Block 6: Graph Representation Learning and Graph Neural Networks

ELEC 573: Network Science and Analytics

Santiago Segarra

Electrical and Computer Engineering
Rice University
segarra@rice.edu

Fall 2021

You are here

Wk.	Date	Topic	HW	Project
1	23-Aug	Introduction to course	HW0 out	
2	30-Aug	Graph theory	HW0 solutions posted	
3	6-Sep	LABOR DAY (no class)	HW1 out	
4	13-Sep	Centrality measures / Community detection		
5	20-Sep	Community detection		
6	27-Sep	Signal Processing and Deep learning for graphs	HW1 due	
7	4-Oct	Signal Processing and Deep learning for graphs	HW2 out	
8	11-Oct	FALL BREAK (no class)		
9	18-Oct	Network models	HW2 due	
10	25-Oct	Network models	HW3 out	Project proposal due
11	1-Nov	Epidemics		
12	8-Nov	Inference of network topologies, features, and processes	HW3 due	
13	15-Nov	Inference of network topologies, features, and processes		
14	22-Nov	Inference of network topologies, features, and processes		Project progress report
15	29-Nov	Inference of network topologies, features, and processes		
	13-Dec	Project presentation (video recording) and final report due		

Some project titles from previous years

- Molecular graph generation using deep generative networks
- EEG based detection of sudden death in epilepsy (SUDEP) biomarker
- In-vivo analysis of nuclei morphology via graph-convolutional neural networks
- Relating phylogenetic distance with social network graph distance of species
- Graph signal processing to understand the relation between structural and functional activation in brain
- KOMBing through bacterial metagenomes
- Characterizing Structural Differences in Graphs During Atrial Fibrillation
- Graph Reordering for Approximate Near Neighbor Search
- Mathematical Language Processing via Tree Embeddings

Borrowed slides

- Tutorial on Graph Representation Learning
- William L. Hamilton and Jian Tang
- KDD 2017, AAAI 2019
- McGill University, HEC, and Mila
- Thank you!

Borrowed slides

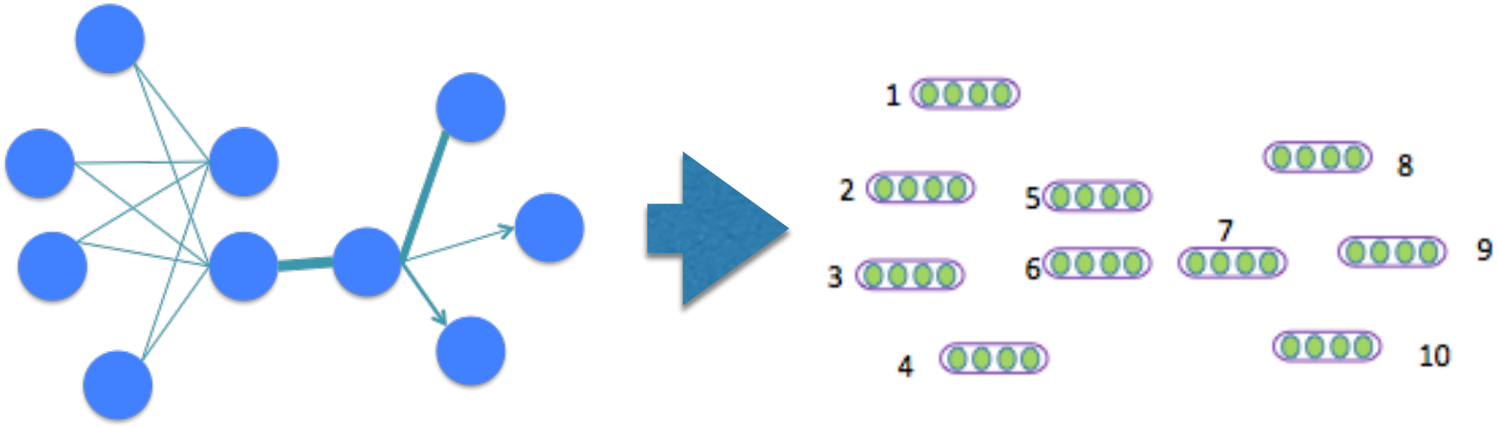
- Tutorial on Graph Representation Learning
- William L. Hamilton and Jian Tang
- KDD 2017, AAAI 2019
- McGill University, HEC, and Mila
- Thank you!
- Also, thanks to Gonzalo Mateos from U. Rochester!

Node Representation Learning

Graph Neural Networks

Problem Definition: Node Embedding

- Given a network/graph G=(V, E, W), where V is the set of nodes, E is the set of edges between the nodes, and W is the set of weights of the edges, the goal of ***node embedding*** is to represent each node *i* with ***a vector*** $\vec{u}_i \in R^d$, which preserves the structure of networks.



Networks Node representations

Related Work

- Classical graph embedding algorithms
  - MDS, IsoMap, LLE, Laplacian Eigenmap, …
  - Hard to scale up
- Graph factorization (Ahmed et al. 2013)
  - Not specifically designed for network representation
  - Undirected graphs only
- Neural word embeddings (Bengio et al. 2003)
  - Neural language model
  - word2vec (skipgram), paragraph vectors, etc.

Laplacian Eigenmap (Belkin and Niyogi, 2003)

- Intuition: the embeddings of similar nodes should be close to each other
- Objective:

$$O = \frac{1}{2}\sum_{(i,j)\in E} w_{ij}(\vec{u}_i - \vec{u}_j)^2 = tr(U^T L U)$$

- Where $U = [\vec{u}_1, \vec{u}_2, \cdots, \vec{u}_N]$, L is the Laplacian matrix $L = D - W$, and $D_{ii} = \sum_j w_{ij}$
- Optimization by finding the eigenvectors of smallest eigenvalues of the Laplacian matrix L:

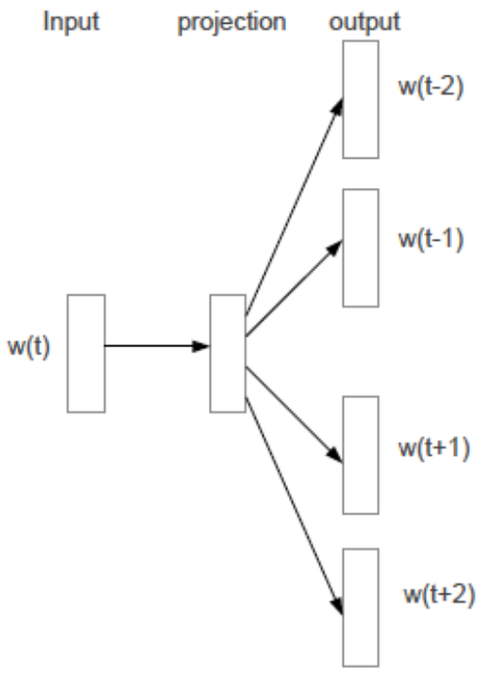
$$Lu = \lambda Du$$

- Computationally expensive for finding eigenvectors when networks are very big

Mikhail Belkin and Partha Niyogi. **Laplacian Eigenmaps for Dimensionality Reduction and Data Representation.** Neural Computation, 2003.

Word2VEC (Mikolov et al. 2014)

- Goal: represent each word *i* with a vector $\vec{v}_i \in R^d$ by training from a sequence $(w_1, w_{2,} \cdots, w_T)$
- Distributional hypothesis (John Rupert Firth): You know a word by the company it keeps
- Skip-gram: learning word representations by predicting the nearby words



$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. **Distributed Representations of Words and Phrases and their Compositionality.** NIPS 2014

Skipgram

- Objective:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

- Where c is the window size
- Direct optimization is computationally expensive due to the softmax function

- Negative sampling:

$$\log\sigma({v'_{w_O}}^{\top}v_{w_I})+\sum_{i=1}^{k}\mathbb{E}_{w_i\sim P_n(w)}\left[\log\sigma(-{v'_{w_i}}^{\top}v_{w_I})\right]$$

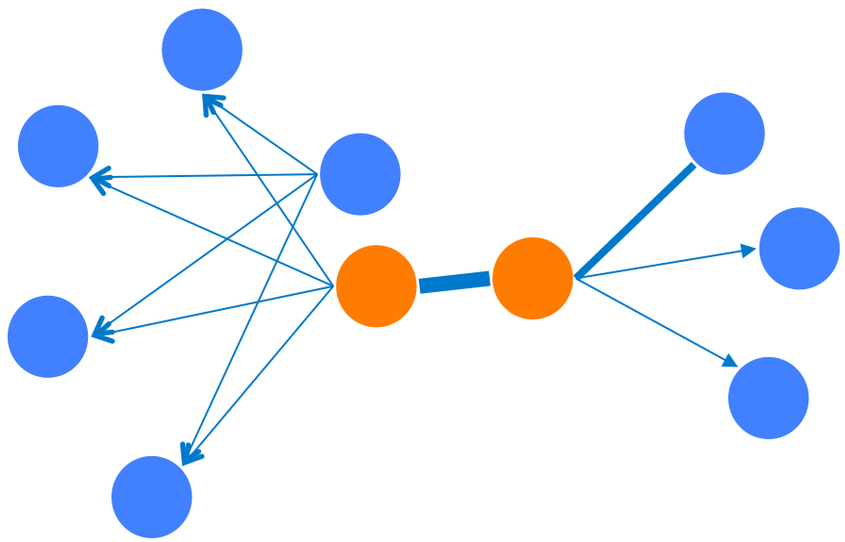
- Where $P_n(w)$ is a noisy distribution

LINE: Large-scale Information Network Embedding (Tang et al., Most Cited Paper of WWW 2015)

- Arbitrary types of networks
  - Directed, undirected, and/or weighted
- Clear objective function
  - Preserve the first-order and second-order proximity
- Scalable
  - Asynchronous stochastic gradient descent
  - Millions of nodes and billions of edges: a coupe of hours on a single machine

Jian Tang, Meng Qu, Mingzhe Wang, Jun Yan, and Qiaozhu Mei. **LINE: Large-scale Information Network Embedding**. WWW'15

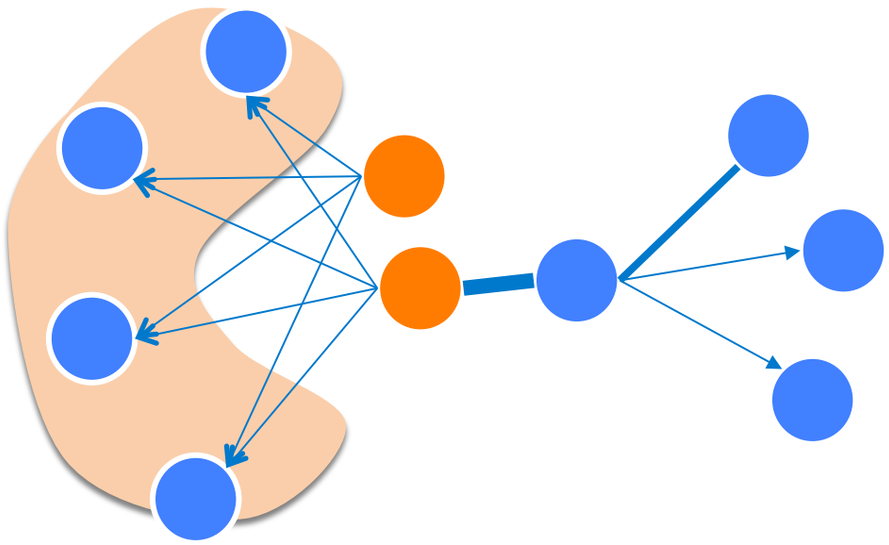
First-order Proximity



- The local pairwise proximity between the nodes
- However, many links between the nodes are not observed
  - Not sufficient for preserving the entire network structure

Second-order Proximity

"The degree of overlap of two people's friendship networks correlates with the strength of ties between them" --Mark Granovetter



"You shall know a word by the company it keeps" --John Rupert Firth

- Proximity between the neighborhood structures of the nodes

Preserving the First-order Proximity (LINE 1st)

- Distributions: : (defined on the undirected edge i - j)

Empirical distribution of first-order proximity:

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(m,n)\in E} w_{mn}}$$

$\vec{u}_i$: Embedding of i

Model distribution of first-order proximity:

$$p_1(v_i, v_j) = \frac{\exp(\vec{u}_i^T \vec{u}_j)}{\sum_{(m,n)\in V\times V} \exp(\vec{u}_m^T \vec{u}_n)}$$

- Objective:

$$O_1 = KL(\hat{p}_1, p_1) = -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$

Preserving the Second-order Proximity (LINE 2nd)

- Distributions: (defined on the directed edge i -> j)

Empirical distribution of neighborhood structure:

$$\hat{p}_2(v_j \mid v_i) = \frac{w_{ij}}{\sum_{k \in V} w_{ik}}$$

Model distribution of neighborhood structure:

$$p_2(v_j \mid v_i) = \frac{\exp(\vec{u}'^T_i \, \vec{u}_j)}{\sum_{k \in V} \exp(\vec{u}'^T_k \, \vec{u}_i)}$$

- Objective:

$$O_2 = \sum_i KL(\hat{p}_2(\cdot \mid v_i), p_2(\cdot \mid v_i)) = -\sum_{(i,j) \in E} w_{ij} \log p_2(v_j \mid v_i)$$

Optimization Tricks

- Stochastic gradient descent + Negative Sampling
  - Randomly sample an edge and multiple negative edges

- The gradient w.r.t the embedding with edge (i, j)

$$\frac{\partial O_2}{\partial \vec{u}_i} = w_{ij} \frac{\partial \log \hat{p}_2(v_j \mid v_i)}{\partial \vec{u}_i}$$

- Problematic when the variances of weights of the edges are large
  - The variance of the gradients are large

- Solution: edge sampling
  - Sample the edges according to their weights and treat the edges as binary

