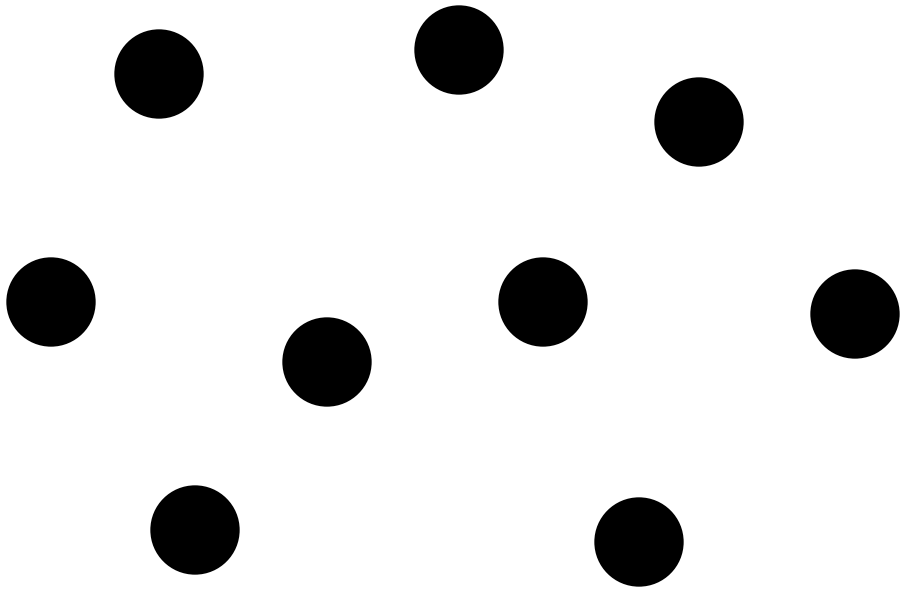
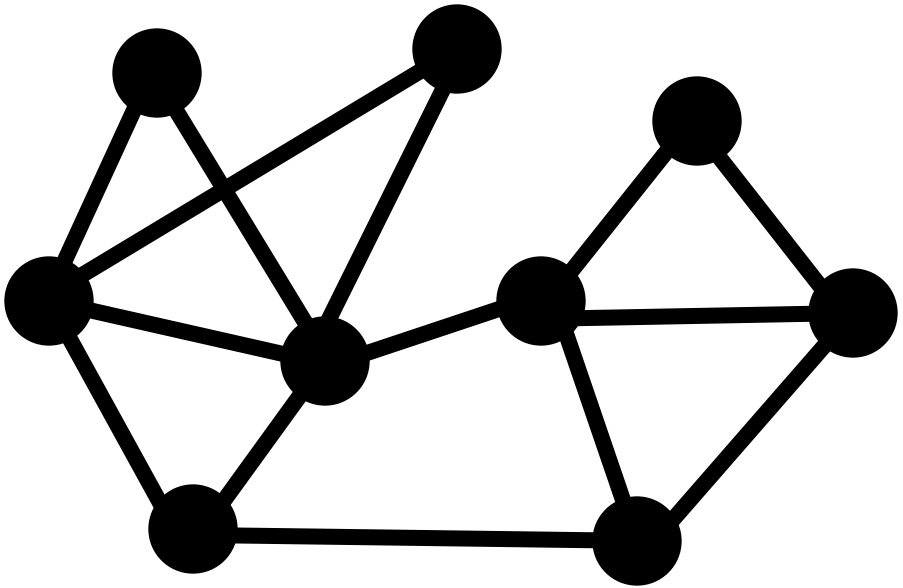
Graph Representation Learning

William L. Hamilton
COMP 551 – Special Topic Lecture

Why graphs?

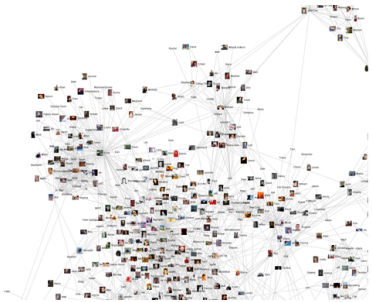
Graphs are a general language for describing and modeling complex systems



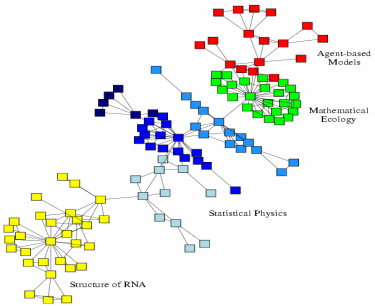


Graph!

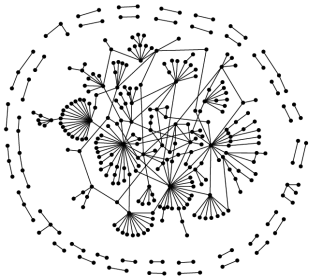
Many Data are Graphs



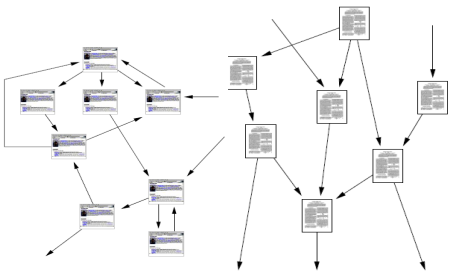
Social networks



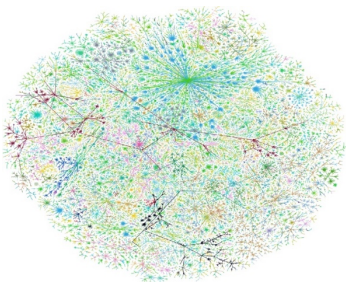
Economic networks



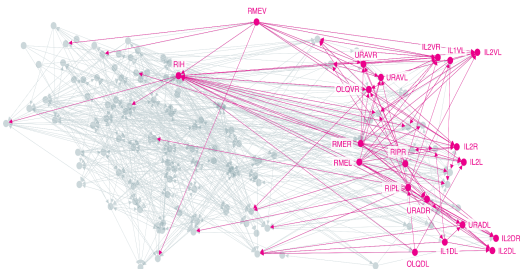
Biomedical networks



Information networks: Web & citations



Internet



Networks of neurons

Why Graphs? Why Now?

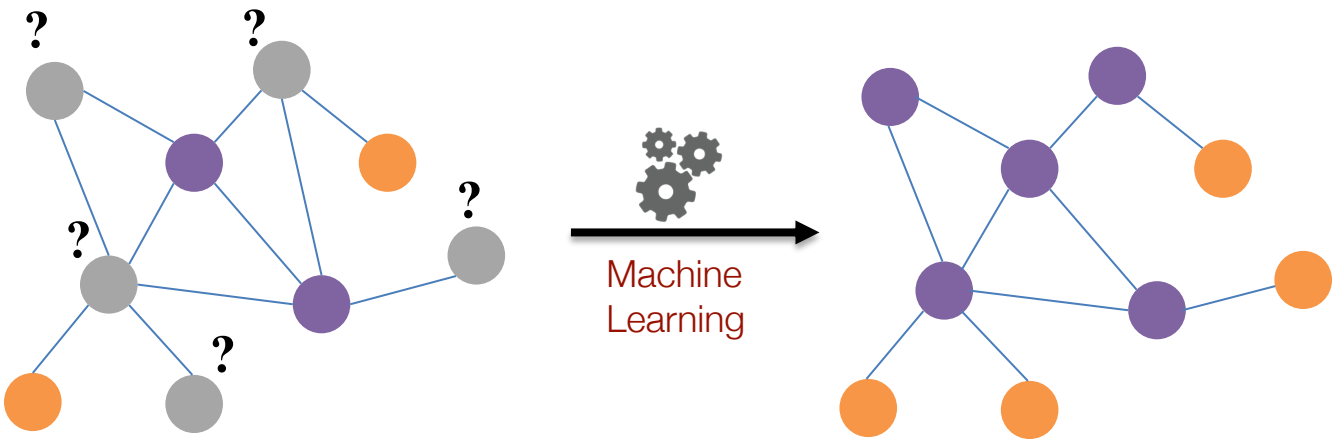
- Universal language for describing complex data
  - Networks/graphs from science, nature, and technology are more similar than one would expect
- Shared vocabulary between fields
  - Computer Science, Social science, Physics, Economics, Statistics, Biology
- Data availability (+computational challenges)
  - Web/mobile, bio, health, and medical
- Impact!
  - Social networking, Social media, Drug design

Machine Learning with Graphs

Classical ML tasks in graphs:

- Node classification
  - Predict a type of a given node
- Link prediction
  - Predict whether two nodes are linked
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks

Example: Node Classification



Example: Node Classification

Classifying the function of proteins in the interactome!

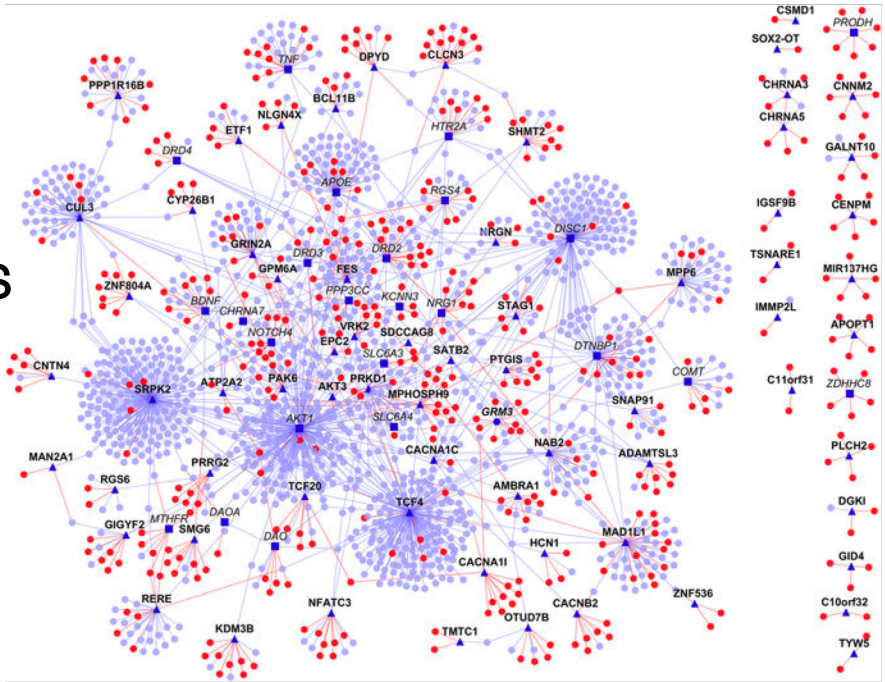
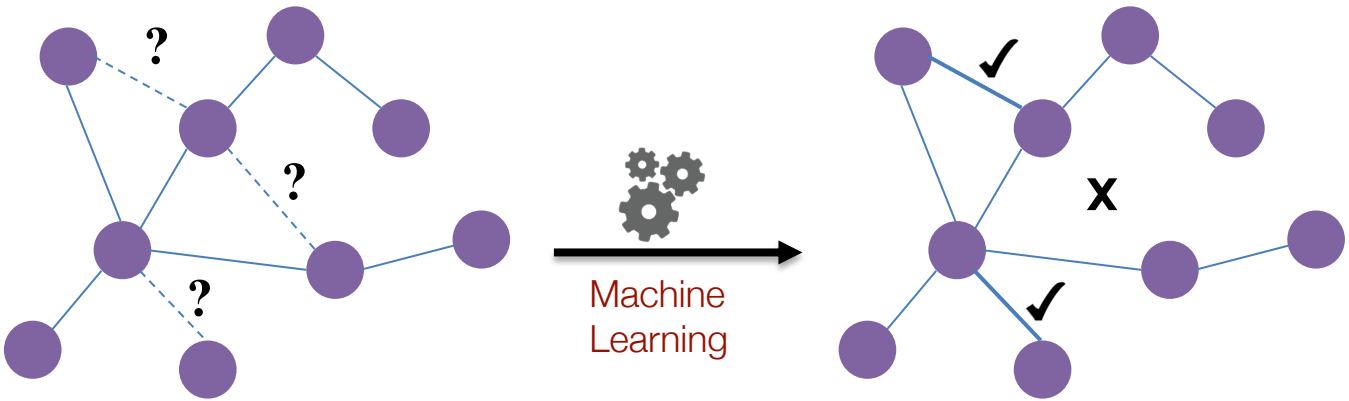


