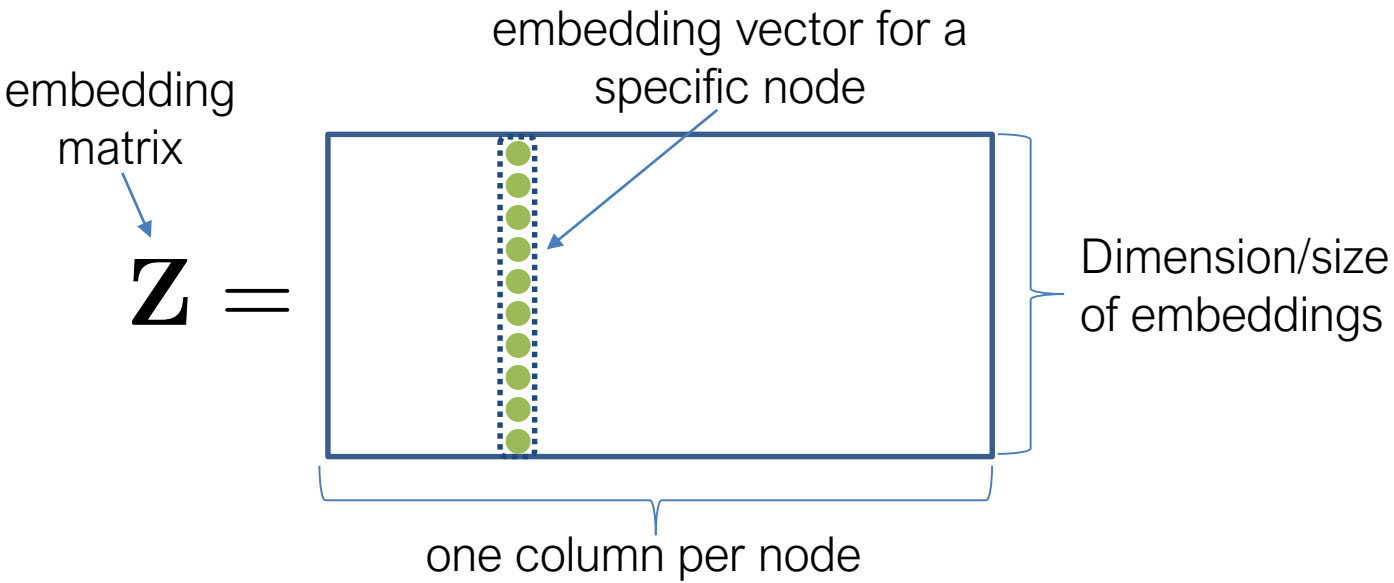
Graph Neural Networks

Presented by: Enas Tarawneh

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 (GCNs)
  3. GraphSAGE

The Basics: Graph Neural Networks

Based on material from:

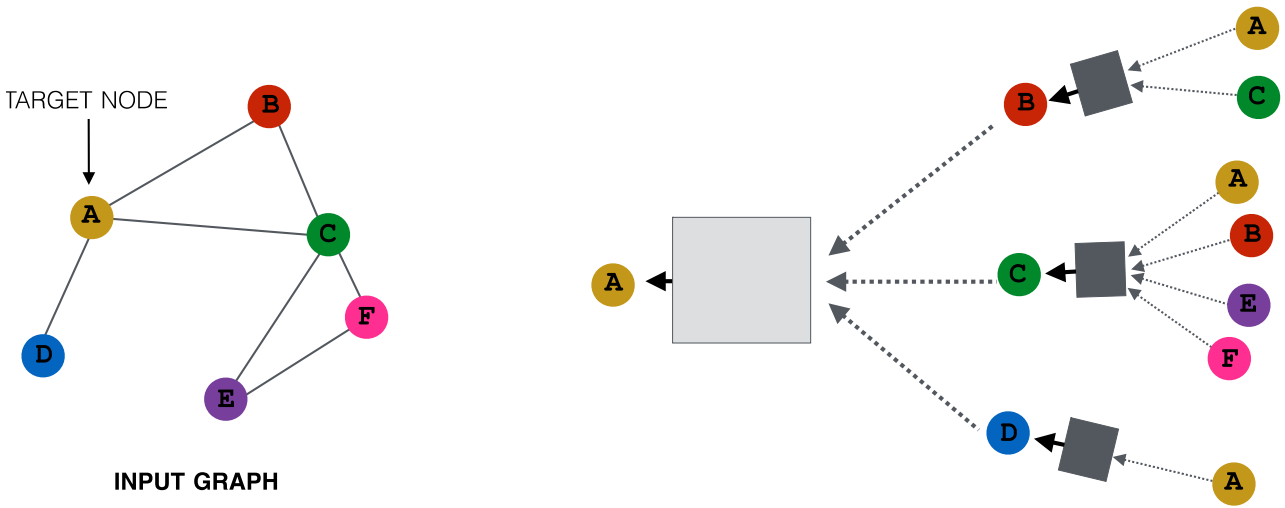
- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
- Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

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)

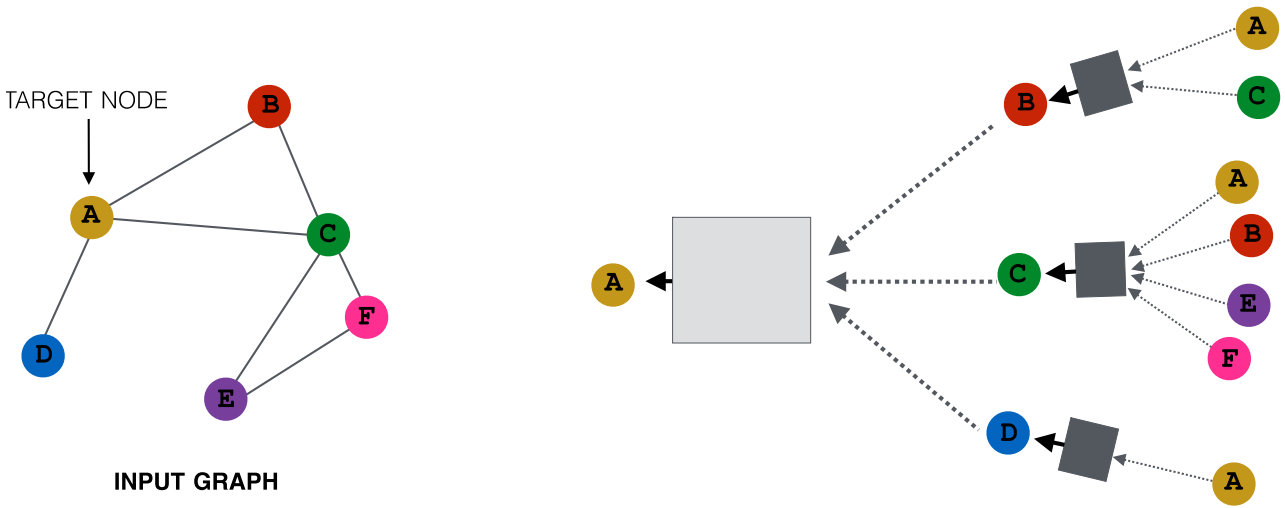
Neighborhood Aggregation

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



Neighborhood Aggregation

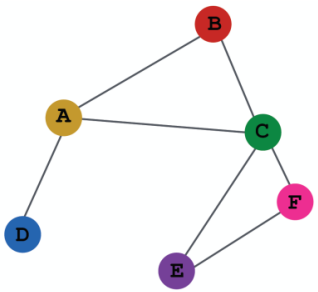
- **Intuition:** Nodes aggregate information from their neighbors using neural networks



