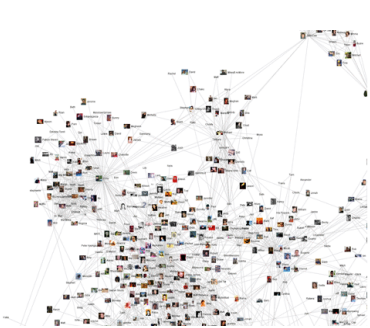
Deep Learning - 2019

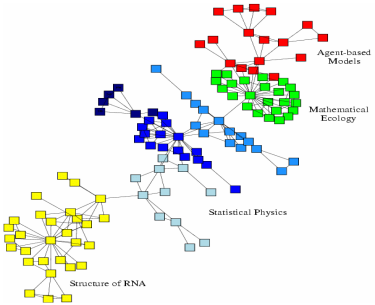
Deep Learning in Graphs

Dr. Megha Khosla

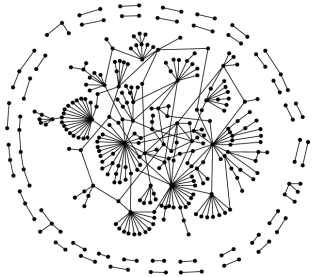
Many Data are Networks



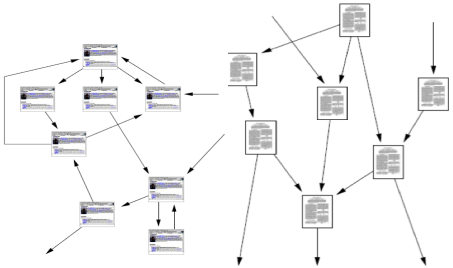
Social networks



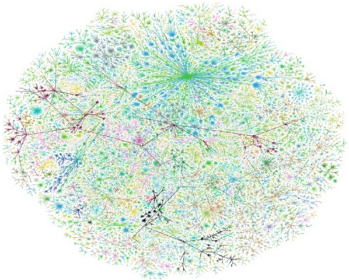
Economic networks



Biomedical networks



Information networks:
Web & citations

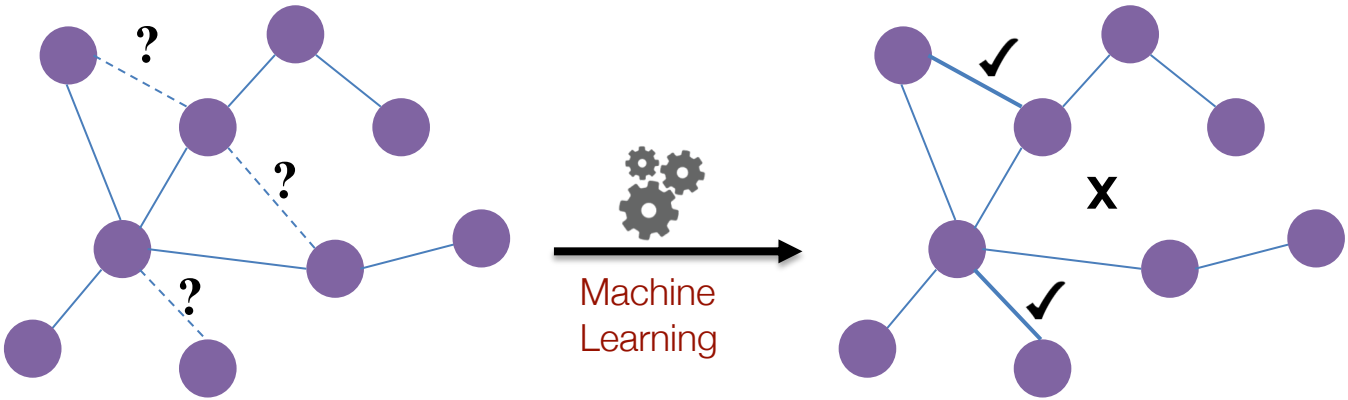


Internet

Networks of neurons

Tasks on Networks

- Predict New Friendships in Social Networks (Link Prediction)



Tasks on Networks

- Predict protein functional labels in biological networks (Node Classification)

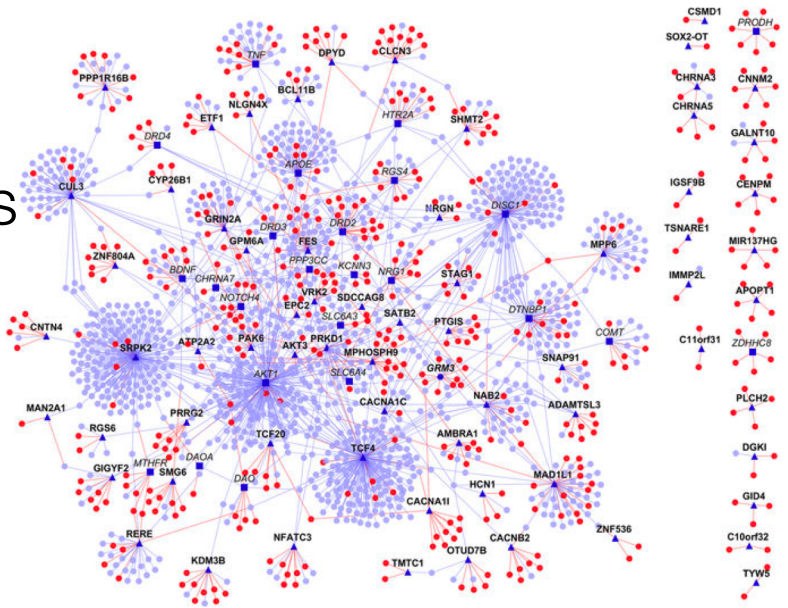
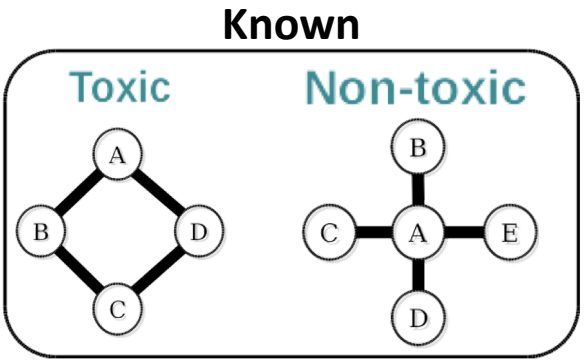


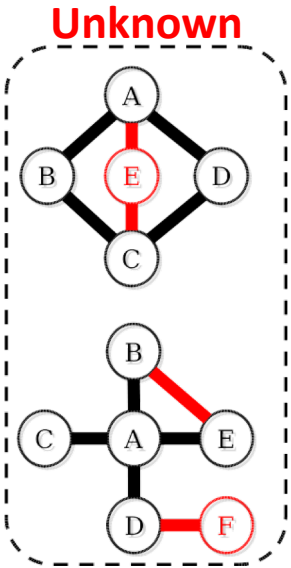
Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

Tasks on Networks

- Predict the functionality of an unseen compound structure in chemical interaction graphs (Graph classification)

Task: predict whether molecules are toxic, given set of known examples





Representation learning on Graphs

- Node Embeddings : Represent each node with a single vector. We would use this embedding to find missing node labels or predict new connections based on node similarities.