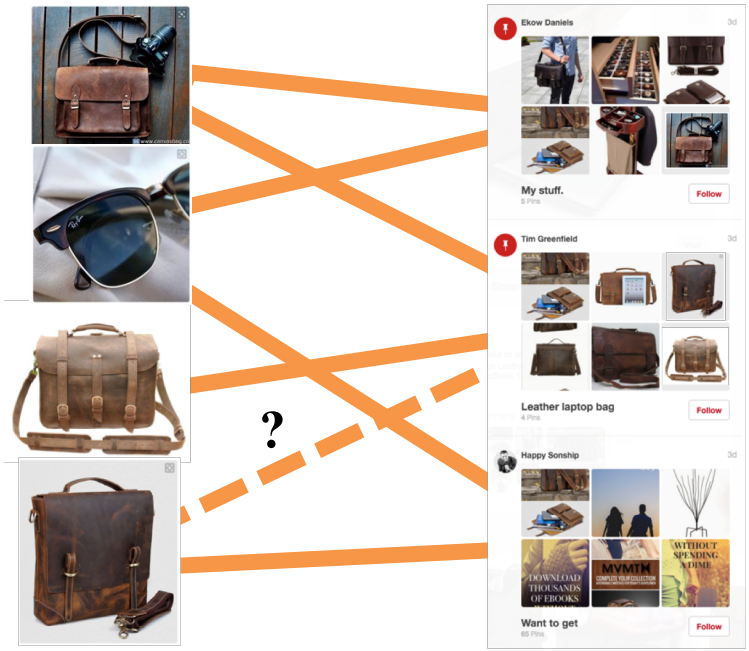
Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

Example: Link Prediction



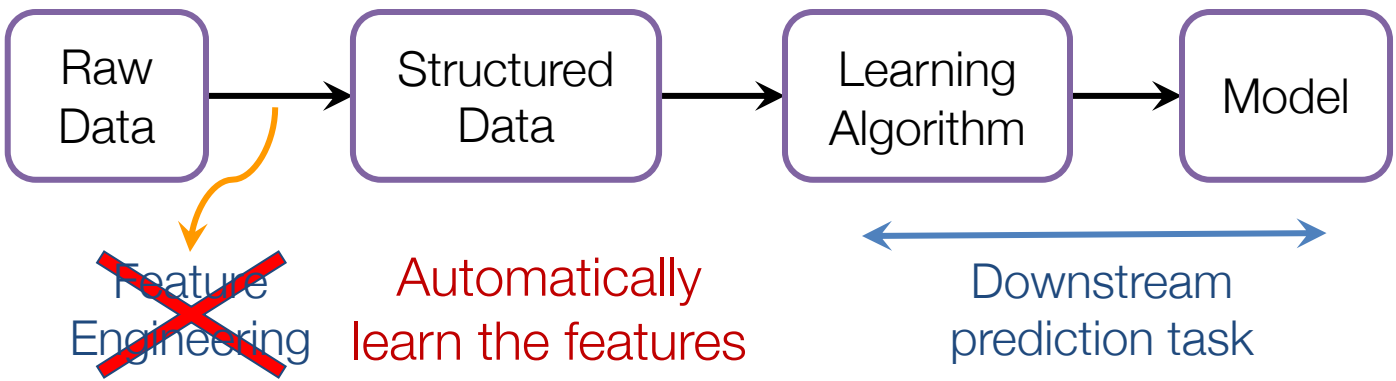
Example: Link Prediction

Content recommendation is link prediction!



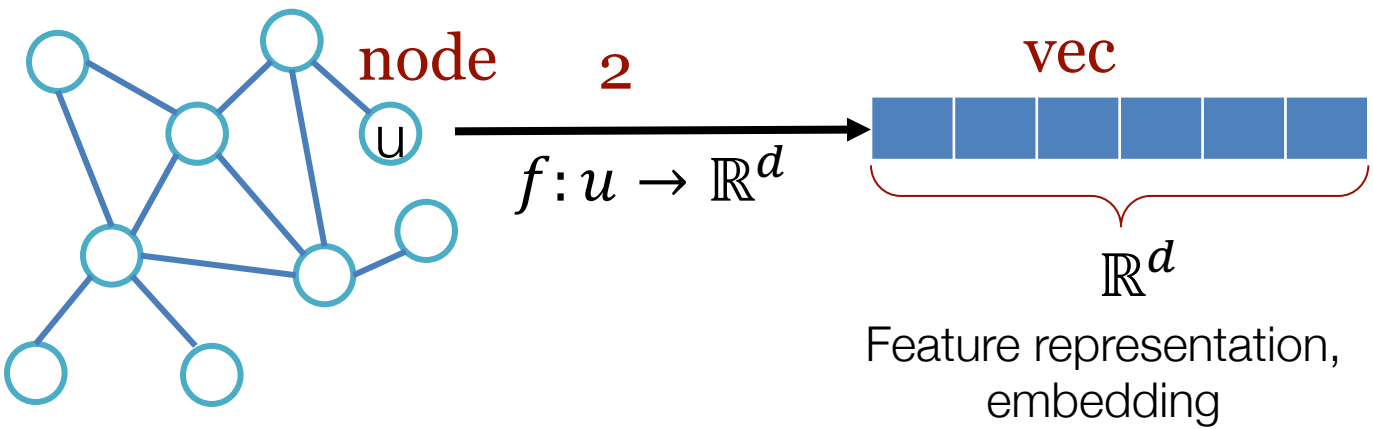
Machine Learning Lifecycle

- (Supervised) Machine Learning Lifecycle: This feature, that feature. Every single time!



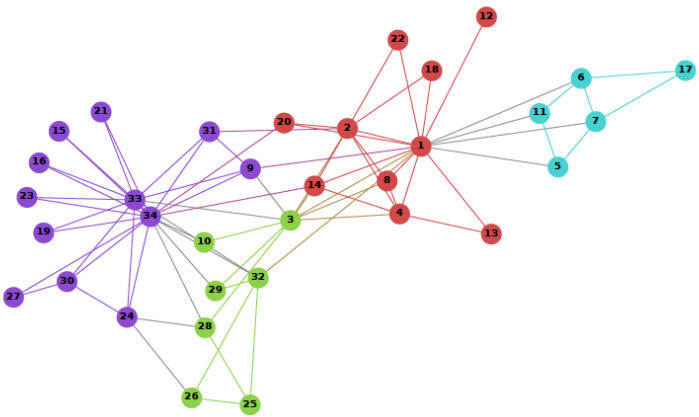
Feature Learning in Graphs

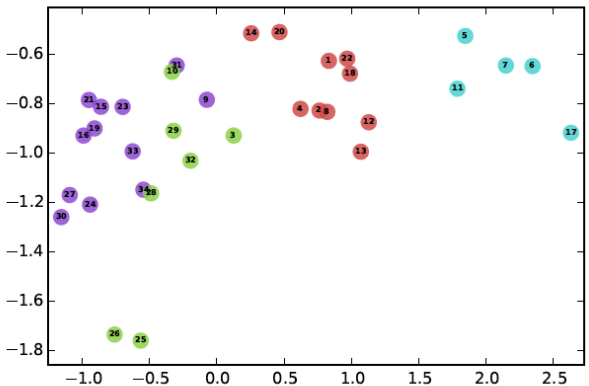
Goal: Efficient task-independent feature learning for machine learning in graphs!



Example

- Zachary's Karate Club Network:



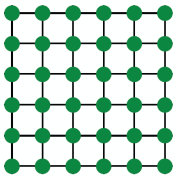
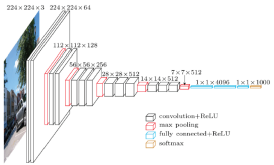


Input Output

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

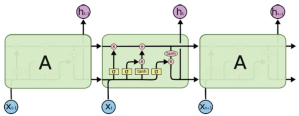
Why Is It Hard?

- Modern deep learning toolbox is designed for simple sequences or grids.
  - CNNs for fixed-size images/grids….

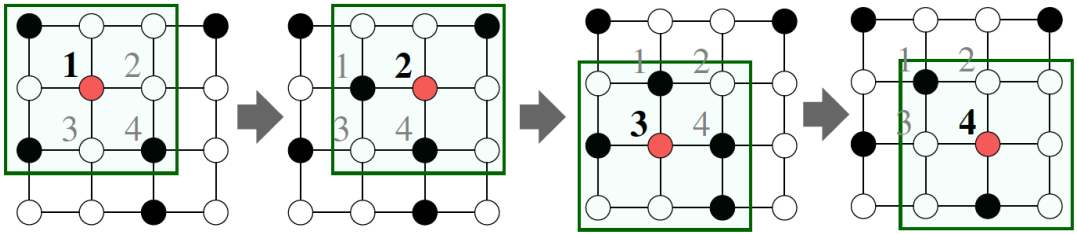
  - RNNs or word2vec for text/sequences…





Why Is It Hard?

- But graphs are far more complex!
  - Complex topographical structure (i.e., no spatial locality like grids)



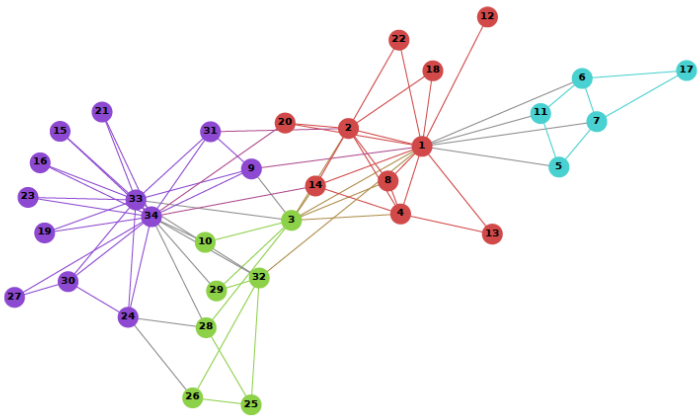
  - No fixed node ordering or reference point (i.e., the isomorphism problem)
  - Often dynamic and have multimodal features.