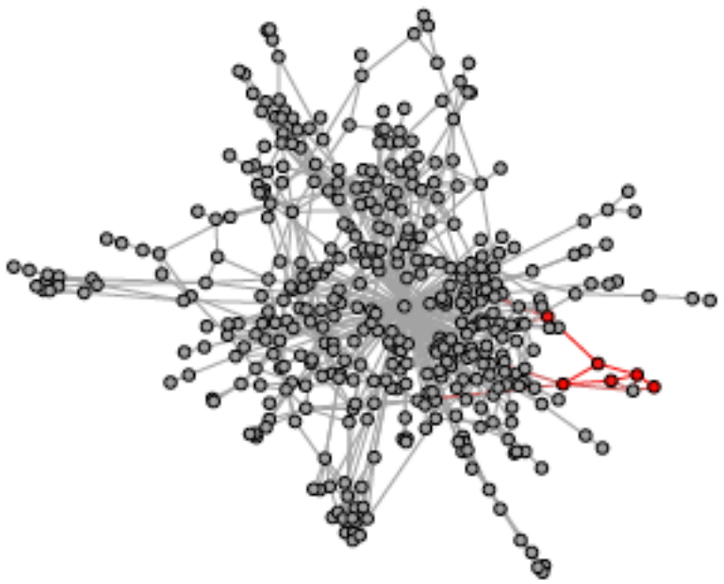
- Complexity: O(d*K*|E|)
  - Linear to the dimensionality d, the number of negative samples K, and the number of edges

Discussion

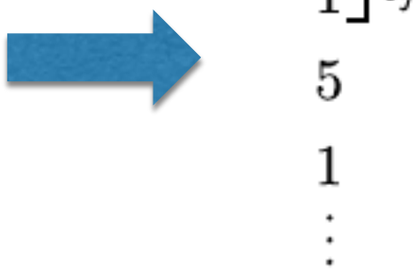
- Embed nodes with few neighbors
  - Expand the neighbors by adding higher-order neighbors
  - **Breadth-first search (BFS)**
  - Adding only second-order neighbors works well in most cases
- Embed new nodes
  - Fix the embeddings of existing nodes
  - Optimize the objective w.r.t. the embeddings of new nodes

DeepWalk (Perozzi et al. 2014)

- Learning node representations with the technique for learning word representations, i.e., Skipgram
- Treat *random walks on networks* as sentences



Random walk generation (generate node contexts through random search)



$$3$$
$$1 \rceil v_j$$
$$5$$
$$1$$
$$\vdots$$

$$p(v_j \mid v_i) = \frac{\exp(\vec{u}'^T_i \vec{u}_j)}{\sum_{k \in V} \exp(\vec{u}'^T_k \vec{u}_i)}$$

Predict the nearby nodes in the random walks

Bryan Perozzi, Rami Al-Rfou, Steven Skiena. DeepWalk: Online Learning of Social Representations. KDD'14

Node2Vec (Grover and Leskovec, 2016)

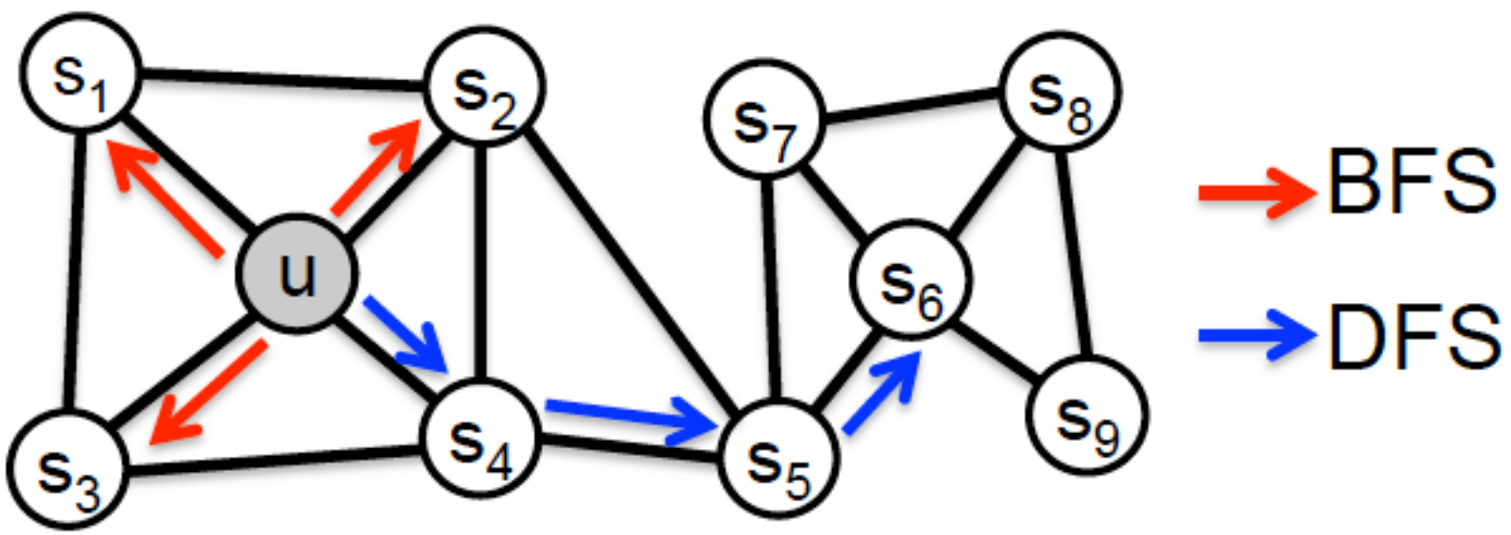
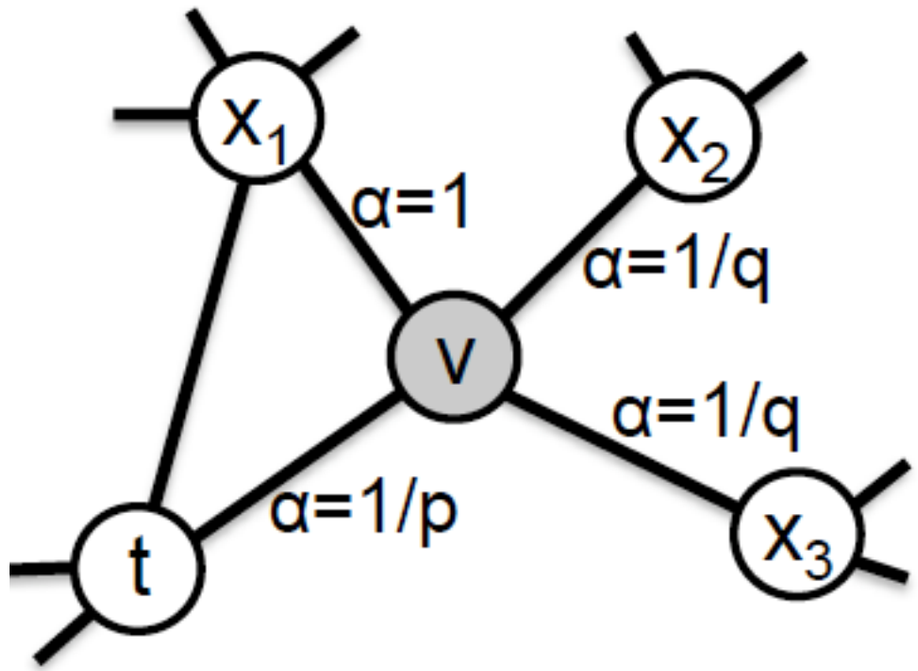


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

- Find the node context by a hybrid strategy of
  - Breadth-first Sampling (BFS): **homophily**
  - Depth-first Sampling (DFS): **structural equivalence**

Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. KDD'16

Expand Node Contexts with Biased Random Walk



- Biased random walk with two parameters p and q
  - p: controls the probability of revisiting a node in the walk
  - q: controls the probability of exploring "outward" nodes
  - Find optimal p and q through cross-validation on labeled data
- Optimized through similar objective as LINE with first-order proximity

Applications

- Node classification (Perozzi et al. 2014, Tang et al. 2015a, Grover et al. 2015)
- Node visualization (Tang et al. 2015a)
- Link prediction (Grover et al. 2015)
- Recommendation (Zhao et al. 2016)
- Text representation (Tang et al. 2015a, Tang et al. 2015b)
- ...

Node Classification

- social network => user representations (features) => classifier
- Community identities as classification labels

	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**32.4**	**34.6**	**35.9**	**36.7**	**37.2**	**37.7**	**38.1**	**38.3**	**38.5**	**38.7**
	SpectralClustering	27.43	30.11	31.63	32.69	33.31	33.95	34.46	34.81	35.14	35.41
	EdgeCluster	25.75	28.53	29.14	30.31	30.85	31.53	31.75	31.76	32.19	32.84
	Modularity	22.75	25.29	27.3	27.6	28.05	29.33	29.43	28.89	29.17	29.2
	wvRN	17.7	14.43	15.72	20.97	19.83	19.42	19.22	21.25	22.51	22.73
	Majority	16.34	16.31	16.34	16.46	16.65	16.44	16.38	16.62	16.67	16.71

Table: Results on Flickr Network (Perozzi et al. 2014)

DeepWalk > Laplacian Eigenmap

Node Classification

- social network => user representations (features) => classifier
- Community identities as classification labels

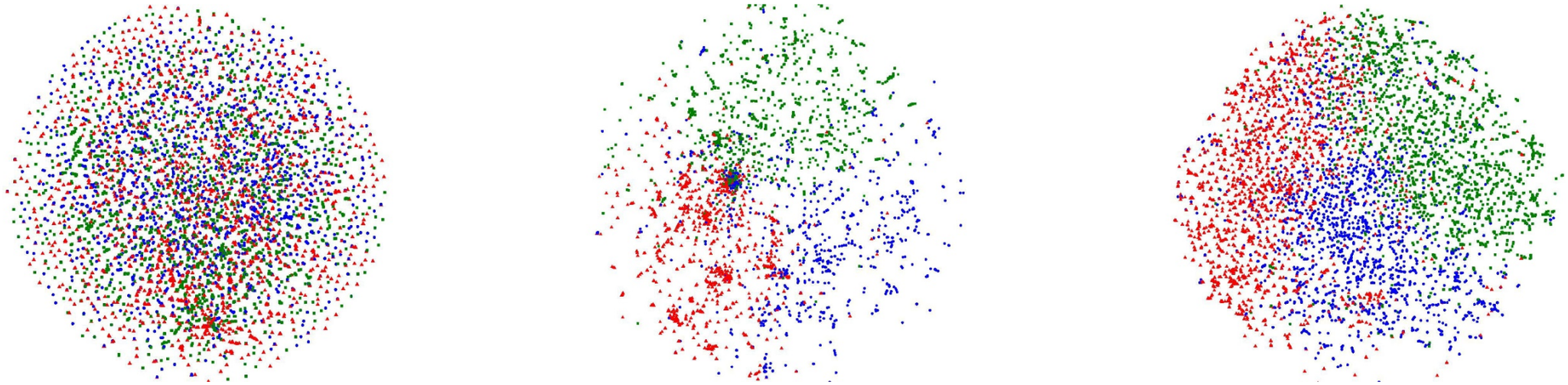
Metric	Algorithm	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1	GF	25.43 (24.97)	26.16 (26.48)	26.60 (27.25)	26.91 (27.87)	27.32 (28.31)	27.61 (28.68)	27.88 (29.01)	28.13 (29.21)	28.30 (29.36)	28.51 (29.63)
	DeepWalk	39.68	41.78	42.78	43.55	43.96	44.31	44.61	44.89	45.06	45.23
	DeepWalk(256dim)	39.94	42.17	43.19	44.05	44.47	44.84	45.17	45.43	45.65	45.81
	LINE(1st)	35.43 (36.47)	38.08 (38.87)	39.33 (40.01)	40.21 (40.85)	40.77 (41.33)	41.24 (41.73)	41.53 (42.05)	41.89 (42.34)	42.07 (42.57)	42.21 (42.73)
	LINE(2nd)	32.98 (36.78)	36.70 (40.37)	38.93 (42.10)	40.26 (43.25)	41.08 (43.90)	41.79 (44.44)	42.28 (44.83)	42.70 (45.18)	43.04 (45.50)	43.34 (45.67)
	LINE(1st+2nd)	39.01* (**40.20**)	41.89 (**42.70**)	43.14 (**43.94****)	44.04 (**44.71****)	44.62 (**45.19****)	45.06 (**45.55****)	45.34 (**45.87****)	45.69** (**46.15****)	45.91** (**46.33****)	46.08** (**46.43****)

Table: Results on Youtube Network(Tang et al. 2015a)

LINE(1^{st} + 2^{nd}) >LINE(2^{nd})> DeepWalk > LINE(1^{st})

Node Visualization (Tang et al. 2015a)

- Coauthor network: authors from three different research fields



● "Data mining" ● "Machine learning" ● "Computer vision"

(a) Graph factorization (b) DeepWalk (c) LINE(2^{nd})

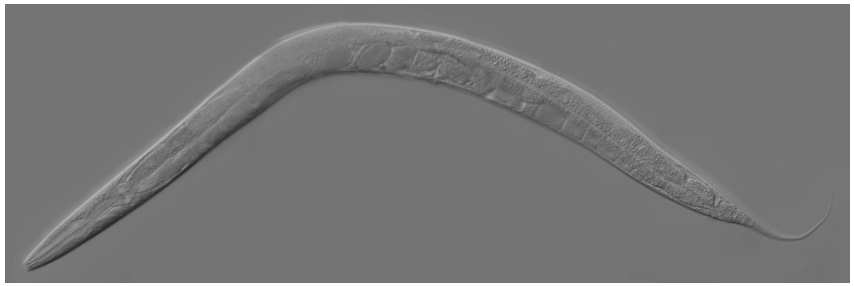
Link Prediction (Grover and Leskovec, 2016)

Op	Algorithm	Dataset		
		Facebook	PPI	arXiv
(a)	Common Neighbors	0.8100	0.7142	0.8153
	Jaccard's Coefficient	0.8880	0.7018	0.8067
	Adamic-Adar	0.8289	0.7126	0.8315
	Pref. Attachment	0.7137	0.6670	0.6996
	Spectral Clustering	0.5960	0.6588	0.5812
	DeepWalk	0.7238	0.6923	0.7066
	LINE	0.7029	0.6330	0.6516
	node2vec	0.7266	0.7543	0.7221
(b)	Spectral Clustering	0.6192	0.4920	0.5740
	DeepWalk	**0.9680**	0.7441	0.9340
	LINE	0.9490	0.7249	0.8902
	node2vec	**0.9680**	**0.7719**	**0.9366**