- Graph embeddings: Represent the whole graph with a single vector. Used to make predictions on the graph level , e.g. labelling of chemical structures.

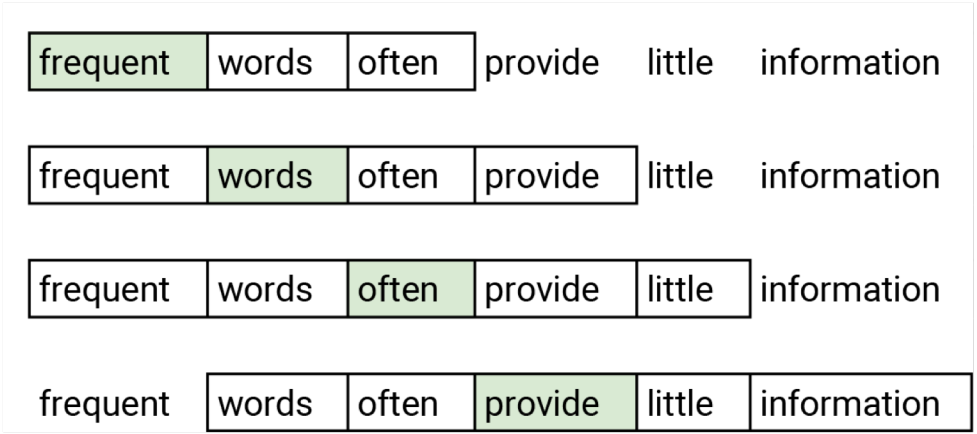
Why do we need vector representations ?

- Limited approaches because of complex graph structure
- Embeddings are compressed representations
- Vector operations are simpler and faster

Desirable Properties

- Mostly task dependent but must usually encode
  - Graph topology
  - Node neighborhood information
  - …
- Scalability to large graphs
- Generalizable to unseen but similar structures

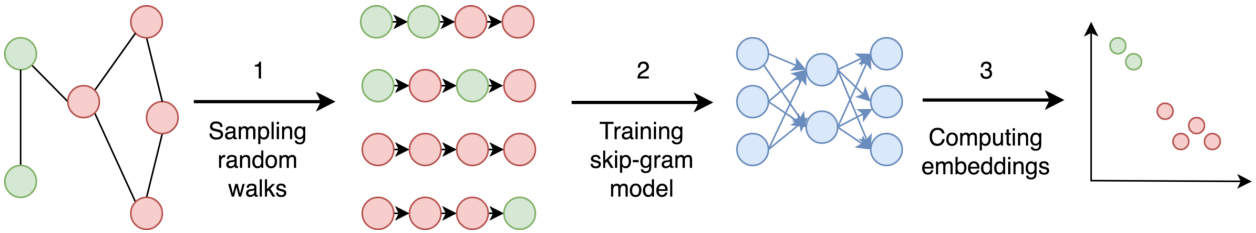
Node Embeddings: Motivation from Word2vec



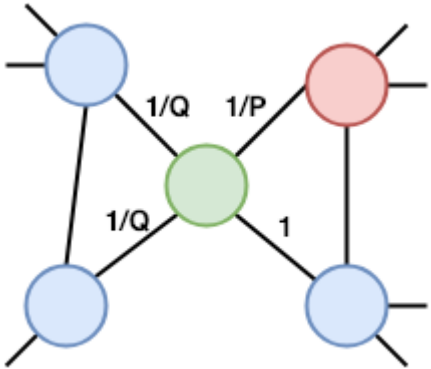
- Input : green colored word
- Predict neighboring words in the window size 2

DeepWalk : A graph Word2Vec

- Generate Sentences from Graphs
  - Via Fixed size random walks
- Given an input node train to predict neighboring nodes within a specified window



Biased Walks in Node2Vec



Let 'red node' be the starting node and the walker is currently at 'green node'

- Go back to the red node with transition probability(t.p.) weighted by 1/P
- Go to the blue node nearer to the red node with transition probability(t.p.) weighted by 1
- Go to any other node with t.p. weighted by 1/Q

Quiz Time

- When do biased walks (of Node2Vec) reduce to uniform random walks as in DeepWalk ?

(Hint: think of directed graphs and graphs with low clustering coefficient)

Random Walk Optimization

1. Run random walks starting from each node on the graph using some strategy $\mathbf{R}$.
2. For each node $\mathbf{u}$ collect $\mathbf{N_R(u)}$, the multiset* of nodes visited on random walks starting from $\mathbf{u}$.
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

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

- Intuition: Optimize embeddings to maximize likelihood of random walk co-occurrences.
- Parameterize $\mathrm{P}(\mathrm{v} \mid \mathrm{z_u})$ 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:

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

sum over all nodes $\mathbf{u}$

sum over nodes $\mathbf{v}$ seen on random walks starting from $\mathbf{u}$

predicted probability of $\mathbf{u}$ and $\mathbf{v}$ co-occuring on random walk

Optimizing random walk embeddings =

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

Random Walk Optimization

But doing this naively is too expensive!!

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

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

Negative Sampling

Solution: Negative sampling

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k}\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

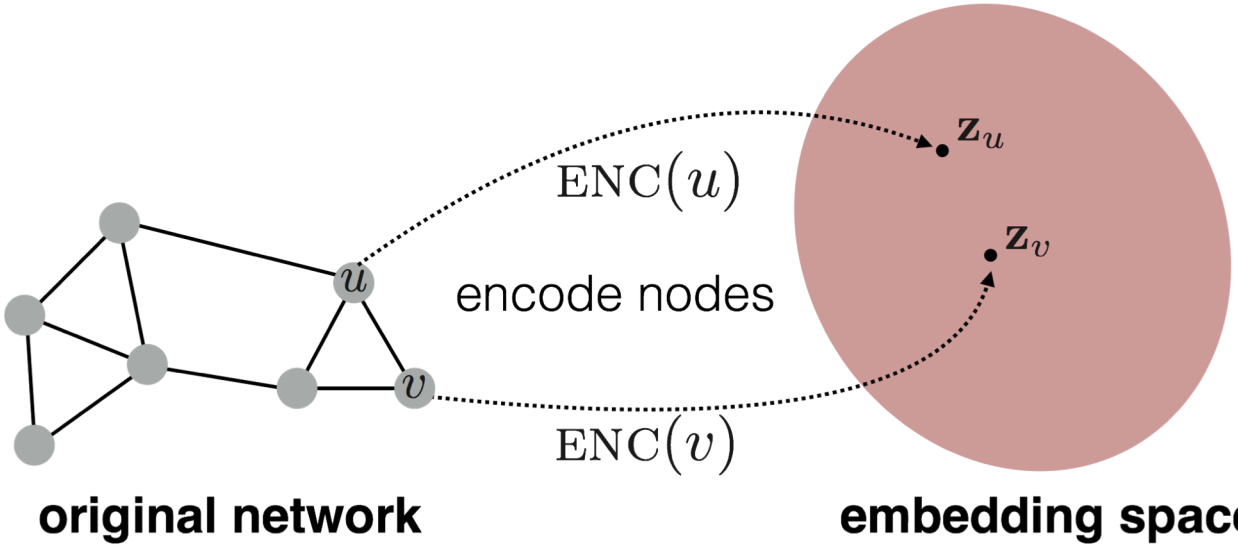
sigmoid function

random distribution over all nodes

i.e., instead of normalizing w.r.t. all nodes, just normalize against $\mathbf{k}$ random "negative samples"

In a nutshell...