This talk

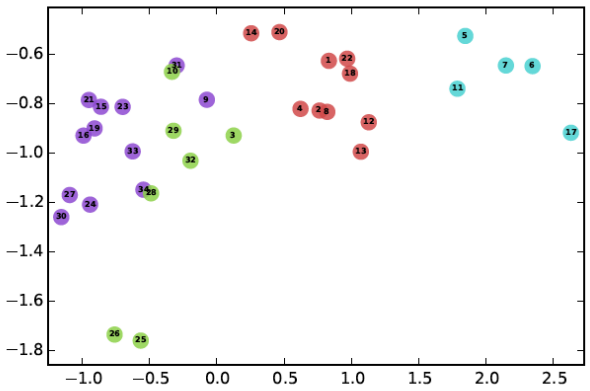
- 1) Node embeddings
  - Map nodes to low-dimensional embeddings.
- 2) Graph neural networks
  - Deep learning architectures for graph-structured data
- 3) Example applications.

Part 1: Node Embeddings

Embedding Nodes



Input



Output

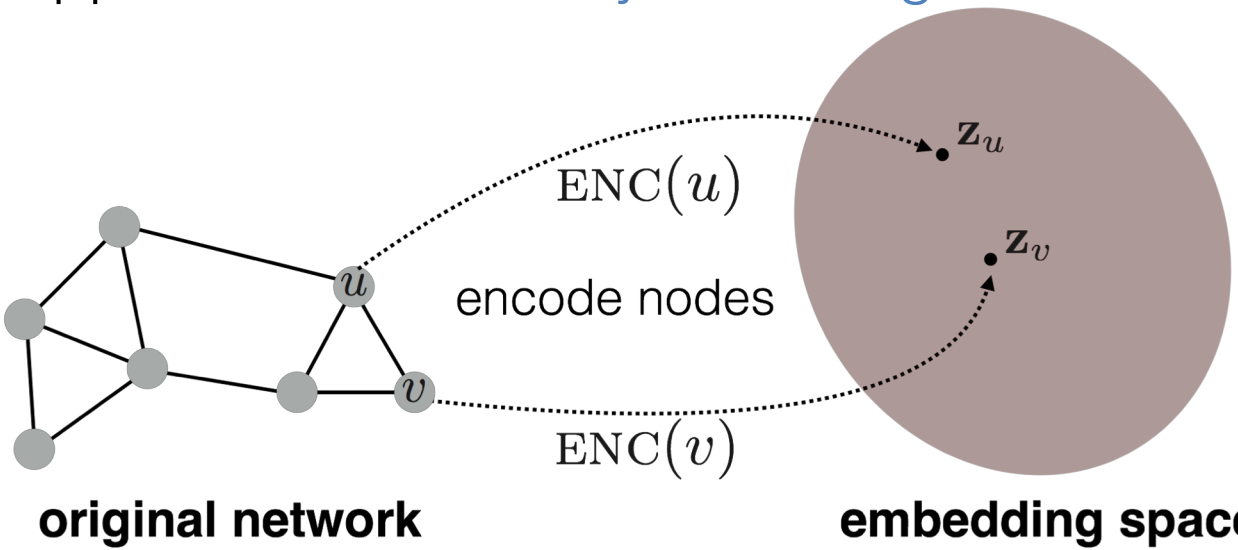
Intuition: Find embedding of nodes to d-dimensions so that "similar" nodes in the graph have embeddings that are close together.

Setup

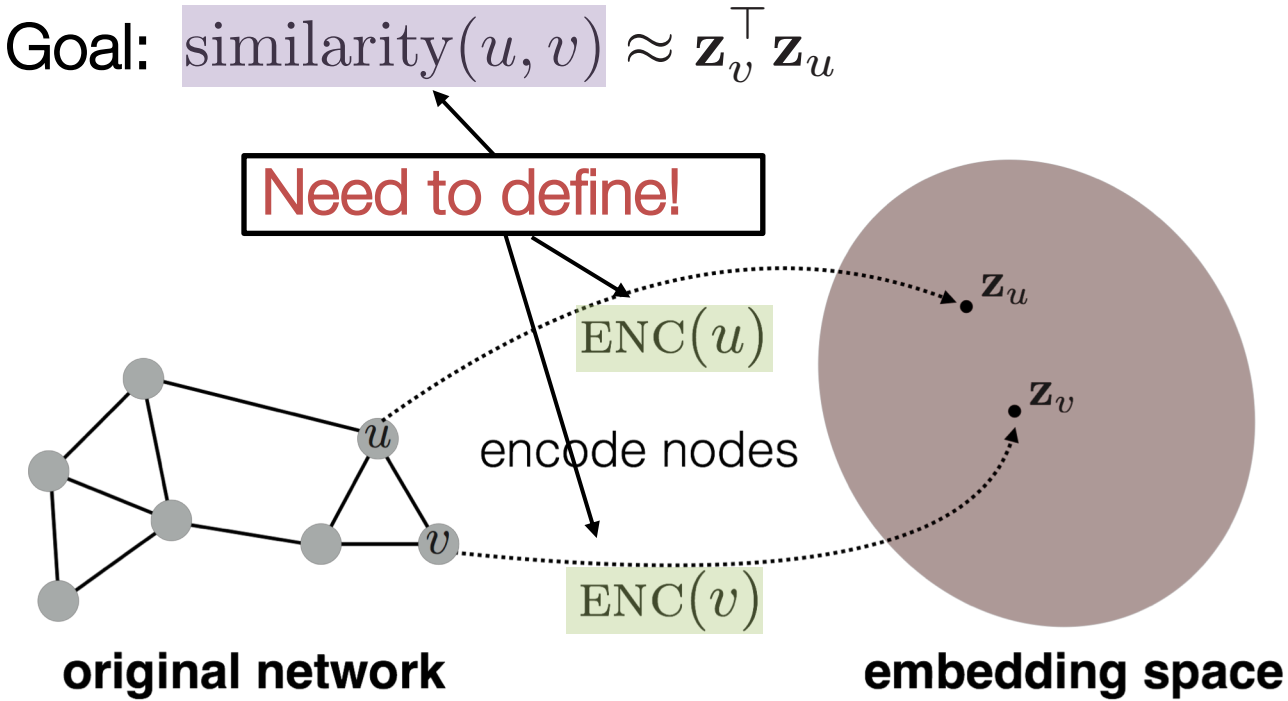
- Assume we have a graph $\mathbf{G}$:
  - $\mathbf{V}$ is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - No node features or extra information is used!

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



Learning Node Embeddings

1. **Define an encoder** (i.e., a mapping from nodes to embeddings)
2. **Define a node similarity function** (i.e., a measure of similarity in the original network).
3. **Optimize the parameters of the encoder so that:**

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

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

“Shallow” Encoding

- Simplest encoding approach: encoder is just an embedding-lookup

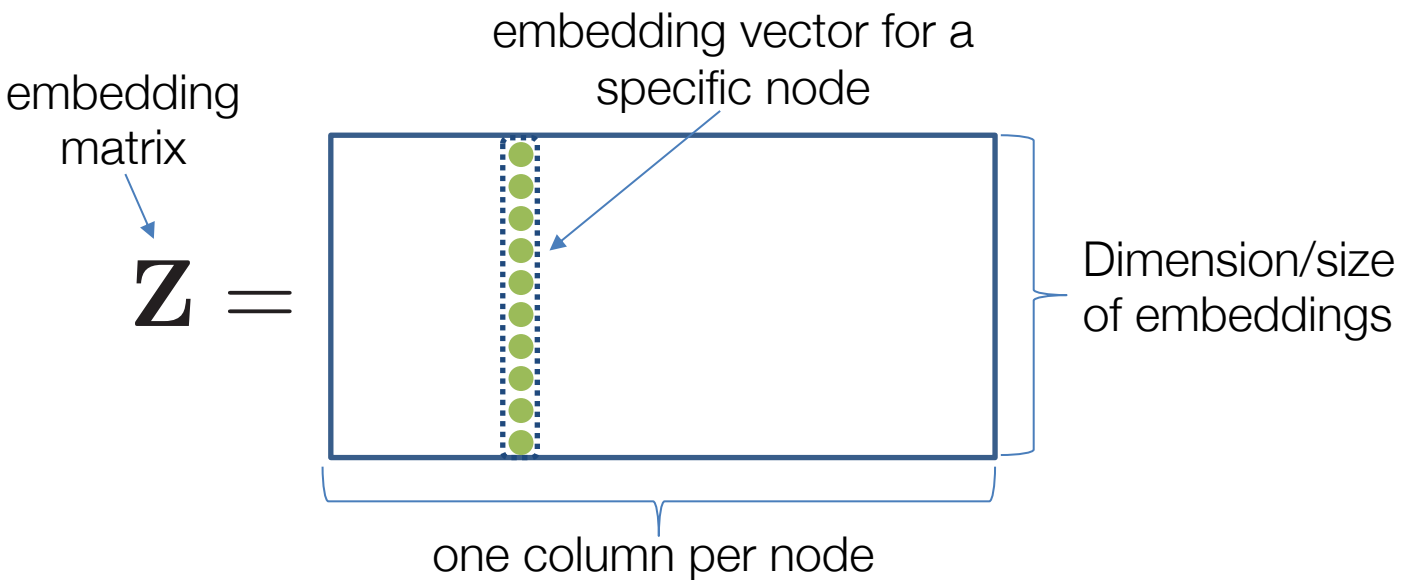
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is node embedding [what we learn!]

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

“Shallow” Encoding

- Simplest encoding approach: encoder is just an embedding-lookup



“Shallow” Encoding

- Simplest encoding approach: encoder is just an embedding-lookup.

i.e., each node is assigned a unique embedding vector.