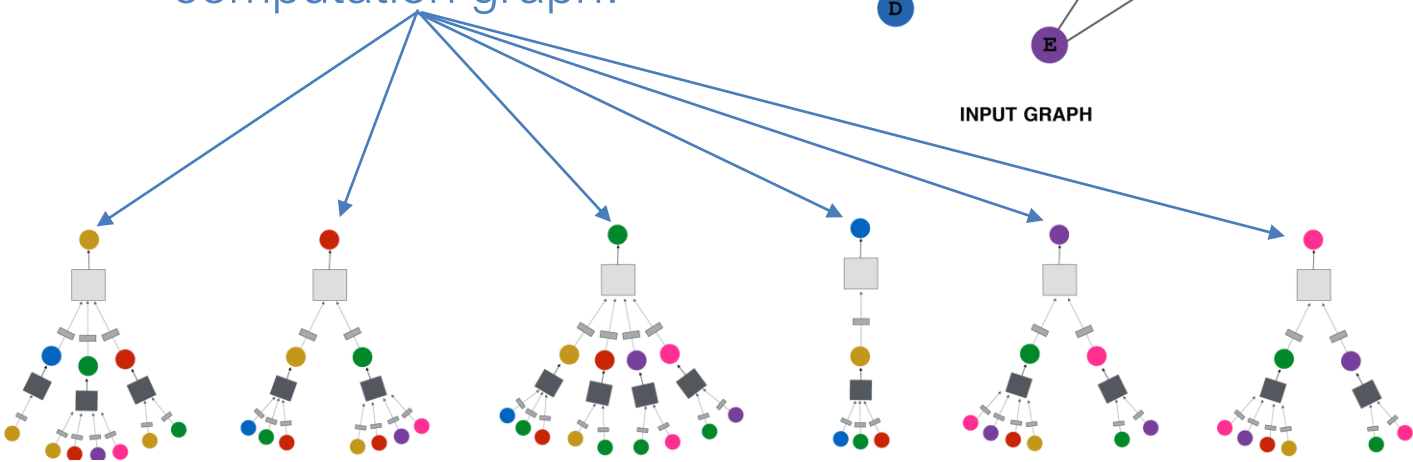
Neighborhood Aggregation

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

Every node defines a unique computation graph!

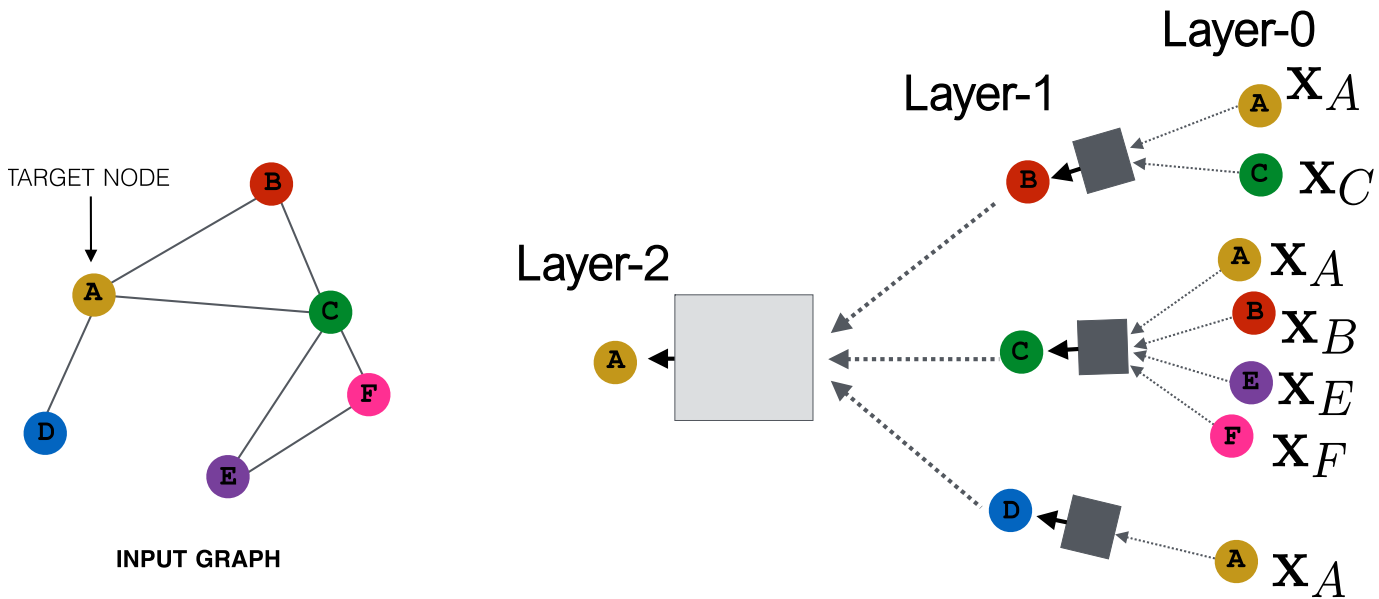


INPUT GRAPH



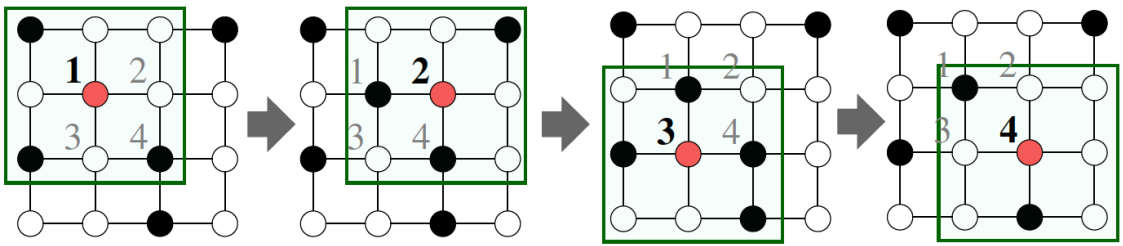
Neighborhood Aggregation

- 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 "Convolutions"

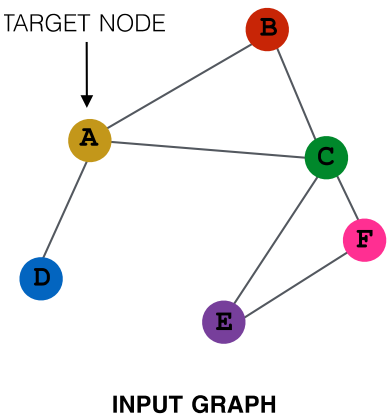
- Neighborhood aggregation can be viewed as a center-surround filter.