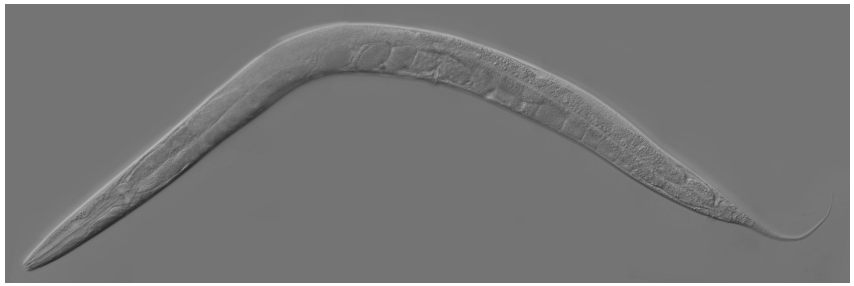
Table: Results of Link Prediction

Node Embeddings (LINE, DeepWalk, node2vec)
> Jaccard's Coefficient > Adamic-Adar

Network of the week

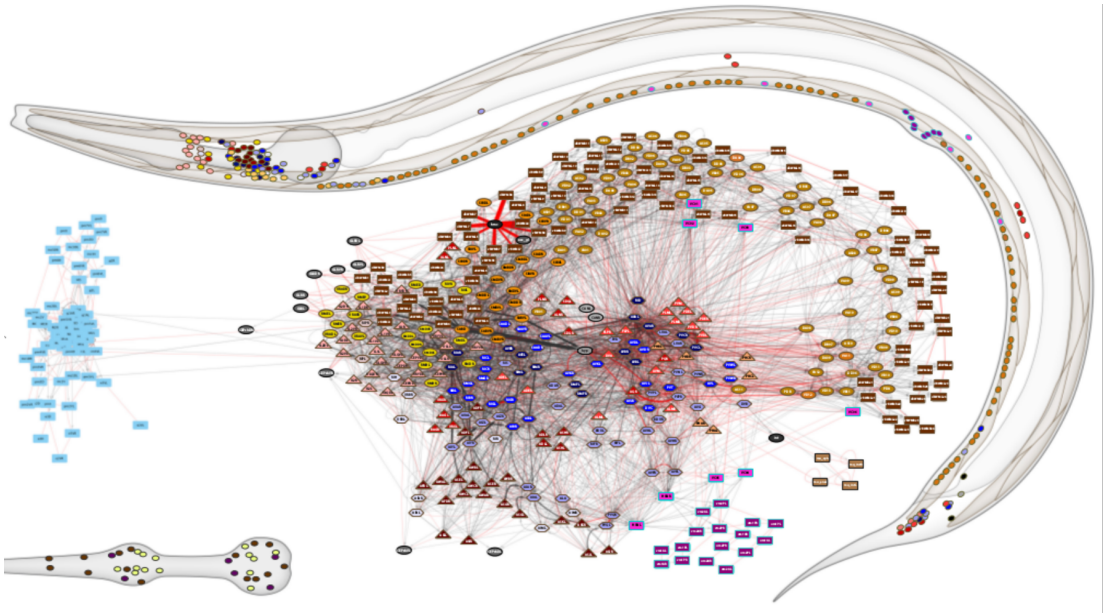


Network of the week

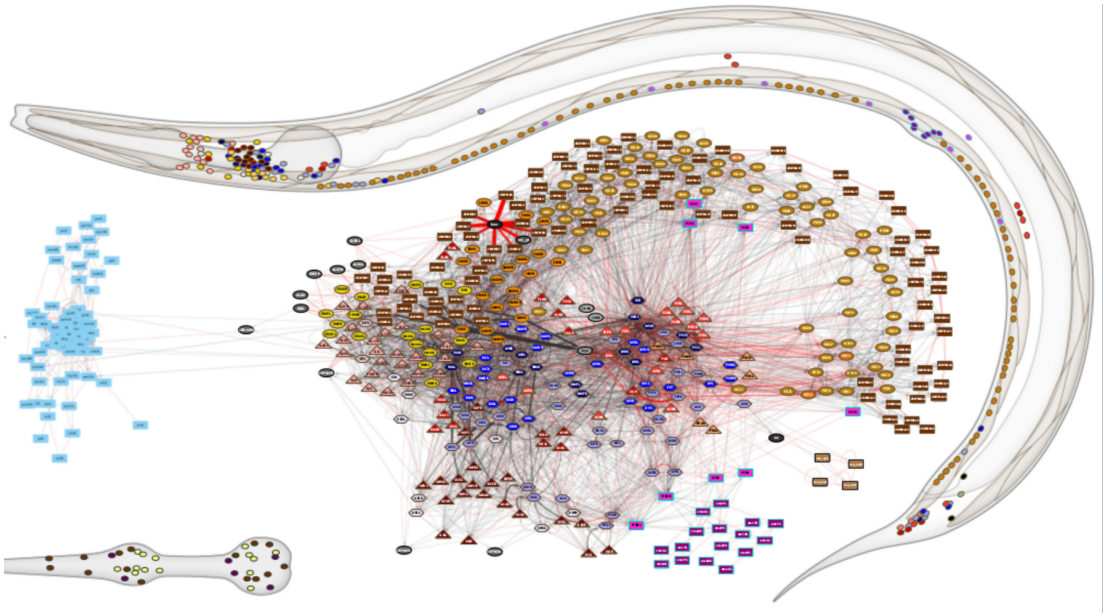


- Caenorhabditis elegans (C. elegans)
  - $\Rightarrow$ Free living, transparent nematode ($\sim$1mm)
  - $\Rightarrow$ caeno- (recent), rhabditis (rod-like) and elegans (elegant)
- First multicellular organism to have its whole genome sequenced
- First (and only) to have its connectome completed

Network of the week



Network of the week



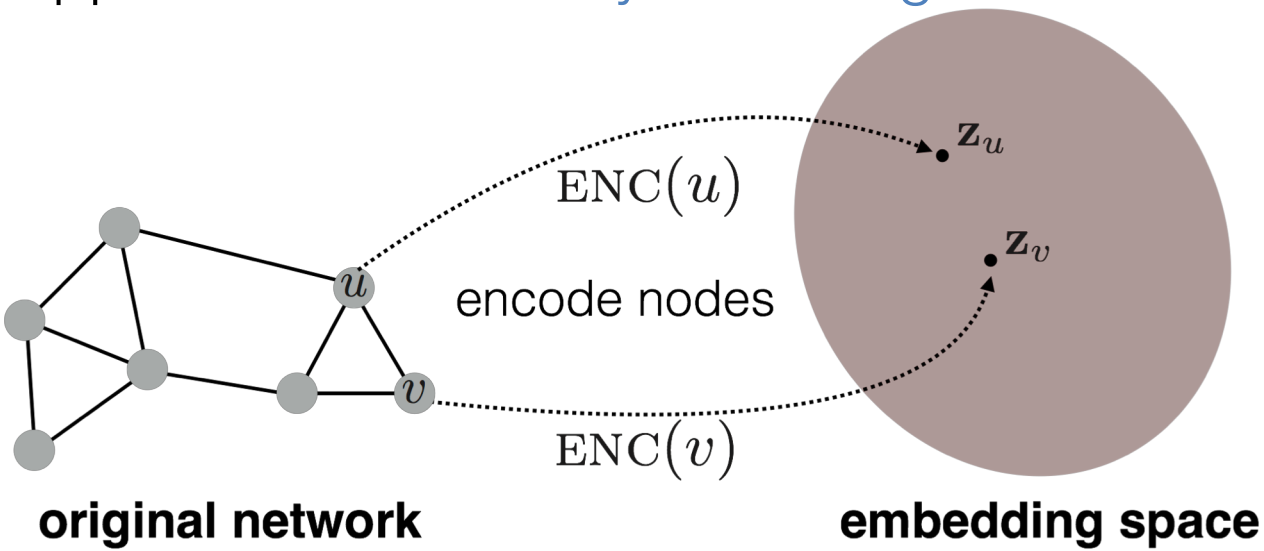
- "The rich club of the C. elegans neuronal connectome"
- "From the connectome to brain function"
- "Topological cluster analysis reveals the systemic organization of the Caenorhabditis elegans connectome"
- "Network control principles predict neuron function in the Caenorhabditis elegans connectome"

Node Representation Learning

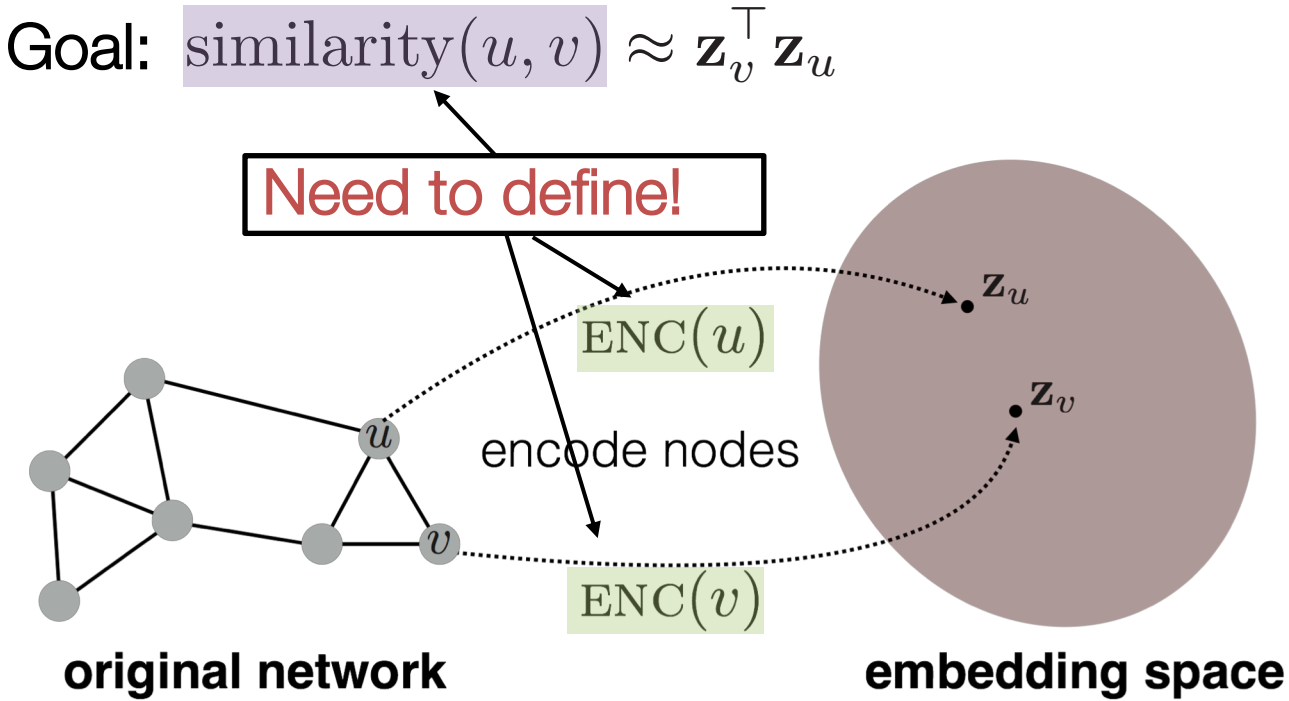
Graph Neural Networks

Embedding Nodes

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.



Embedding Nodes



Two Key Components

- Encoder maps each node to a low-dimensional vector.

$$\text{ENC}(v) = \mathbf{z}_v$$

$\mathbf{d}$-dimensional embedding

node in the input graph

- Similarity function specifies how relationships in vector space map to relationships in the original network.

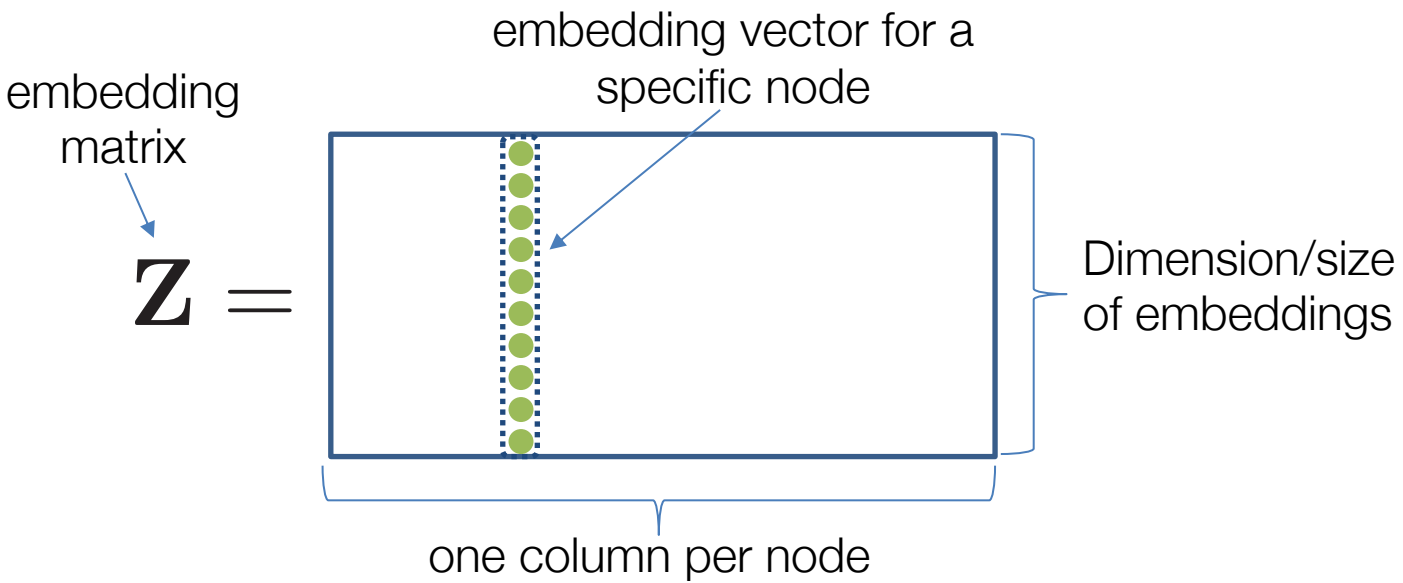
$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

From "Shallow" to "Deep"

- So far we have focused on **"shallow" encoders**, i.e. embedding lookups:



From “Shallow” to “Deep”

- Limitations of shallow encoding:
  - O(|V|) parameters are needed: there no parameter sharing and every node has its own unique embedding vector.
  - Inherently “transductive”: It is impossible to generate embeddings for nodes that were not seen during training.
  - Do not incorporate node features: Many graphs have features that we can and should leverage.