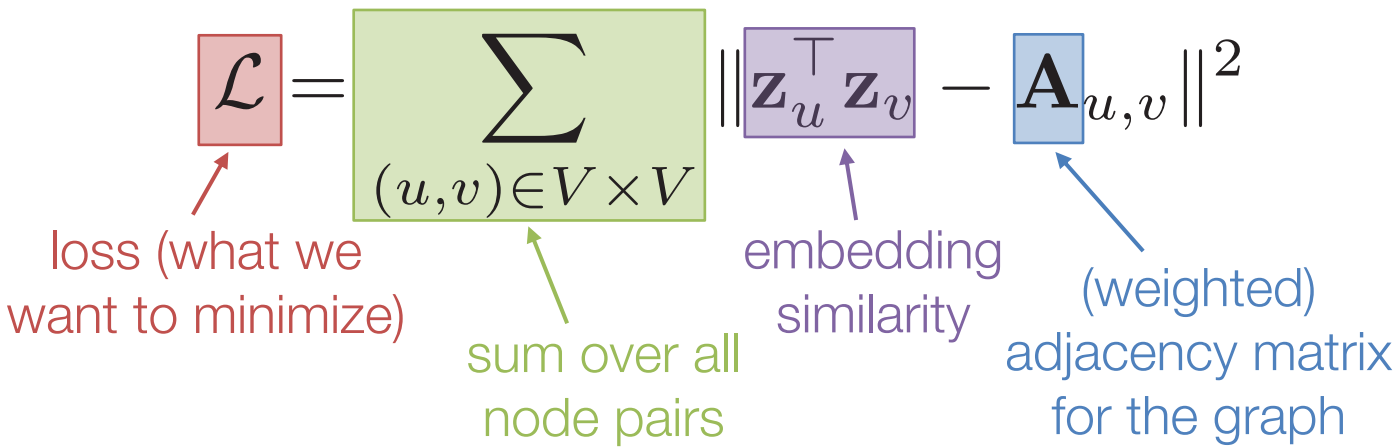
- E.g., node2vec, DeepWalk, LINE

How to Define Node Similarity?

- Key distinction between "shallow" methods is how they define node similarity.
- E.g., should two nodes have similar embeddings if they….
  - are connected?
  - share neighbors?
  - have similar "structural roles"?
  - …?

Adjacency-based Similarity

- Similarity function is just the edge weight between u and v in the original network.
- Intuition: Dot products between node embeddings approximate edge existence.



Adjacency-based Similarity

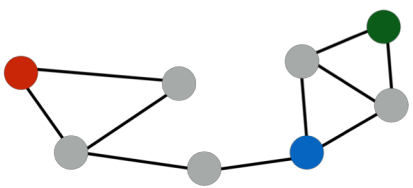
$$\mathcal{L} = \sum_{(u,v) \in V \times V} \left\| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \right\|^2$$

- Find embedding matrix $\mathbf{Z} \in \mathbb{R}^{\boldsymbol{d} \times |\boldsymbol{V}|}$ that minimizes the loss $\mathcal{L}$
  - Option 1: Use stochastic gradient descent (SGD) as a general optimization method.
    - Highly scalable, general approach
  - Option 2: Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines).
    - Only works in limited cases.

Adjacency-based Similarity

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

- Drawbacks:
  - $O(|V|^2)$ runtime. (Must consider all node pairs.)
    - Can make $O(|E|)$ by only summing over non-zero edges and using regularization (e.g., Ahmed et al., 2013)
  - $O(|V|)$ parameters! (One learned vector per node).
  - Only considers direct, local connections.



e.g., the blue node is obviously more similar to green compared to red node, despite none having direct connections.

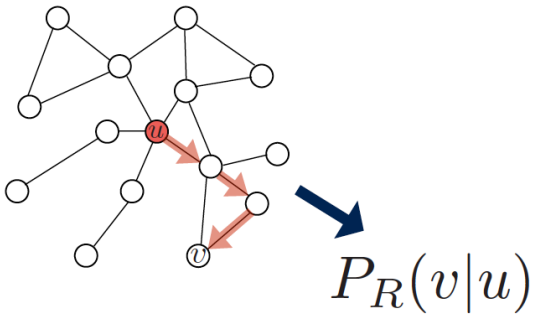
Random-walk Embeddings

$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

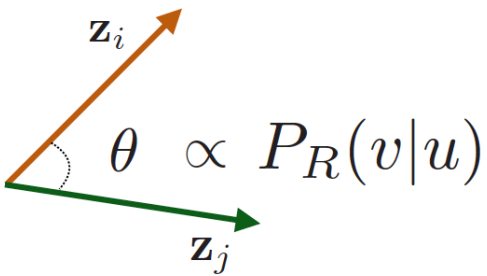
probability that u and v co-occur on a random walk over the network

Random-walk Embeddings

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



2. Optimize embeddings to encode these random walk statistics.



Why Random Walks?

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

Random Walk Optimization

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

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

* $\mathrm{N}_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks.

Random Walk Optimization

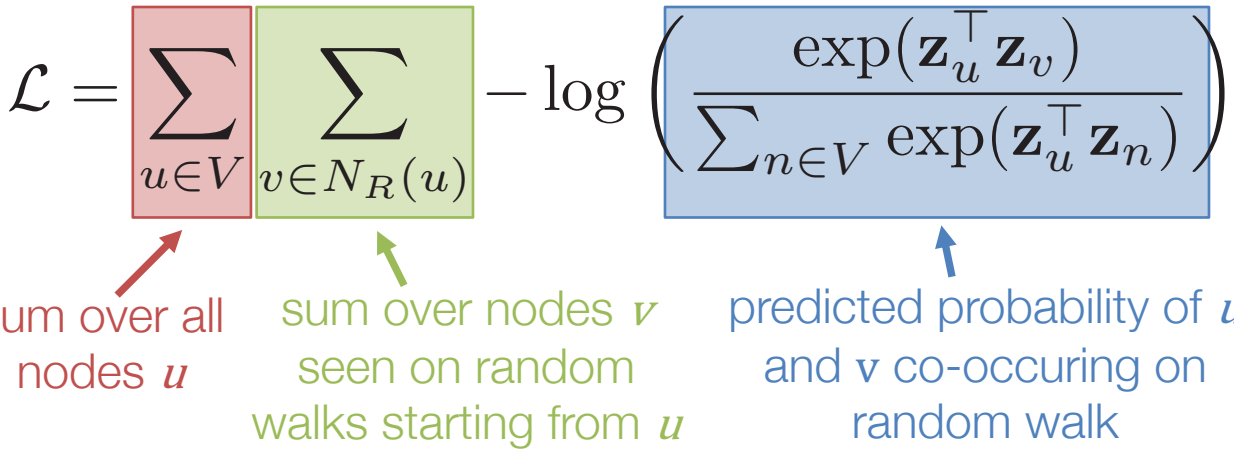
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- Intuition: Optimize embeddings to maximize likelihood of random walk co-occurrences.
- Parameterize $\mathrm{P}(v|\,\mathbf{z}_u)$ using softmax:

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

Random Walk Optimization

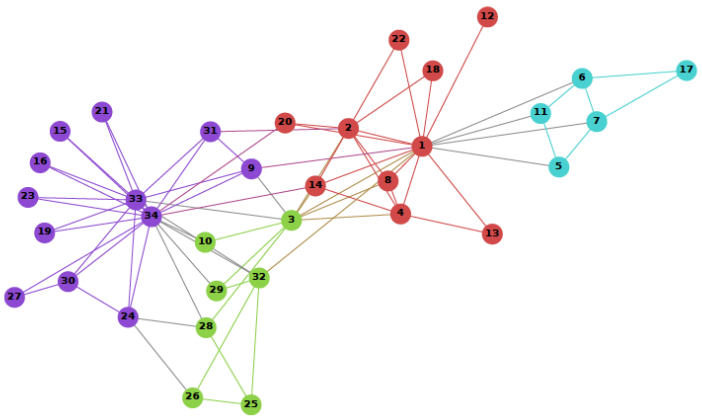
Putting things together:



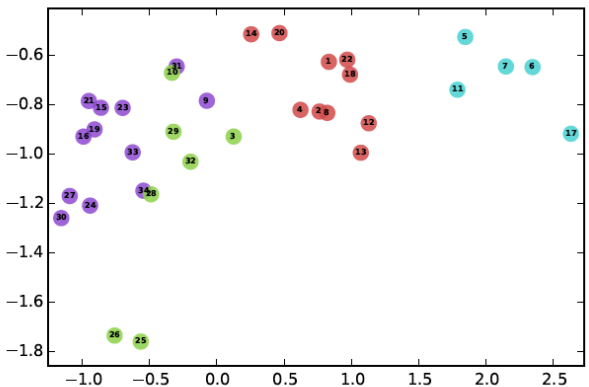
Optimizing random walk embeddings =

Finding embeddings $\mathbf{z}_u$ that minimize $\mathcal{L}$

Example: DeepWalk



Input

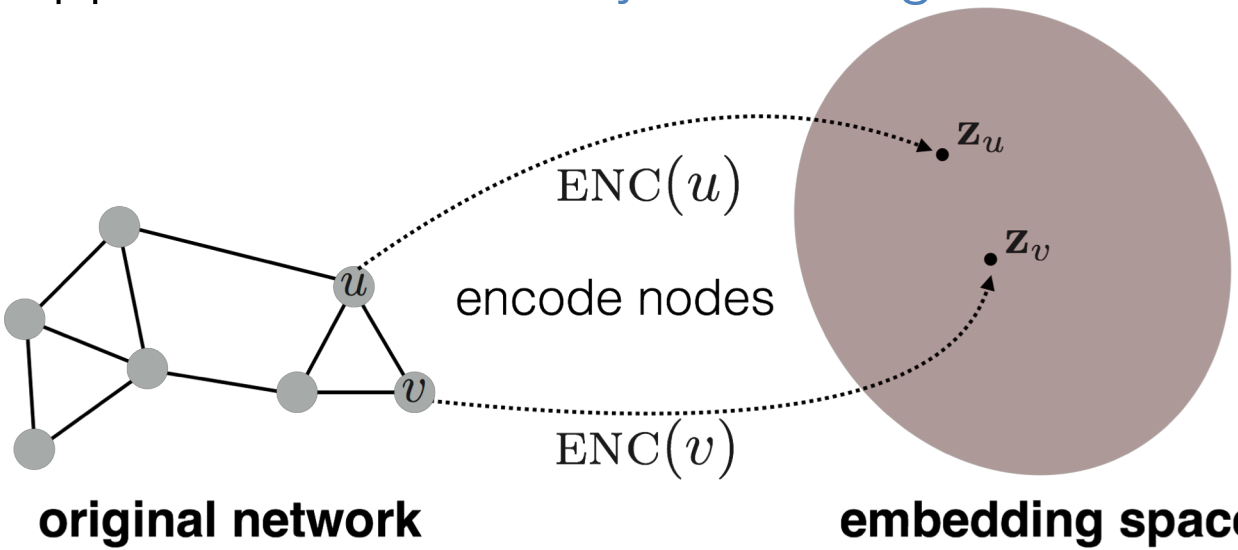


Output

Part 2: Graph Neural Networks

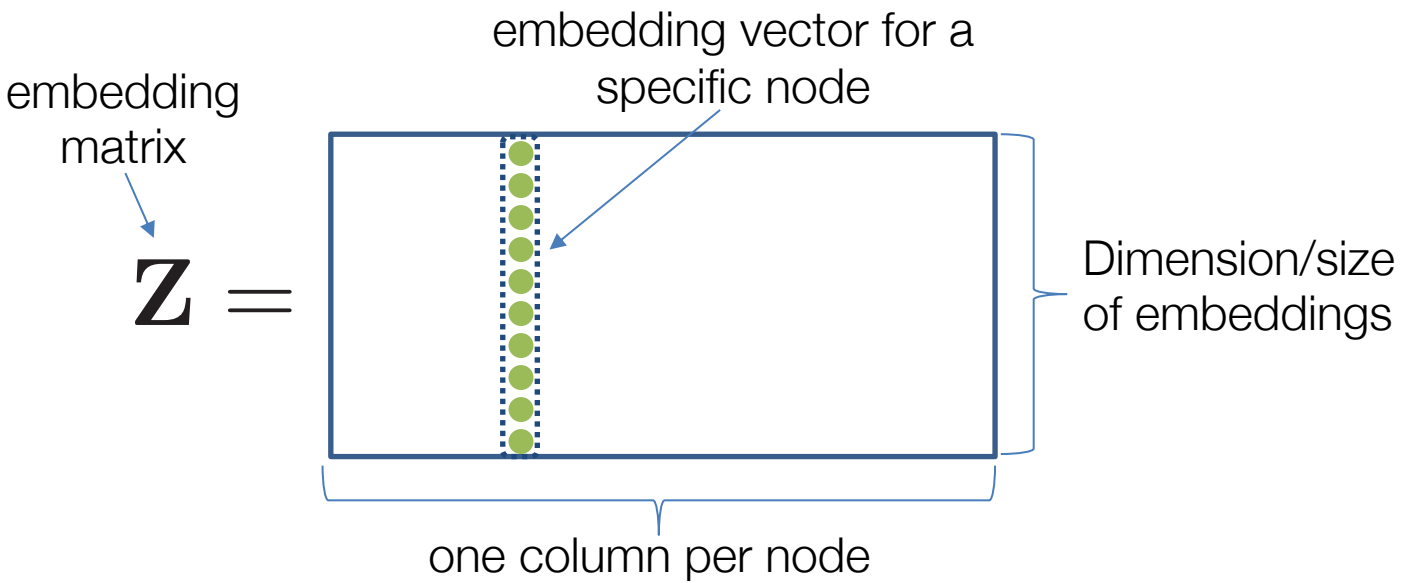
Embedding Nodes

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



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.

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

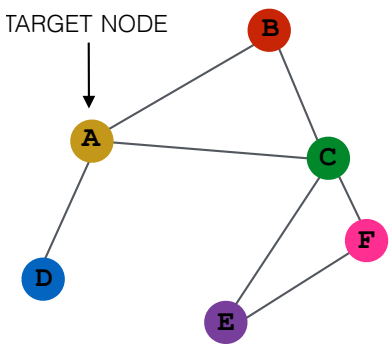
- In general, all of these more complex encoders can be combined with the similarity functions from the previous section.

Setup

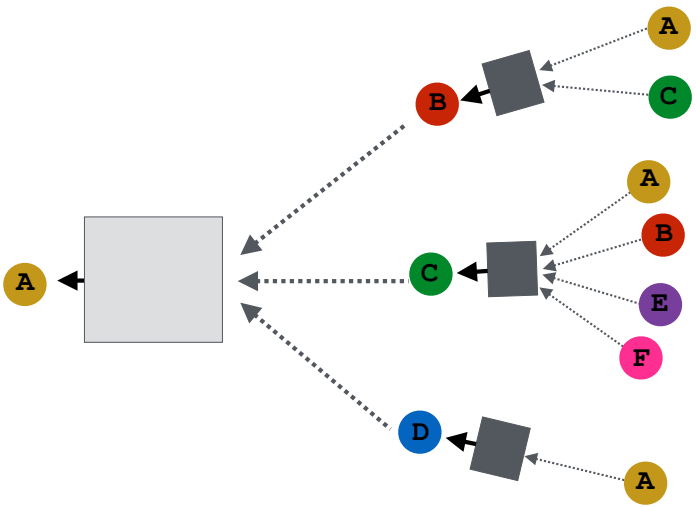
- Assume we have a graph $\mathbf{G}$:
  - $\mathbf{V}$ is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - $\mathbf{X} \in \mathbf{R}^{m \times |V|}$ is a matrix of node features.
    - Categorical attributes, text, image data
      - E.g., profile information in a social 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.

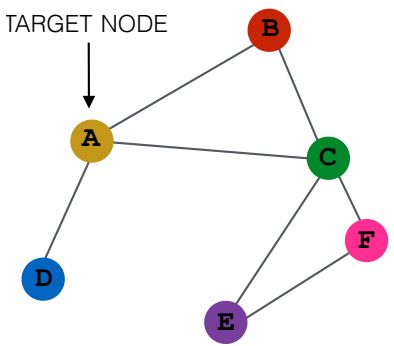


INPUT GRAPH

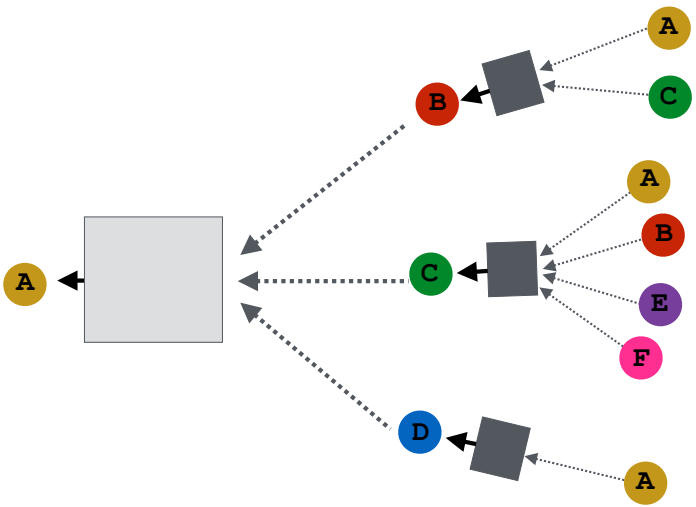


Neighborhood Aggregation

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



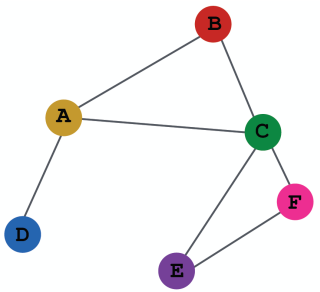
INPUT GRAPH



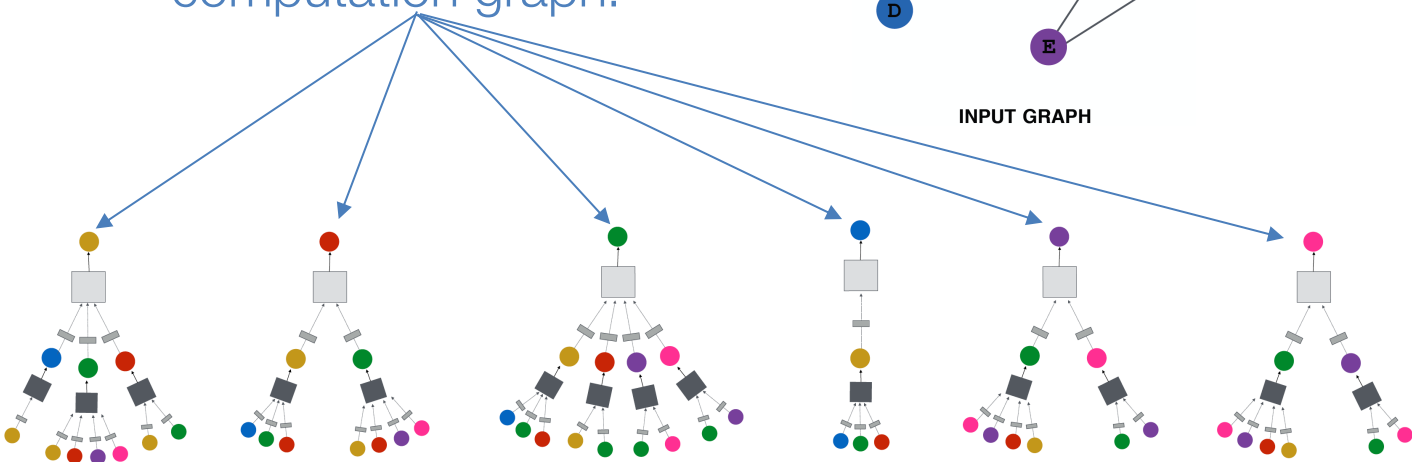
Neighborhood Aggregation

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

Every node defines a unique computation graph!

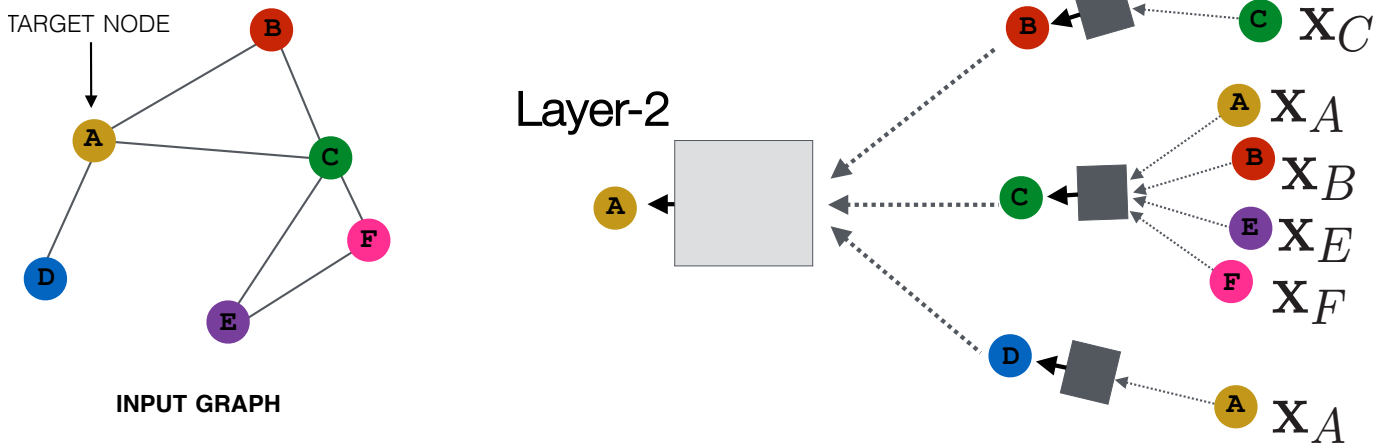


INPUT GRAPH



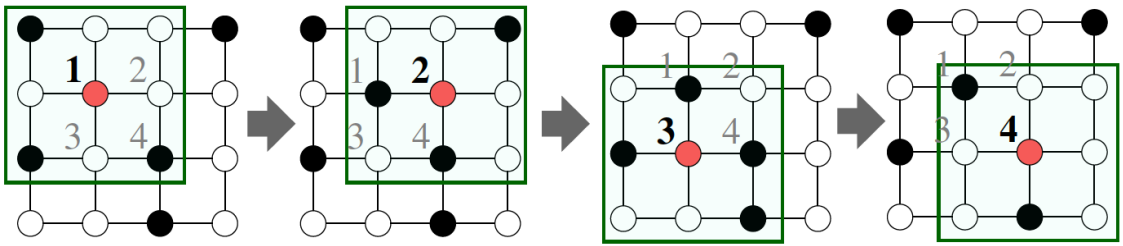
Neighborhood Aggregation

- 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"

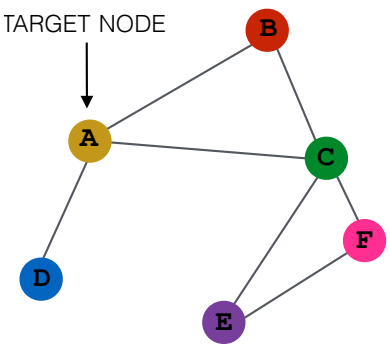
- Neighborhood aggregation can be viewed as a center-surround filter.