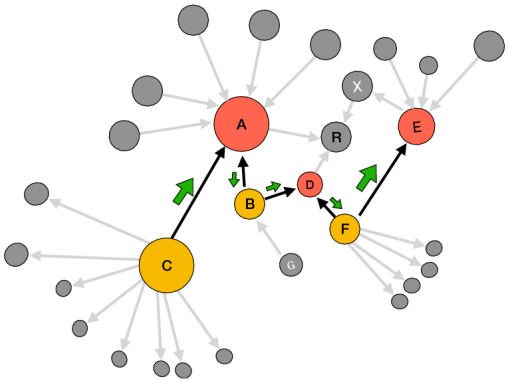
- Learning representations for nodes is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.

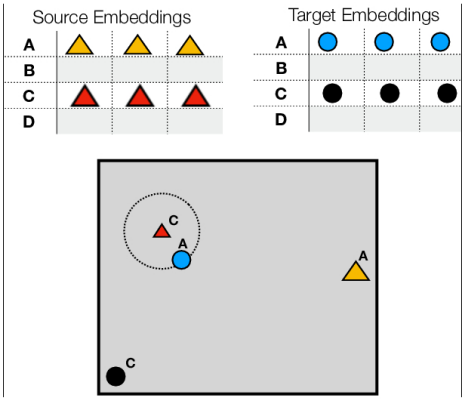


Directed Graphs

- What happens when we use only one embedding per node?
- Can it encode directed relations (consider follower networks) ?

Alternating Random Walks



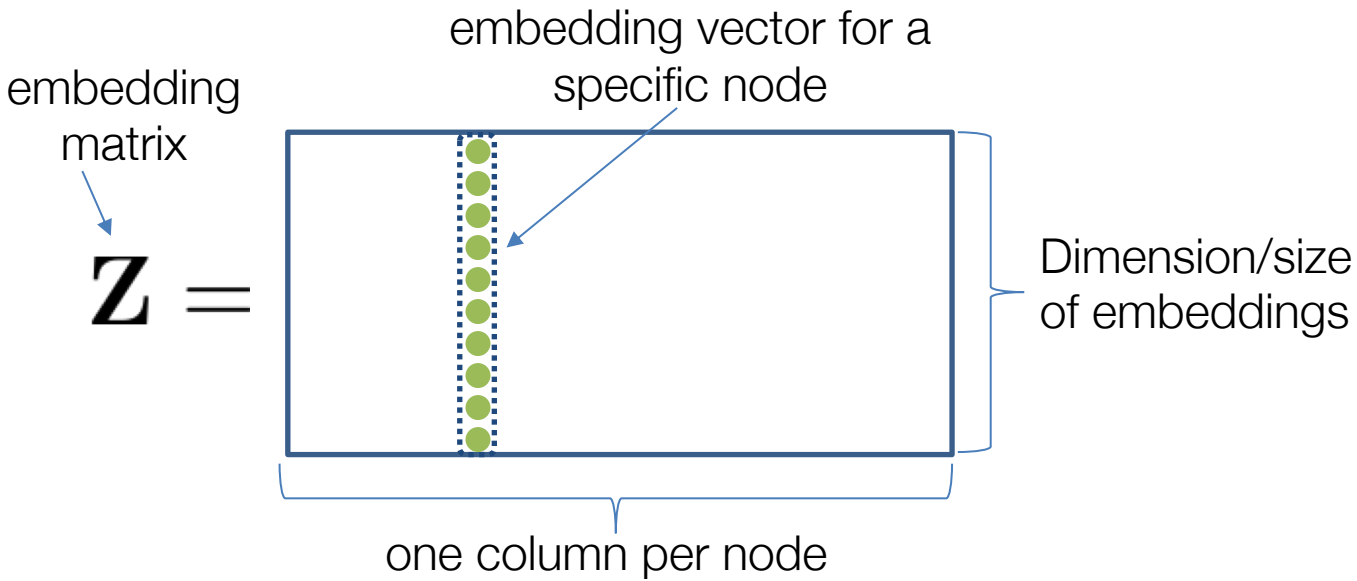


- NERD : Use alternating walks to sample neighborhood and learn two embeddings per node

Graph Neural Networks

From “Shallow” to “Deep”

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



From “Shallow” to “Deep”

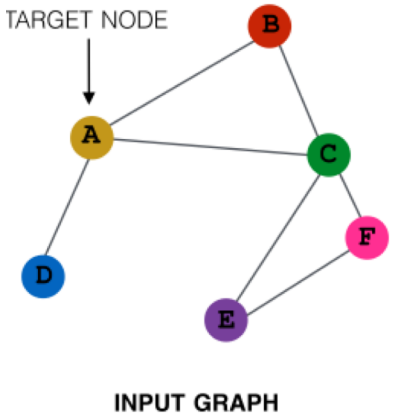
- Limitations of shallow encoding:
  - O(|V|) parameters are needed: there no parameter sharing and every node has its own unique embedding vector.
  - Inherently “transductive”: It is impossible to generate embeddings for nodes that were not seen during training.
  - Do not incorporate node attributes: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

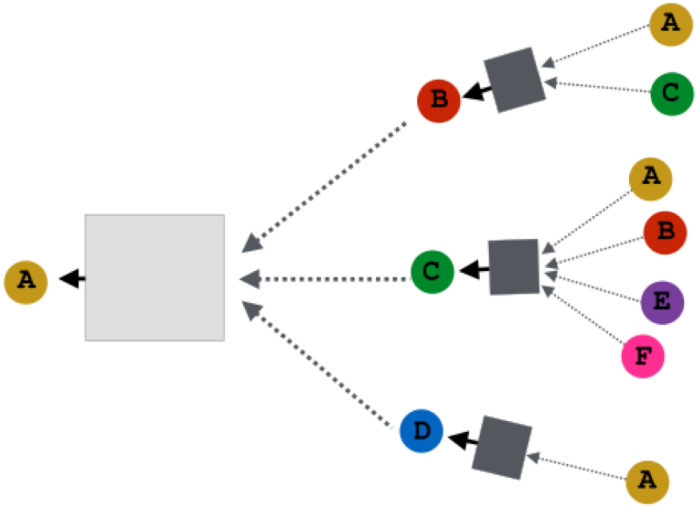
Setup

- Assume we have a graph $\mathbf{G}$:
  - $\mathbf{V}$ is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - $\mathbf{X} \in \mathbb{R}^{m \times |V|}$ **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

