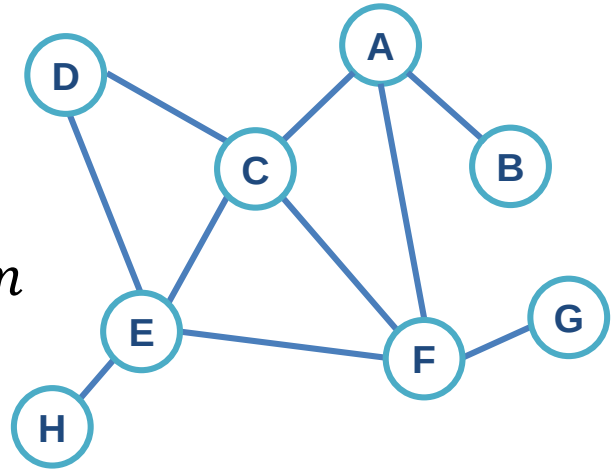


[1] M. Niepert, M. Ahmed, and K. Kutzkov. "Learning convolutional neural networks for graphs." International Conference on Machine Learning. 2016.
[2] Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." Advances in neural information processing systems. 2015.
[3] Atwood, James, and Don Towsley. "Diffusion-convolutional neural networks." Advances in Neural Information Processing Systems. 2016.

Graph Basics

Assume we have a graph G:

- V is the vertex set
- A is the adjacency matrix (assumed binary)
- $X \in R^{m \times |V|}$ is a matrix of node features X_v having $|X_v| = m$



Node features could be of different type

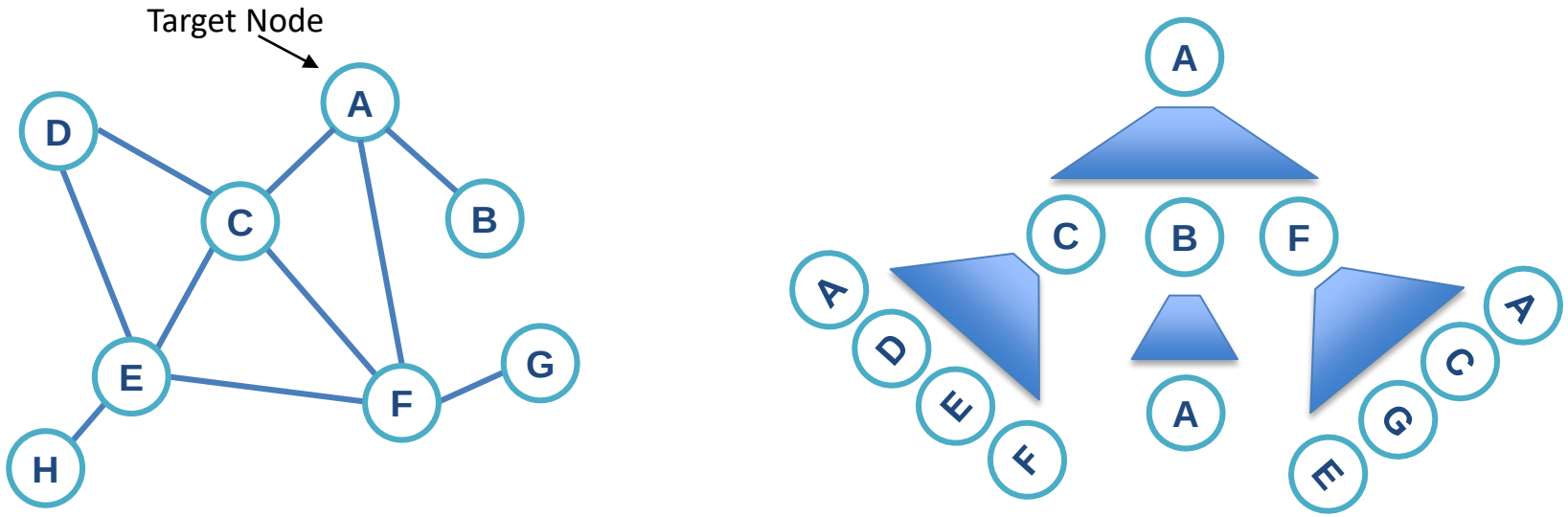
- Categorical attributes, text, image data, etc.
- Vertexex encoded by indicator vectors (i.e., one-hot encoding of each node)

Neighborhood Aggregation

Generate node embeddings based on local neighborhoods.

Nodes aggregate information from their neighbors using neural networks

Every node defines a (unique) computation graph



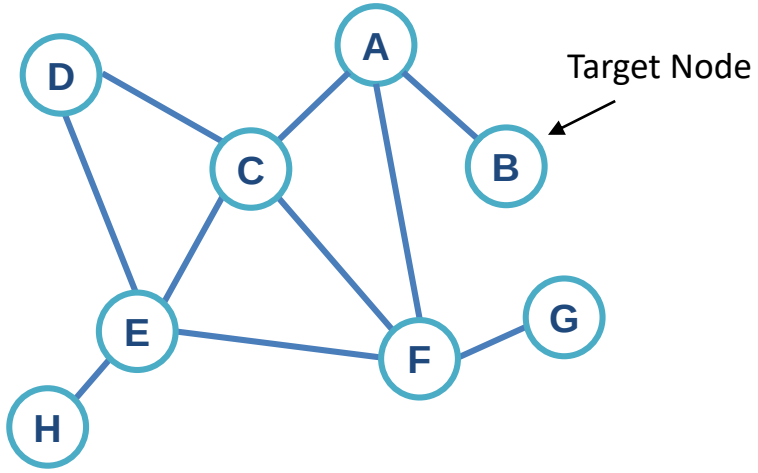
Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

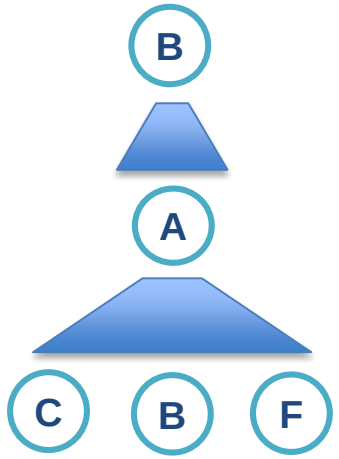
Neighborhood Aggregation

Generate node embeddings based on local neighborhoods.

Nodes aggregate information from their neighbors using neural networks

Every node defines a (unique) computation graph





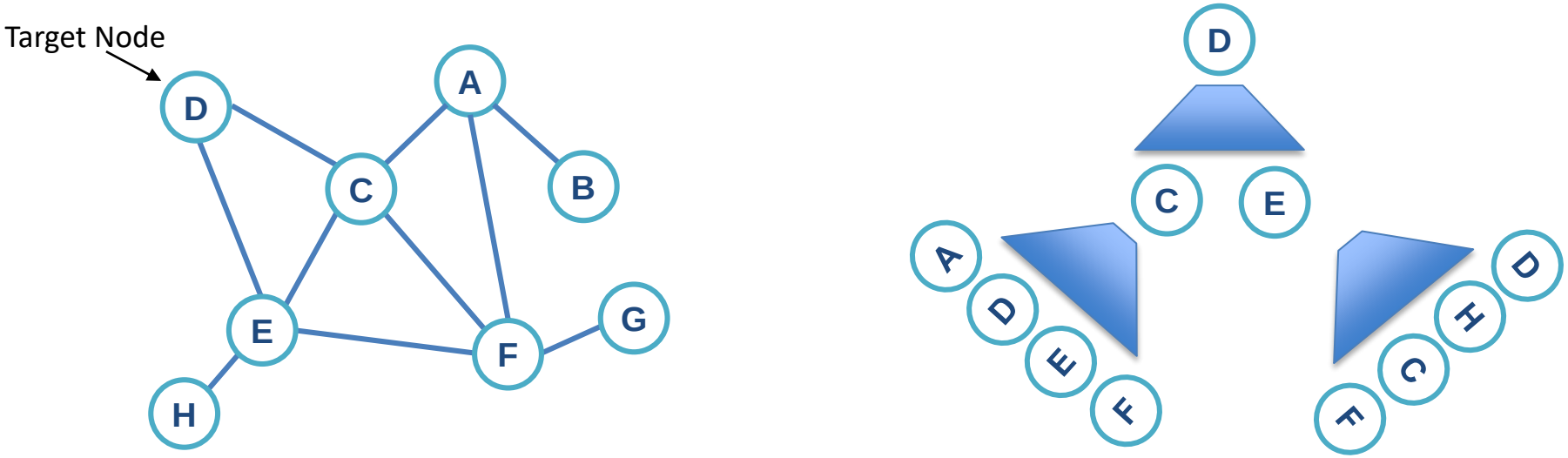
Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Neighborhood Aggregation

Generate node embeddings based on local neighborhoods.

Nodes aggregate information from their neighbors using neural networks

Every node defines a (unique) computation graph



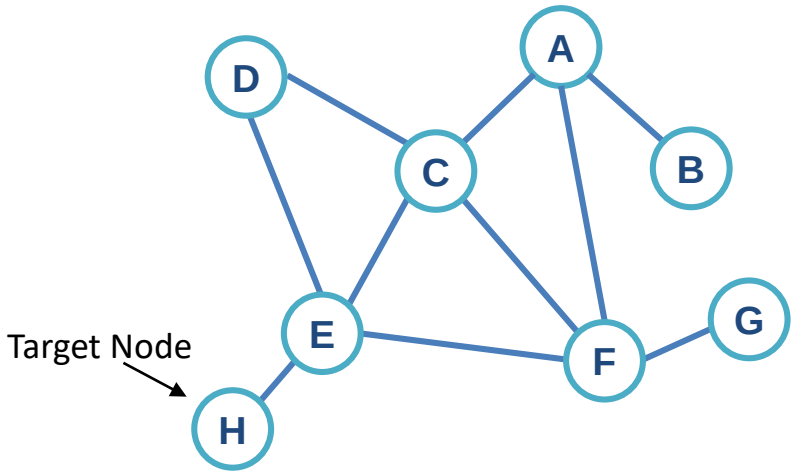
Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

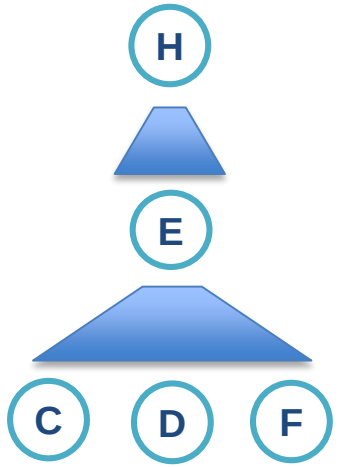
Neighborhood Aggregation

Generate node embeddings based on local neighborhoods.

Nodes aggregate information from their neighbors using neural networks

Every node defines a (unique) computation graph





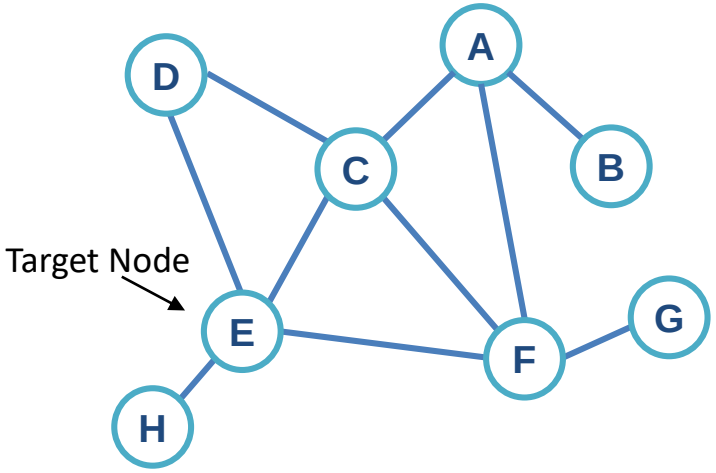
Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

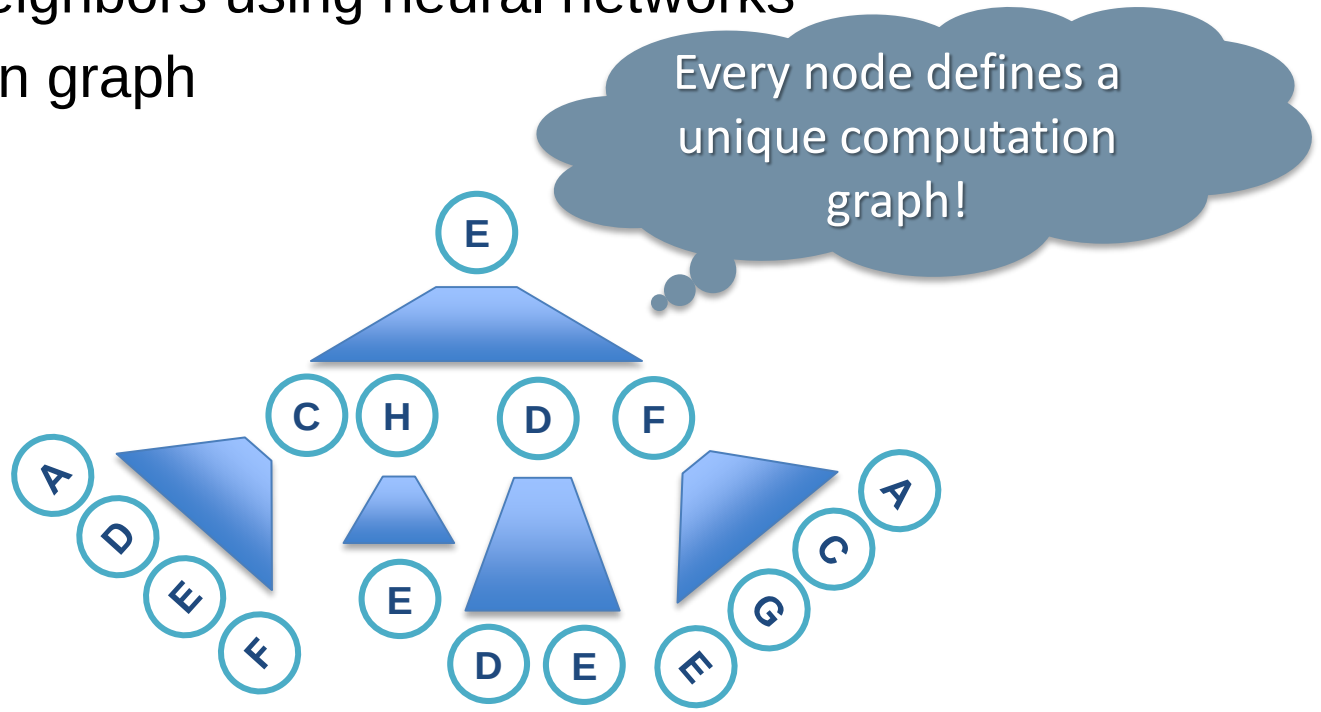
Neighborhood Aggregation

Generate node embeddings based on local neighborhoods.

Nodes aggregate information from their neighbors using neural networks

Every node defines a (unique) computation graph



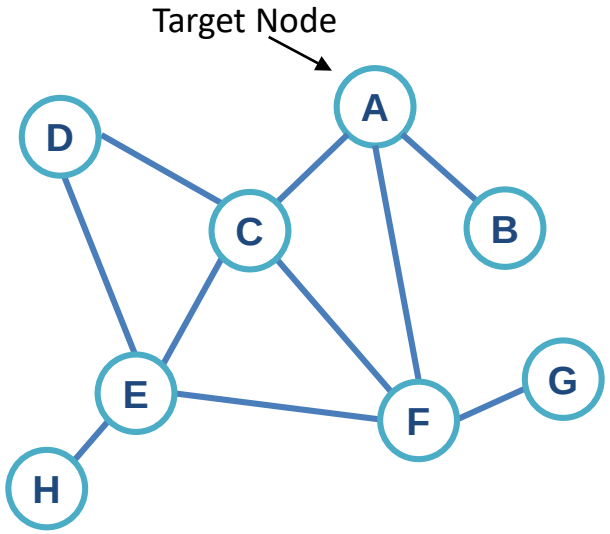


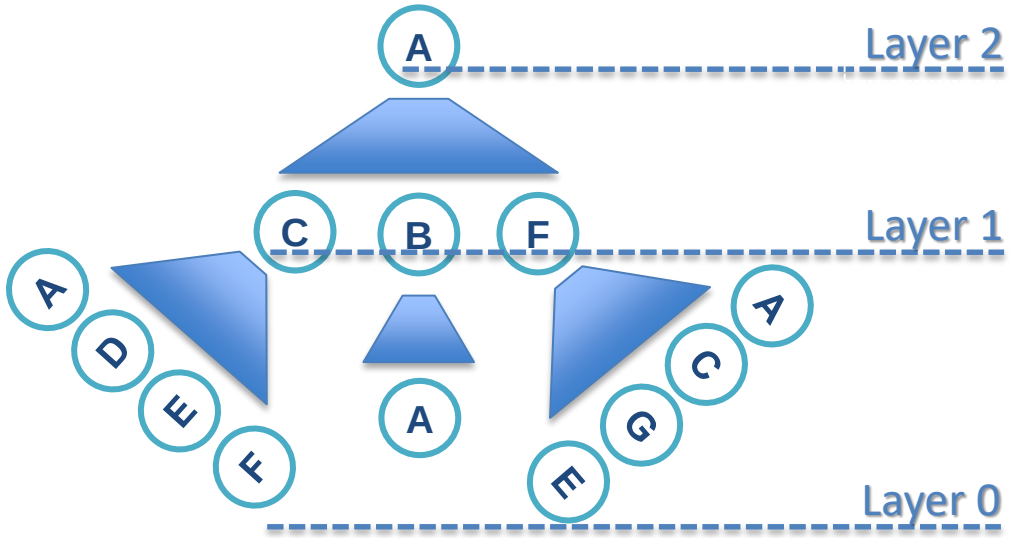
Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Neighborhood Aggregation

Nodes have embeddings at each layer, and the model can be arbitrary depth.