From “Shallow” to “Deep”

- We will now discuss “deeper” methods based on graph neural networks.

$$\text{ENC}(v) = \quad \text{complex function that depends on graph structure.}$$

- In general, all of these more complex encoders can be combined with the similarity functions from the previous section.

Outline for this Section

- We will now discuss “deeper” methods based on graph neural networks.
  1. The Basics
  2. Graph Convolutional Networks
  3. GraphSAGE
  4. Gated Graph Neural Networks
  5. Graph Attention Networks
  6. Subgraph embeddings

The Basics: Graph Neural Networks

Based on material from:

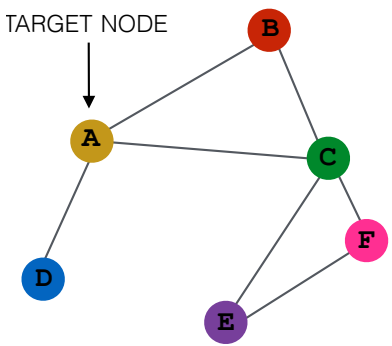
- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
- Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Setup

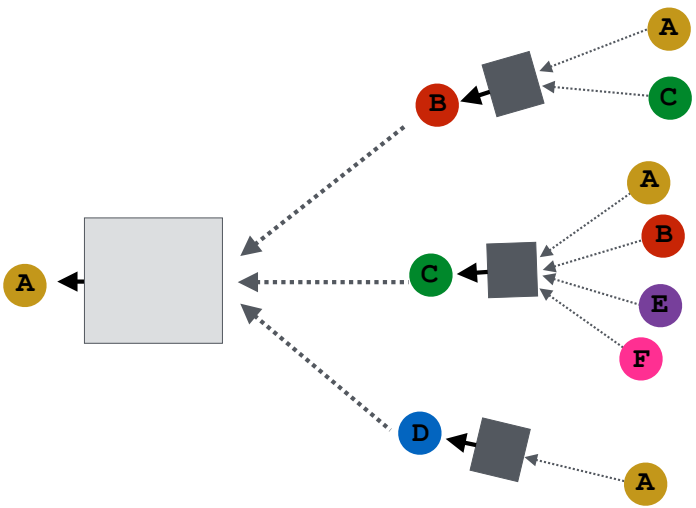
- Assume we have a graph $\mathbf{G}$:
  - $\mathbf{V}$ is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - $\mathbf{X} \in \mathbf{R}^{m \times |V|}$ is a matrix of node features.
    - Categorical attributes, text, image data
      - E.g., profile information in a social network.
    - Node degrees, clustering coefficients, etc.
    - Indicator vectors (i.e., one-hot encoding of each node)

Neighborhood Aggregation

- **Key idea:** Generate node embeddings based on local neighborhoods.

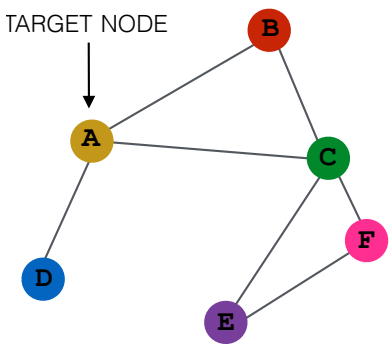


INPUT GRAPH

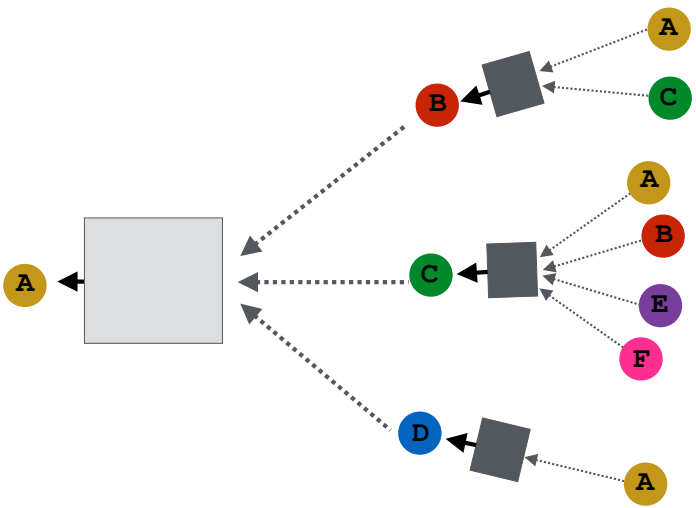


Neighborhood Aggregation

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



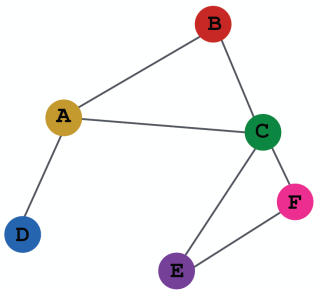
INPUT GRAPH



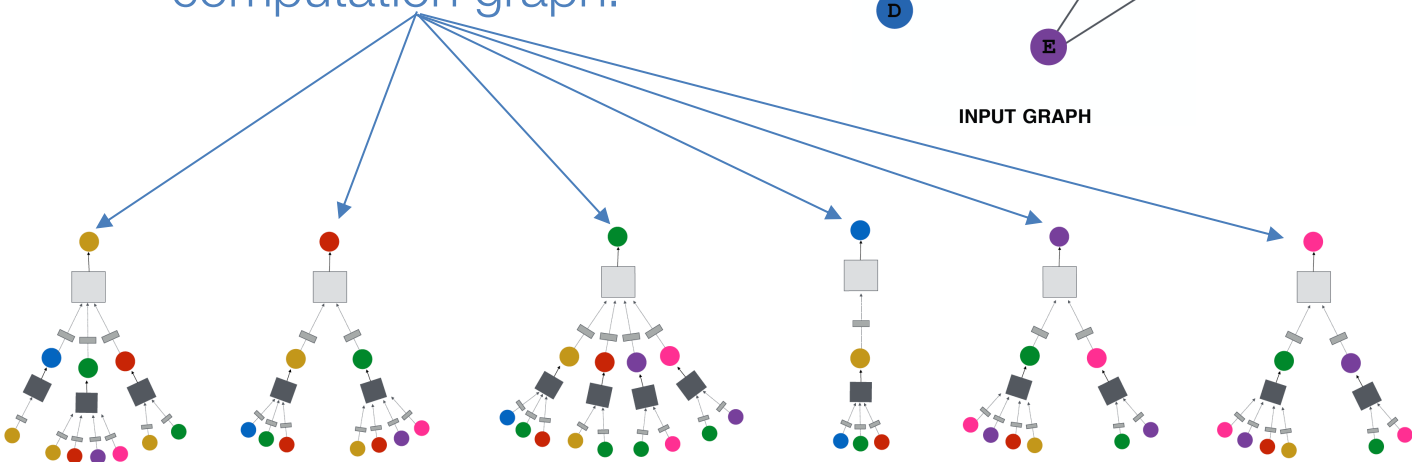
Neighborhood Aggregation

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a unique computation graph!

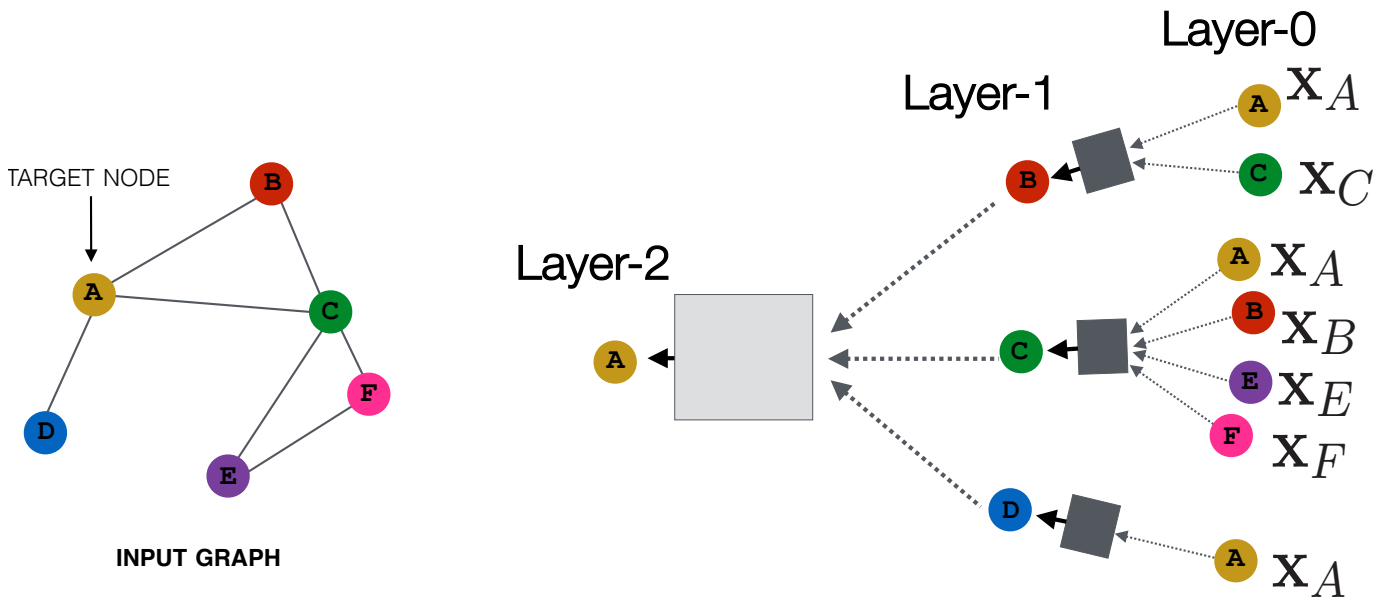


INPUT GRAPH



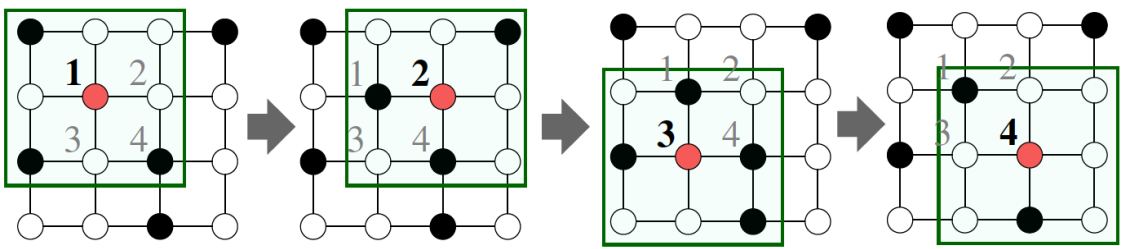
Neighborhood Aggregation

- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- "layer-0" embedding of node $\mathbf{u}$ is its input feature, i.e. $\mathbf{x}_u$.



Neighborhood "Convolutions"

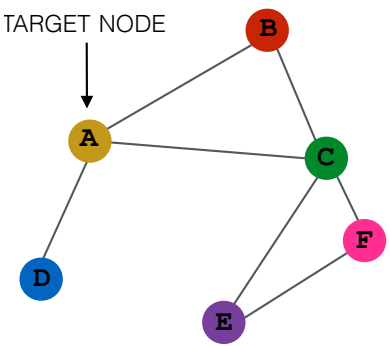
- Neighborhood aggregation can be viewed as a center-surround filter.



- Mathematically related to spectral graph convolutions (see Bronstein et al., 2017)

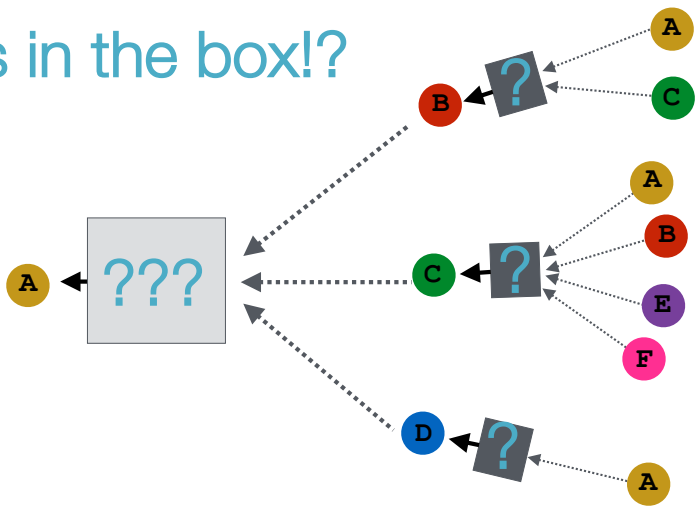
Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate information across the layers.



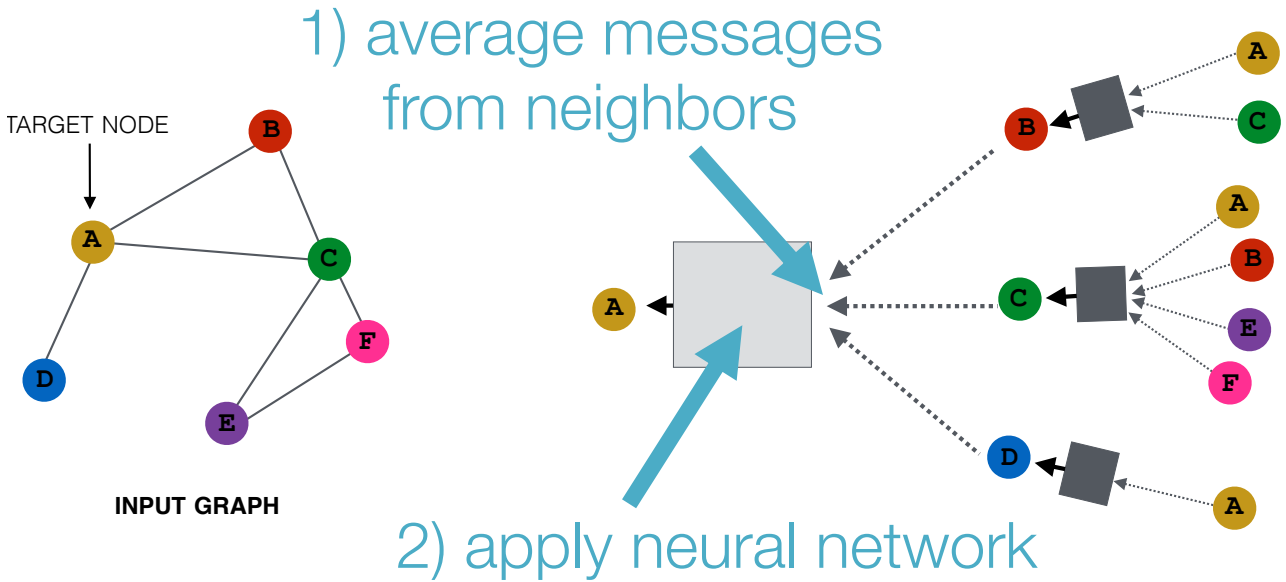
INPUT GRAPH

what's in the box!?