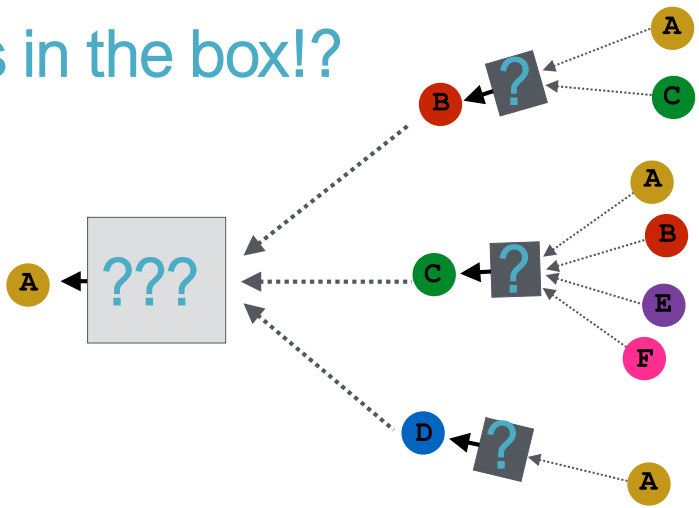
- Mathematically related to spectral graph convolutions (see Bronstein et al., 2017)

Neighborhood Aggregation

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

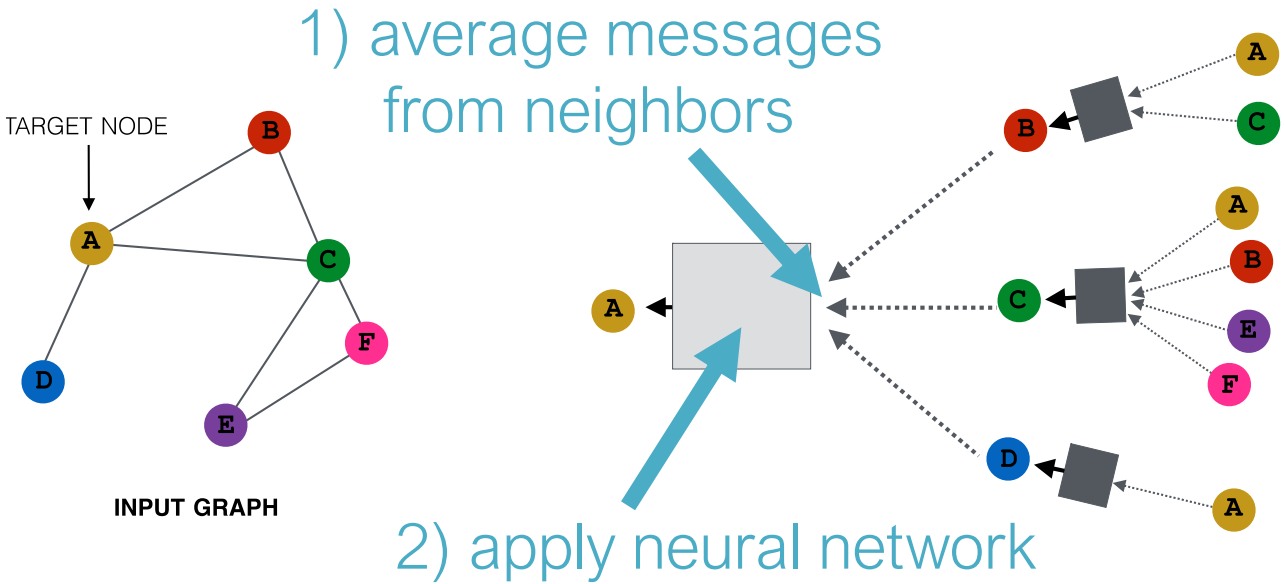


what's in the box!?