Embedding of node v at "layer-0" is its input feature X_v



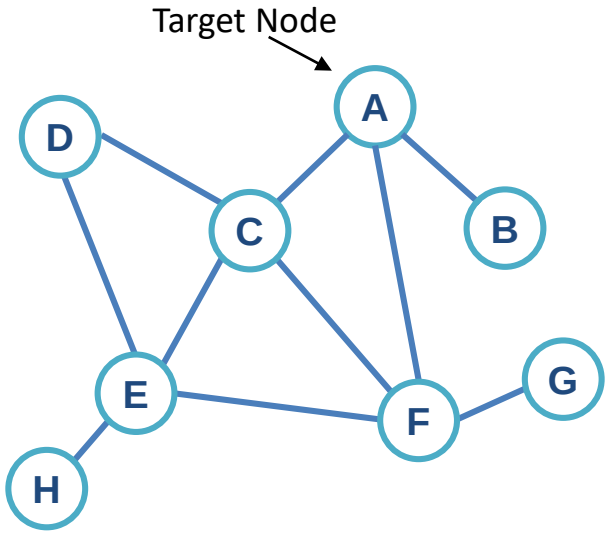


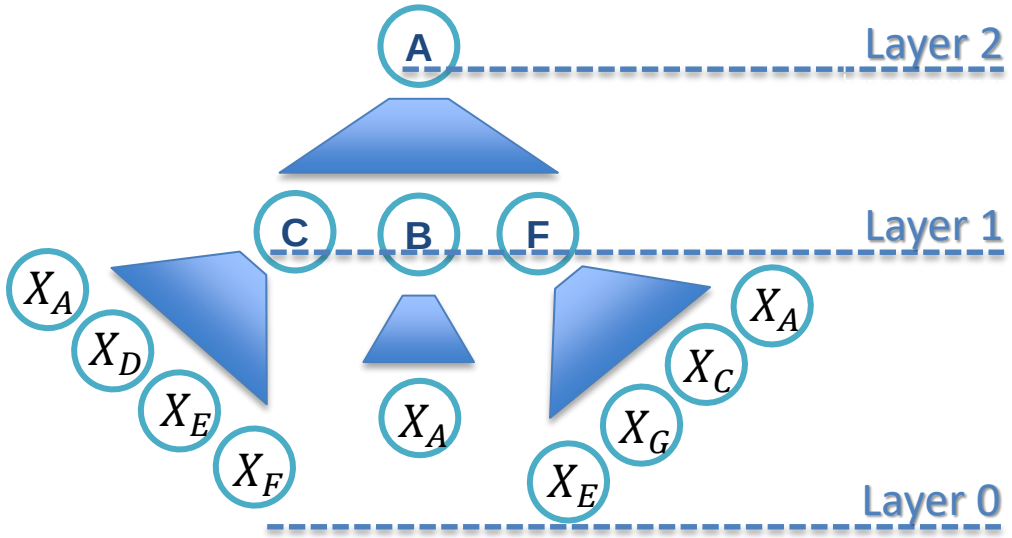
Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Neighborhood Aggregation

Nodes have embeddings at each layer, and the model can be arbitrary depth.

Embedding of node v at "layer-0" is its input feature X_v

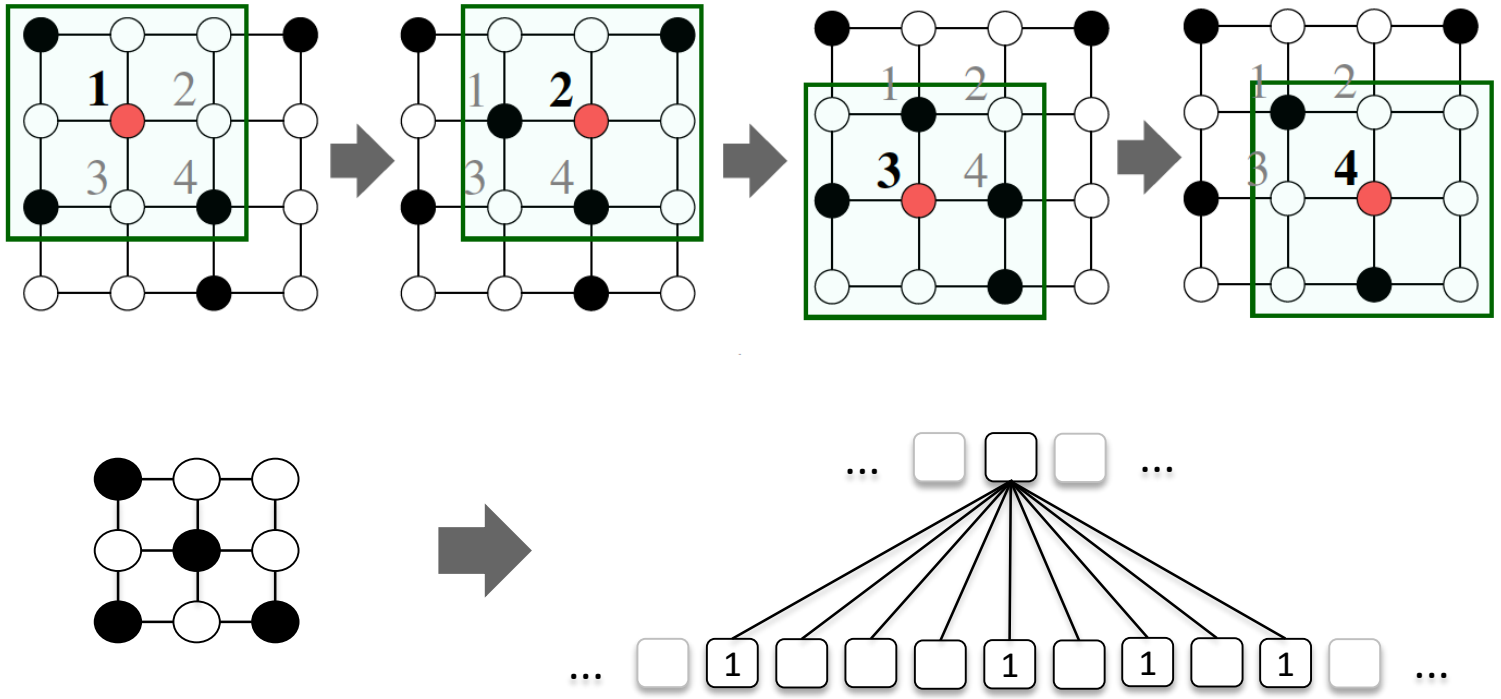




Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Neighborhood «Convolutions»

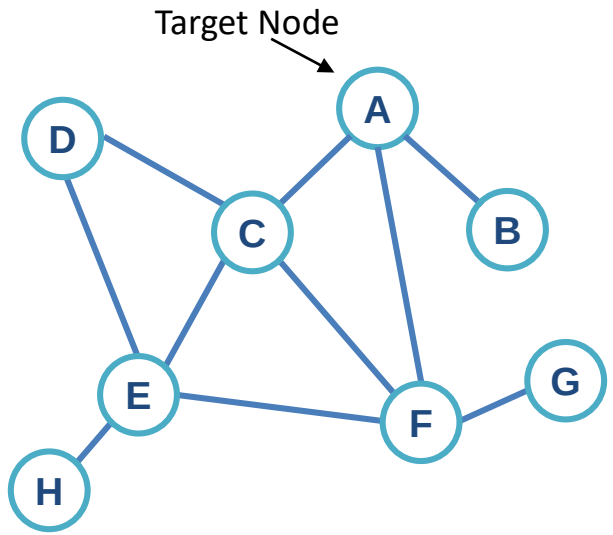
Neighborhood aggregation can be viewed as a center-surround filter; mathematically related to spectral graph convolutions (see Bronstein et al., 2017).

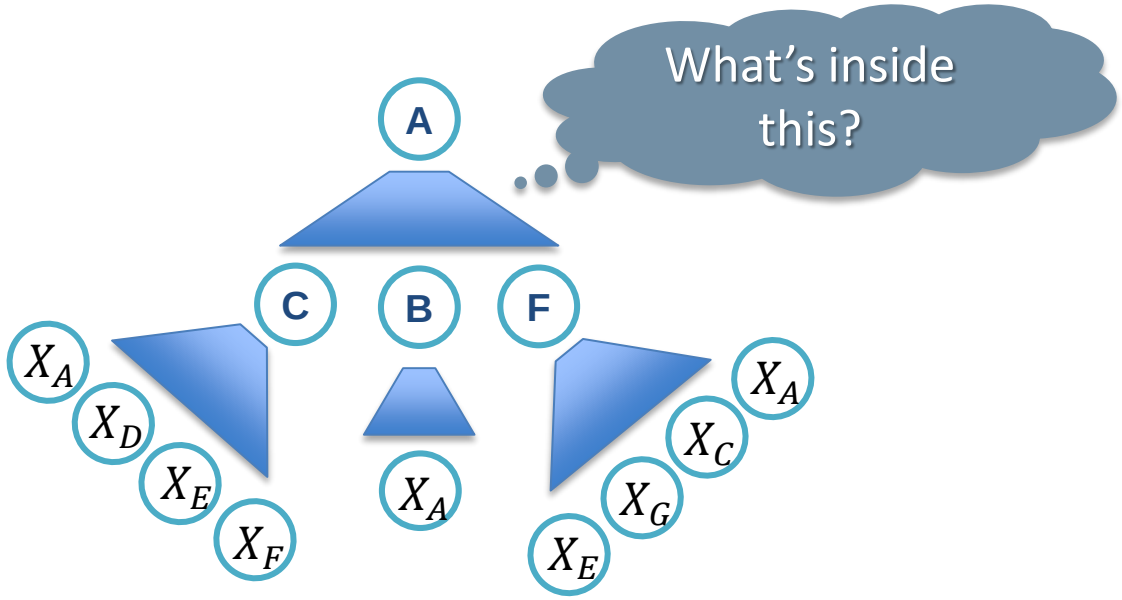


Neighborhood Aggregation

Key distinctions among algorithms are in the way they aggregate information, e.g.,

- Average information from neighbors messages
- Apply a non linear transformataion

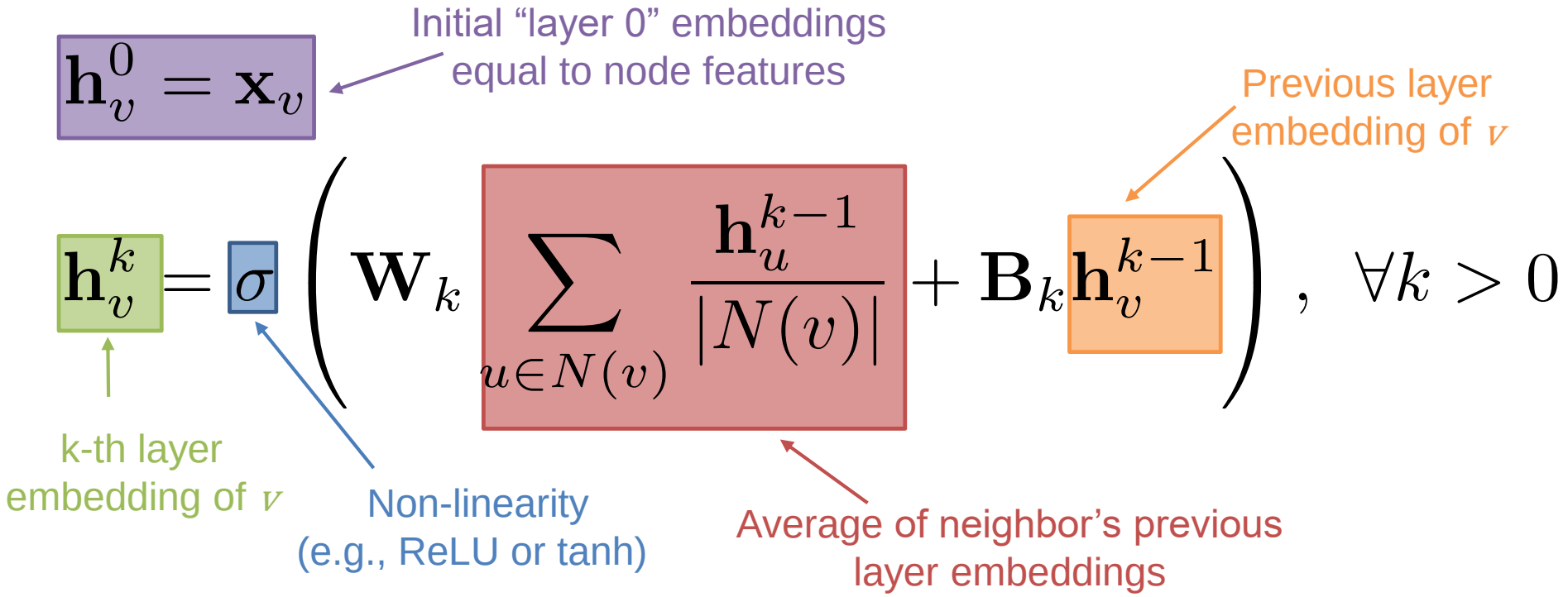




Neighborhood Aggregation

Key distinctions among algorithms are in the way they aggregate information, e.g.,

- Average information from neighbors messages
- Apply a non linear transformataion



Neighborhood Aggregation

Key distinctions among algorithms are in the way they aggregate information, e.g.,

- Average information from neighbors messages
- Apply a non linear transformataion
- Train the embedding to minimize a loss function

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

Trainable matrices
(i.e., what we learn)