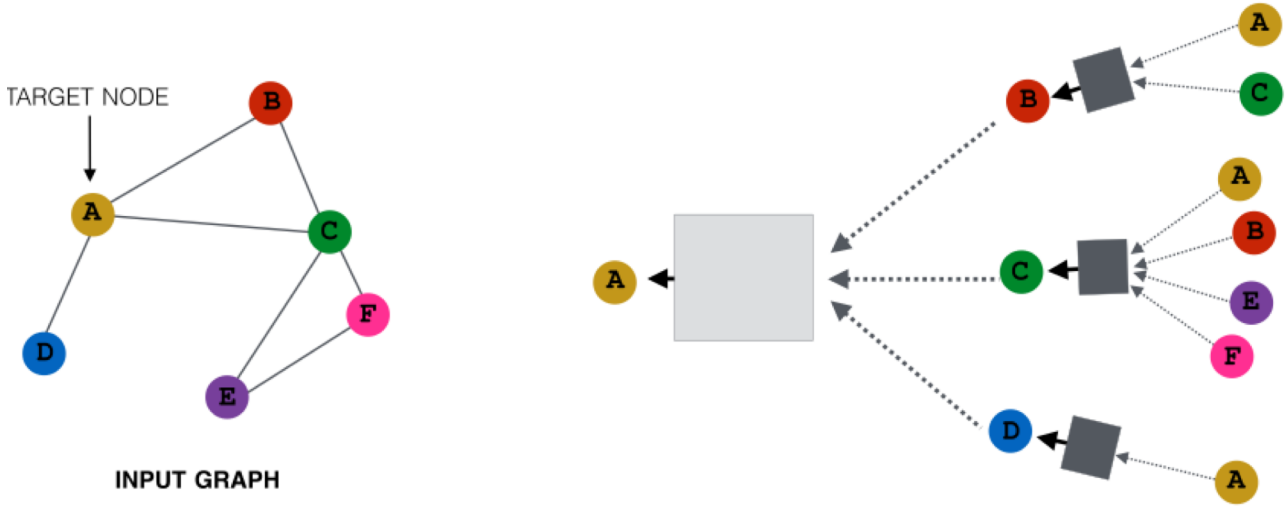
- Key idea: Generate node embeddings based on local neighborhoods.





Neighborhood Aggregation

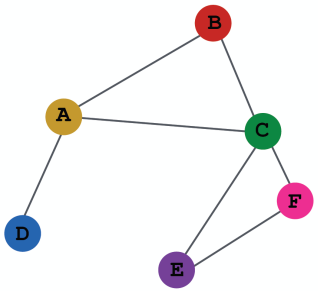
- Intuition: Nodes aggregate information from their neighbors using neural networks



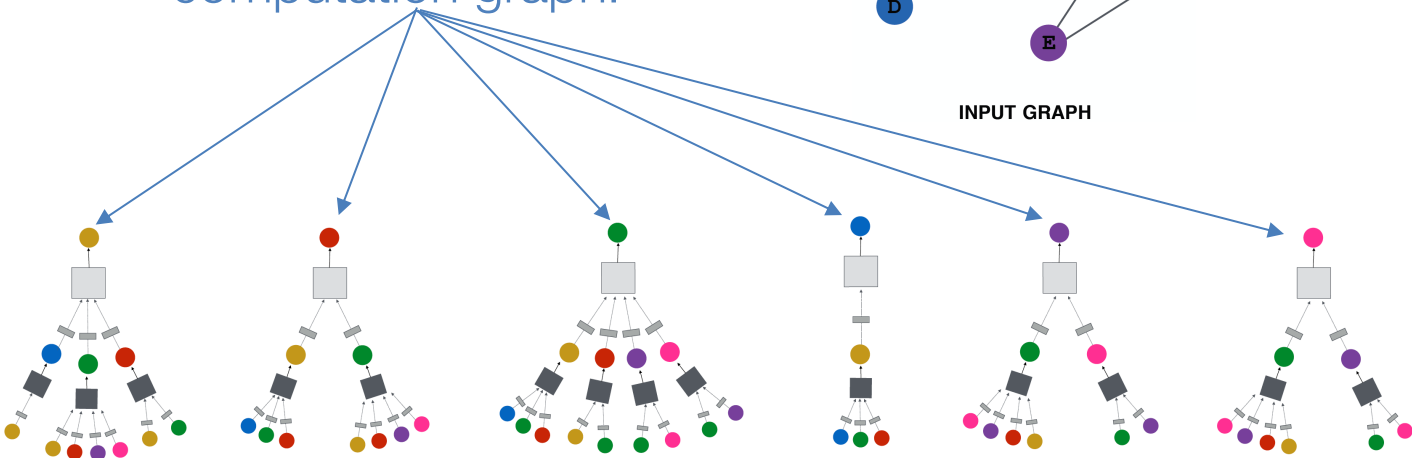
Neighborhood Aggregation

- Intuition: Network neighborhood defines a computation graph

Every node defines a unique computation graph!