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

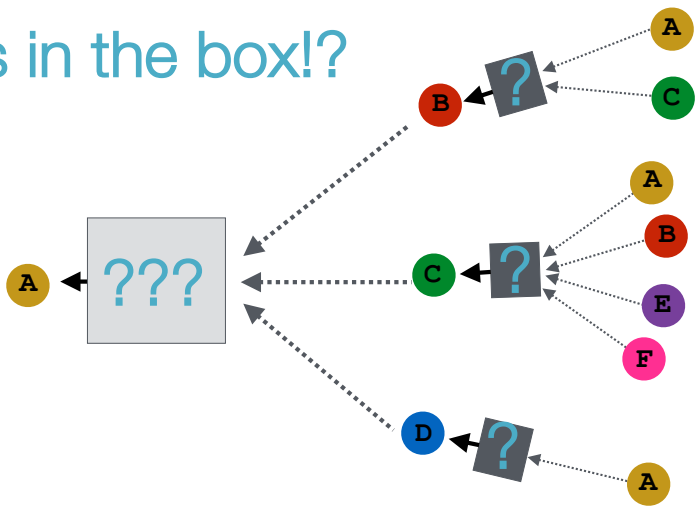
Neighborhood Aggregation

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



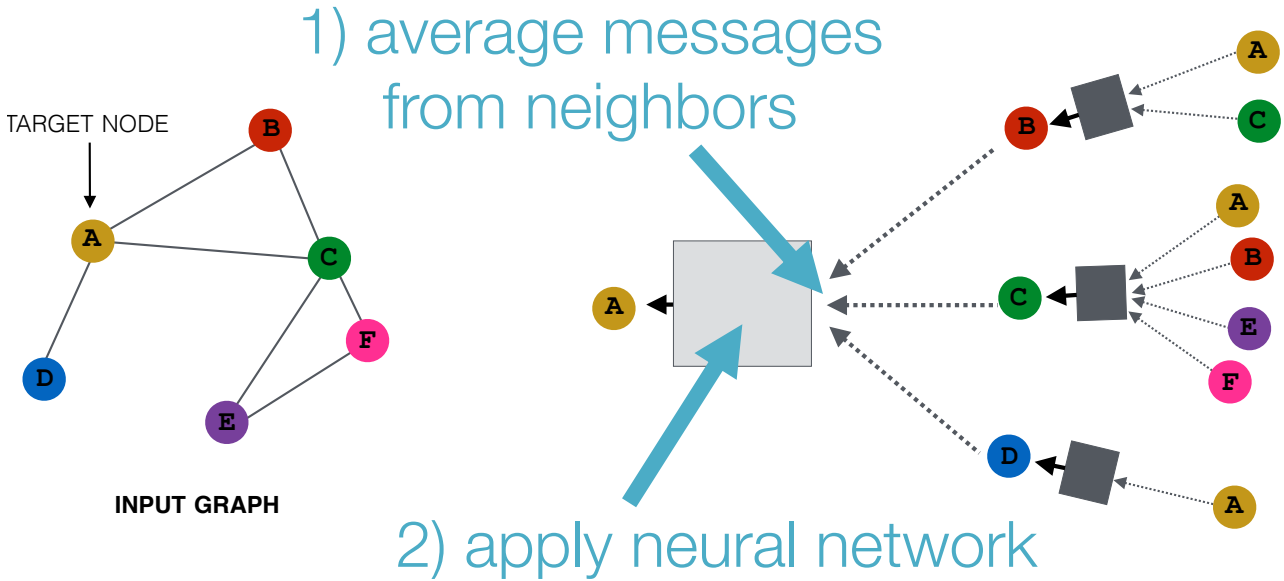
INPUT GRAPH

what's in the box!?



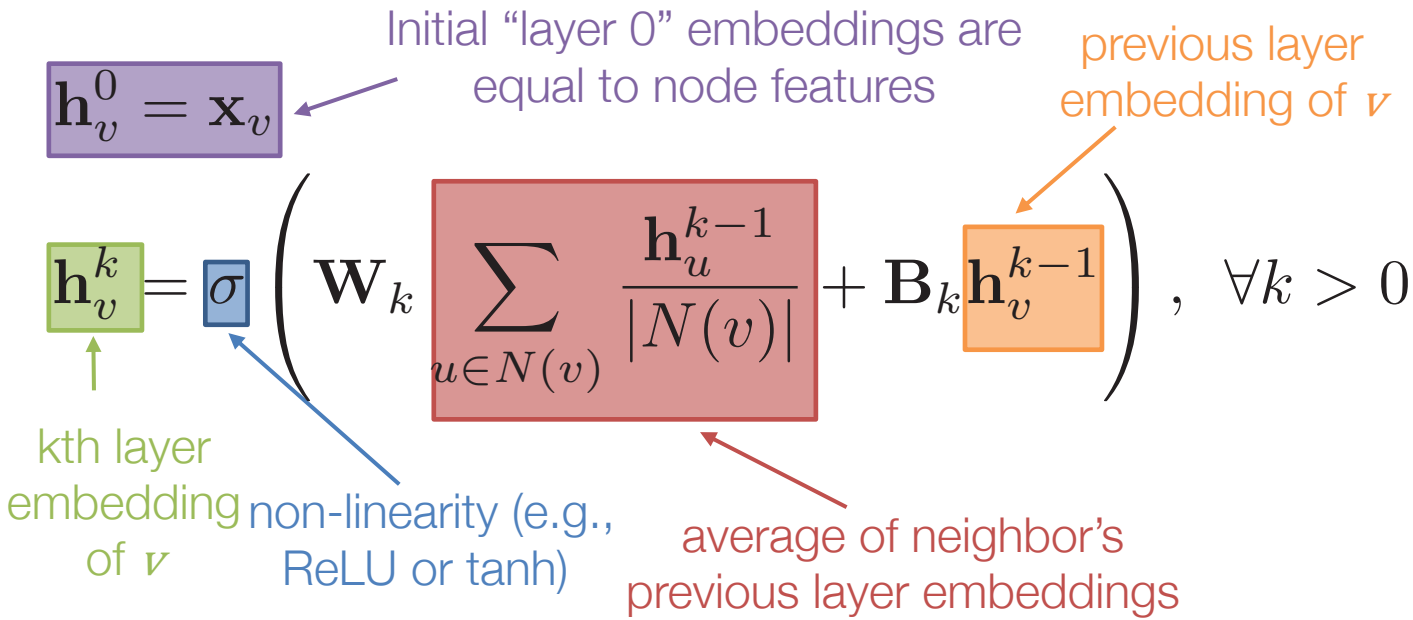
Neighborhood Aggregation

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



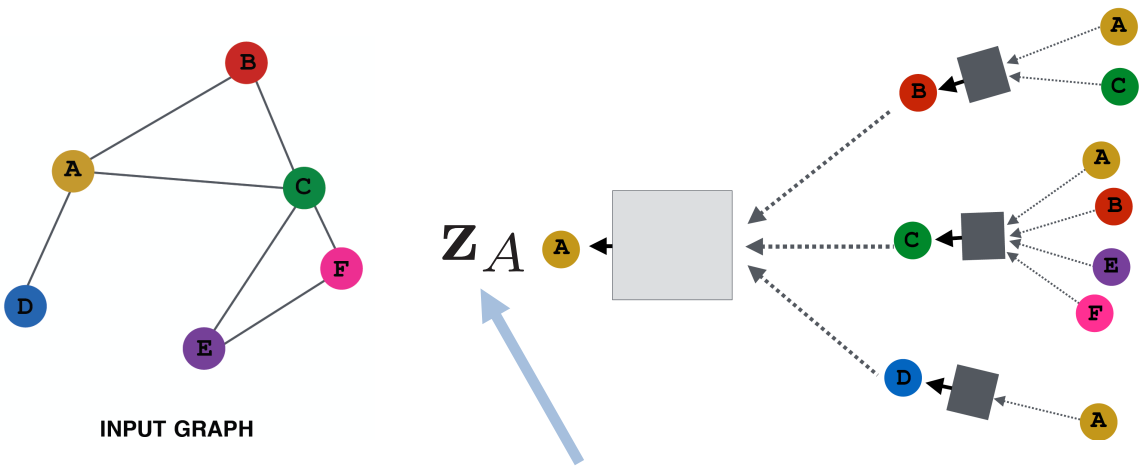
The Math

- **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)

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

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

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

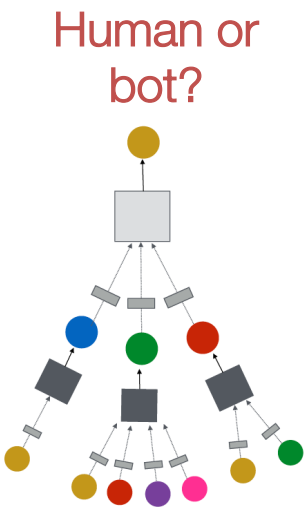
- After K-layers of neighborhood aggregation, we get output embeddings for each node.
- We can feed these embeddings into any loss function and run stochastic gradient descent to train the aggregation parameters.