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

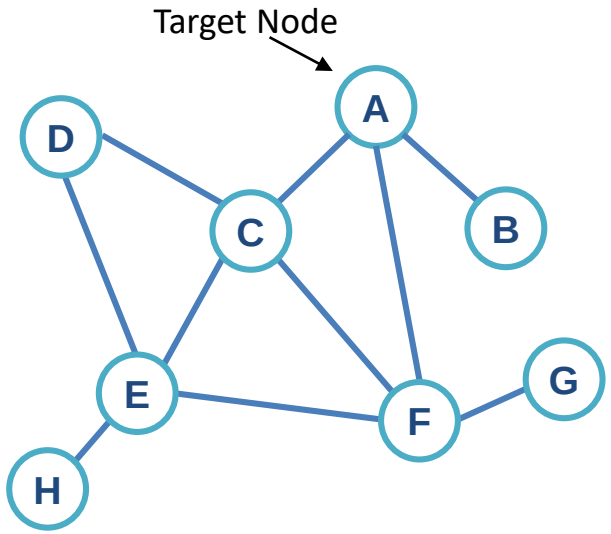
$$\mathbf{z}_v = \mathbf{h}_v^K$$

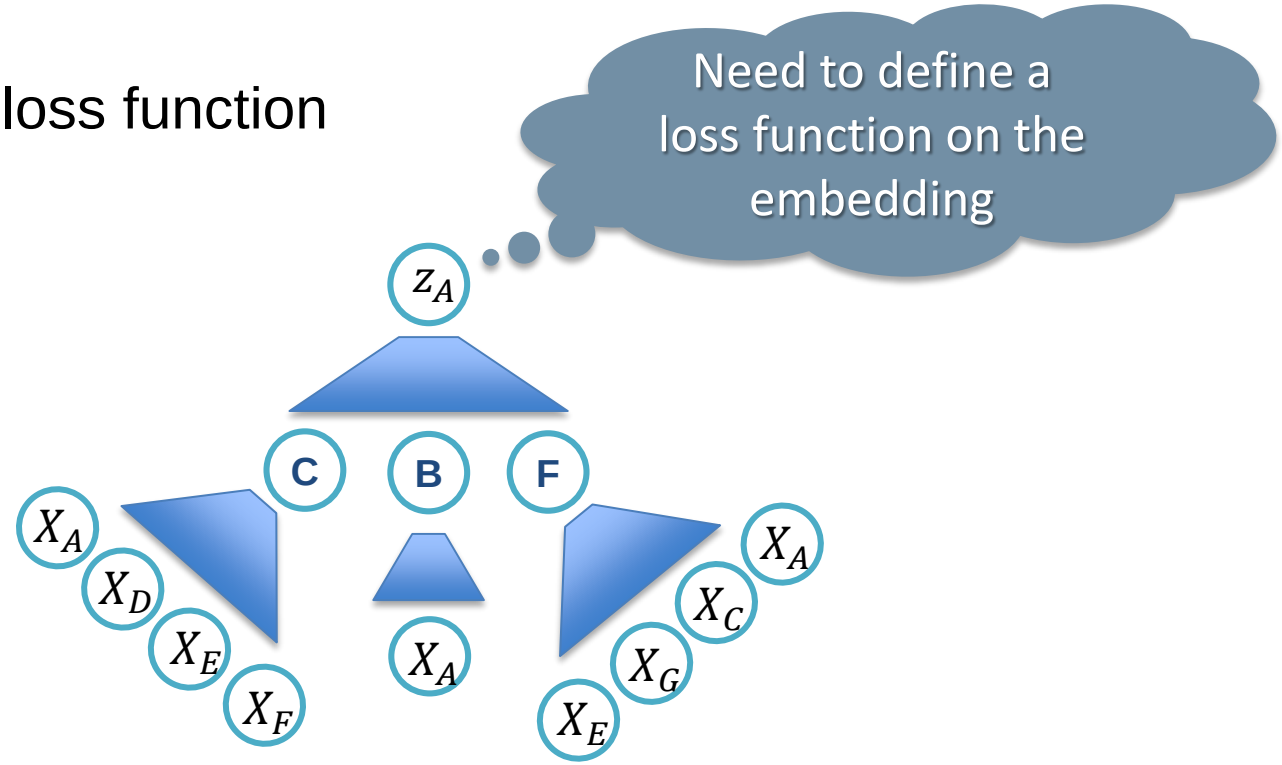
Embedding obtained
after ***K layers***

Neighborhood Aggregation

Key distinctions among algorithms are in the way they aggregate information, e.g.,

- Average information from neighbors messages
- Apply a non linear transformataion
- Train the embedding to minimize a loss function





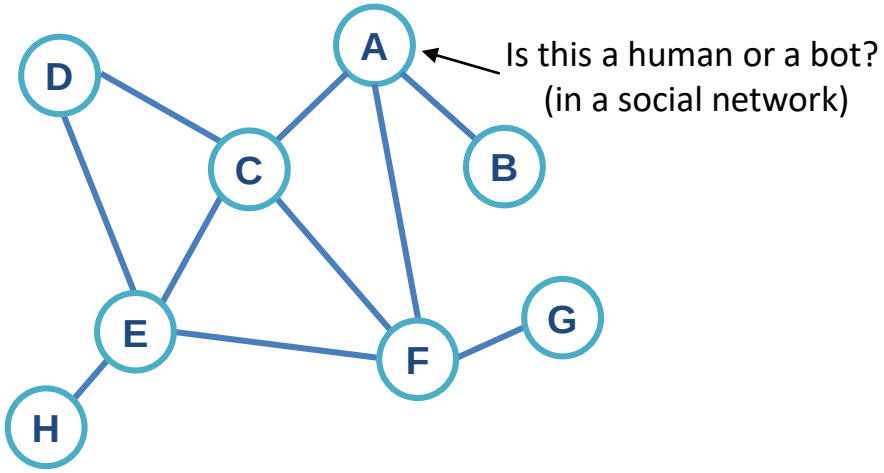
Training the Model

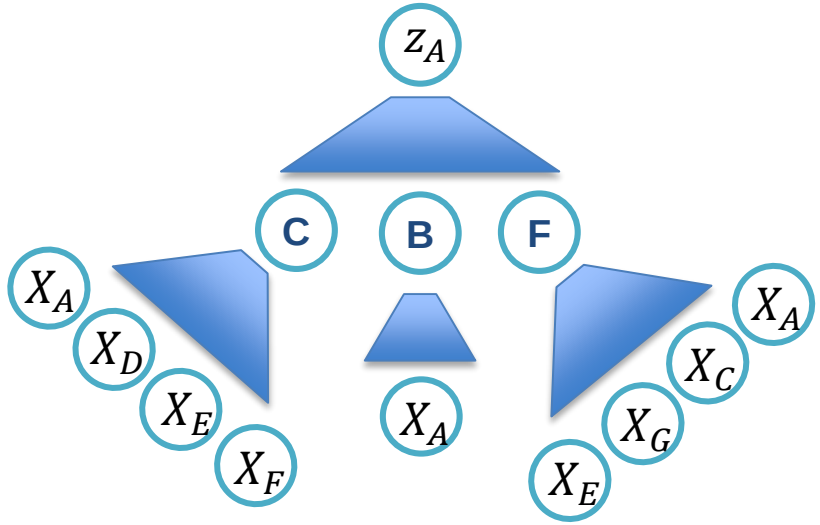
Features are not optimized for a specific task

Train in an ***unsupervised manner*** using only the graph structure [illegible] loss function, e.g., based on:

- Random walks (node2vec, DeepWalk)
- Graph factorization (i.e., “similar” nodes have similar embeddings)

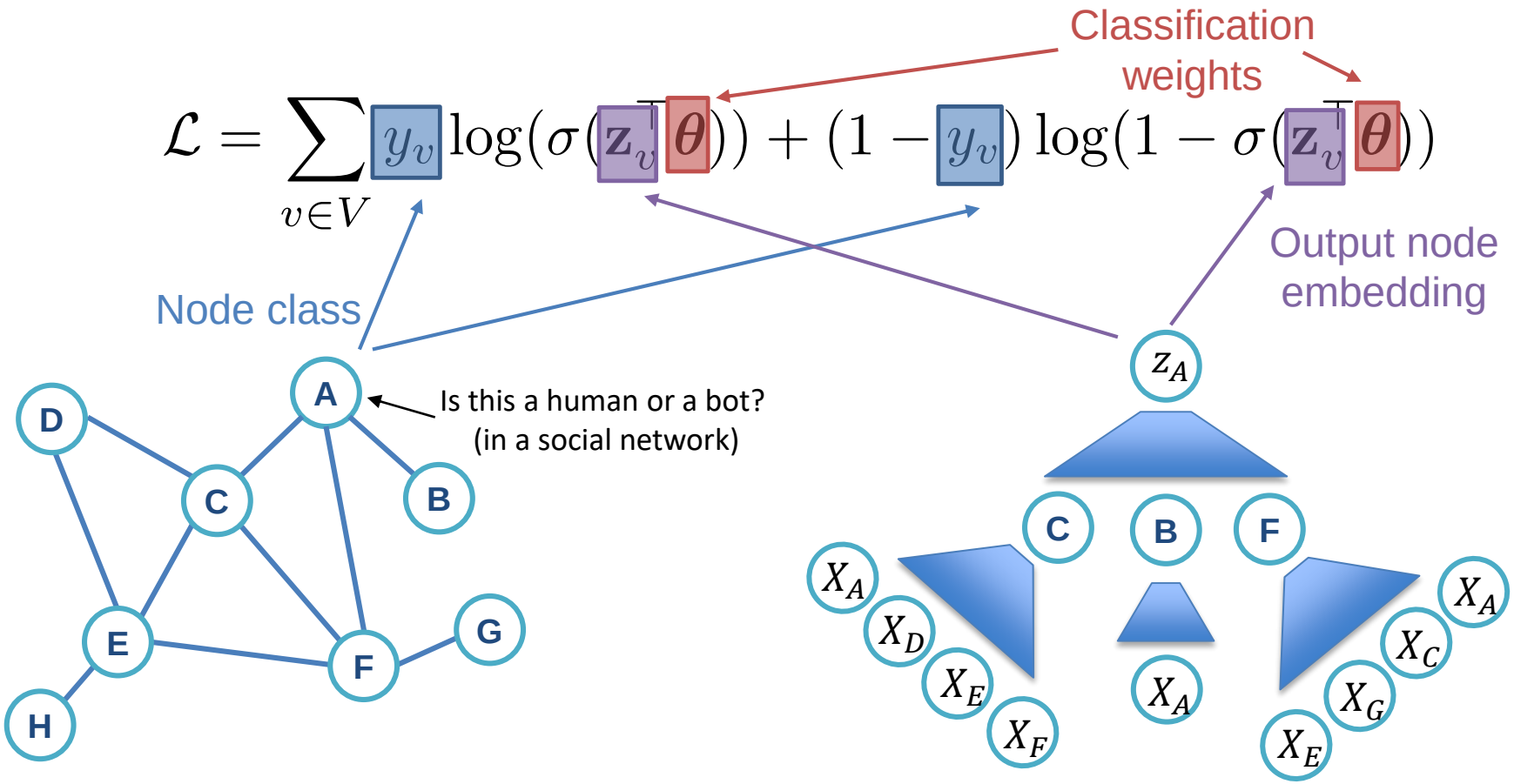
Directly train the model for a ***supervised task*** (e.g., node classification):





Training the Model

Directly train the model for a ***supervised task*** (e.g., node classification):

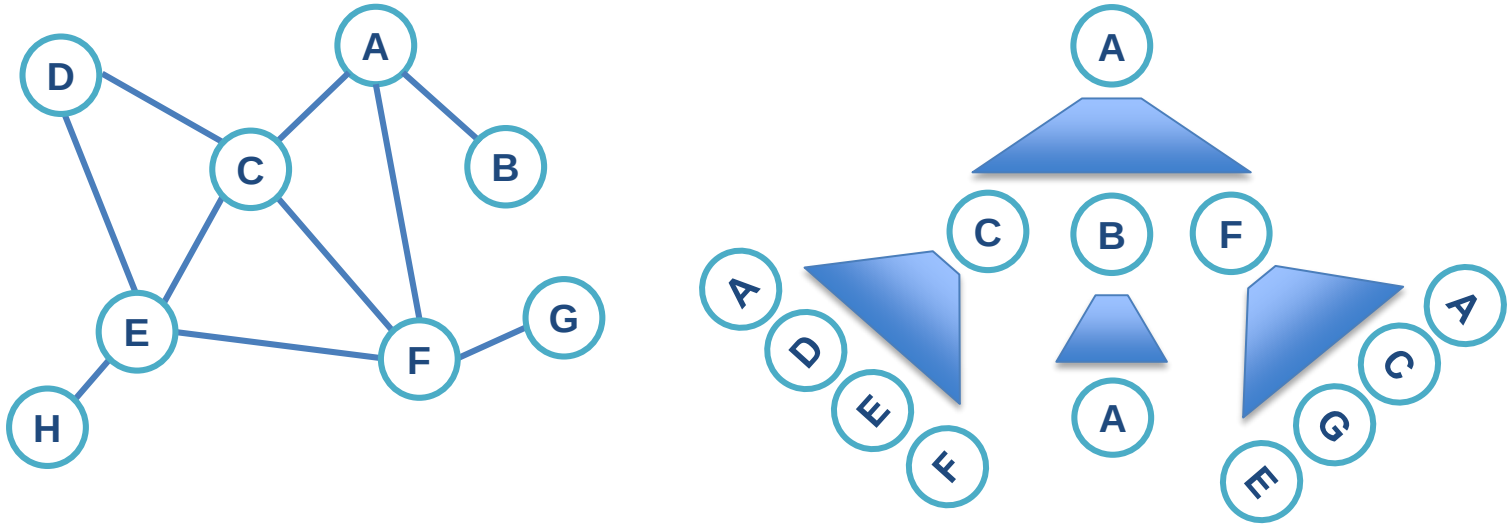


Training and Generalization

- Define a neighborhood aggregation function

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

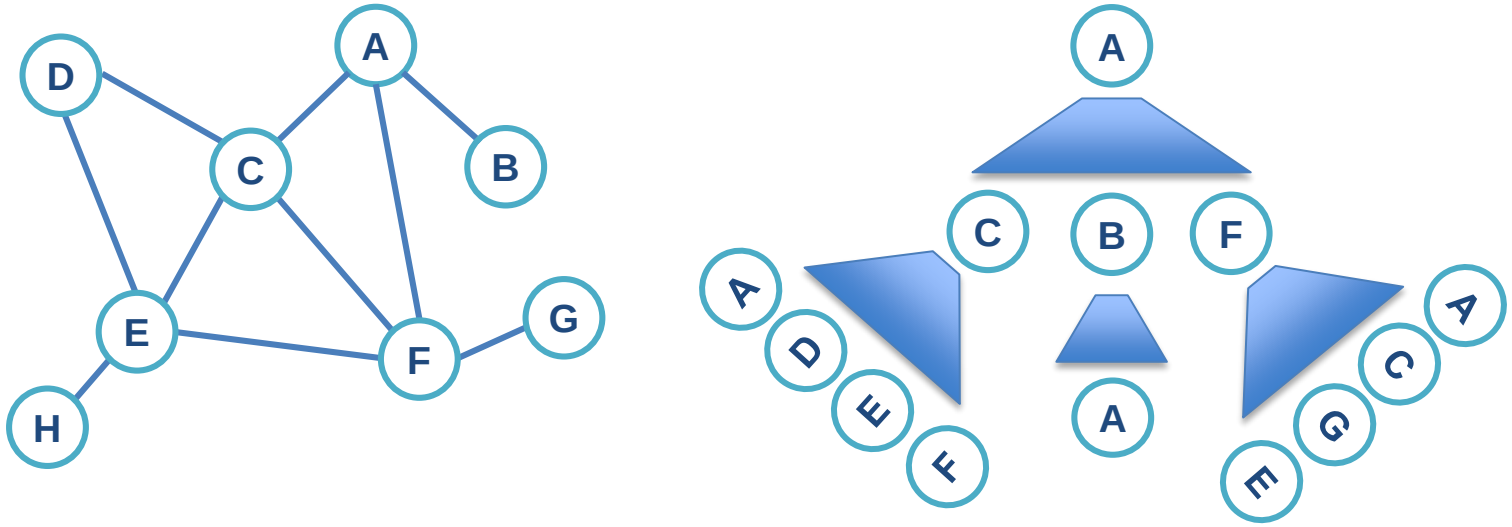
$$\mathbf{z}_v = \mathbf{h}_v^K$$



Training and Generalization

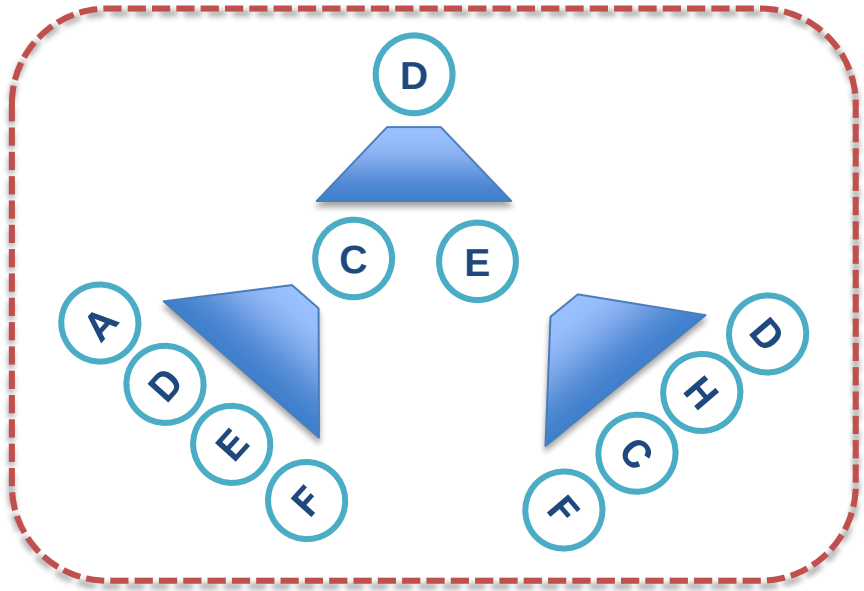
- Define a neighborhood aggregation function
- Define a loss function on the embedding