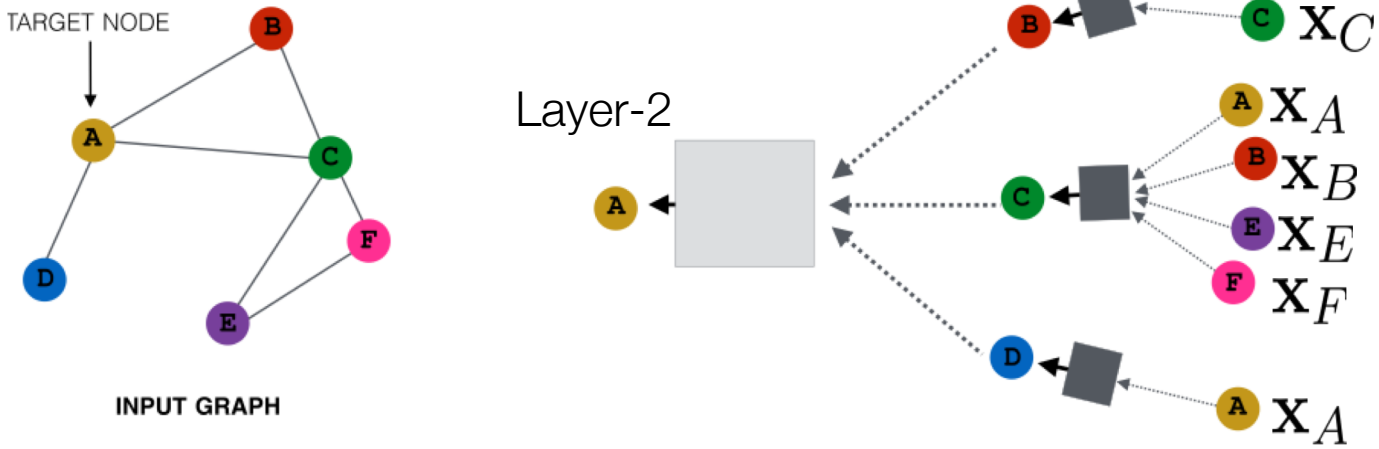


INPUT GRAPH



Neighborhood Aggregation

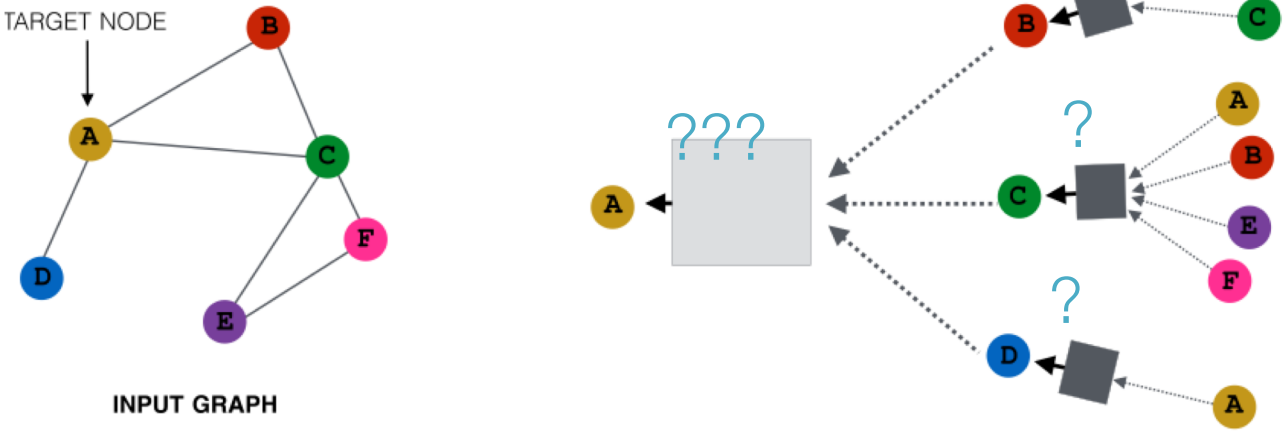
- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- "layer-0" embedding of node $\mathbf{u}$ is its input feature, i.e. $\mathbf{x}_u$.



Neighborhood Aggregation

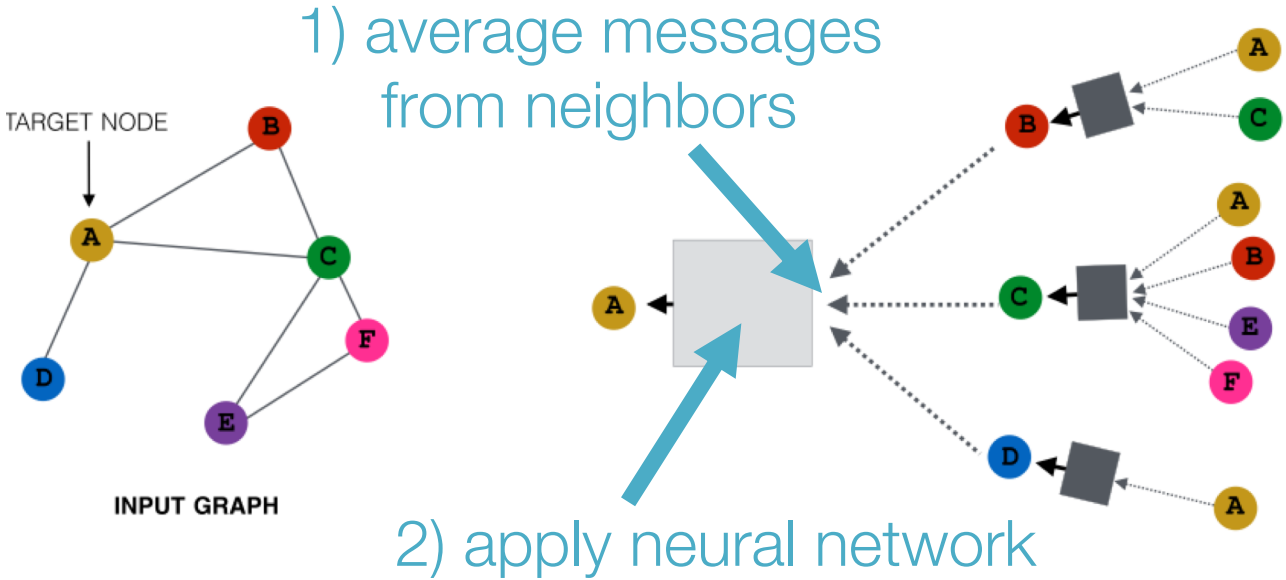
- Key distinctions are in how different approaches aggregate information across the layers.

what's in the box!?



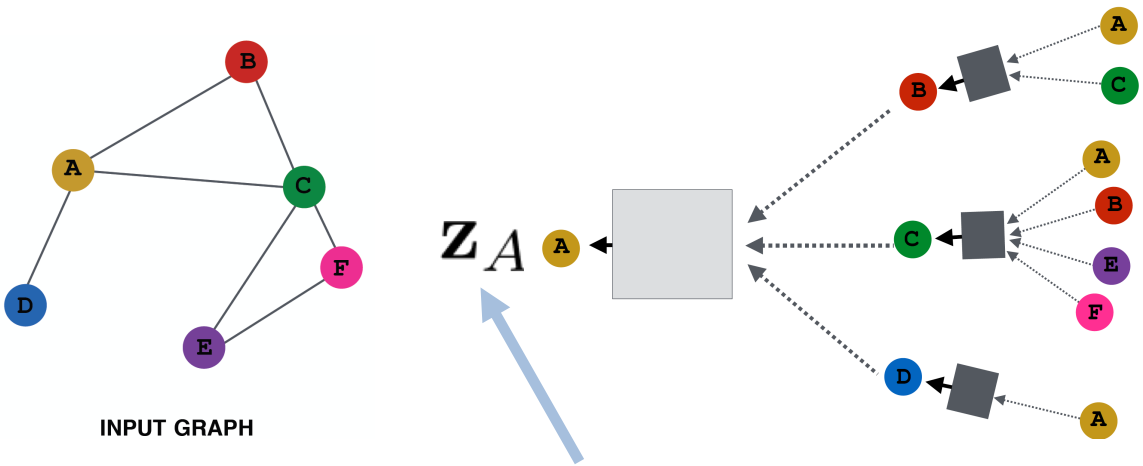
Neighborhood Aggregation

- Basic approach: Average neighbor information 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

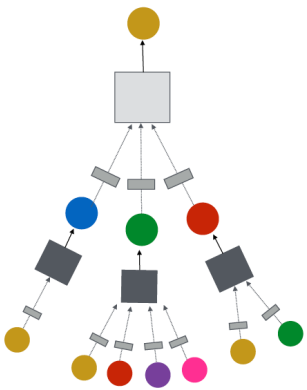
- Train in an unsupervised manner using only the graph structure.