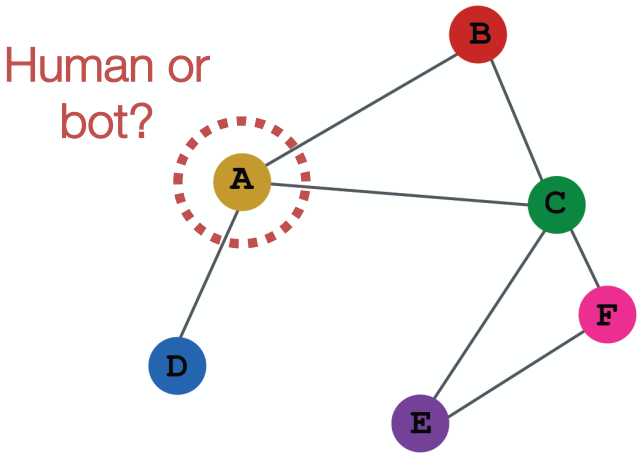
Training the Model

- 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):

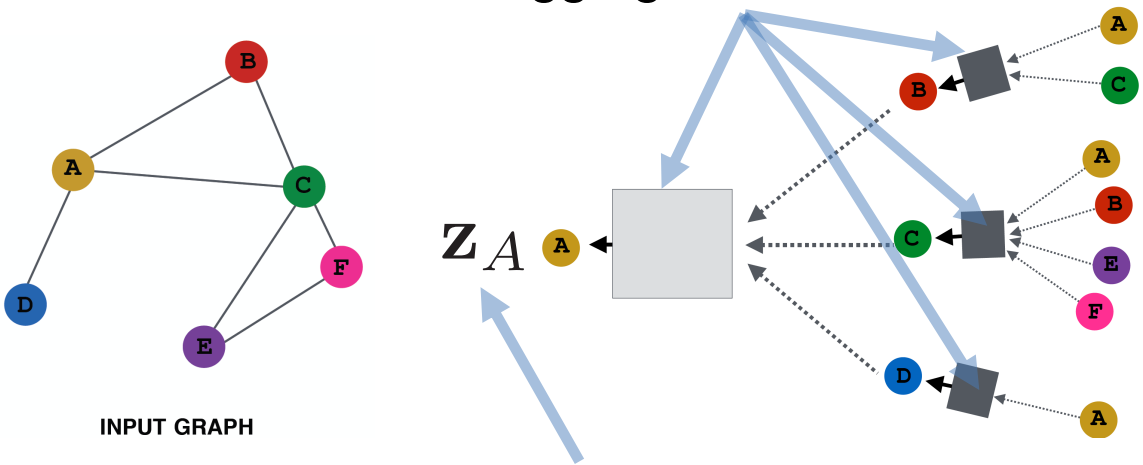




e.g., an online social network

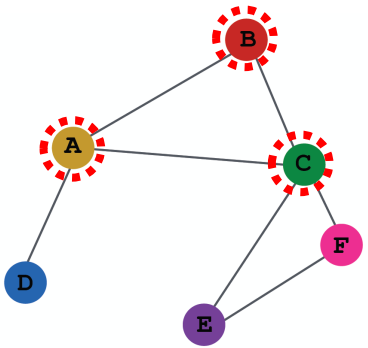
Overview of Model Design

1) Define a neighborhood aggregation function.



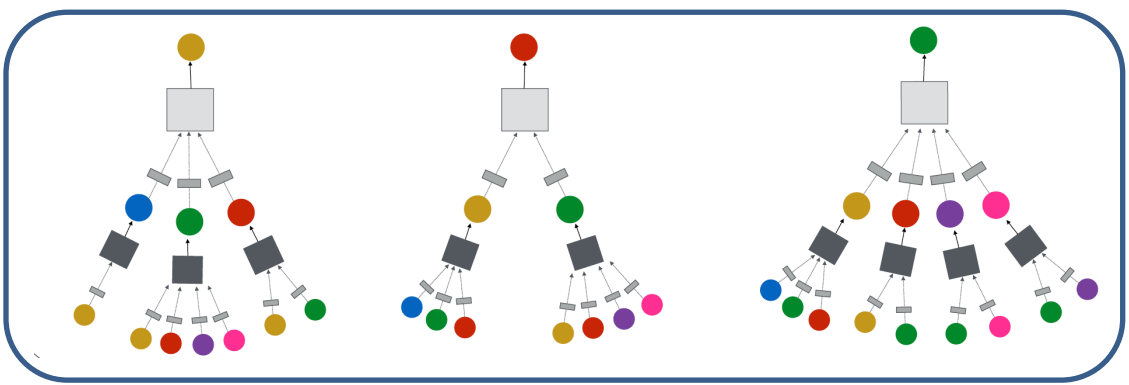
2) Define a loss function on the embeddings, $\mathcal{L}(\mathbf{z}_u)$

Overview of Model Design

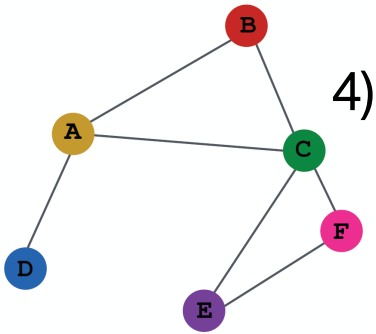


INPUT GRAPH

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



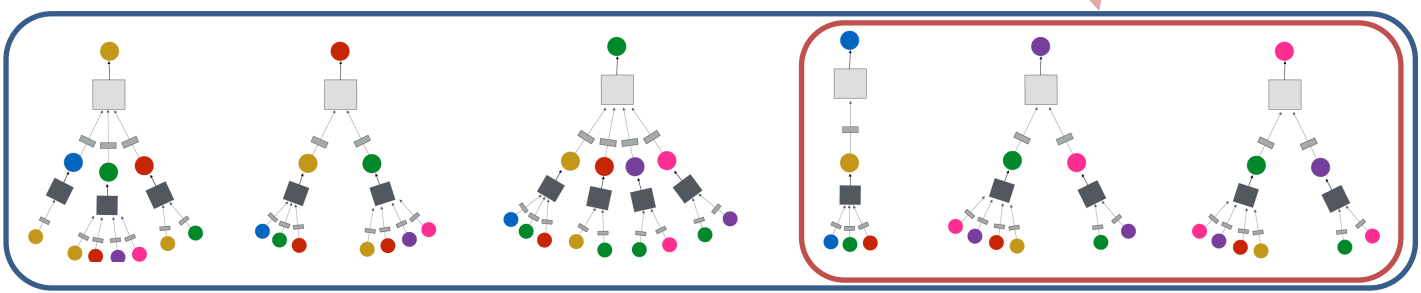
Overview of Model



INPUT GRAPH

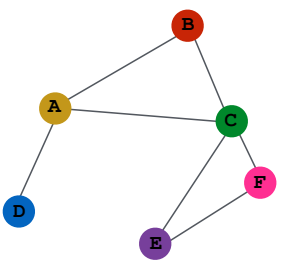
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

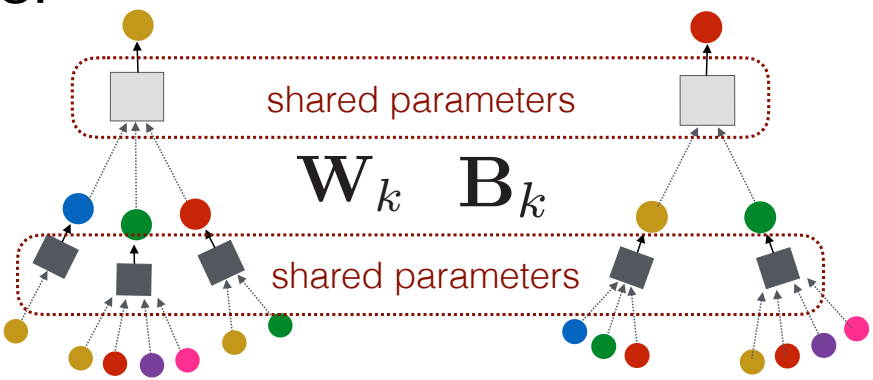


Inductive Capability

- 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!

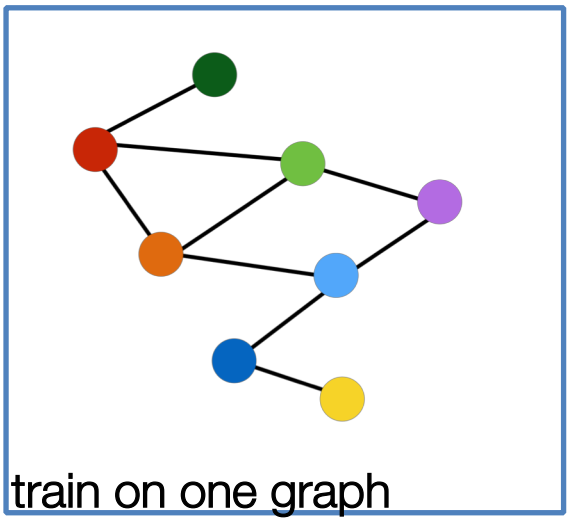


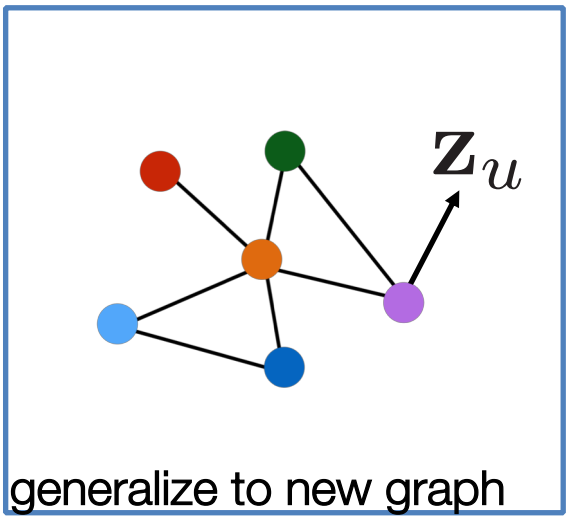
INPUT GRAPH



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

Inductive Capability

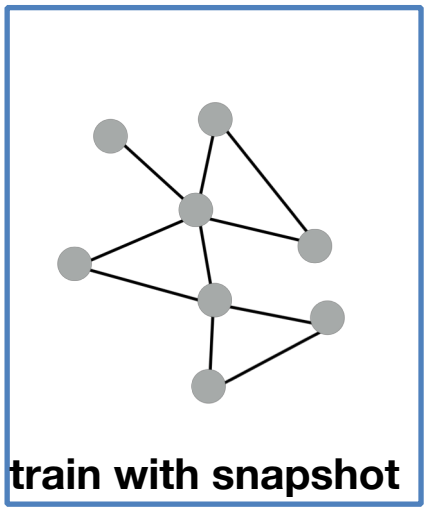


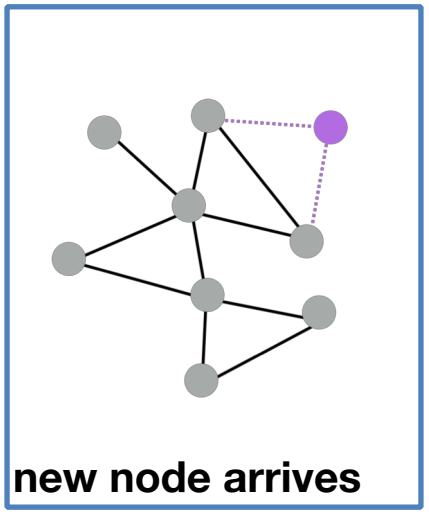


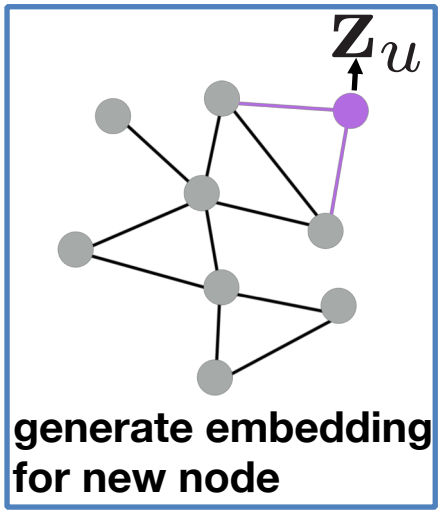
Inductive node embedding → generalize to entirely unseen graphs

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, ….