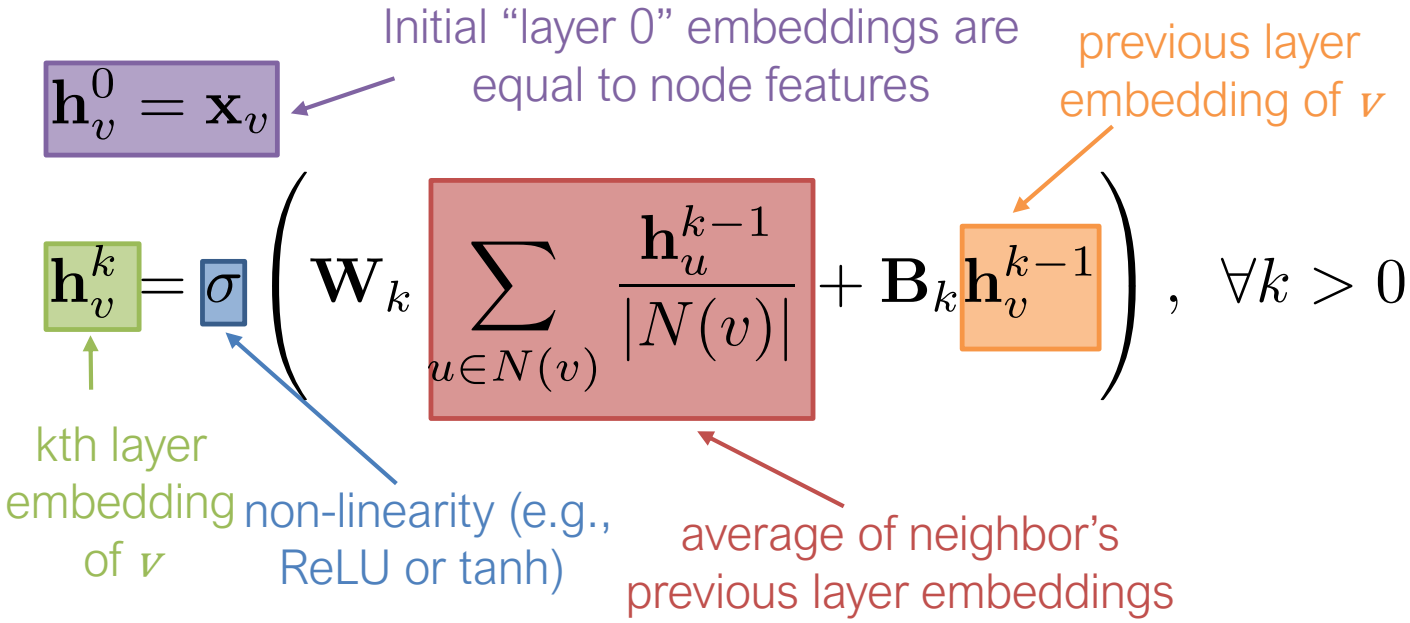
Neighborhood Aggregation

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



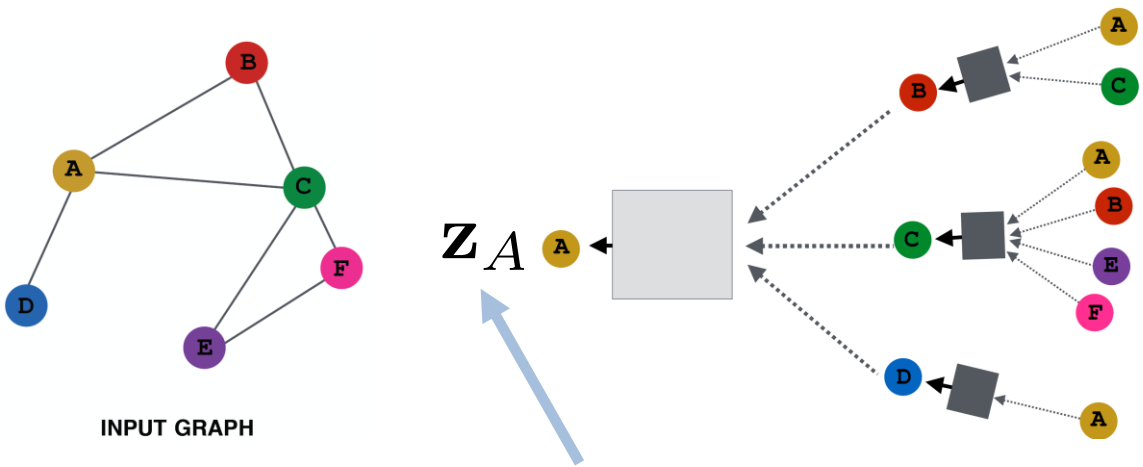
The Math

- **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, $\mathbf{L}(\mathbf{z}_u)$!

Training the Model

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

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

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

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

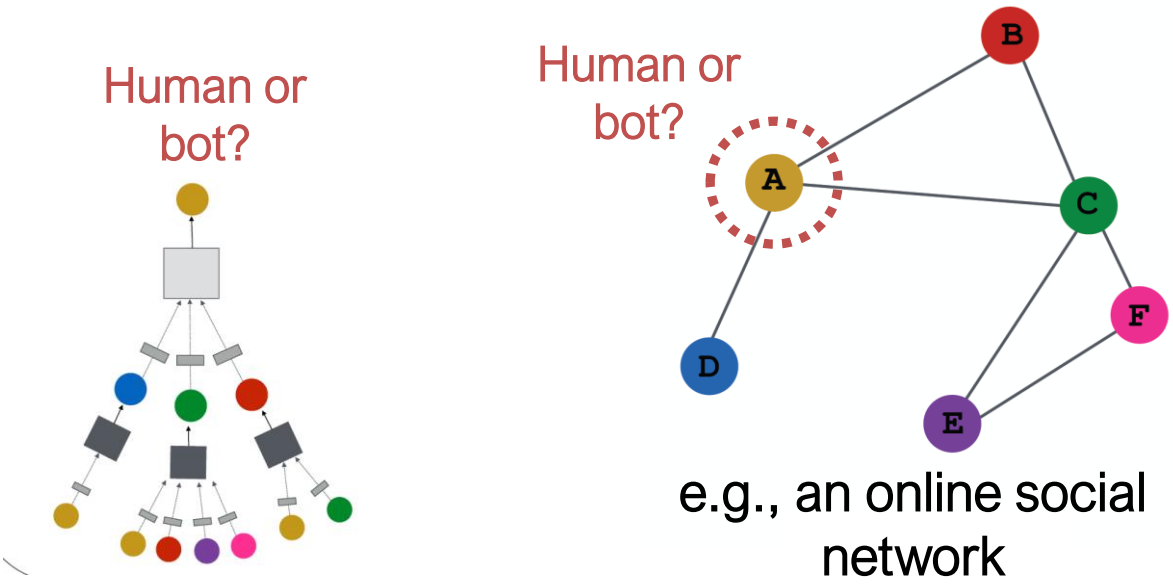
- After K-layers of neighborhood aggregation, we get output embeddings for each node.
- **We can feed these embeddings into any loss function** and run stochastic gradient descent to train the aggregation parameters.

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.