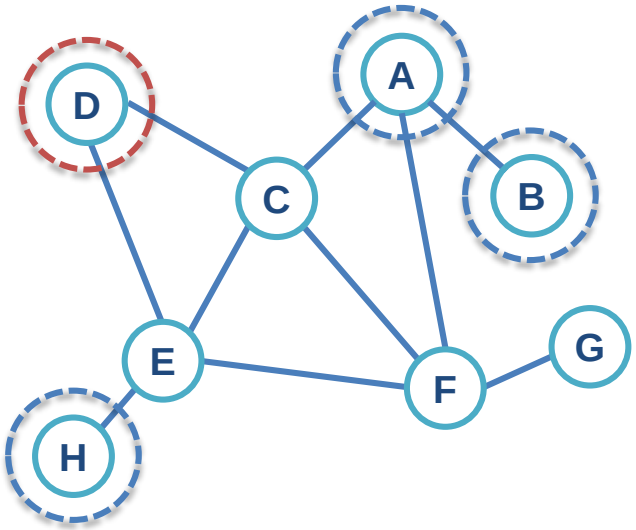
$$\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(\mathbf{z}_v^\top \boldsymbol{\theta})) + (1 - y_v) \log(1 - \sigma(\mathbf{z}_v^\top \boldsymbol{\theta}))$$

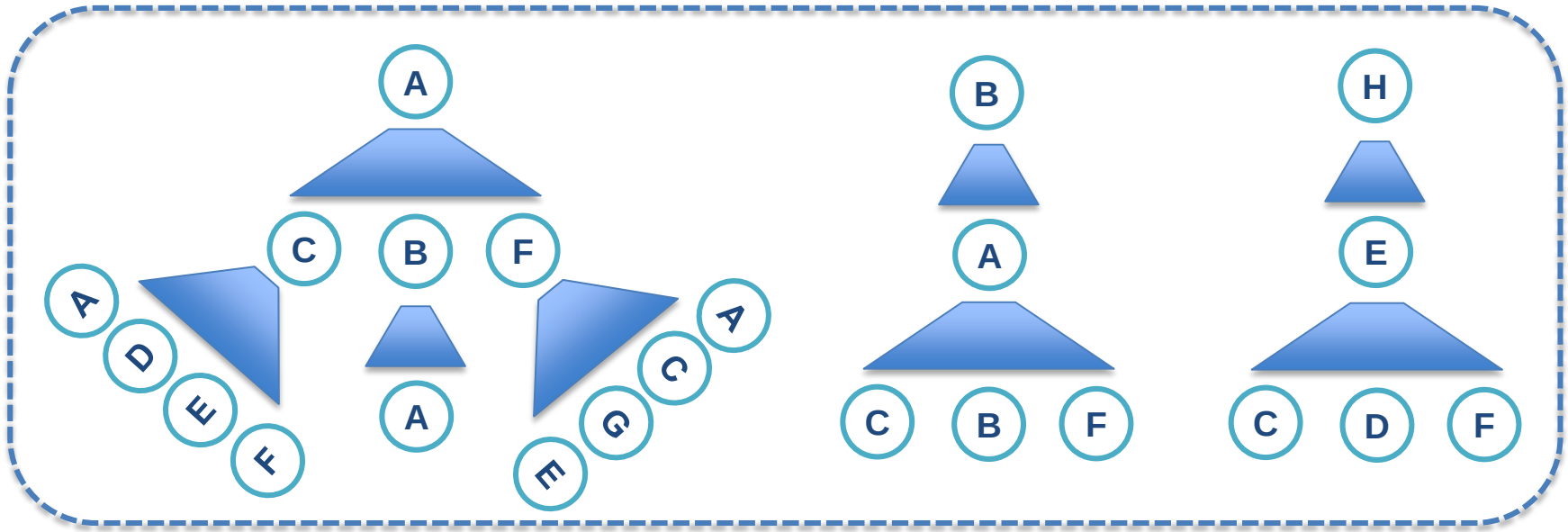


Training and Generalization

- Define a neighborhood aggregation function
- Define a loss function on the embedding
- Train on a set of computing graphs in a batch
- Generate embedding for nodes as needed



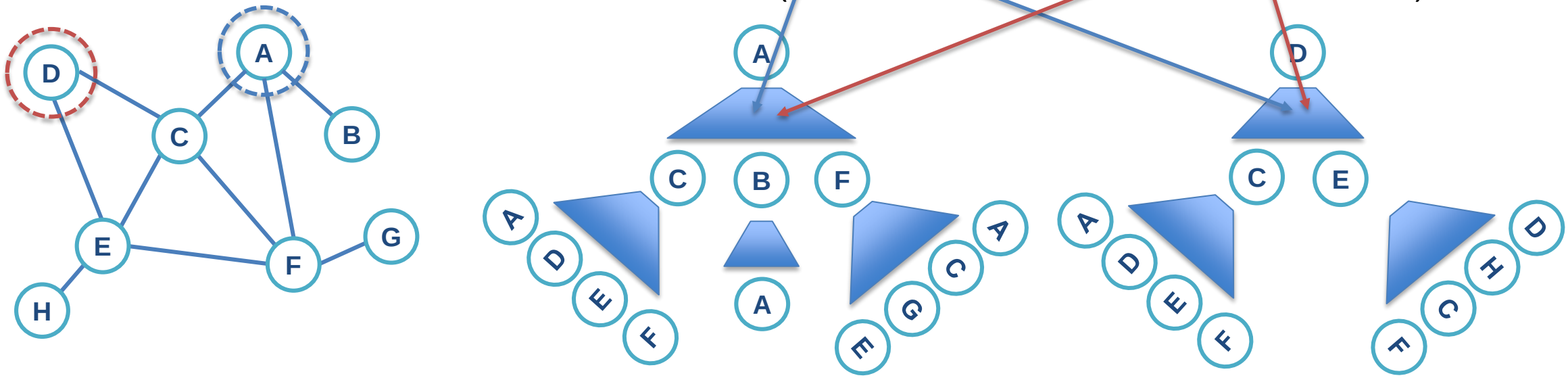




Inductive Capacity of Neighbours Aggregation

The same aggregation parameters are shared for all nodes; the number of model parameters is sublinear in |V| and we can generalize to unseen nodes!

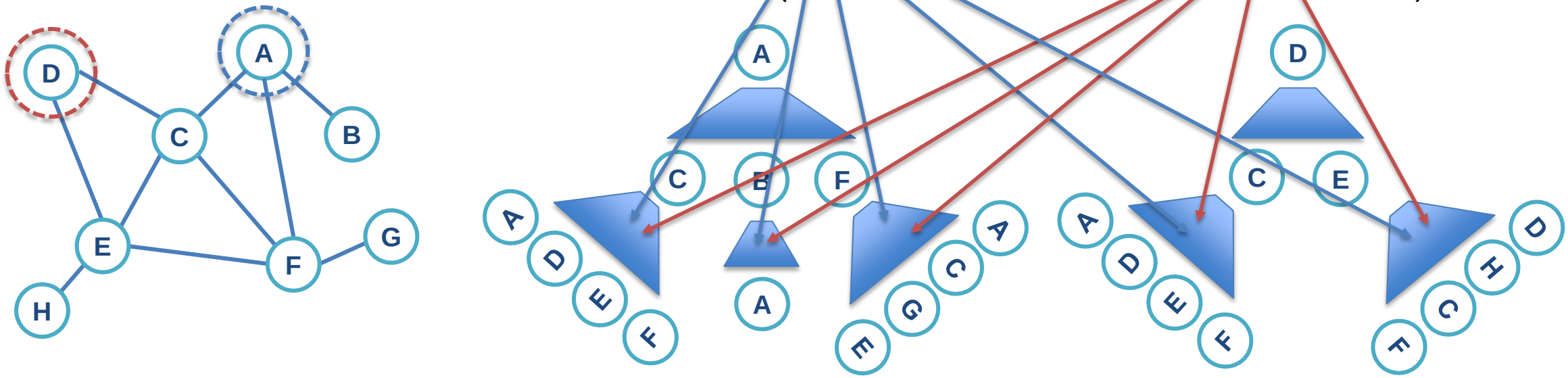
$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$



Inductive Capacity of Neighbours Aggregation

The same aggregation parameters are shared for all nodes; the number of model parameters is sublinear in |V| and we can generalize to unseen nodes!

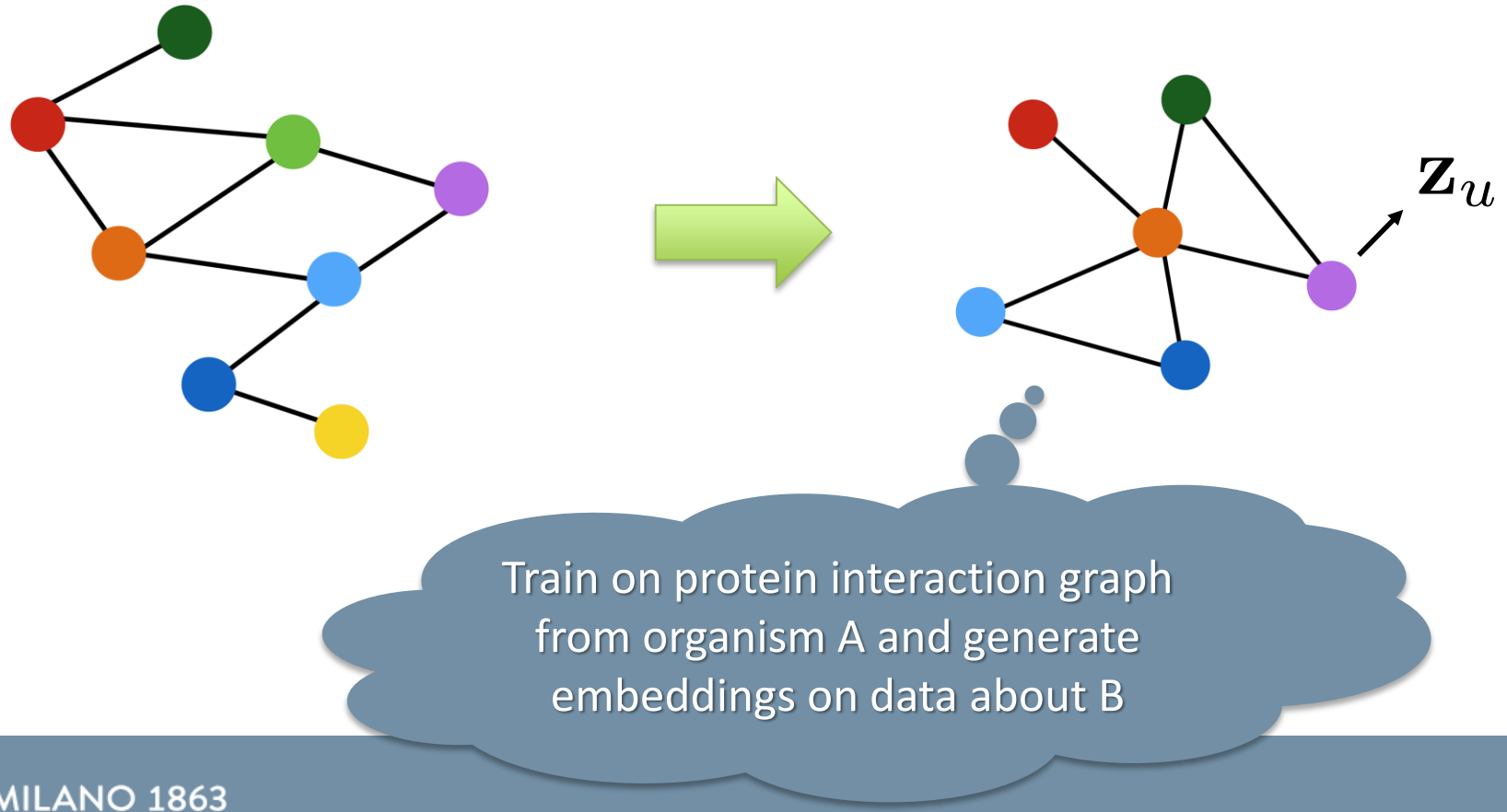
$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$



Inductive Capacity

The inductive capacity allows to:

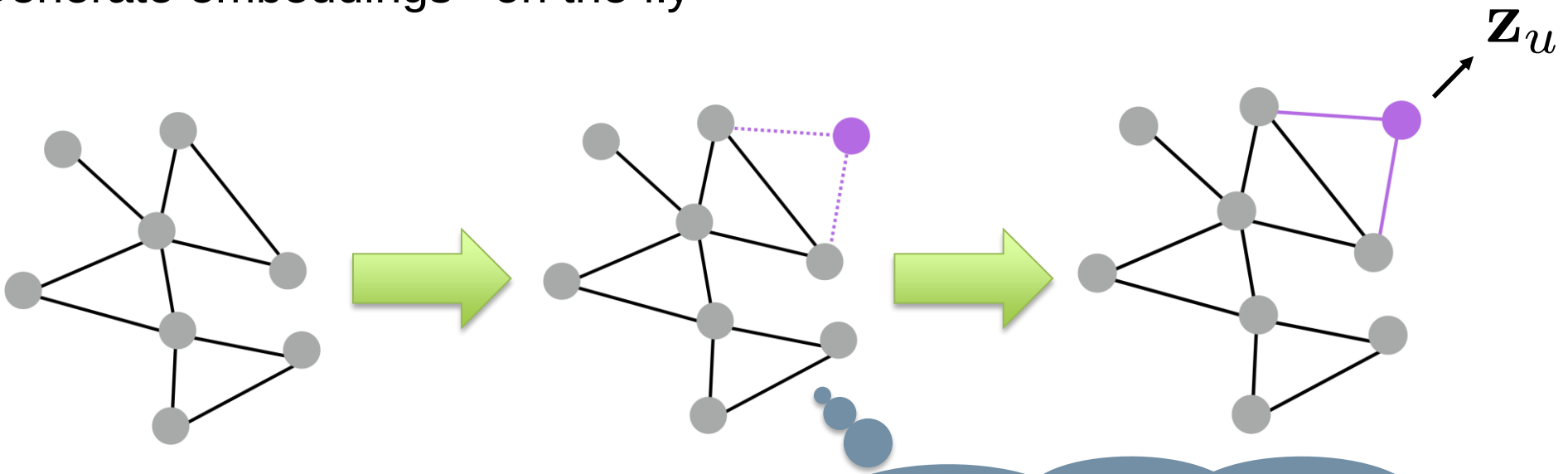
- Train on one graph and generalize to a new one



Inductive Capacity

The inductive capacity allows to:

- Train on one graph and generalize to a new one
- Generate embeddings «on the fly»



New data appears all the days, e.g.,
Reddit, YouTube, GoogleScholar, ….

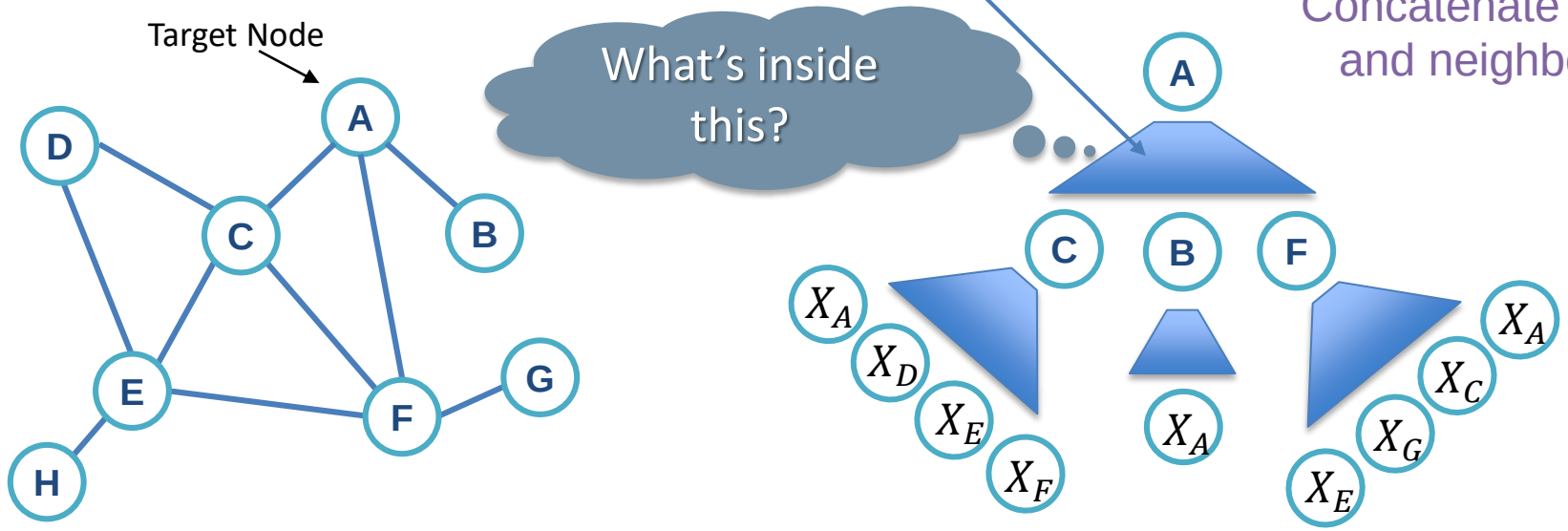
GraphSAGE

We can go beyond simple weighted average

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

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

Concatenate self embedding and neighbor embedding



Material from: Hamilton et al., 2017. Inductive Representation Learning on Large Graphs. *NIPS.*

GraphSAGE

What about convolutions?