- Unsupervised loss function can be anything, e.g., based on random walk methods

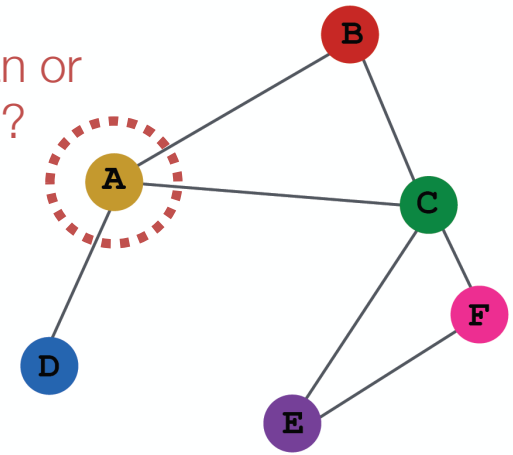
Training the Model

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

Human or bot?



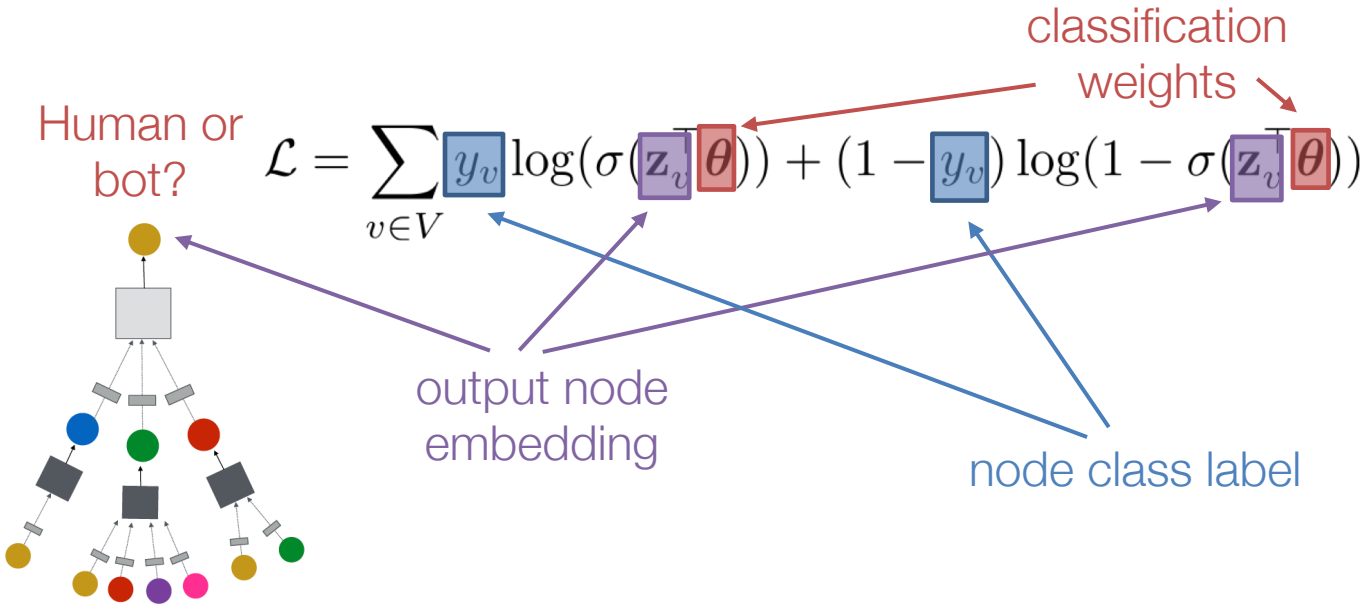
Human or bot?



e.g., an online social network

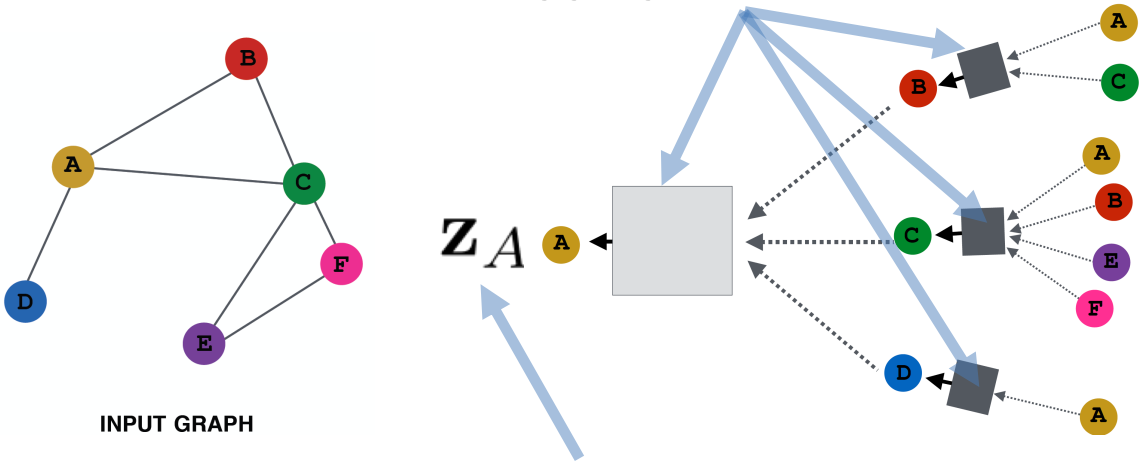
Training the Model

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



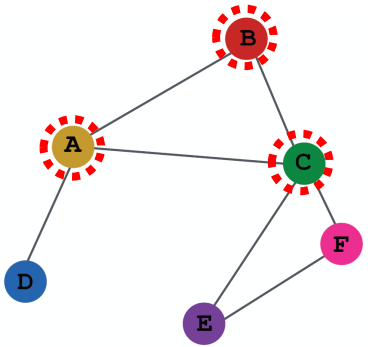
Overview of Model Design

1) Define a neighborhood aggregation function.



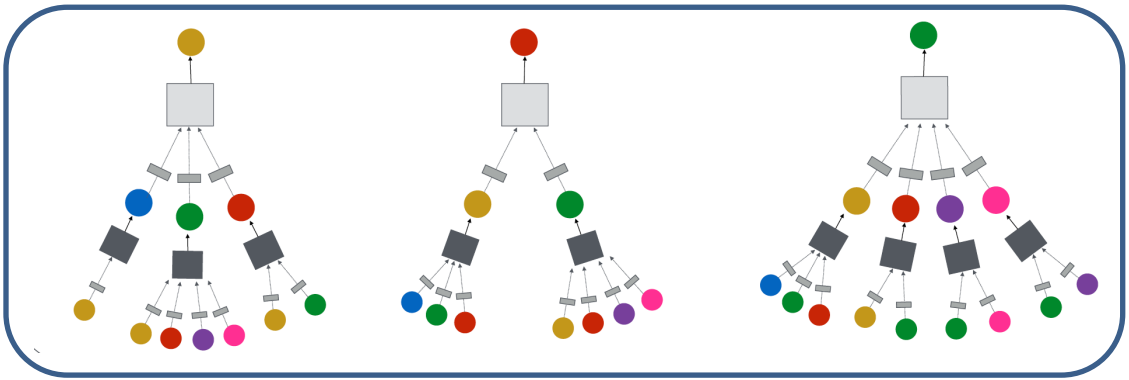
2) Define a loss function on the embeddings, $\mathcal{L}(\mathbf{z}_u)$