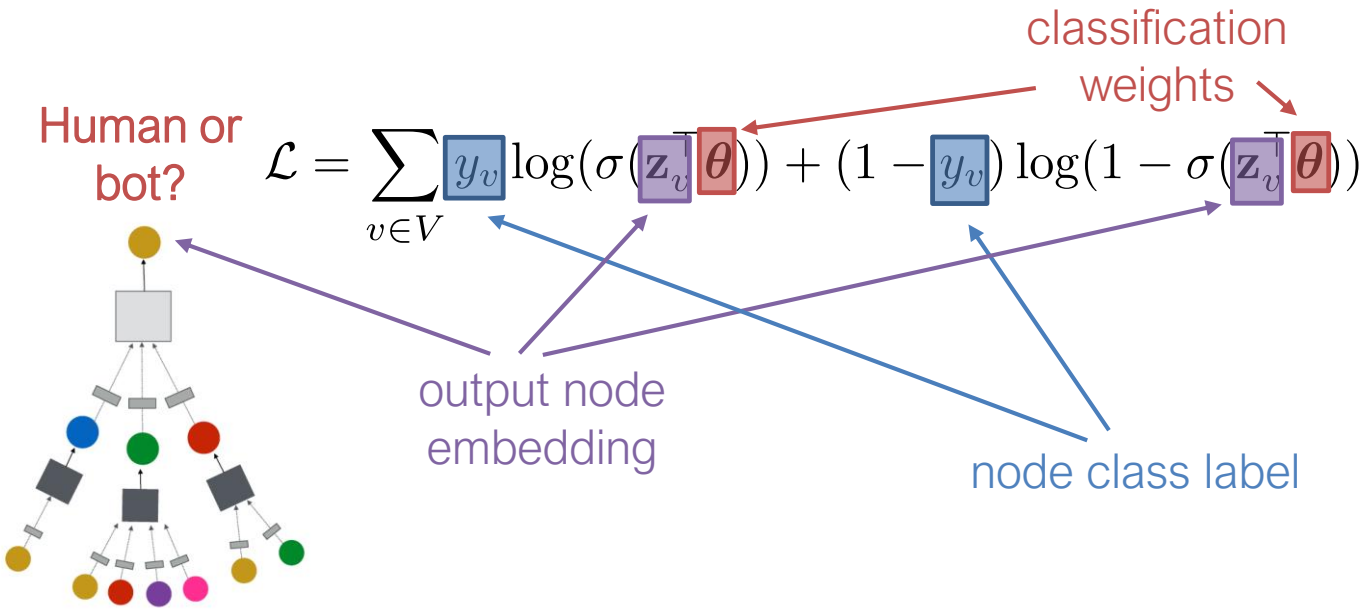
Training the Model

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



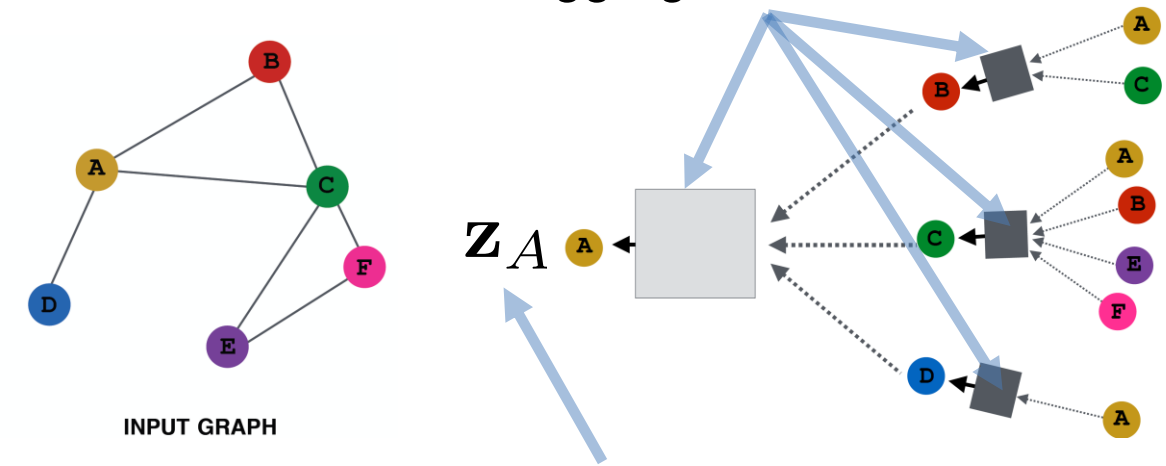
Training the Model

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



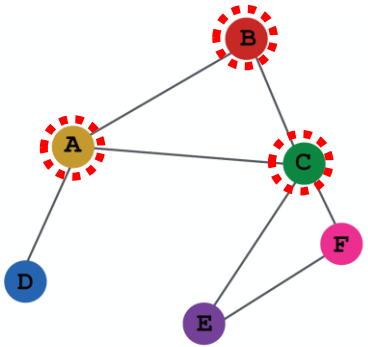
Overview of Model Design

1) Define a neighborhood aggregation function.



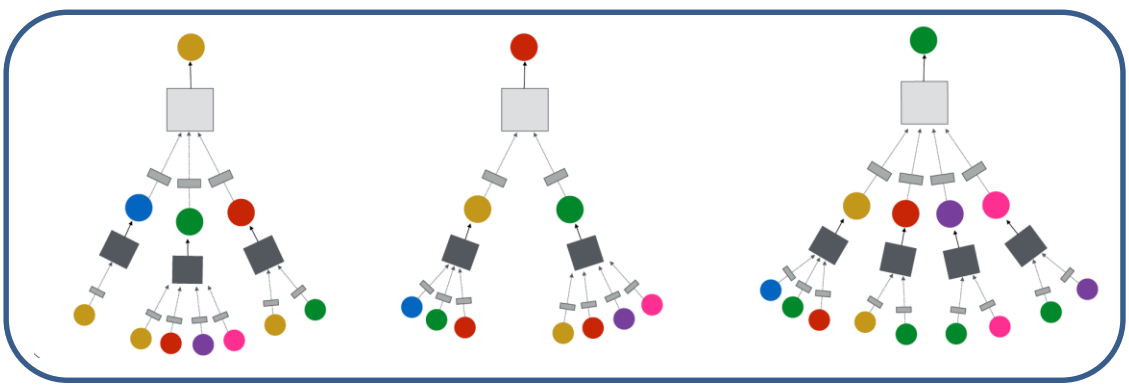
2) Define a loss function on the embeddings, $\mathbf{L}(\mathbf{z}_u)$

Overview of Model Design

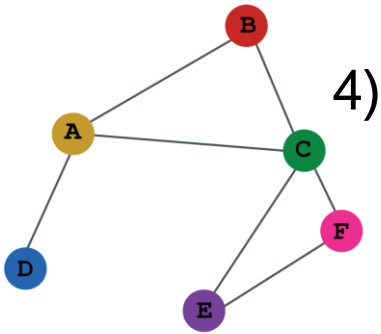


INPUT GRAPH

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



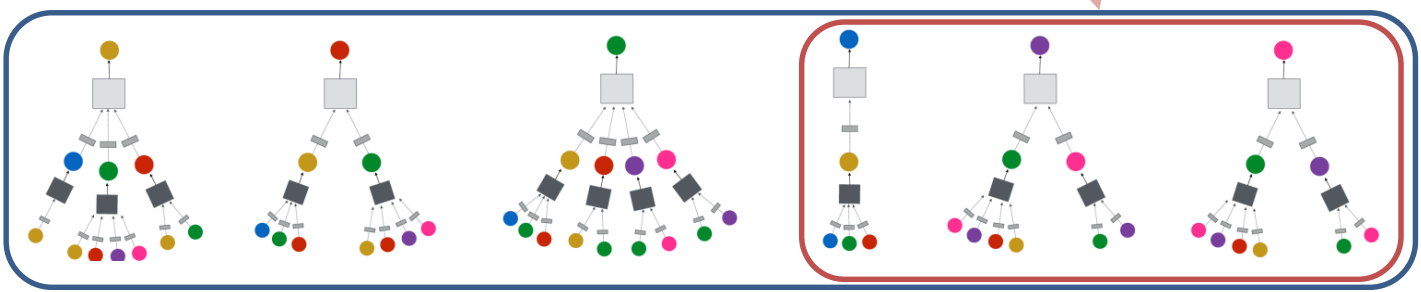
Overview of Model



INPUT GRAPH

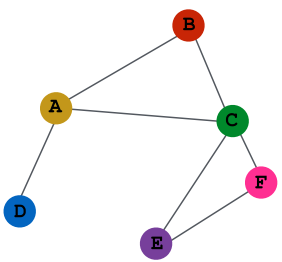
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