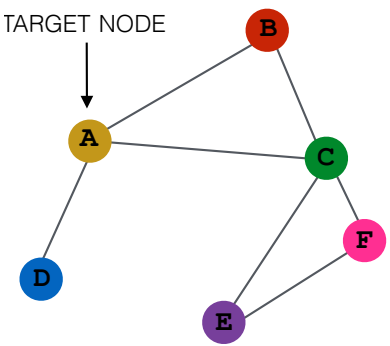
Need to generate new embeddings "on the fly"

Quick Recap

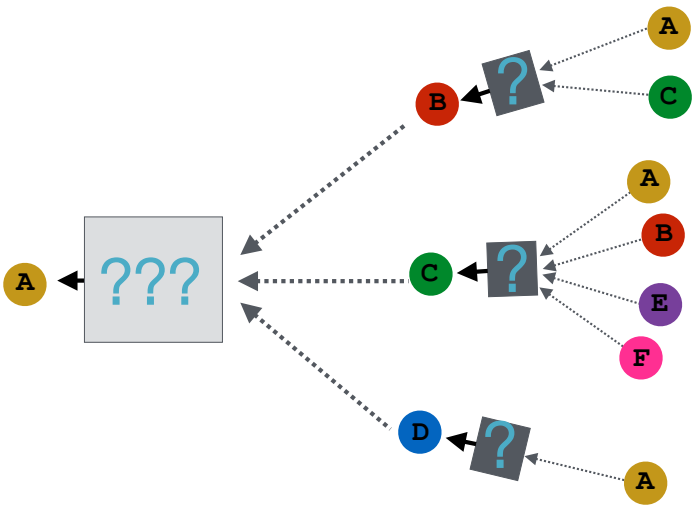
- Recap: Generate node embeddings by aggregating neighborhood information.
  - Allows for parameter sharing in the encoder.
  - Allows for inductive learning.
- We saw a **basic variant of this idea**…

Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages



INPUT GRAPH



Part 3: Applications

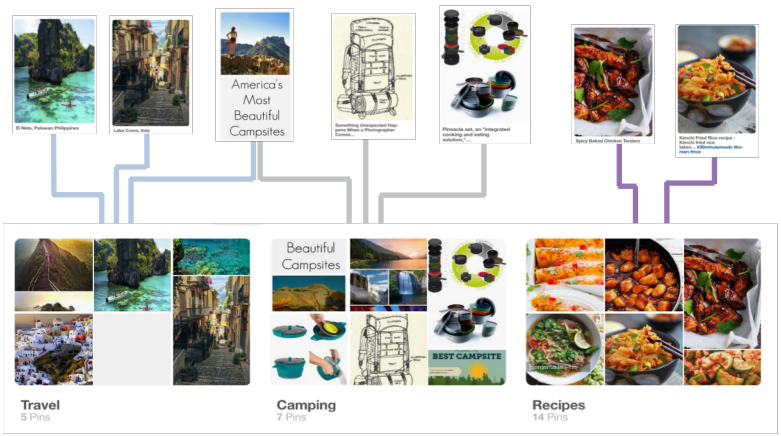
Social recommendations

[KDD 2018]

Collaboration with Pinterest

Pins: Visual bookmarks
text, images, links

Boards
collection of pins



~200 million monthly active users
~2 billion pins, ~1 billion boards, ~17 billion edges

Social recommendations

[KDD 2018]

Task: Given a query pin, recommend related pins.

Social recommendations

[KDD 2018]

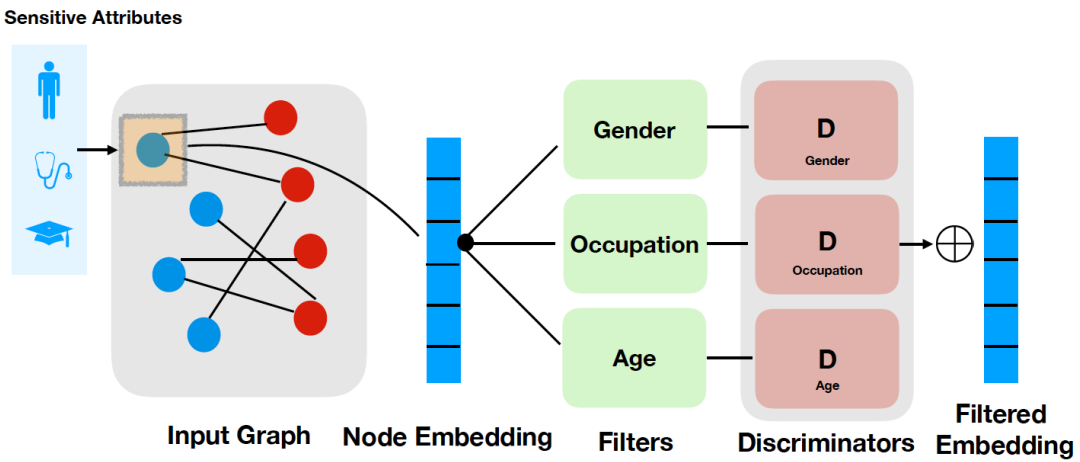
Compared with current production system (highly optimized random-walk based recommendations).

5000 query images, 20000 head-to-head comparisons

Users preferred GNN recommendations 60% of the time.

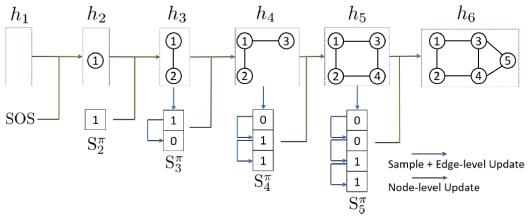
“Fair” recommendations

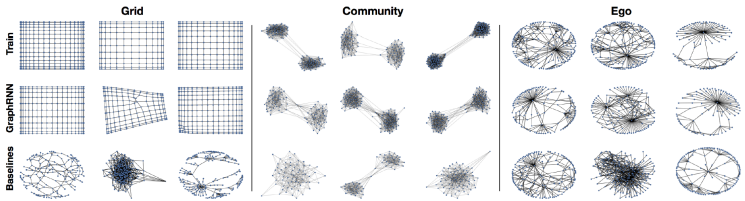
[Under Review]

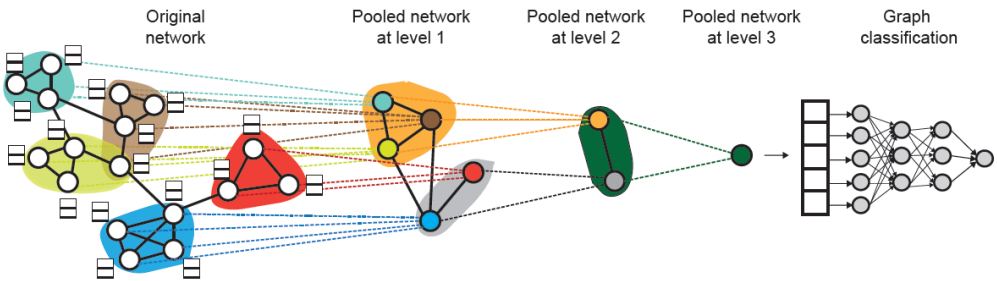


Graph generation and drug design

[ICML 2018; NeurIPS 2018; AAAI 2019]







From learning to reasoning

- Growing interest in models that are capable of logical induction and combinatorial generalization.

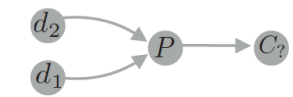
- Learn "rules" from training data that can generalize to unseen types of data instances (e.g., larger, different structures, …).

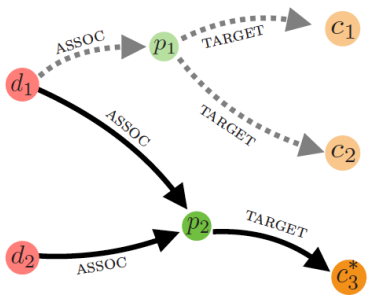
Reasoning with graphs

[NeurIPS 2018]

$C_?.\exists P : \text{ASSOC}(d_1, P) \wedge \text{ASSOC}(d_2, P) \wedge \text{TARGET}(P, C_?)$

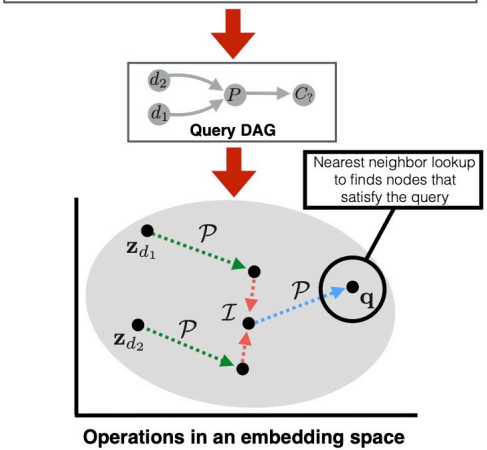
"Predict drugs $C_?$ that might target proteins that are associated with the given disease nodes d_1 and d_2 "





$C_?.\exists P : \text{TARGETS}(C_?, P) \wedge \text{ASSOC}(P, d_2) \wedge \text{ASSOC}(\tilde{p}_i, d_2)$

Input query

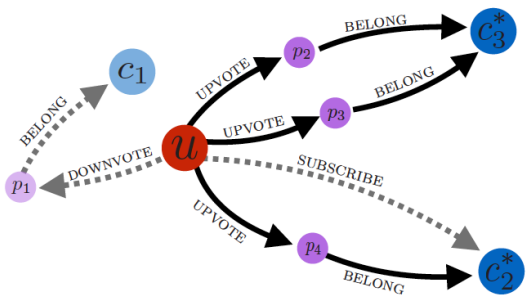


Operations in an embedding space

Reasoning with graphs

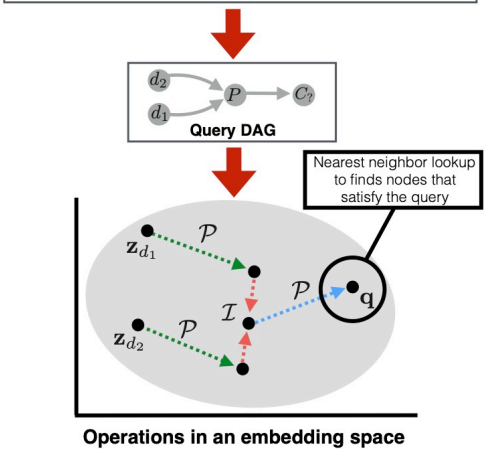
$C_?.\exists P : \text{UPVOTE}(u, P) \wedge \text{BELONG}(P, C_?)$

"Predict communities $C_?$ in which user u is likely to upvote a post"



$C_?.\exists P : \text{TARGETS}(C_?, P) \wedge \text{ASSOC}(P, d_2) \wedge \text{ASSOC}(\tilde{p}_i, d_2)$

Input query



Questions?