We can go beyond simple weighted average:

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

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

Concatenate self embedding and neighbor embedding

- Mean: $\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$
- Pool: transform neighbor vectors and apply symmetric vector function

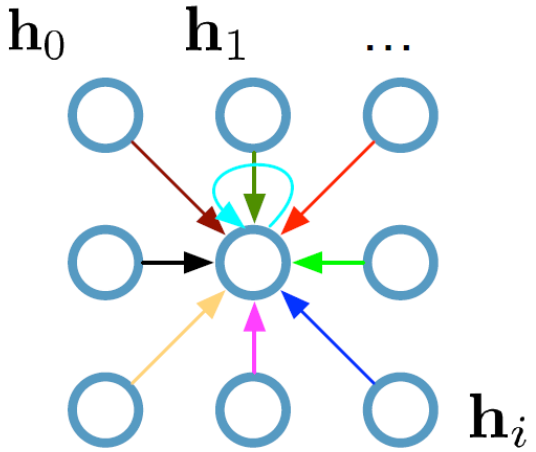
$$\text{AGG} = \gamma\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

Element-wise mean/max

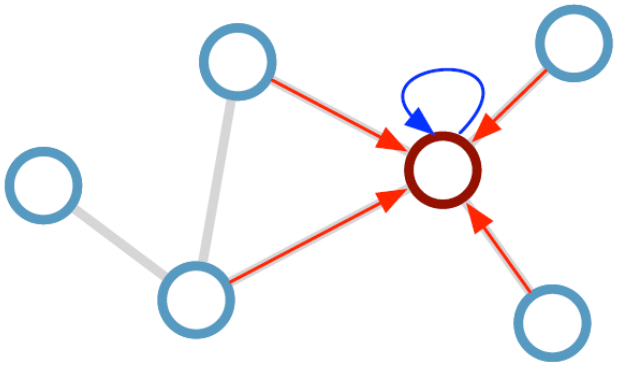
- LSTM: apply LSTM to random permutation of neighbors.

$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

From 2D Convolutions to Graphs Convolutions



$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$



$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (per edge)

Graph Convolutional Networks

Graph Convolutional Networks (GCNs) are a variation on neighborhood aggregation:

Basic Neighborhood Aggregation

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

More parameter sharing. Down-weights hight degree neighbors.

Same matrix for self and neighbor embeddings

GCN Neighborhood Aggregation

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

Per-neighbor normalization

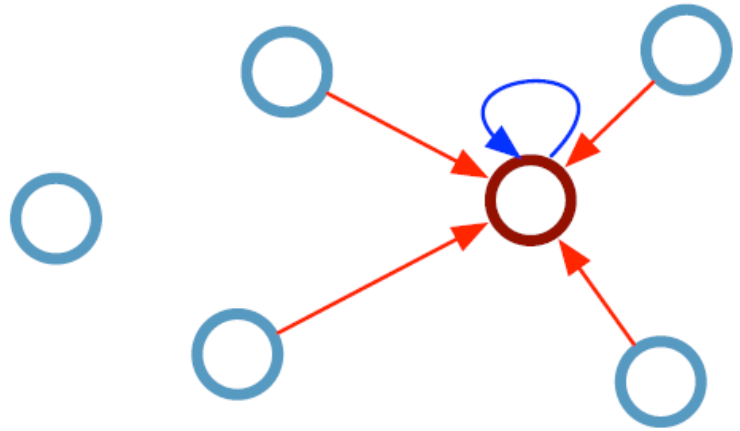
Kipf et al., 2017. Semisupervised Classification with Graph Convolutional Networks. *ICLR*.

Vectorized Implementation (Faster!)

$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)^T}, \ldots, \mathbf{h}_N^{(l)^T}]^T$$

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

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

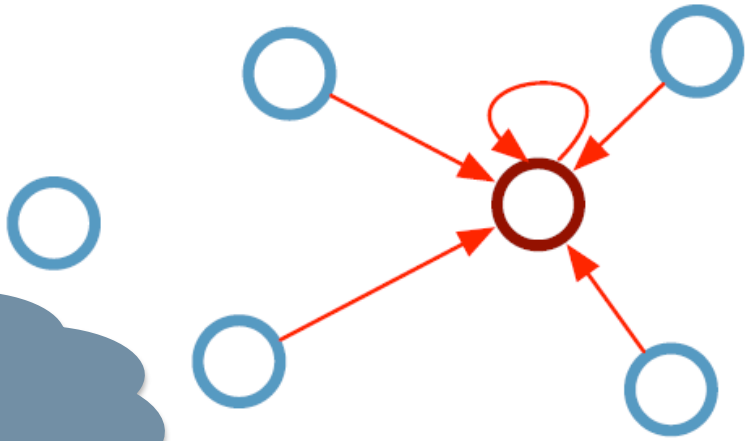


$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

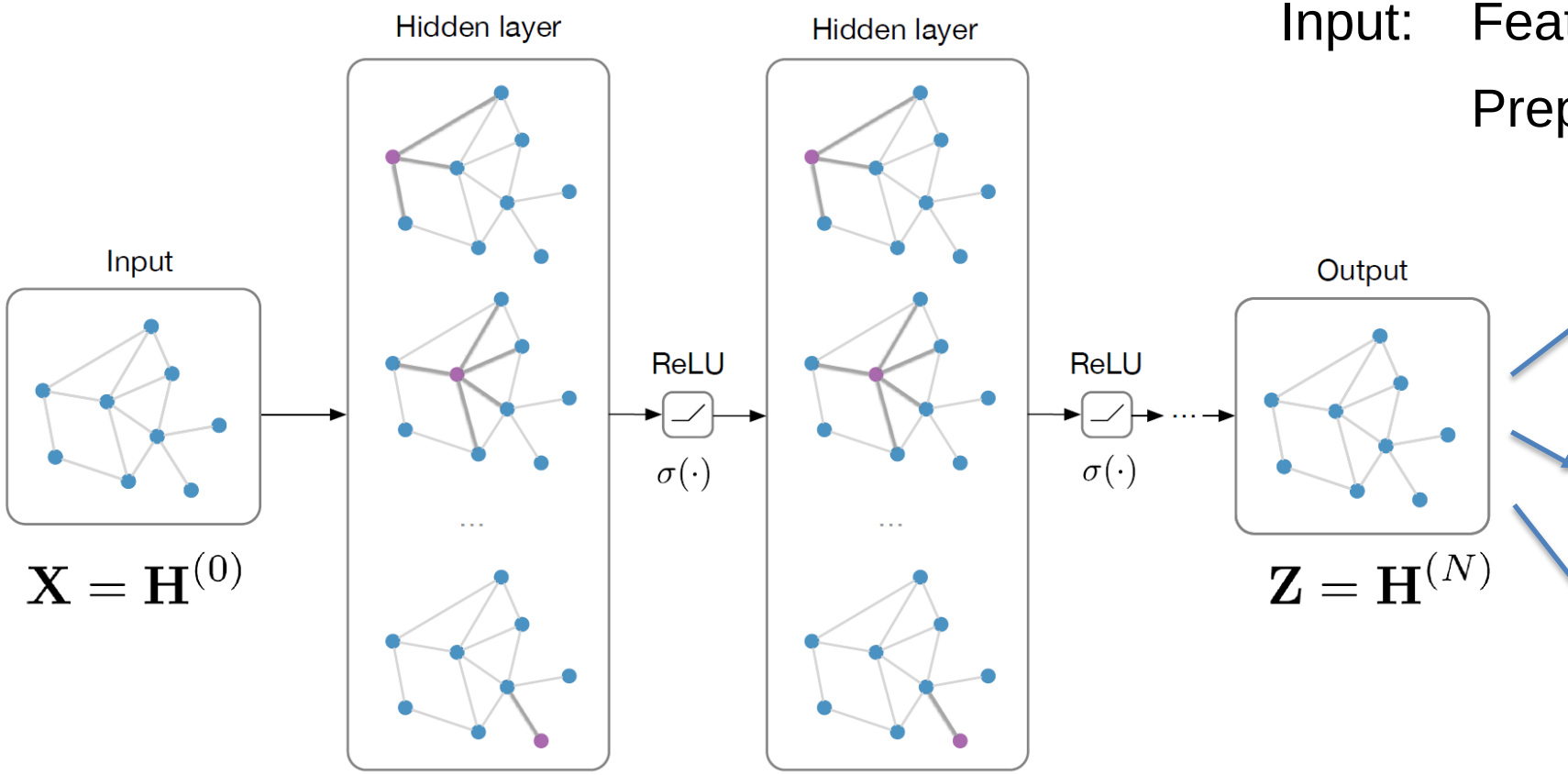
$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_N)\tilde{\mathbf{D}}^{-\frac{1}{2}}$$

$$\tilde{D}_{ii} = \sum_j (A_{ij} + \delta_{ij})$$

A is generally sparse, obtained via fast sparse multiplications!



Graph Convolutional Networks Applications



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Input: Features matrix $\mathbf{X} \in \mathbf{R}^{\mathbf{m} \times |\mathbf{V}|}$

Preprocessed adiacency $\widehat{\boldsymbol{A}}$

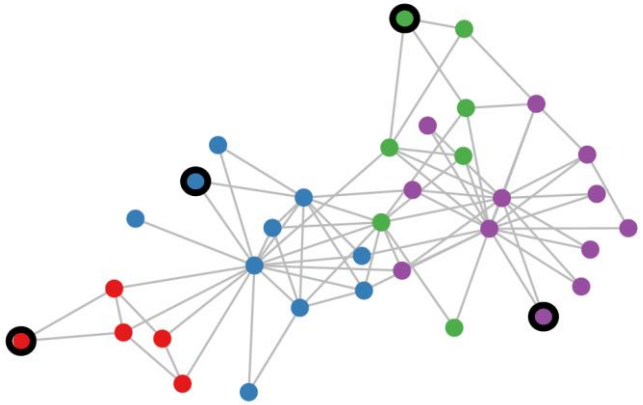
Node Classification: $\text{softmax}(\mathbf{z_n})$
(e.g., Kipf & Welling ICLR 2017)

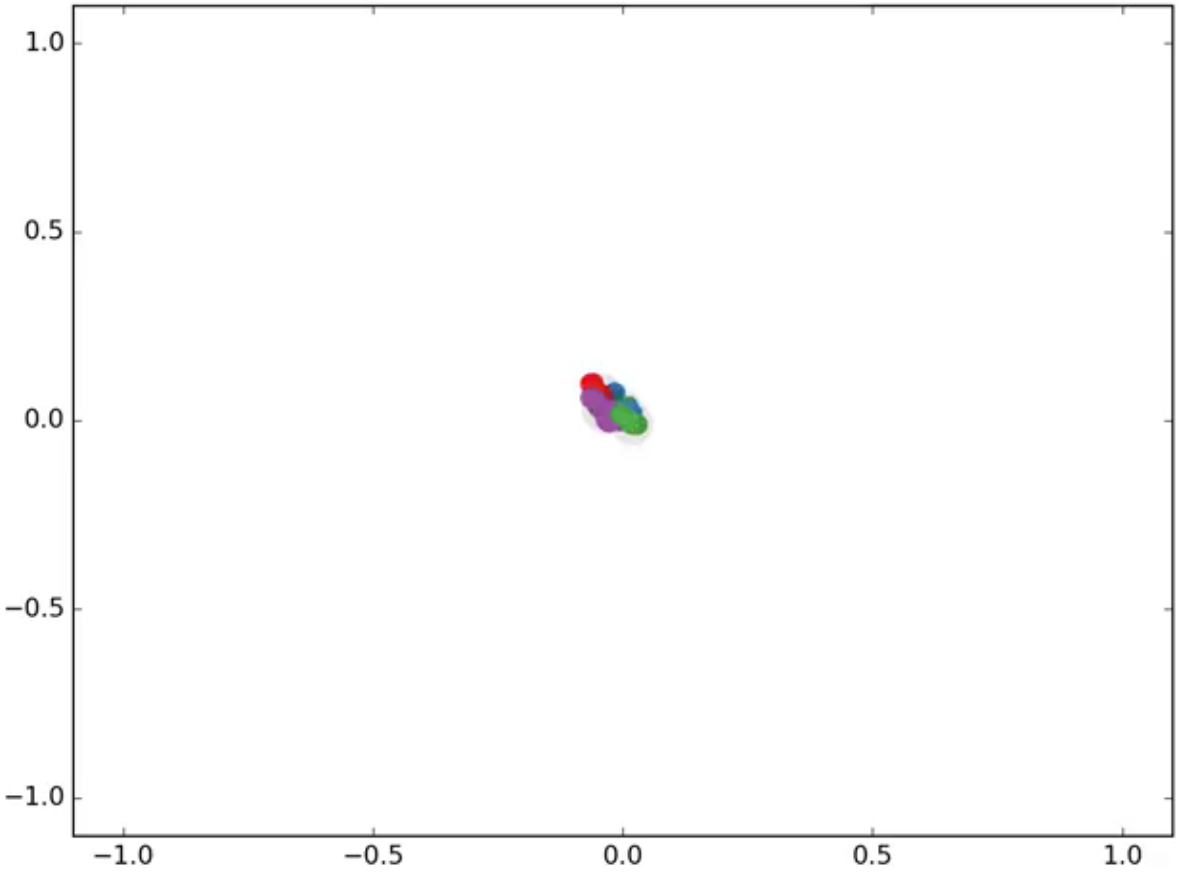
Graph Classification: $\text{softmax}(\sum_{\boldsymbol{n}} \boldsymbol{z_n})$
(e.g., Duvenaud et al. NIPS 2015)

Link Prediction: $p(A_{ij}) = \sigma(\boldsymbol{z_i^T z_j})$
(e.g., Kipf & Welling NIPS BDL 2016)

Exaple: Semi-supervised Learning

Let assume only few nodes are labeled and initialize the network weights randomly





Evaluate loss on labeled nodes only

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

Exaple: Semi-supervised Learning (Citation network)

Citation networks are a typical example of social networks:

- nodes are papers, edges are citation links
- bag-of-words features on nodes (optional)

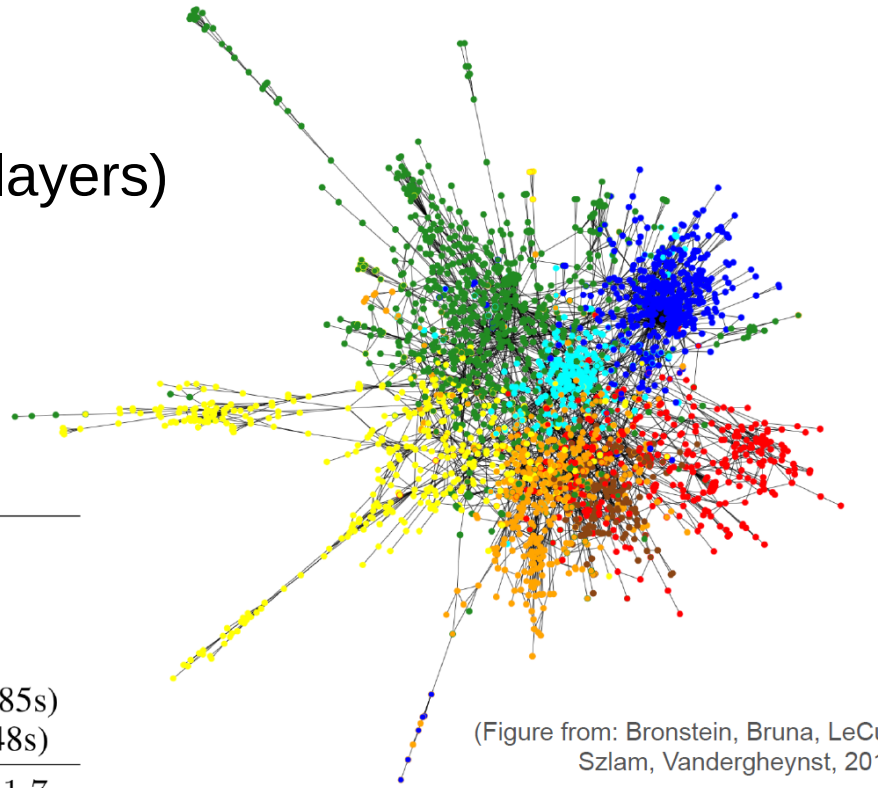
Graph convolutional network to predict categories (2 layers)