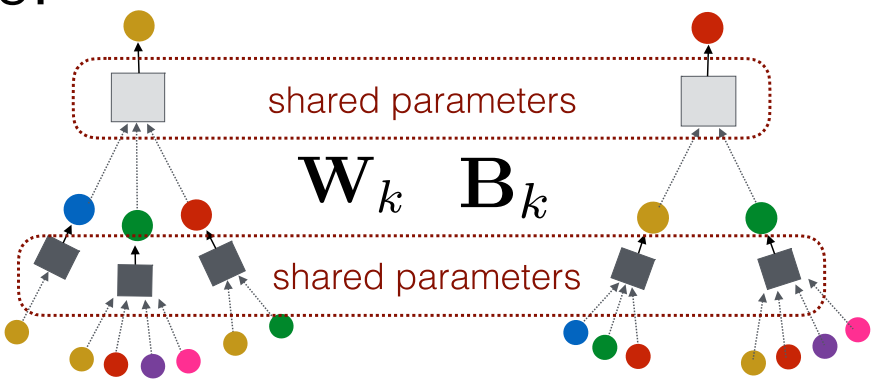


Inductive Capability

- 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!

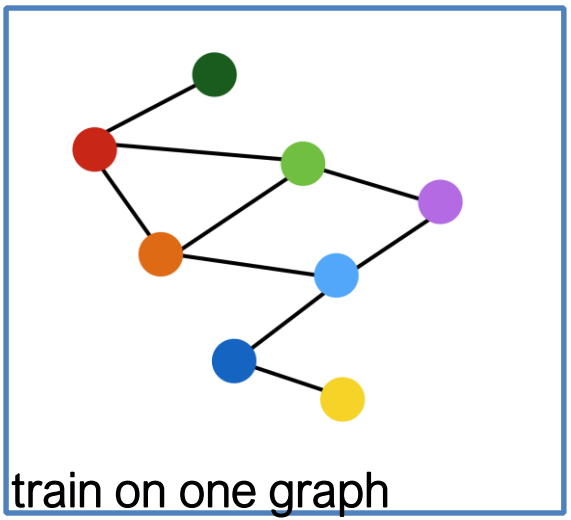


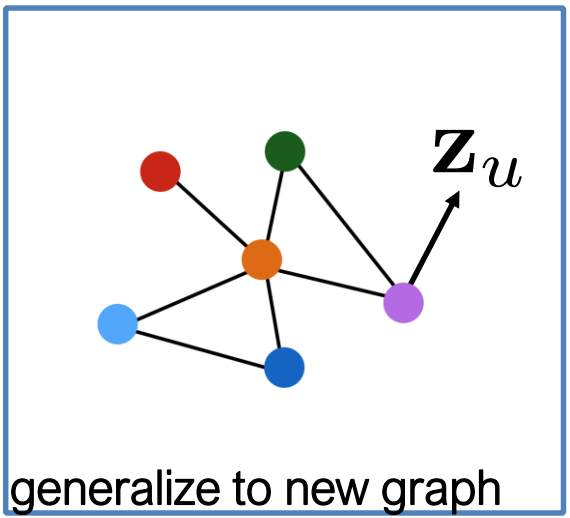
INPUT GRAPH



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

Inductive Capability

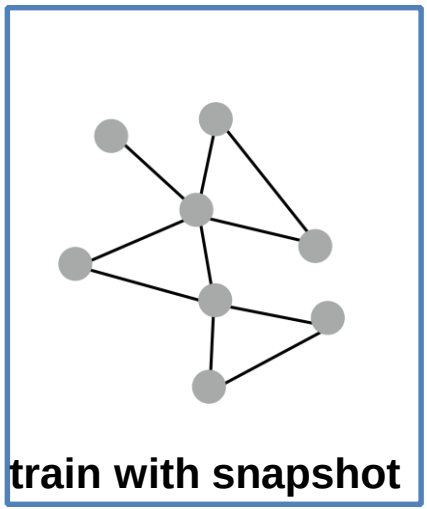


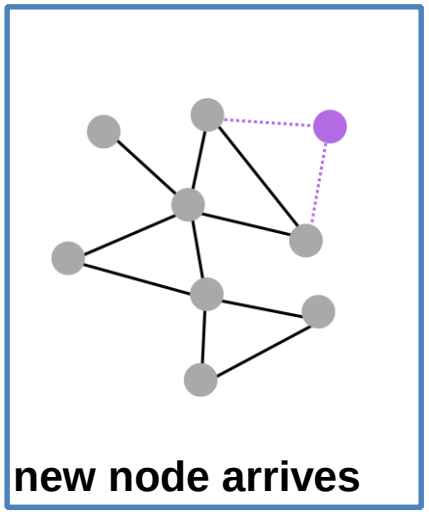


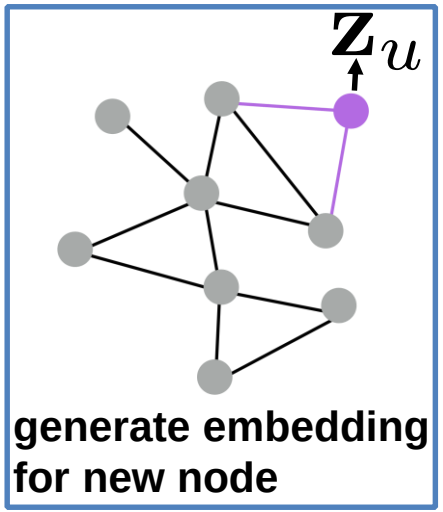
Inductive node embedding → generalize to entirely unseen graphs

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, ….

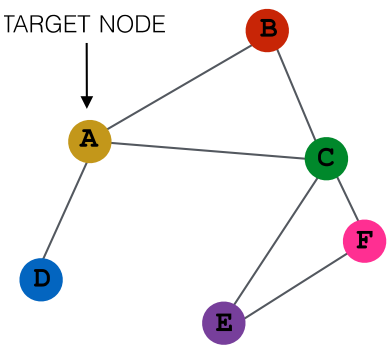
Need to generate new embeddings "on the fly"

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**… now we will cover some state of the art variants from the literature.