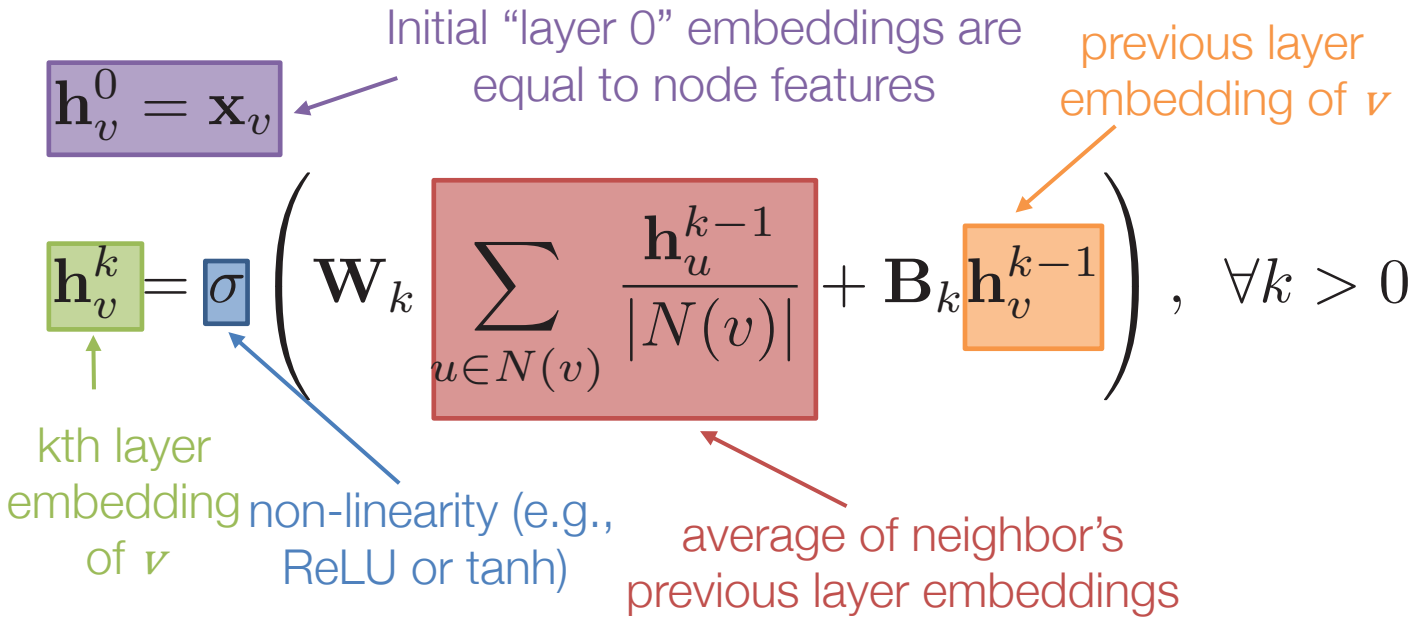
Neighborhood Aggregation

- **Basic approach:** Average neighbor information and apply a neural network.



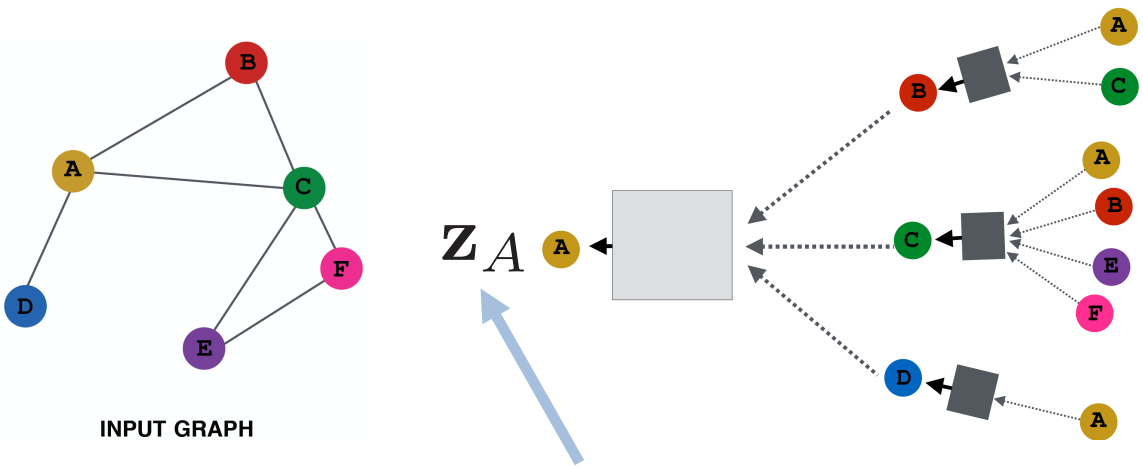
The Math

- **Basic approach:** Average neighbor messages and apply a neural network.



Training the Model

- How do we train the model to generate "high-quality" embeddings?



Need to define a loss function on the embeddings, $\mathcal{L}(\mathbf{z}_u)$!

Training the Model

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

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

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

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

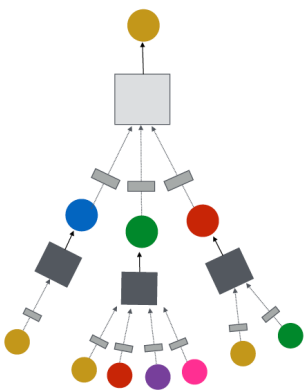
- After K-layers of neighborhood aggregation, we get output embeddings for each node.
- We can feed these embeddings into any loss function and run stochastic gradient descent to train the aggregation parameters.

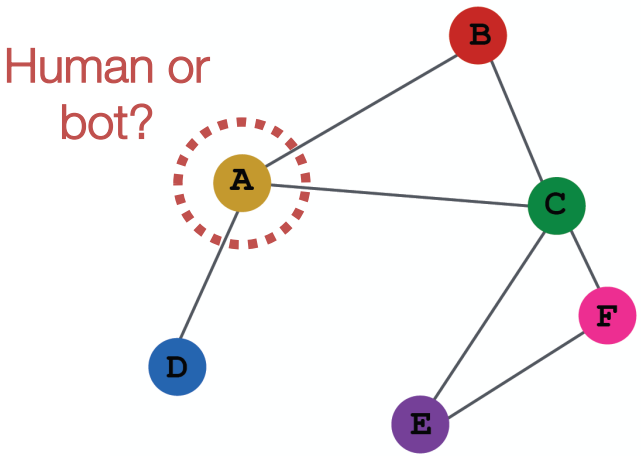
Training the Model

- Train in an **unsupervised manner** using only the graph structure.
- Unsupervised loss function can be anything from the last section, e.g., based on
  - Random walks (node2vec, DeepWalk)
  - Graph factorization
  - i.e., train the model so that "similar" nodes have similar embeddings.

Training the Model

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

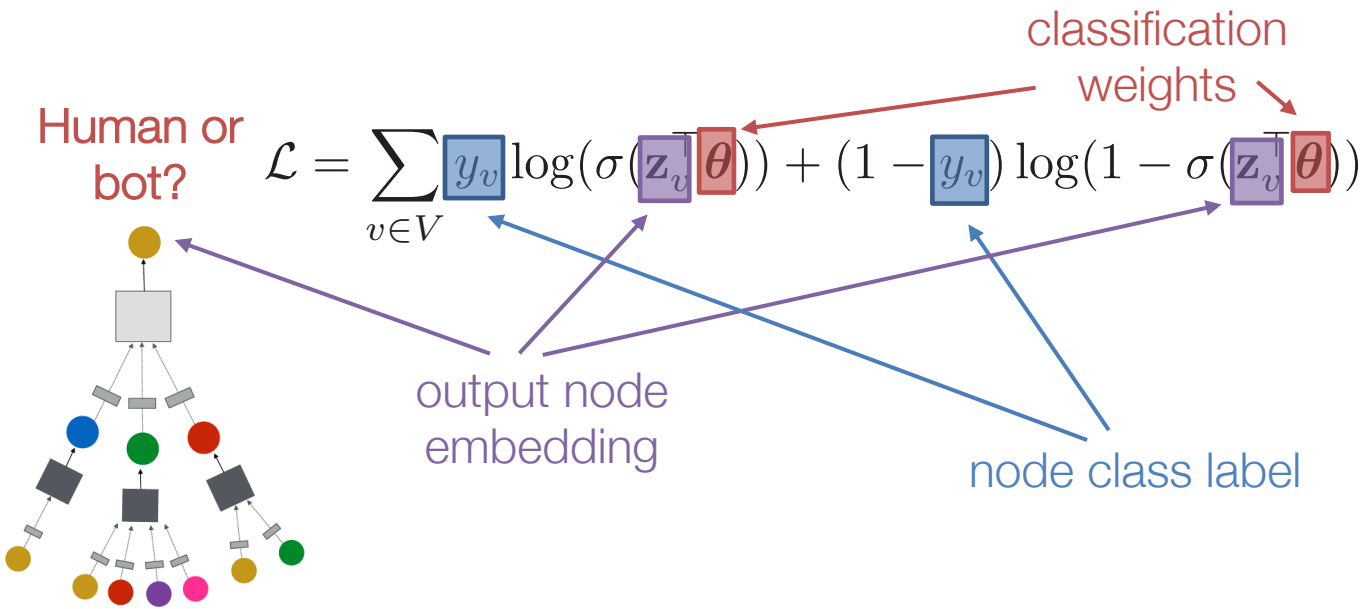




e.g., an online social network

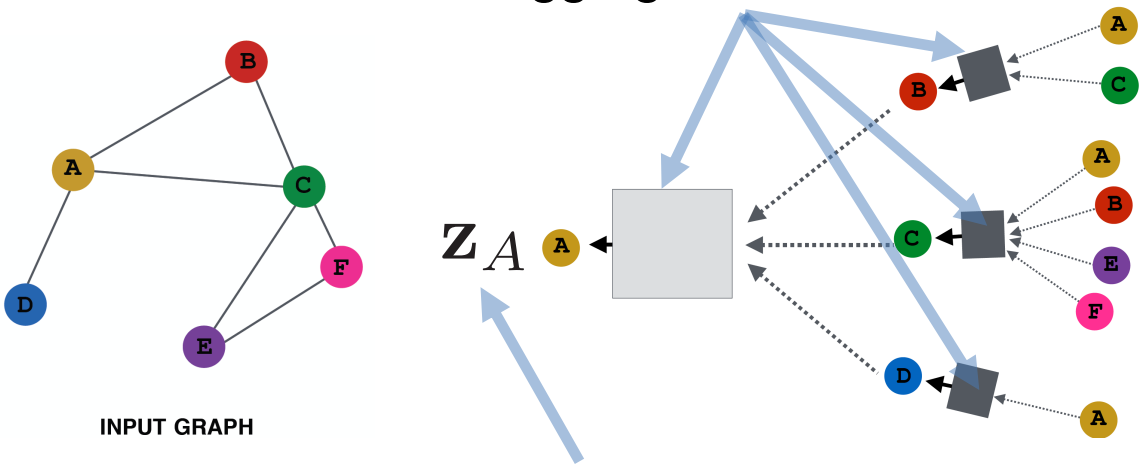
Training the Model

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



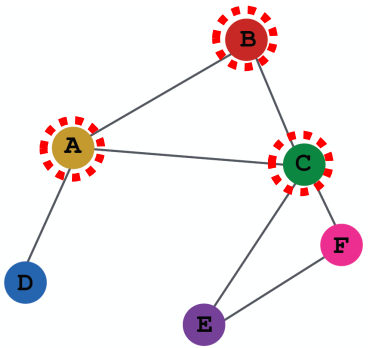
Overview of Model

1) Define a neighborhood aggregation function.