Overview of Model Design

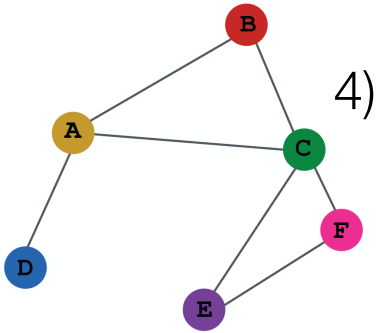


INPUT GRAPH

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



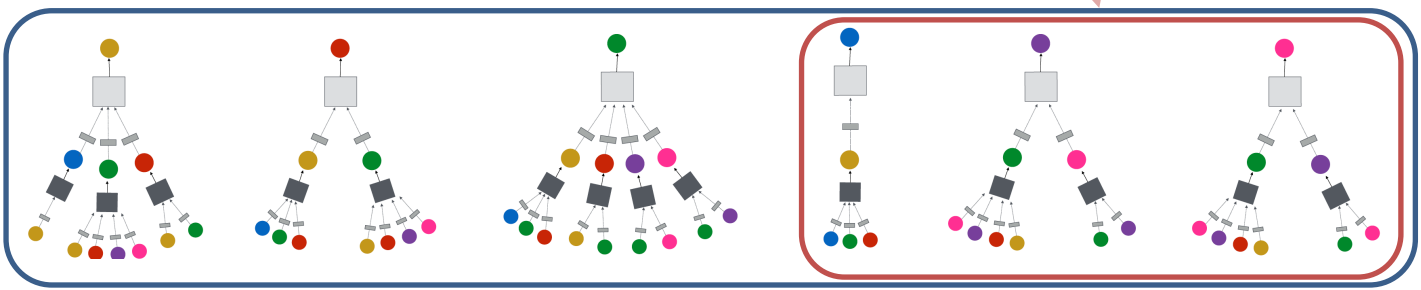
Overview of Model



INPUT GRAPH

4) Generate embeddings for nodes as needed

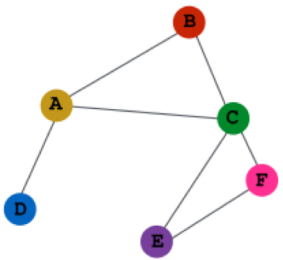
Even for nodes we never trained on!!!!



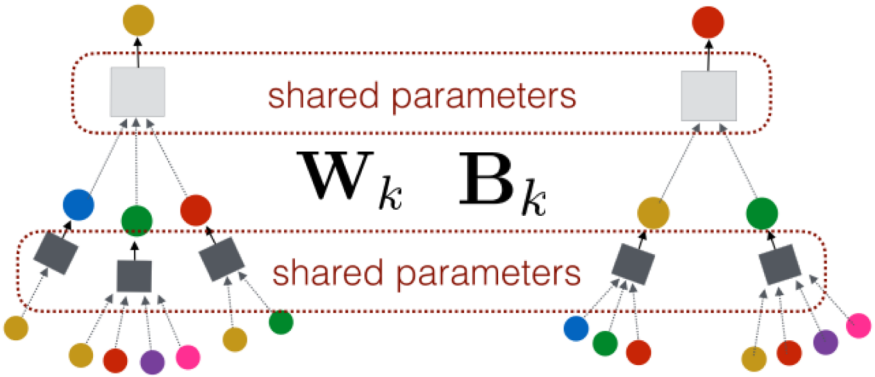
Inductive Capability

- 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!



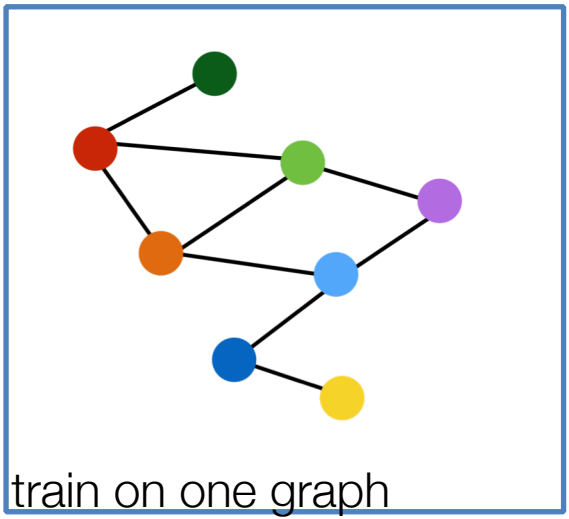
INPUT GRAPH

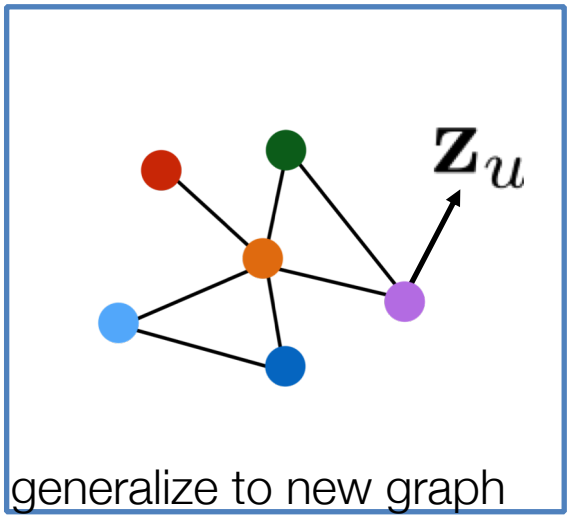


Compute graph for node A

Compute graph for node B

Inductive Capability

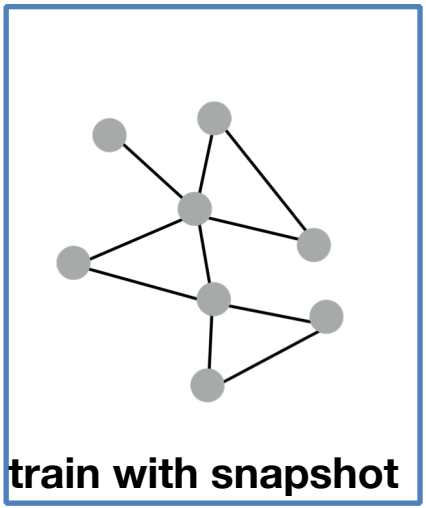


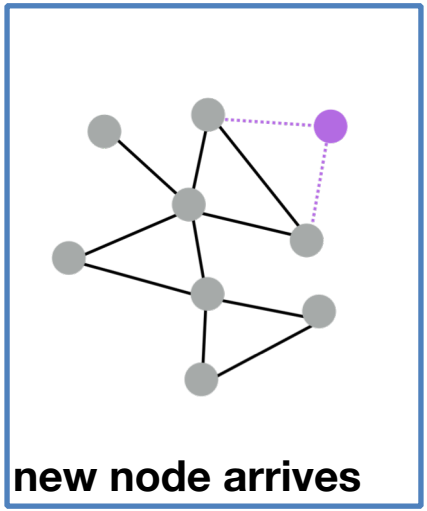


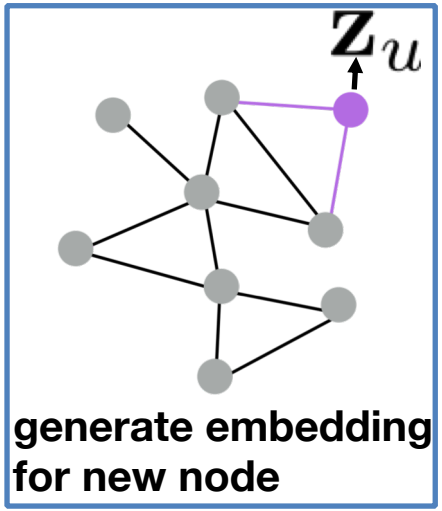
Inductive node embedding → generalize to entirely unseen graphs