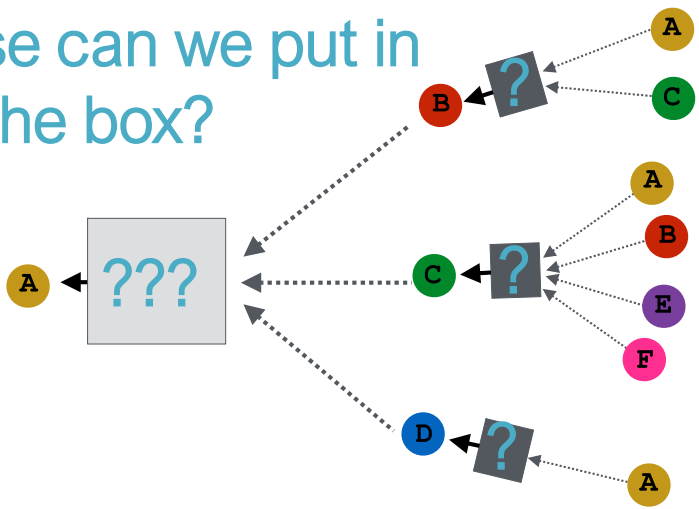
Neighborhood Aggregation

- 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 

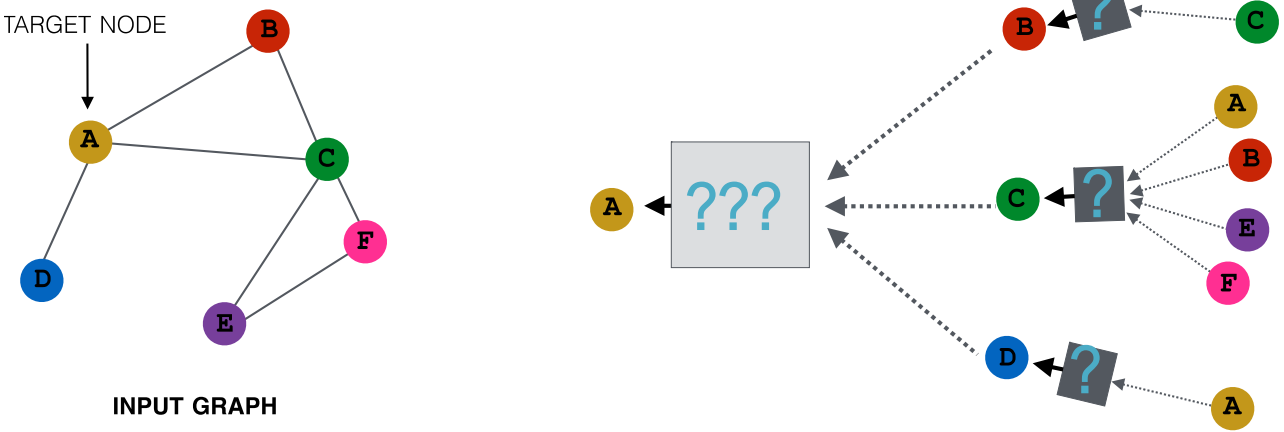
GraphSAGE

Based on material from:

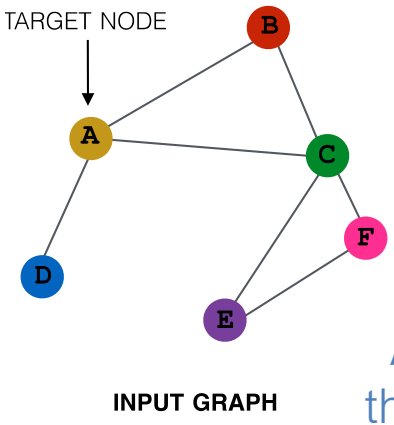
- Hamilton et al., 2017. Inductive Representation Learning on Large Graphs. *NIPS.*

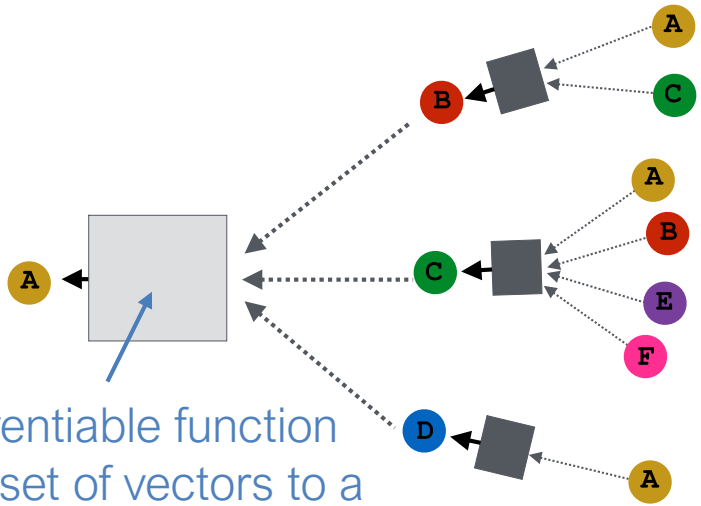
GraphSAGE Idea

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



GraphSAGE Idea





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

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

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.

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

Summary so far

- Graph convolutional networks
  - Average neighborhood information and stack neural networks.
- GraphSAGE
  - Generalized neighborhood aggregation.

This Talk

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

Application

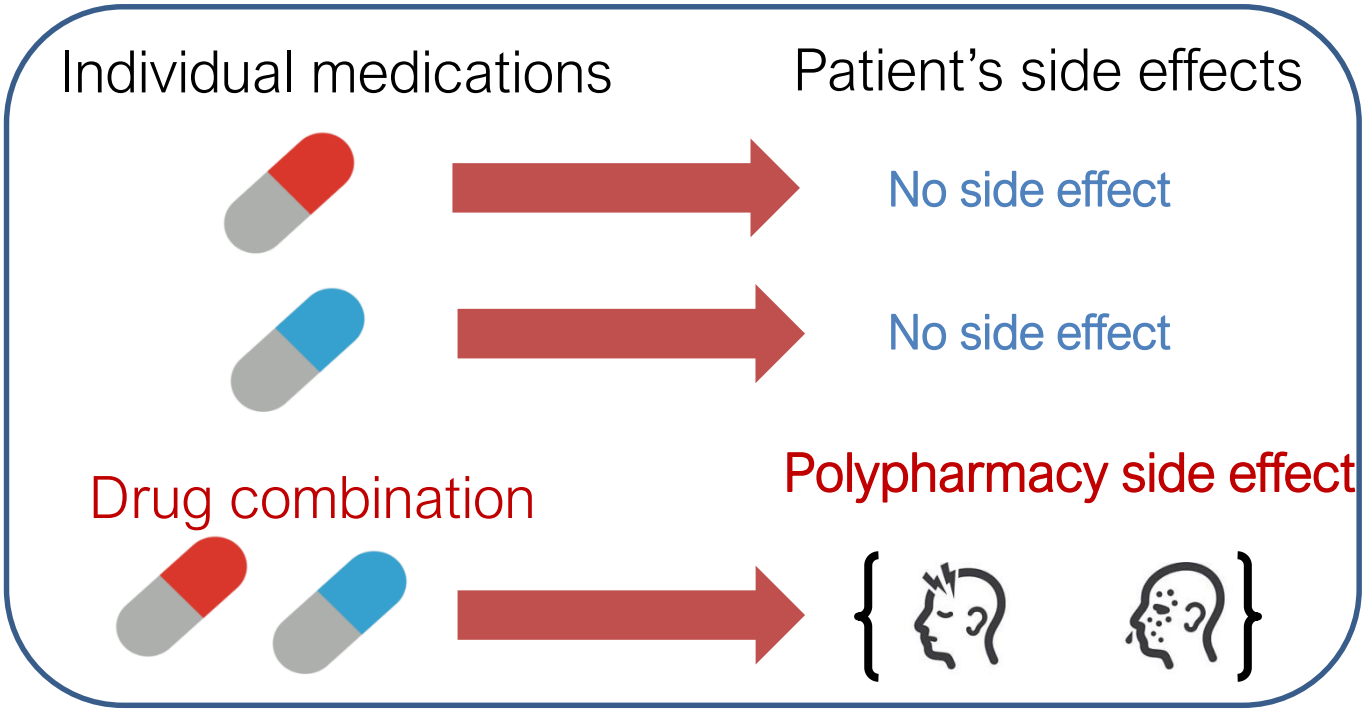
Decagon:
A Graph Convolutional Approach to Polypharmacy Side Effects