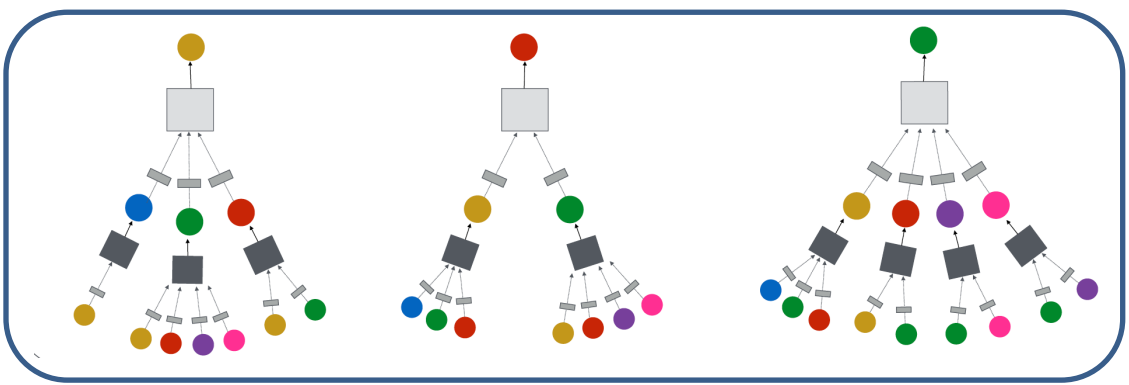
2) Define a loss function on the embeddings, $\mathcal{L}(\mathbf{z}_u)$

Overview of Model

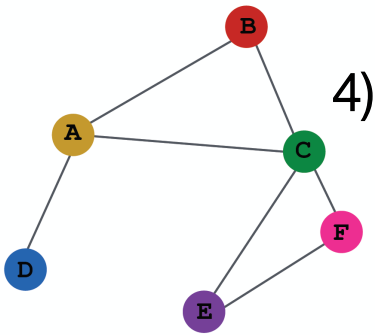


INPUT GRAPH

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



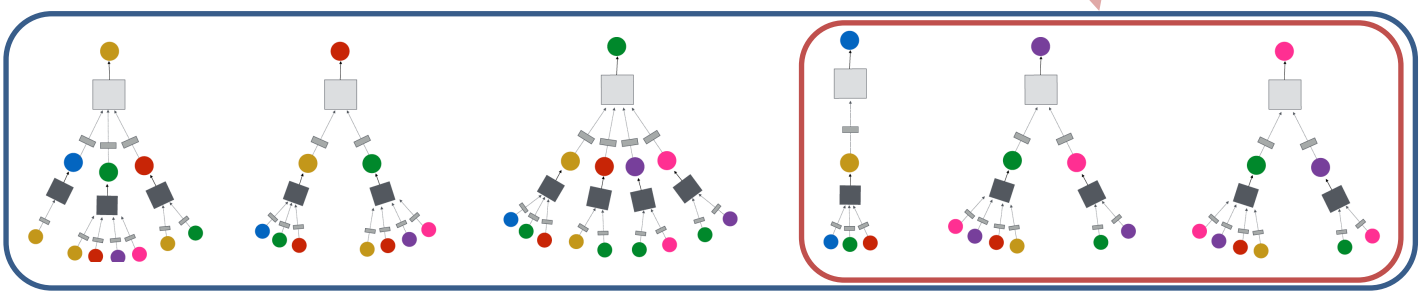
Overview of Model



INPUT GRAPH

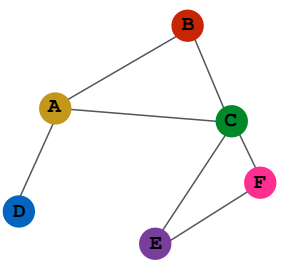
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

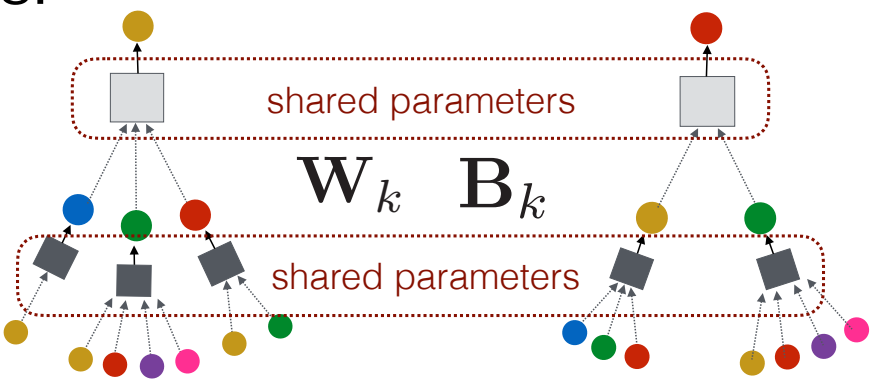


Inductive Capability

- 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!

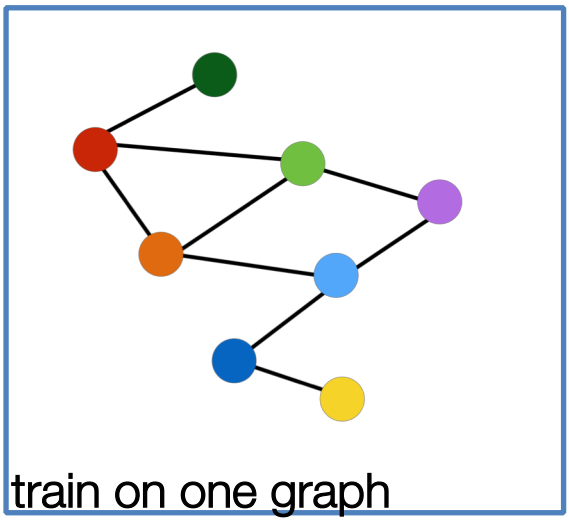


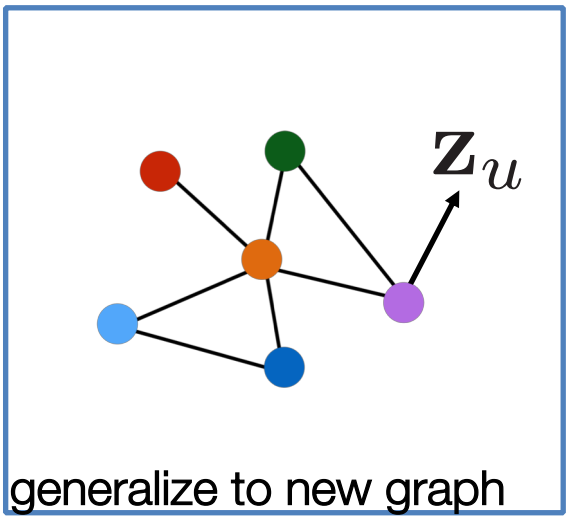
INPUT GRAPH



Compute graph for node A **Compute graph for node B**

Inductive Capability

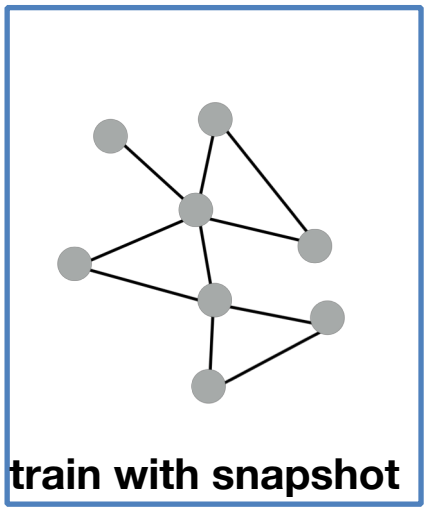


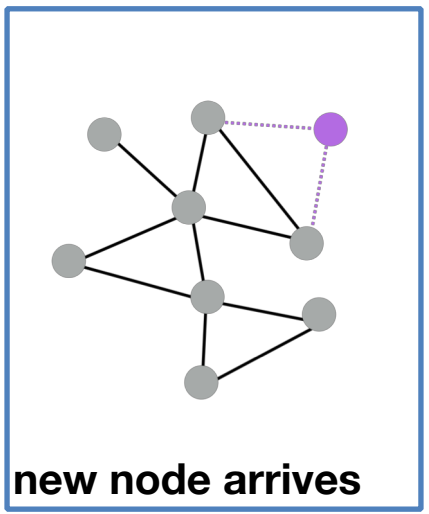


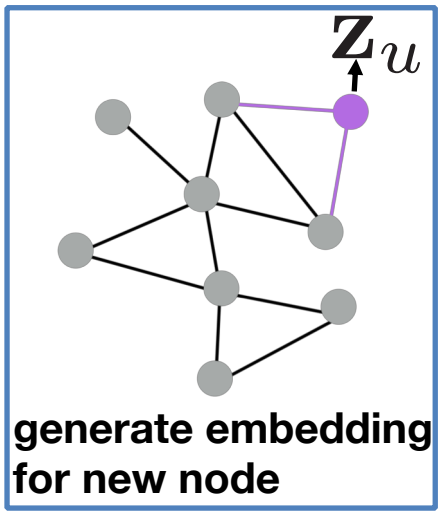
Inductive node embedding → generalize to entirely unseen graphs

e.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Inductive Capability







Many application settings constantly encounter previously unseen nodes.

e.g., Reddit, YouTube, GoogleScholar, ….

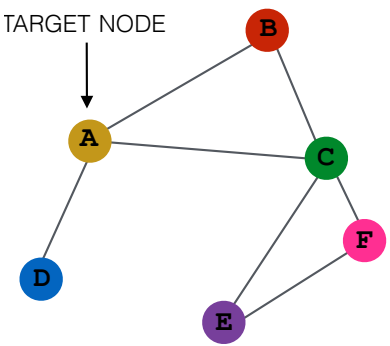
Need to generate new embeddings "on the fly"

Quick Recap

- Recap: Generate node embeddings by aggregating neighborhood information.
  - Allows for parameter sharing in the encoder.
  - Allows for inductive learning.
- We saw a **basic variant of this idea**… now we will cover some state of the art variants from the literature.

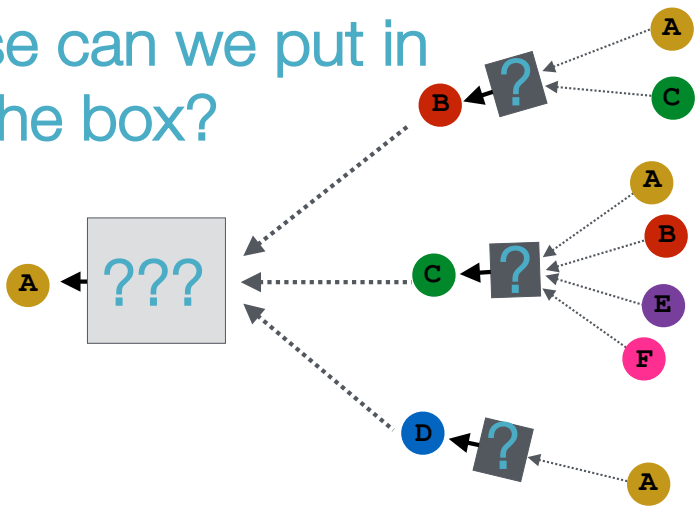
Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages



INPUT GRAPH

What else can we put in the box?



Graph Convolutional Networks

Based on material from:

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

Graph Convolutional Networks

- Kipf et al.'s **Graph Convolutional Networks (GCNs)** are a slight variation on the neighborhood aggregation idea:

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

Graph Convolutional Networks

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

Graph Convolutional Networks

- Empirically, they found this configuration to give the best results.
  - More parameter sharing.
  - Down-weights high degree neighbors.

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

use the same transformation matrix for self and neighbor embeddings

instead of simple average, normalization varies across neighbors

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE 
4. Gated Graph Neural Networks
5. Graph Attention Networks
6. Subgraph Embeddings

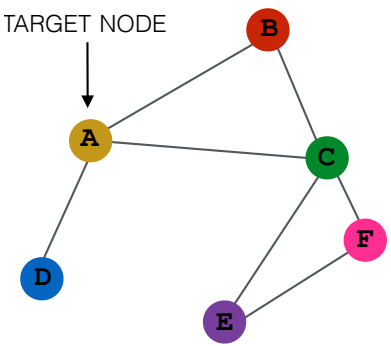
GraphSAGE

Based on material from:

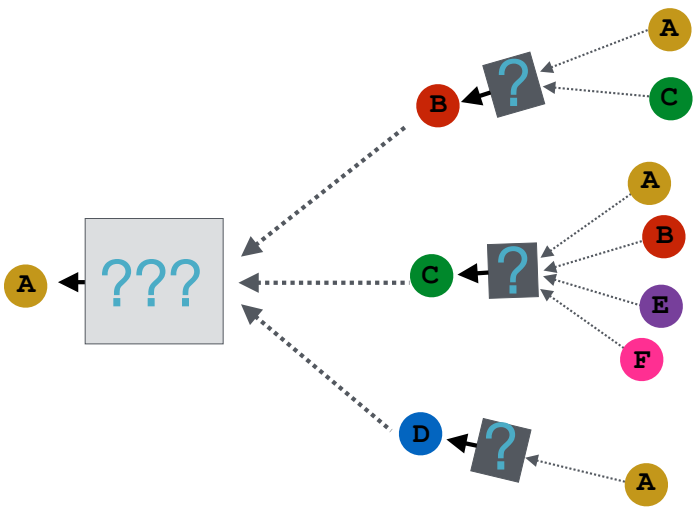
- Hamilton et al., 2017. Inductive Representation Learning on Large Graphs. *NIPS.*

GraphSAGE Idea

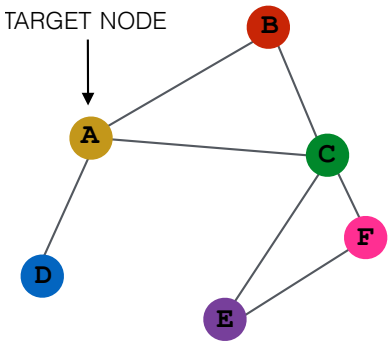
- So far we have aggregated the neighbor messages by taking their (weighted) average, can we do better?



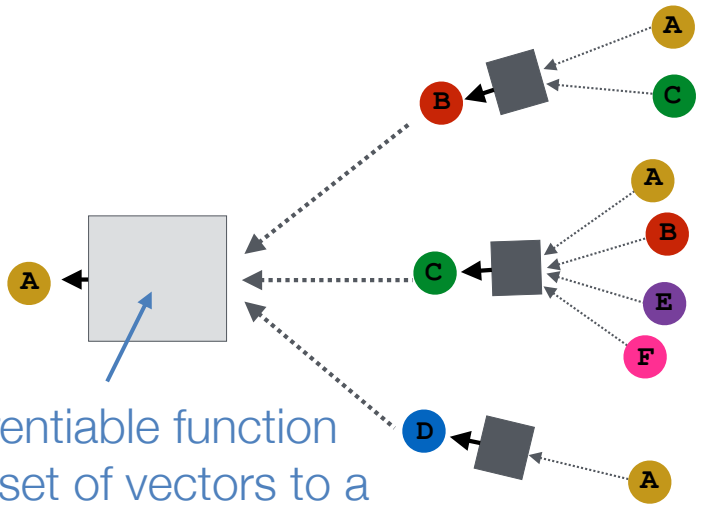
INPUT GRAPH



GraphSAGE Idea



INPUT GRAPH



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

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

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

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

generalized aggregation

GraphSAGE Variants

- 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.

element-wise mean/max

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

- LSTM:
  - Apply LSTM to random permutation of neighbors.