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

Inductive Capability







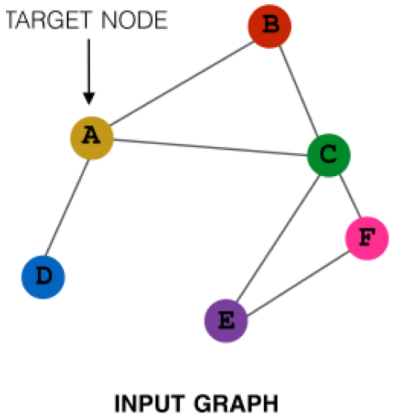
Many application settings constantly encounter previously unseen nodes.

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

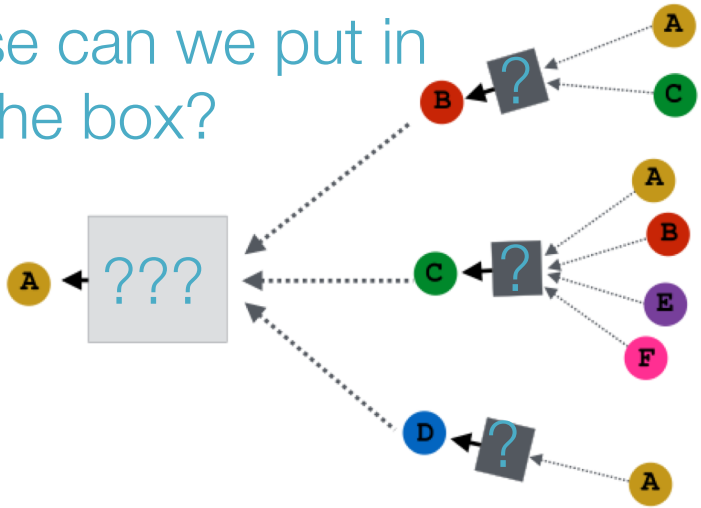
Need to generate new embeddings "on the fly"

Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages



What else can we put in the box?



Graph Convolutional Networks

- Kipf et al.'s Graph Convolutional Networks (GCNs) provides a 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

$$\mathbf{H}^{(k+1)} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(k)}\mathbf{W}_k\right)$$

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$$

$$\mathbf{D}_{ii} = \sum_j \mathbf{A}_{i,j}$$

Quiz Time

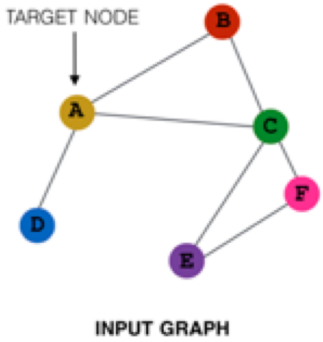
- Can you imagine a drawback of neighborhood aggregation methods??
  - Hint : consider high degree nodes

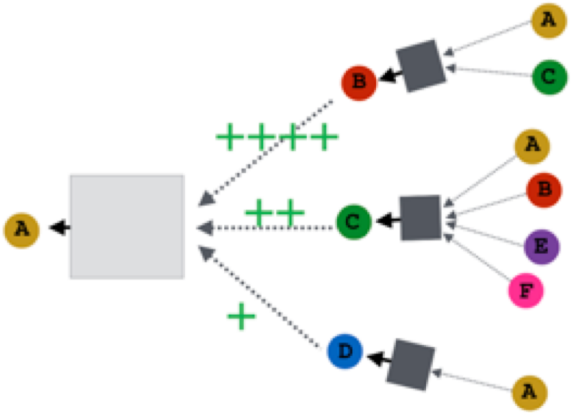
Scaling Variants

- GraphSage
  - Samples neighbors instead of taking the complete neighborhood
  - Provides a number of aggregation operations
- FastGCN
  - Samples nodes in each convolutional layer

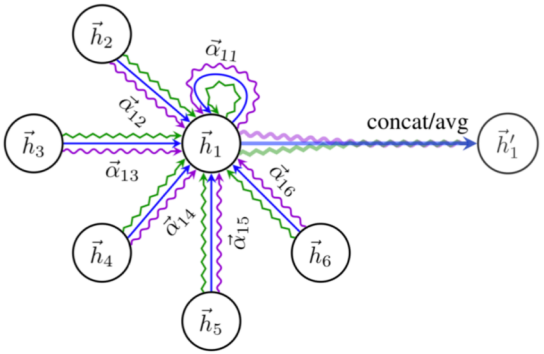
Neighbor Importance

What if some neighbors are more important than others?





Attentions over Neighbors

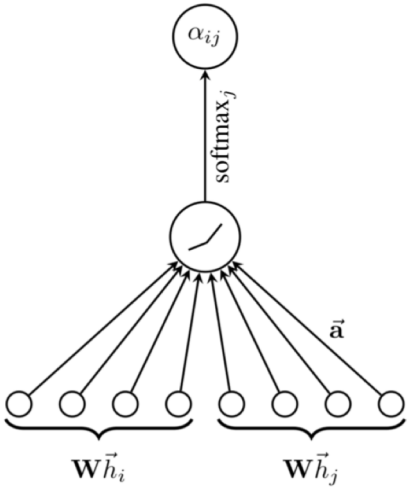


[Figure from Veličković et al. (ICLR 2018)]

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{k}\mathbf{W}^{k}\vec{h}_j\right)$$

- *Every neighbor i of node 1 sends its own vector of attentional coefficients* α_{1i}^{k} *one per each attention head*
- Aggregate K separate linear combinations of neighbors' features to obtain the next level feature of node 1

Attention Coefficients



$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

References

Perozzi et al.. "Deepwalk: Online learning of social representations." *In KDD 2014*

Grover and Leskovec. "node2vec: Scalable feature learning for networks." *In KDD 2016*

Khosla et al. Node Representation Learning for Directed Graphs. In ECML-PKDD 2019 (https://arxiv.org/abs/1810.09176)

Khosla et al. A Comprehensive Comparison of Unsupervised Network Representation Learning Methods. (https://arxiv.org/abs/1903.07902)

Hamilton et al.. "Inductive representation learning on large graphs." *Advances in Neural Information Processing Systems*. 2017.

Veličković, Petar, et al. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

An extensive list of papers including surveys can be found at https://github.com/thunlp/NRLPapers