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$$

Classification results (accuracy)

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [24]	59.6	59.0	71.1	26.7
LP [27]	45.3	68.0	63.0	26.5
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

no input features



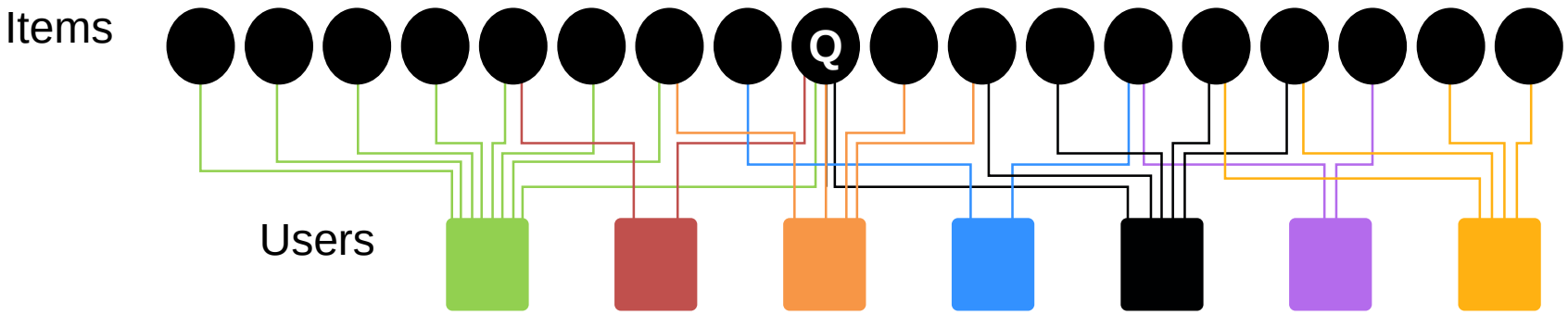
(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

Example: Recommendation System

Recommender systems are bipartite graphs:

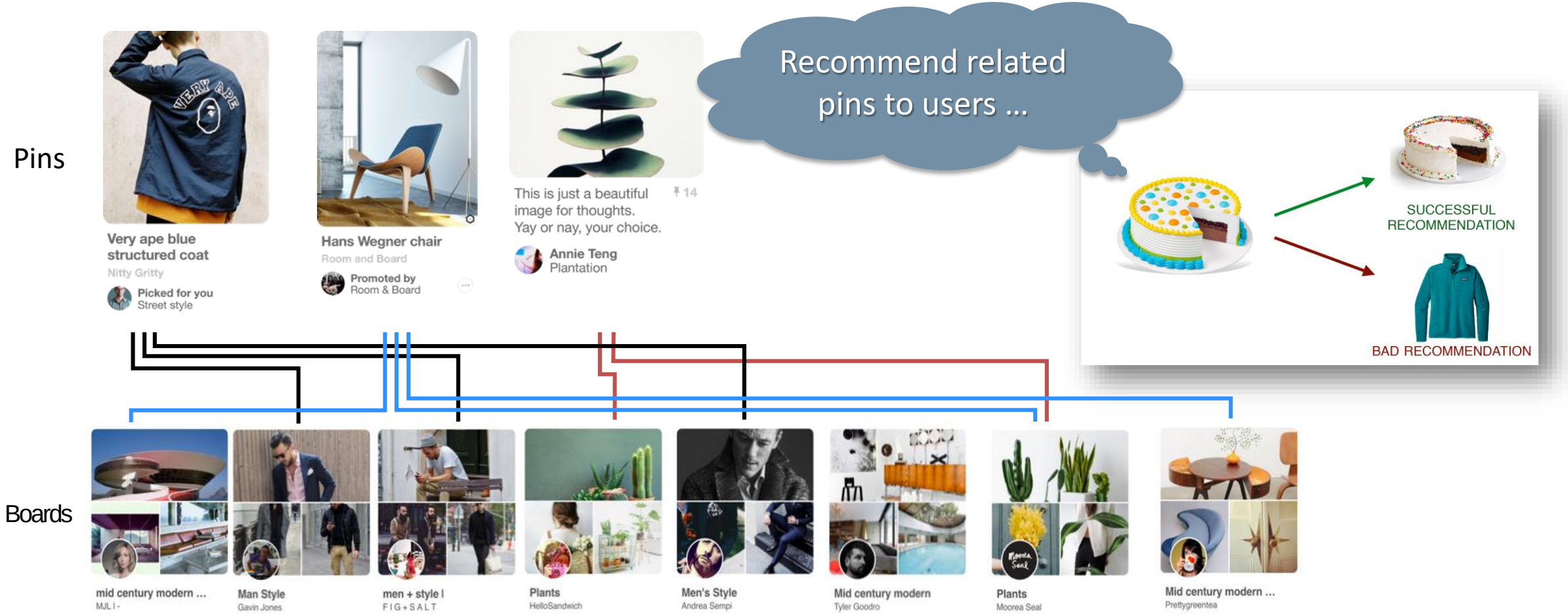
- Content features: user and item features, in the form of images, categories etc.
- Network structure: user-item interactions, in the form of graph/network structure.



Ying et al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems.

Example: Recommendation System (Pinterest)

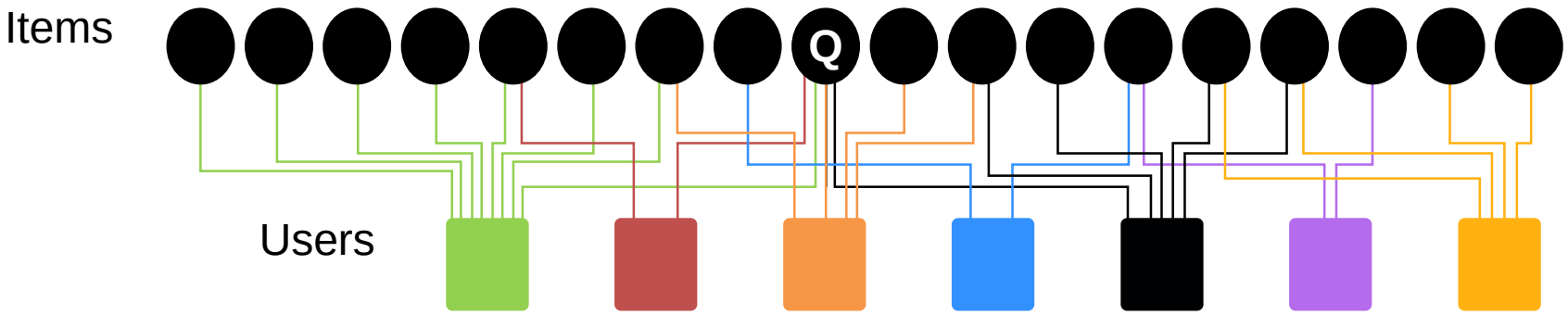
Pins are visual bookmarks users save from the internet to their boards.



Example: Recommendation System

Recommender systems are bipartite graphs:

- Content features: user and item features, in the form of images, categories etc.
- Network structure: user-item interactions, in the form of graph/network structure.



- Graph is dynamic, i.e., need to apply to new nodes without model retraining

Proposed approach: Random Walk +Graph Convolution Networks (RW-GCN)

Ying et al. Graph Convolutional Neural Networks for Web-Scale Recommender Systems.

Example: Recommendation System

After collecting billions of training pairs from user logs.

- Train so that pins that are consecutively clicked have similar embeddings

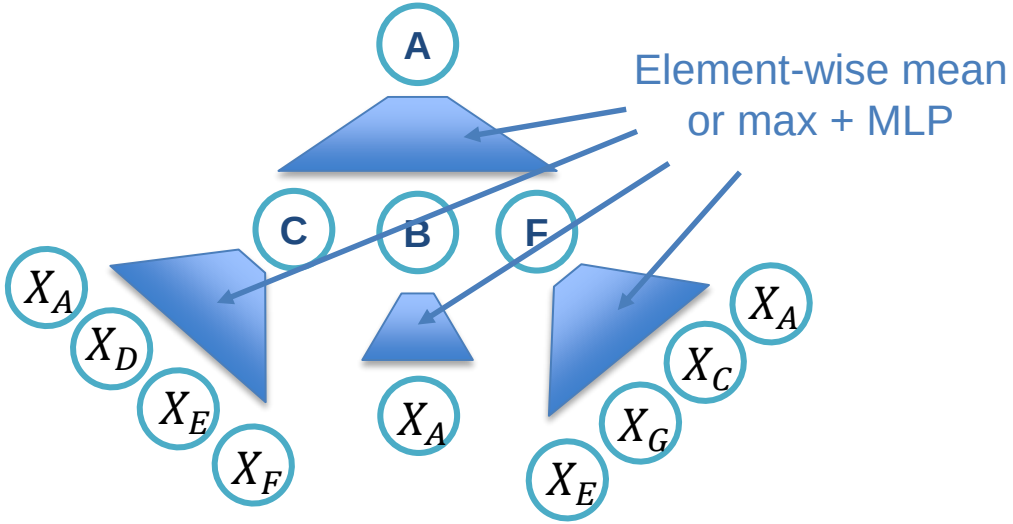
$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \max(0, -\mathbf{z}_u^\top \mathbf{z}_v + \mathbf{z}_u^\top \mathbf{z}_n + \Delta)$$

Set of training pairs from user logs

Positive/true training pair

Negative sample

Margin

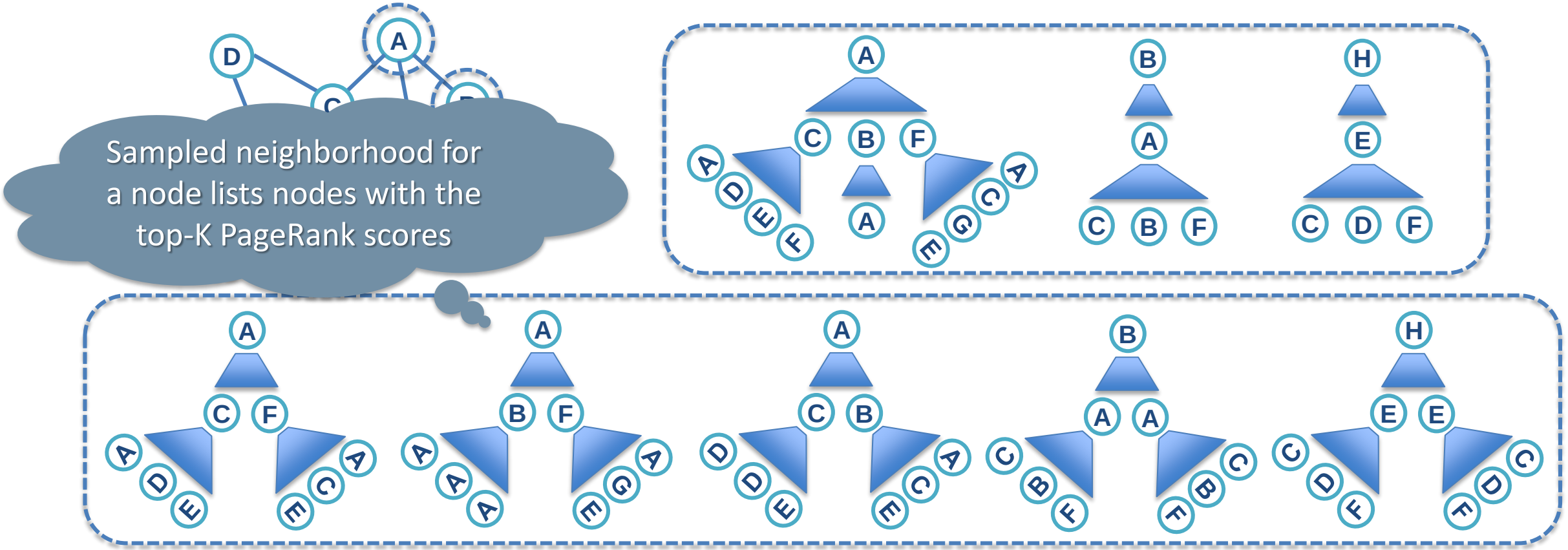


- Generate embeddings for all pins.
- Make recommendations using nearest neighbor search in the embedding space (real-time).

Example: Recommendation System

RW-GCN Tips and Tricks:

- Sub-sample neighborhoods for efficient GPU batching



Example: Recommendation System

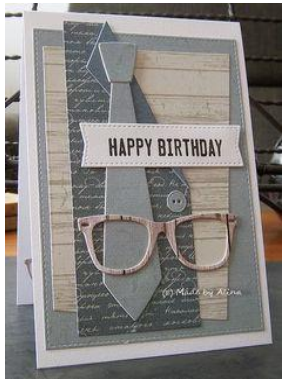
RW-GCN Tips and Tricks:

- Sub-sample neighborhoods for efficient GPU batching
- Curriculum learning for negative samples