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

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE ✓
4. Gated Graph Neural Networks 
5. Graph Attention Networks
6. Subgraph Embeddings

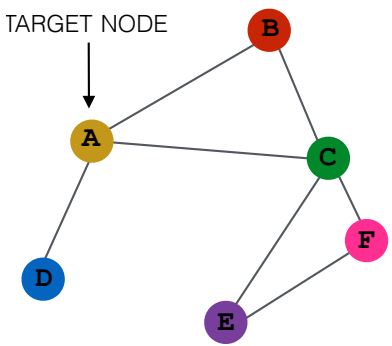
Gated Graph Neural Networks

Based on material from:

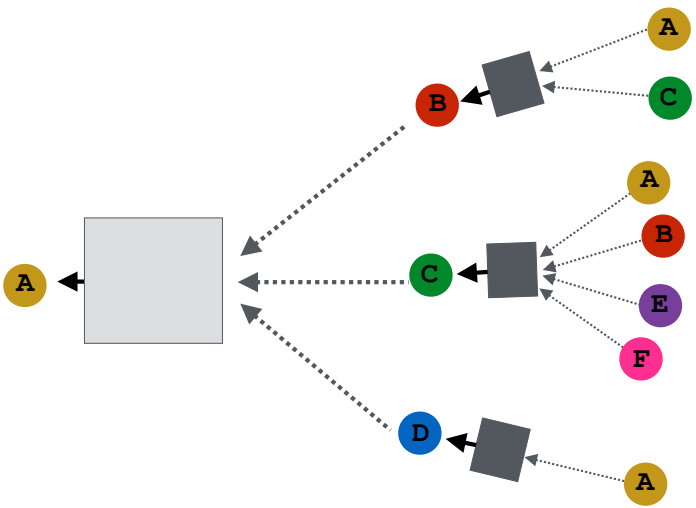
- Li et al., 2016. Gated Graph Sequence Neural Networks. *ICLR.*
- Gilmer et al., 2017. Neural Message Passing for Quantum Chemistry. *ICML.*

Neighborhood Aggregation

- **Basic idea:** Nodes aggregate "messages" from their neighbors using neural networks

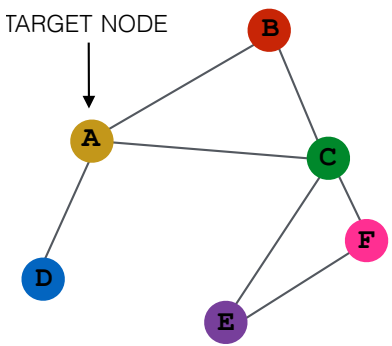


INPUT GRAPH

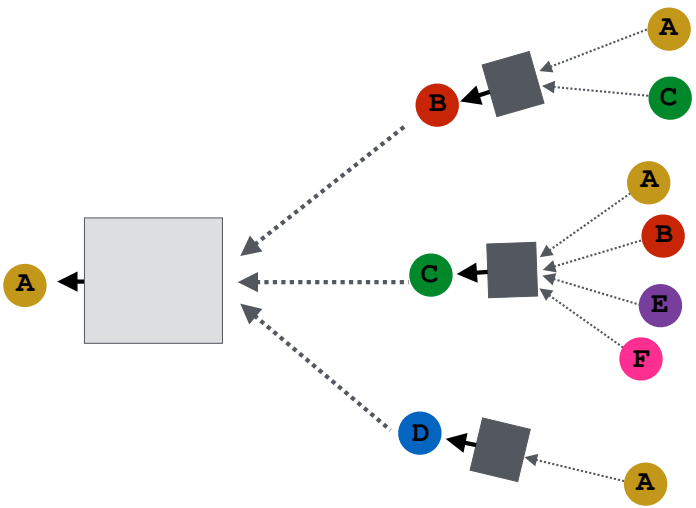


Neighborhood Aggregation

- GCNs and GraphSAGE **generally only 2-3 layers deep.**

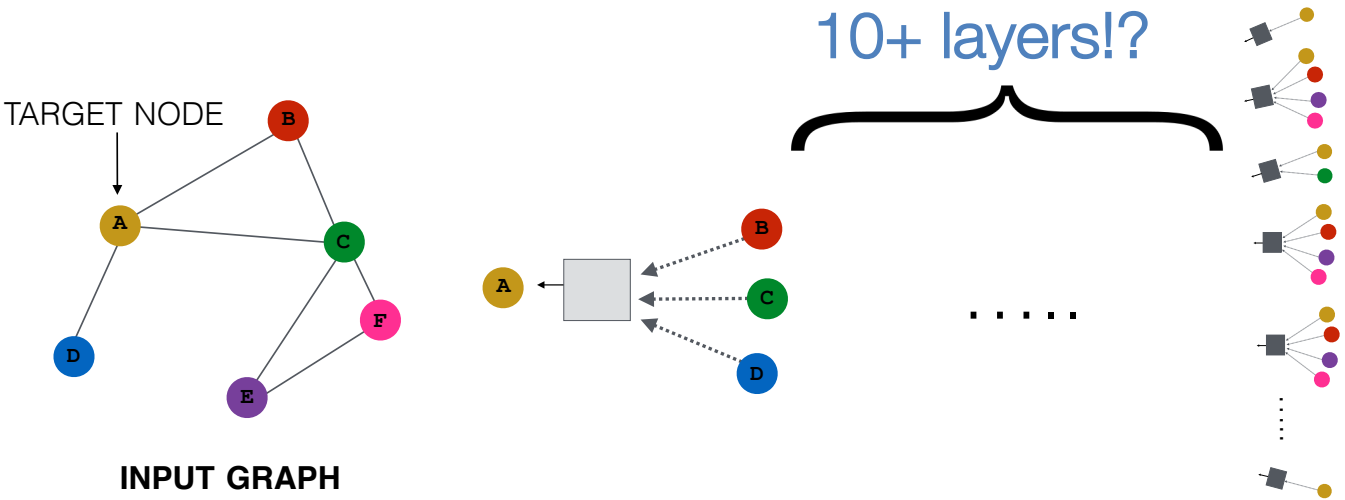


INPUT GRAPH



Neighborhood Aggregation

- But what if we want to go deeper?

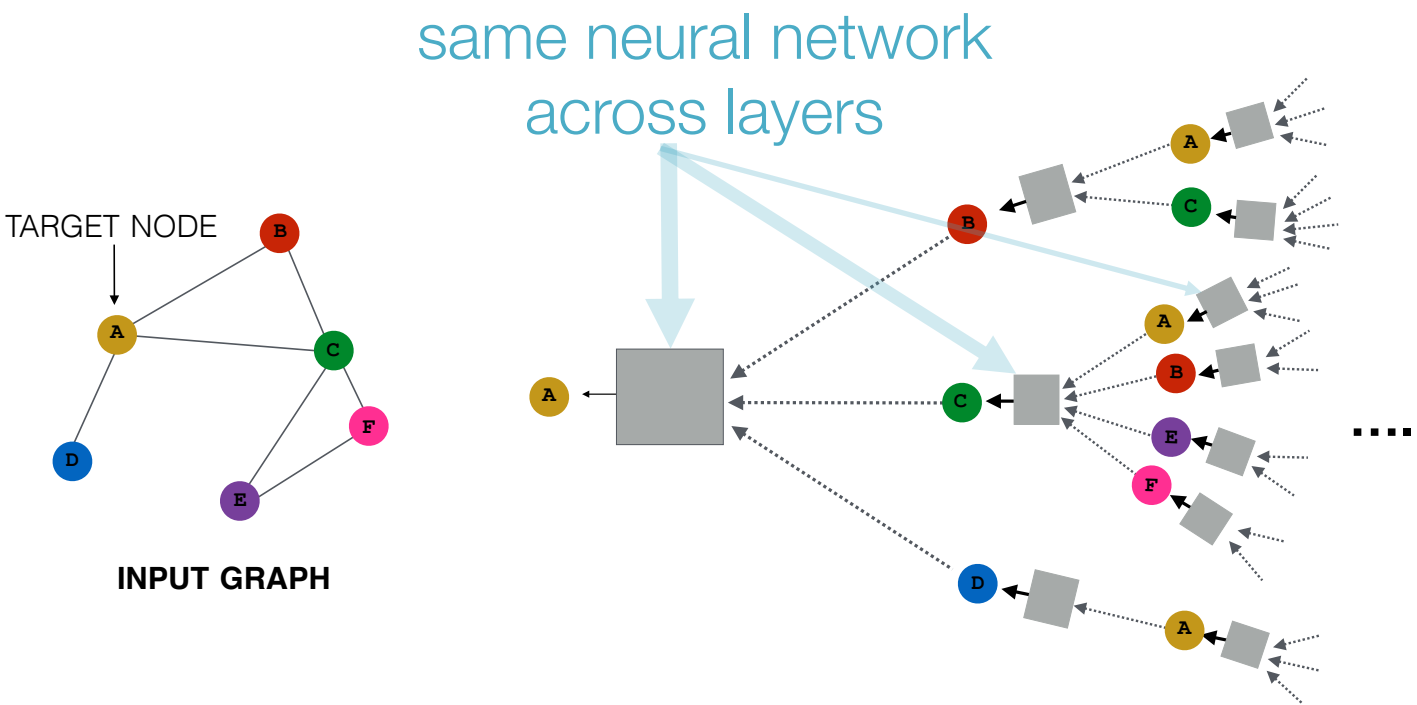


Gated Graph Neural Networks

- How can we build models with many layers of neighborhood aggregation?
- Challenges:
  - Overfitting from too many parameters.
  - Vanishing/exploding gradients during backpropagation.
- **Idea:** Use techniques from modern recurrent neural networks!

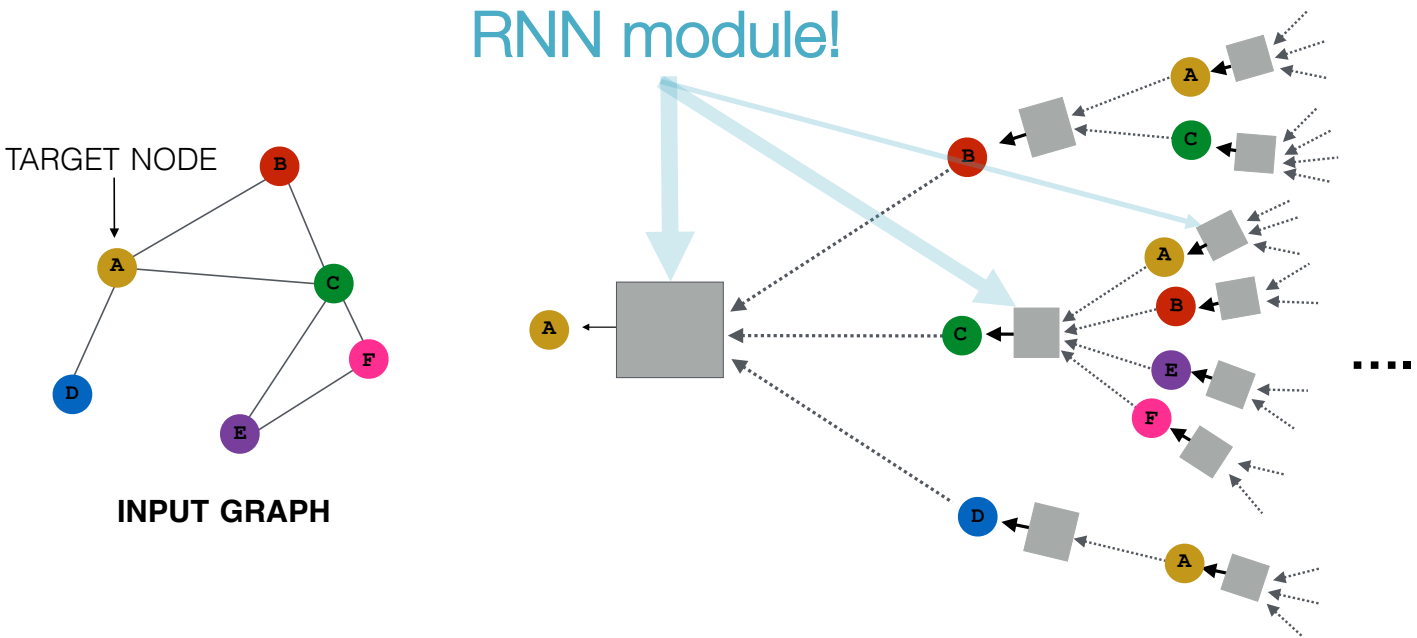
Gated Graph Neural Networks

- **Idea 1:** Parameter sharing across layers.



Gated Graph Neural Networks

- **Idea 2:** Recurrent state update.



The Math

- **Intuition:** Neighborhood aggregation with RNN state update.
  1. Get "message" from neighbors at step k:

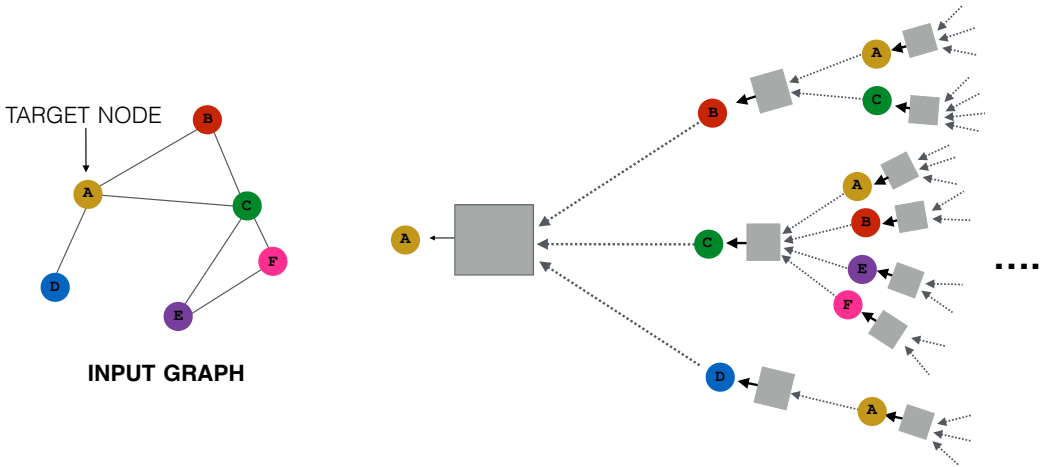
$$\mathbf{m}_v^k = \mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}$$

aggregation function does not depend on $\mathbf{k}$

  2. Update node "state" using Gated Recurrent Unit (GRU). New node state depends on the old state and the message from neighbors:

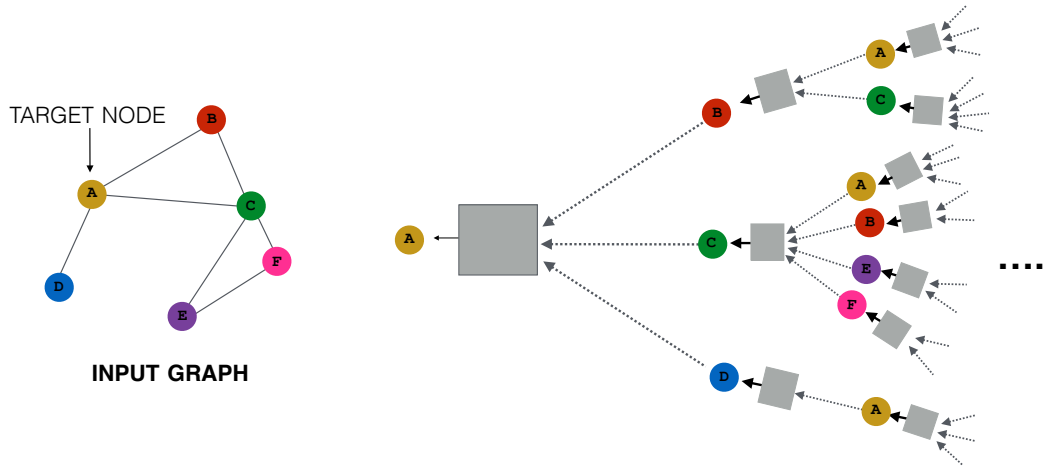
$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

Gated Graph Neural Networks



- Can handle models with >20 layers.
- Most real-world networks have small diameters (e.g., less than 7).
- Allows for complex information about global graph structure to be propagated to all nodes.

Gated Graph Neural Networks



- Useful for complex networks representing:
  - Logical formulas.
  - Programs.

Message-Passing Neural Networks

- **Idea:** We can generalize the gated graph neural network idea:

1. Get "message" from neighbors at step k:

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

Can incorporate edge features.

Generic "message" function (e.g., sum or MLP).

2. Update node "state":

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

Generic update function (e.g., LSTM or GRU)

Message-Passing Neural Networks

- This is a general conceptual framework that subsumes most GNNs.
  1. Get "message" from neighbors at step k:

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

  2. Update node "state":

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

- Gilmer et al., 2017. Neural Message Passing for Quantum Chemistry. *ICML.*

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE ✓
4. Gated Graph Neural Networks ✓
5. Graph Attention Networks 
6. Subgraph Embeddings

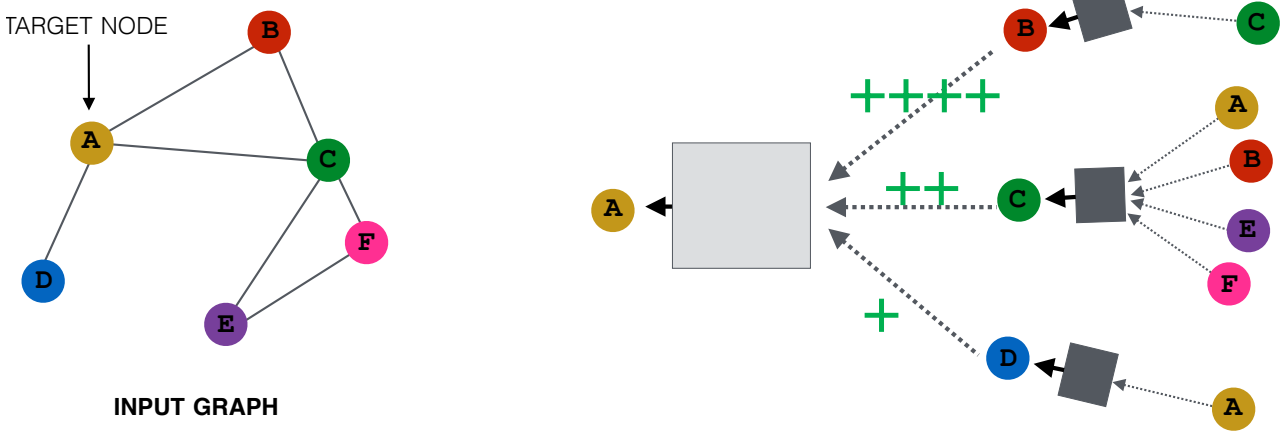
Graph Attention Networks

Based on material from:

- Velickovic et al., 2018. Graph Attention Networks. *ICLR.*

Neighborhood Attention

- What if some neighbors are more important than others?



Graph Attention Networks

- Augment basic graph neural network model with attention.

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

Non-linearity

Sum over all neighbors (and the node itself)

Learned attention weights!

Attention weights

- Various attention models are possible.
- The original GAT paper uses:

$$\alpha_{v,u} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_u]\right)\right)}{\sum_{u' \in N(v) \cup \{v\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_{u'}]\right)\right)}$$

- Achieved SOTA in 2018 on a number of standard benchmarks.

Attention in general

- Various attention mechanisms can be incorporated into the "message" step:
  1. Get "message" from neighbors at step k:

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

  2. Update node "state":

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

Incorporate attention here.

Recent advances in graph neural nets (not covered in detail here)

- Generalizations based on spectral convolutions:
  - Geometric Deep Learning (Bronstein et al., 2017)
  - Mixture Model CNNs (Monti et al., 2017)
- Speed improvements via subsampling:
  - FastGCNs (Chen et al., 2018)
  - Stochastic GCNs (Chen et al., 2017)
- And much more!!!

So what is SOTA?

- No consensus…
- Standard benchmarks ~2017-2018
  - Cora, CiteSeer, PubMed
  - Semi-supervised node classification.
  - **Extremely noisy evaluation and basic GNN/GCNs are very strong…**
- Attention, gating, and other modifications have shown improvements in specific settings (e.g., molecule classification, recommender systems).

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE ✓
4. Gated Graph Neural Networks ✓
5. Graph Attention Networks ✓
6. Subgraph Embeddings 

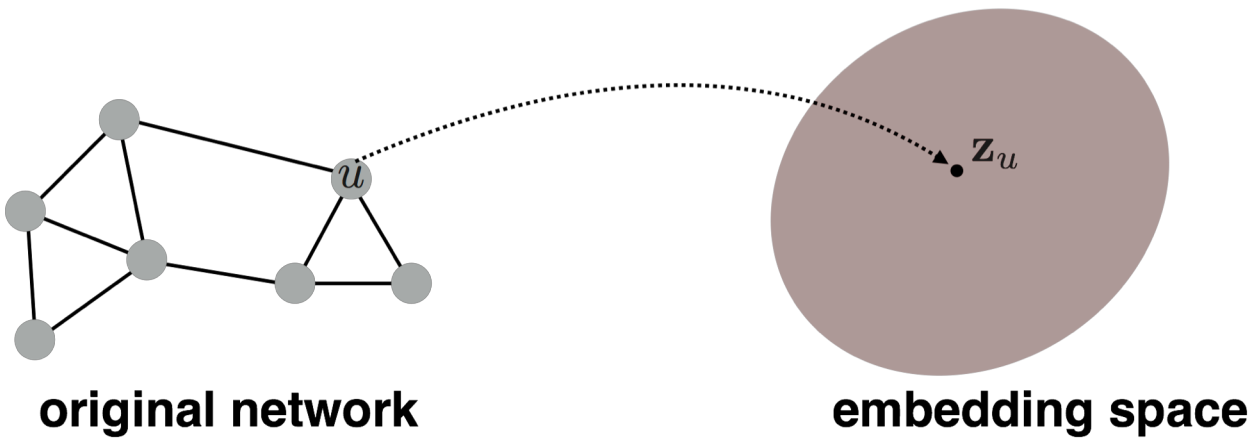
(Sub)graph Embeddings

Based on material from:

- Duvenaud et al. 2016. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *ICML.*
- Li et al. 2016. Gated Graph Sequence Neural Networks. *ICLR.*
- Ying et al, 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *NeurIPS*.

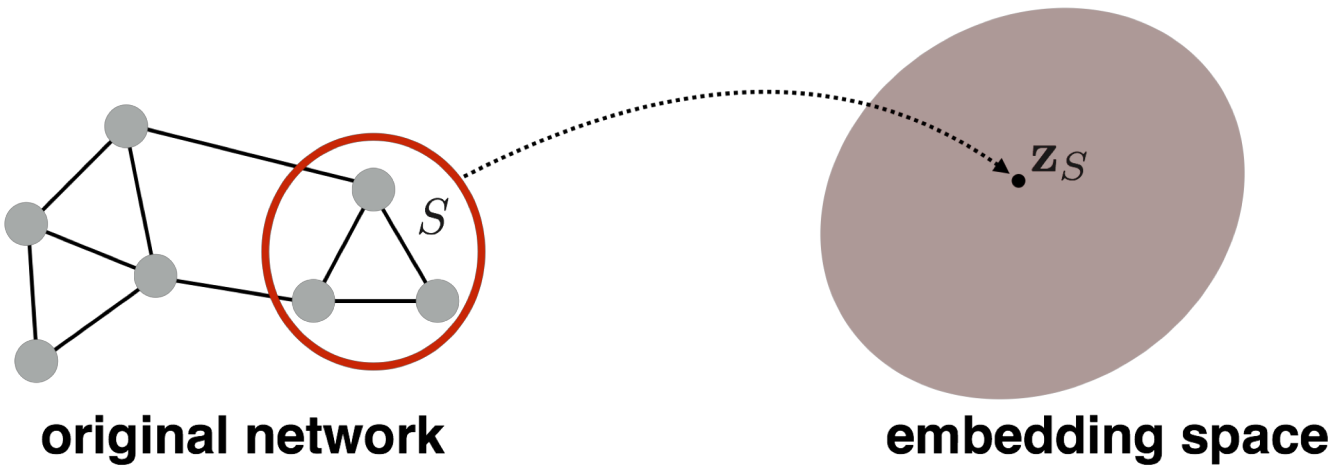
(Sub)graph Embeddings

- So far we have focused on node-level embeddings…



(Sub)graph Embeddings

- But what about subgraph embeddings?



Approach 1

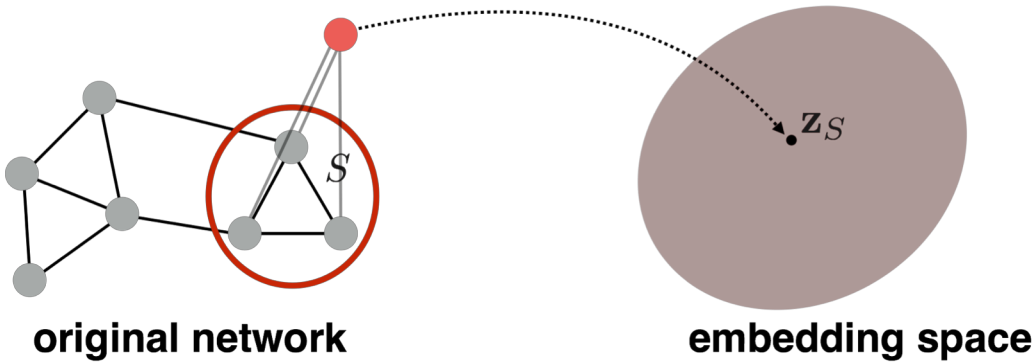
- **Simple idea:** Just sum (or average) the node embeddings in the (sub)graph

$$\mathbf{z}_S = \sum_{v \in S} \mathbf{z}_v$$

- Used by Duvenaud et al., 2016 to classify molecules based on their graph structure.

Approach 2

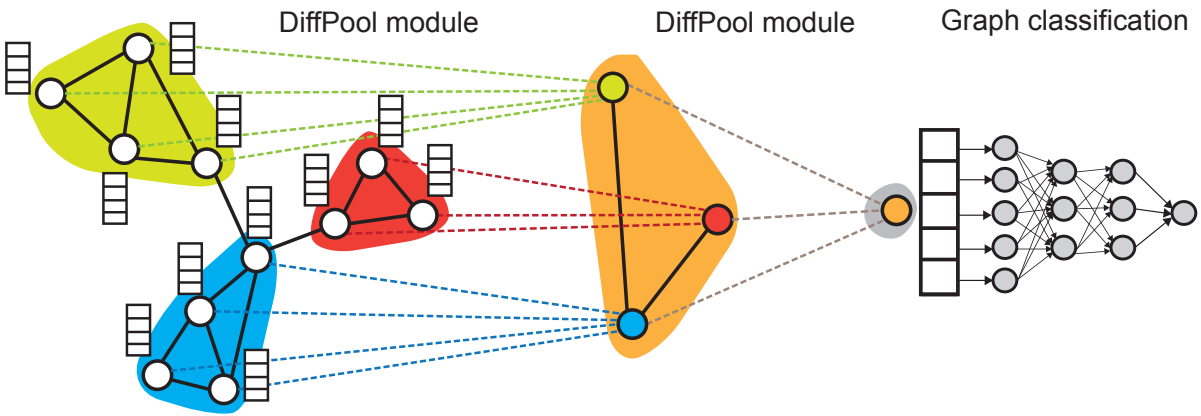
- **Idea:** Introduce a "virtual node" to represent the subgraph and run a standard graph neural network.



- Proposed by Li et al., 2016 as a general technique for subgraph embedding.

Approach 3

- **Idea:** Learn how to hierarchically cluster the nodes.



- First proposed by Ying et al., 2018 and currently SOTA(?).

Approach 3

- **Idea:** Learn to hierarchically cluster the nodes.
- Basic overview:
  1. Run GNN on graph and get node embeddings.
  2. Cluster the node embeddings together to make a "coarsened" graph.
  3. Run GNN on "coarsened" graph.
  4. Repeat.
- Different approaches to clustering:
  - Soft clustering via learned softmax weights (Ying et al., 2018)
  - Hard clustering (Cangea et al., 2018 and Gao et al., 2018)