Source pin

Positive

Easy negative

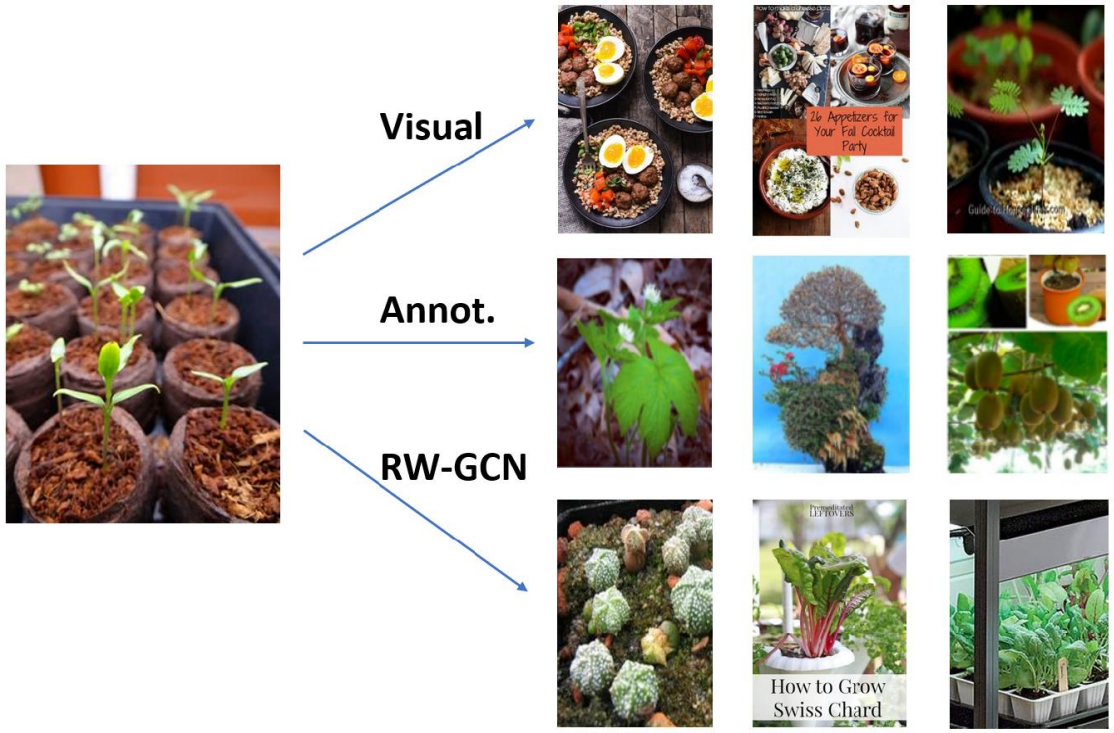


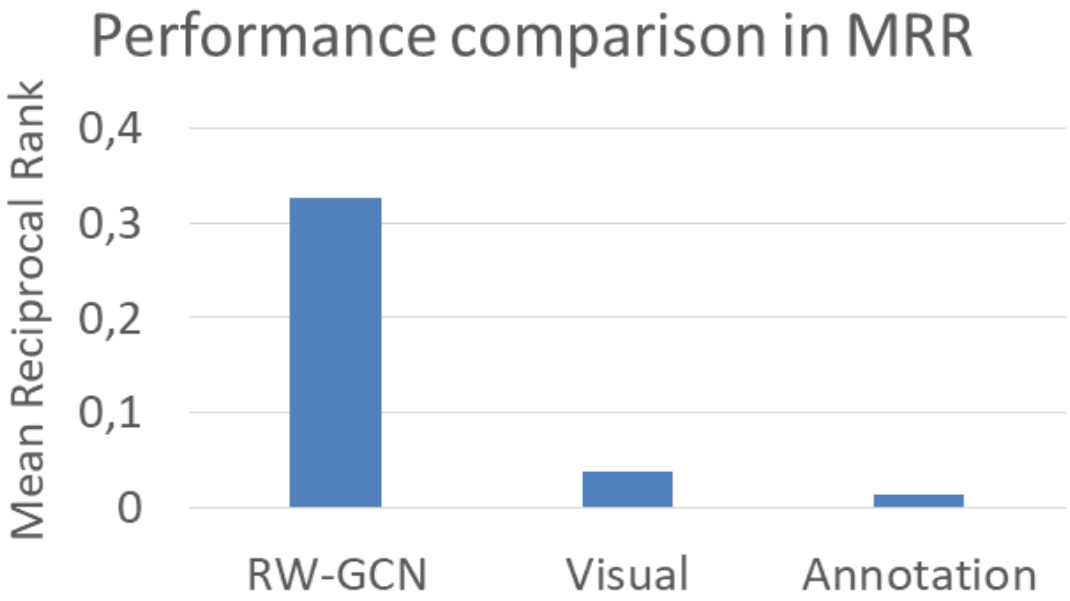
Hard negative

You know this from the distance in the embedding space …

Example: Recommendation System

Rank true “next-clicked” pin against 10^9 other candidates

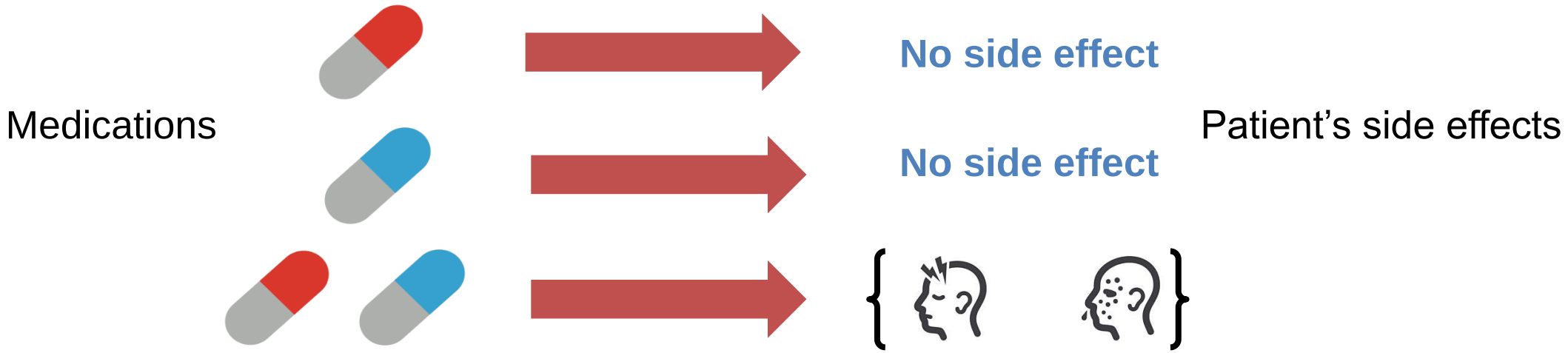




Example: Polypharmacy Side Effects

Predict side effects of taking multiple drugs

- Rare, occur only in a subset of patients
- Not observed in clinical testing

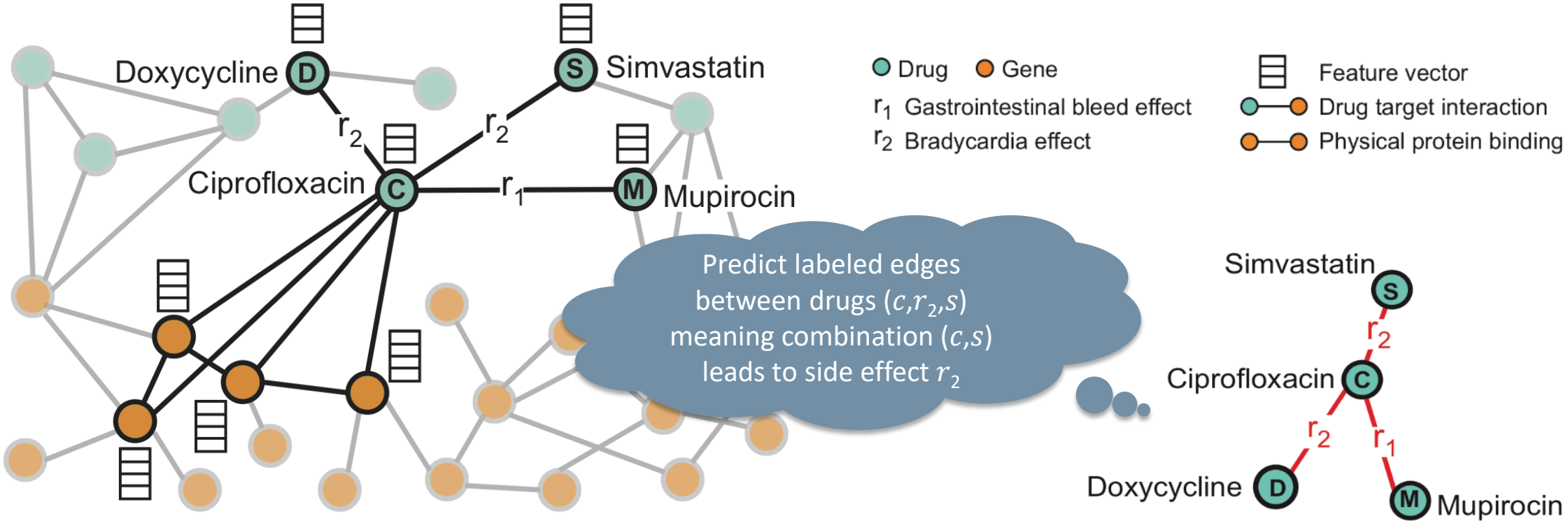


Zitnik et al. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics & ISMB*.

Example: Polypharmacy Side Effects

Computationally screen/predict polypharmacy side effects

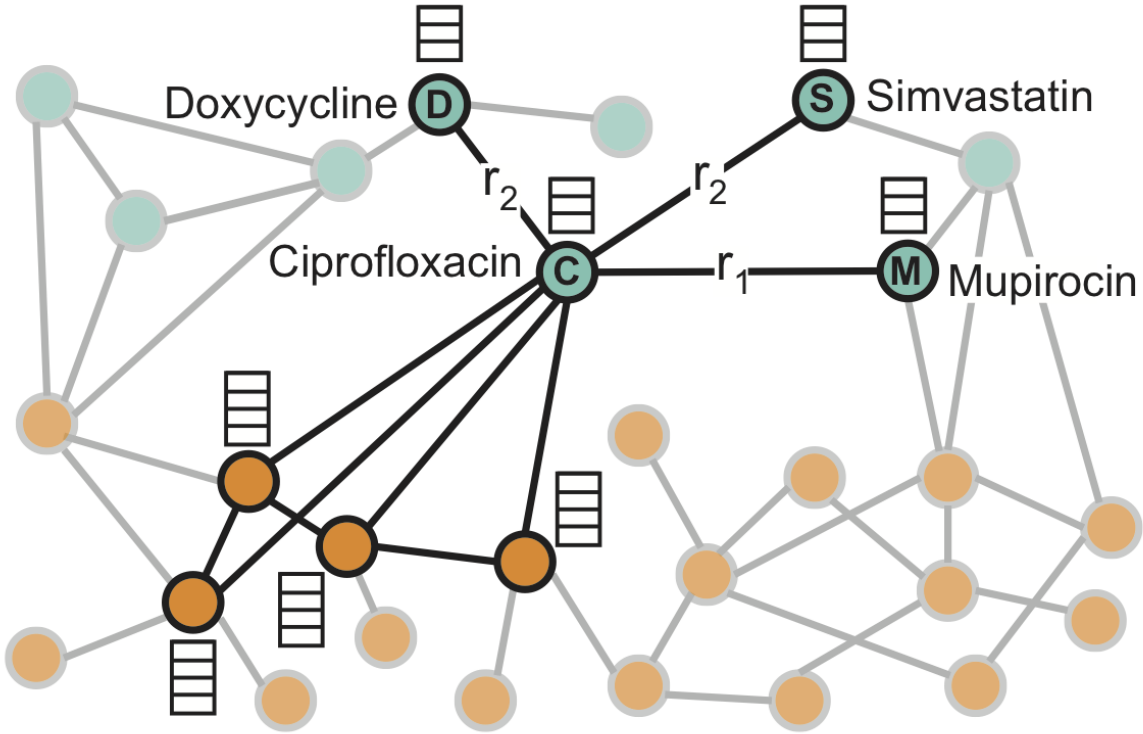
- Use molecular, pharmacological and patient population data
- Guide strategies for combination treatments in patients

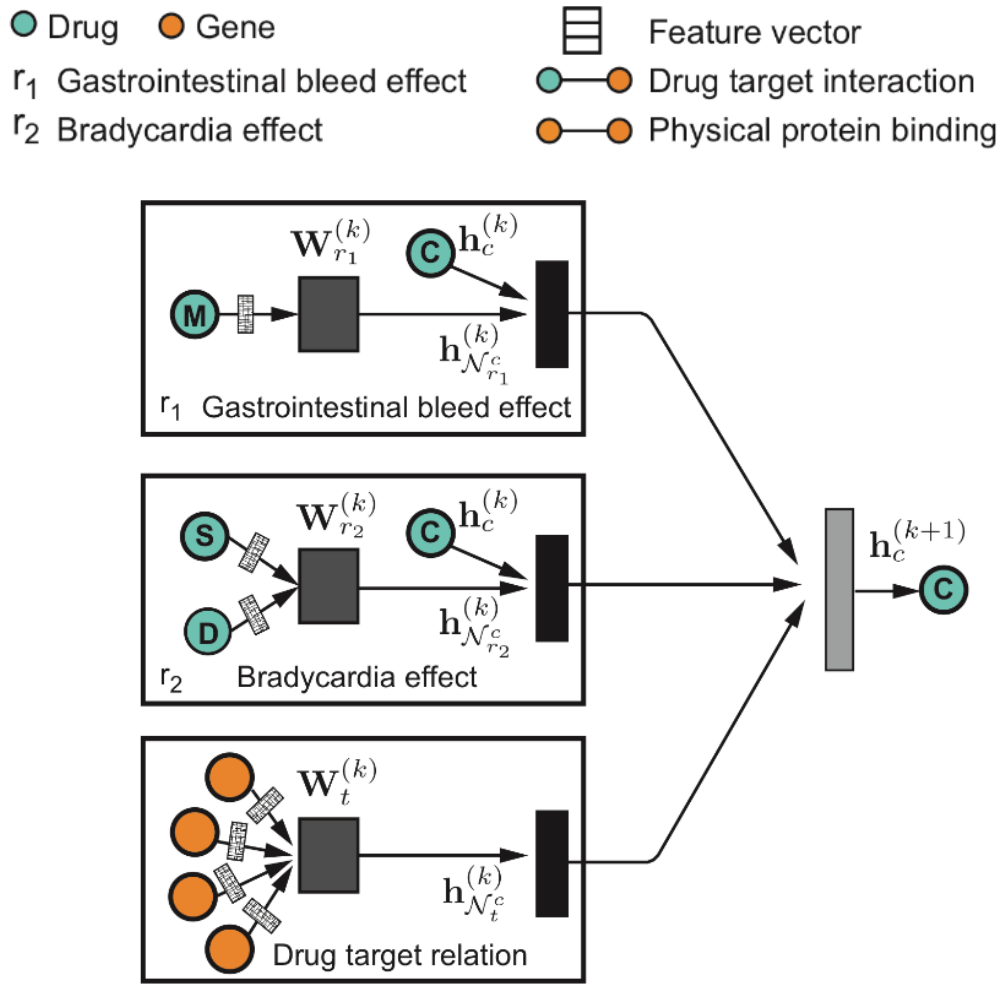


Example: Polypharmacy Side Effects

Encoder deconder network:

- Embedding for nodes

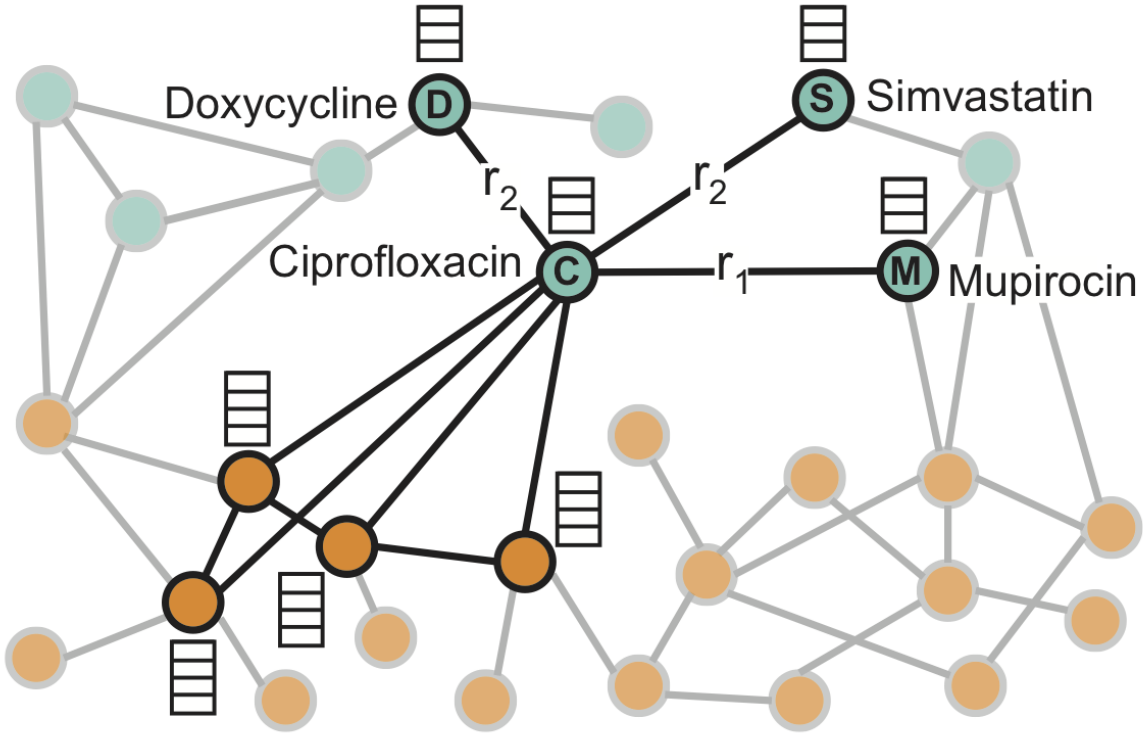


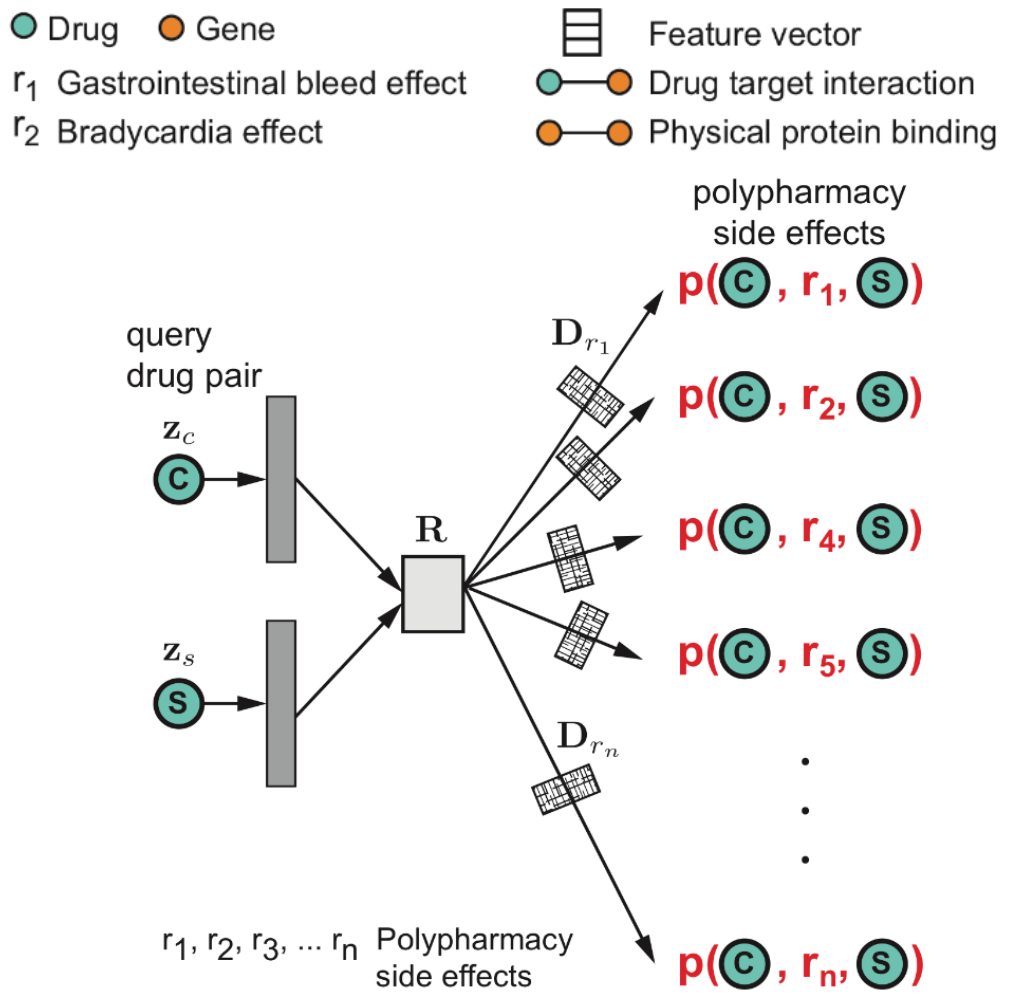


Example: Polypharmacy Side Effects

Encoder deconder network:

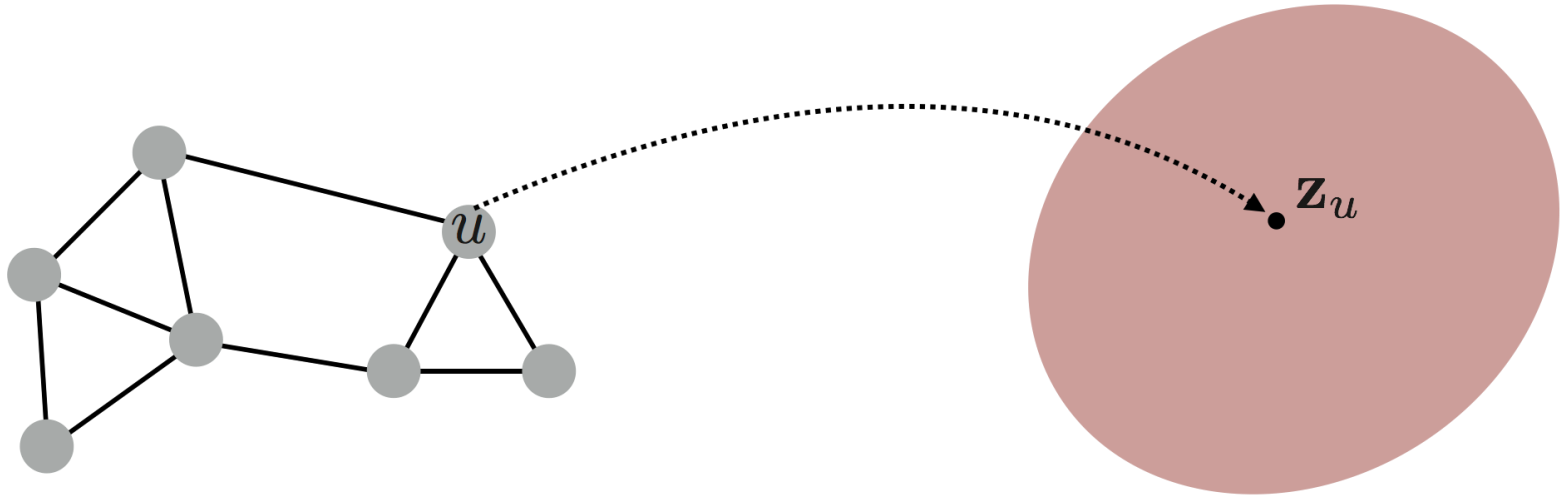
- Embedding for nodes
- Predict edges (side effects)





A note on (sub-)graph embedding

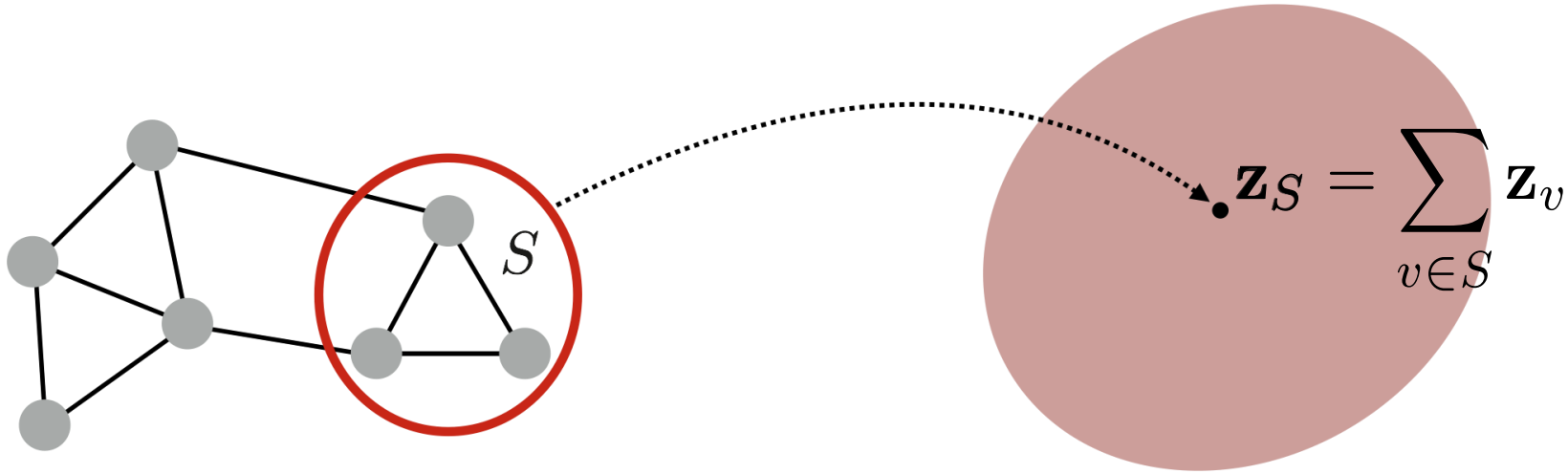
So far we have focused on node level embedding ...



Duvenaud et al. 2016. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *ICML.*
Li et al. 2016. Gated Graph Sequence Neural Networks. *ICLR.*

A note on (sub-)graph embedding

So far we have focused on node level embedding ... what about sub-graphs?

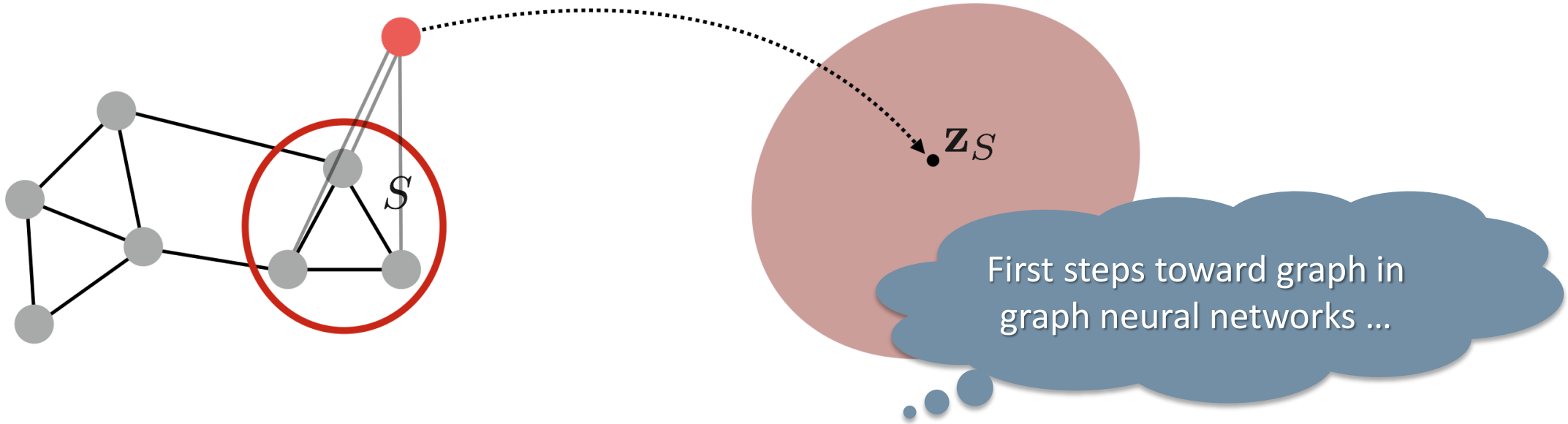


- Sum (or average) node embeddings in the (sub)graph (Duvenaud et al., 2016)

Duvenaud et al. 2016. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *ICML.*
Li et al. 2016. Gated Graph Sequence Neural Networks. *ICLR.*

A note on (sub-)graph embedding

So far we have focused on node level embedding … what about sub-graphs?

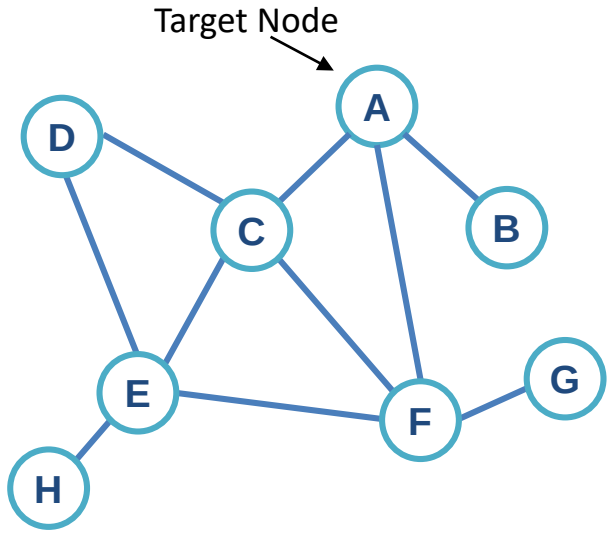


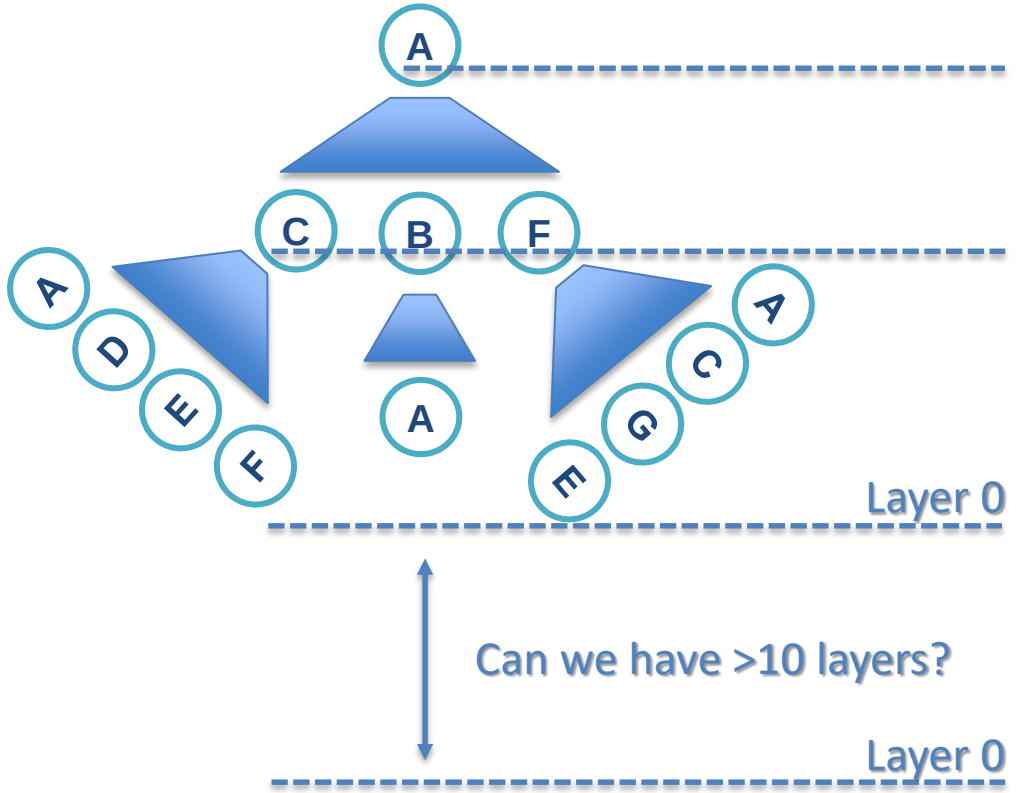
- Sum (or average) node embeddings in the (sub)graph (Duvenaud et al., 2016)
- Introduce a “virtual node” to represent the subgraph (Li et al. 2016)
- …

Duvenaud et al. 2016. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *ICML*.
Li et al. 2016. Gated Graph Sequence Neural Networks. *ICLR*.

Gated Graph Neural Networks

Neighborhood aggregation combines messages from neighbors using neural networks





Li et al., 2016. Gated Graph Sequence Neural Networks. *ICLR.*

Gated Graph Neural Networks

Neighborhood aggregation combines messages from neighbors using neural networks

Challenges:

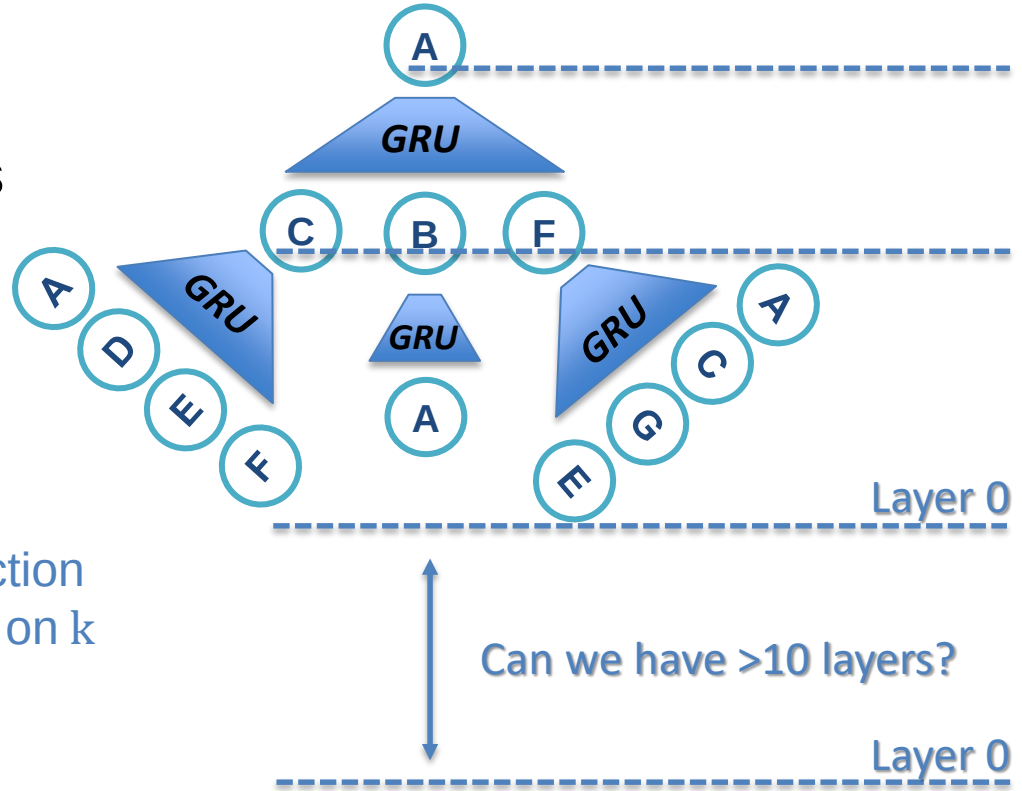
- Overfitting from too many parameters
- Vanishing/exploding gradients during backpropagation.

Aggregation via recurrent networks:

$$\mathbf{m}_v^k = \mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}$$

Aggregation function does not depend on k

$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$



Li et al., 2016. Gated Graph Sequence Neural Networks. *ICLR.*

Dynamic Graphs

Many graphs evolve over time:

- Recommender systems
- Financial transaction
- Graphs from videos
- Social networks

Applications:

- Predict/classify graph evolution (e.g., activity recognition)
- Anomaly detection (e.g., fraud)

Acknowledgements

These slides are highly based on material taken from the following websites/blogs/presentations:

- “Representation Learning on Networks”, http://snap.stanford.edu/proj/embeddings-www
- “Hunt For The Unique, Stable, Sparse and Fast Feature Learning on Graphs” Saurabh Verma, University of Minnesota Twin Cities
- “Structured deep models: Deep learning on graphs and beyond Thomas Kipf, 25 May 2018 (n collaboration with Ethan Fetaya, Rianne van den Berg, Michael Schlichtkrull, Petar Veličković, Ivan Titov, Max Welling, Richard Zemel et al.)