Based on material from:

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

Polypharmacy Side Effects

Goal: Predict side effects of taking multiple drugs.



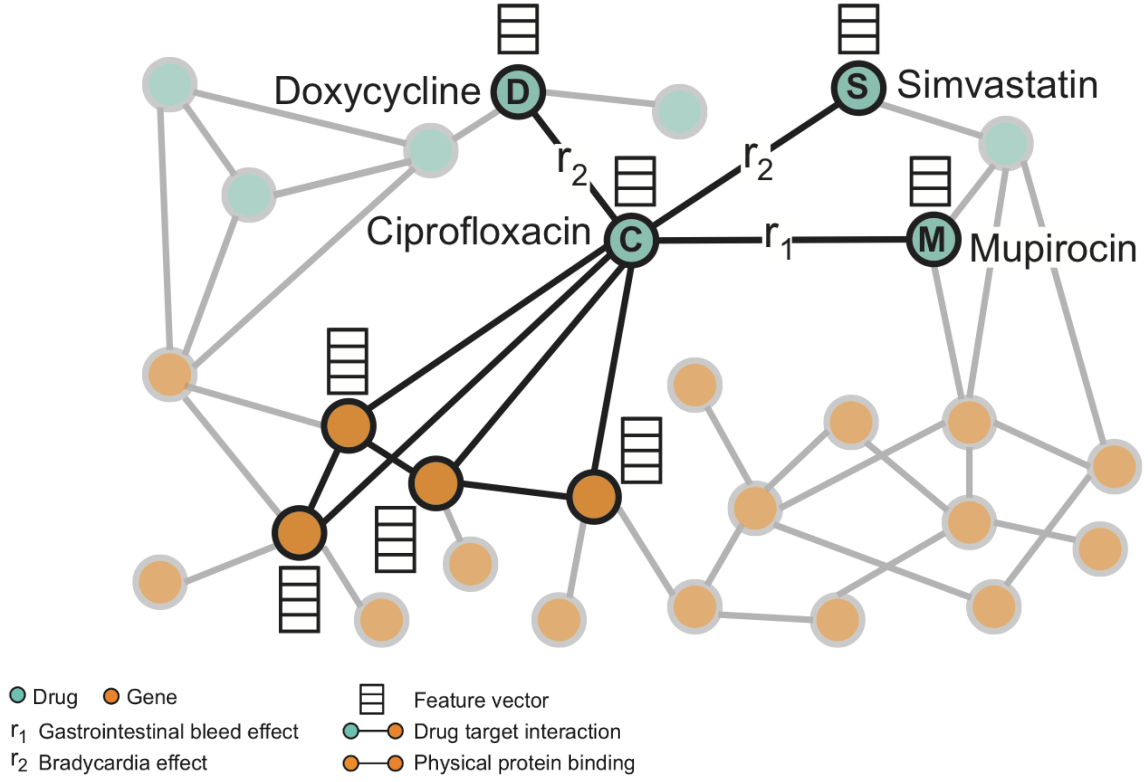
Polypharmacy Side Effects

- Polypharmacy is common to treat complex diseases and co-existing conditions
- High risk of side effects due to interactions
- 15% of the U.S. population affected
- Annual costs exceed $177 billion
- Difficult to identify manually:
  - Rare, occur only in a subset of patients
  - Not observed in clinical testing

Modeling Polypharmacy

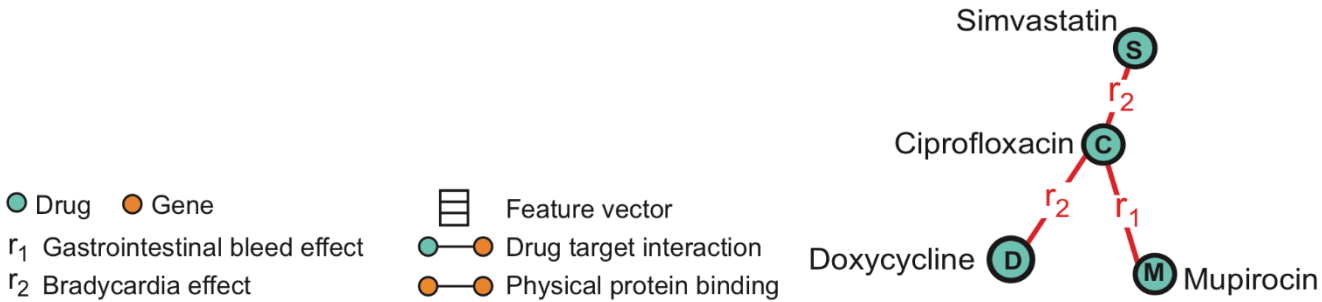
- Systematic experimental screening of drug interactions is challenging
- Idea: Computationally screen/predict polypharmacy side effects
  - Use molecular, pharmacological and patient population data
  - Guide strategies for combination treatments in patients

Data: Heterogeneous Graphs

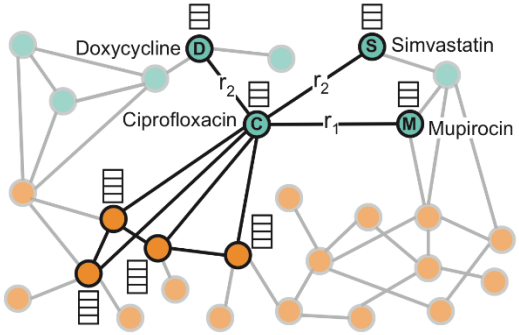


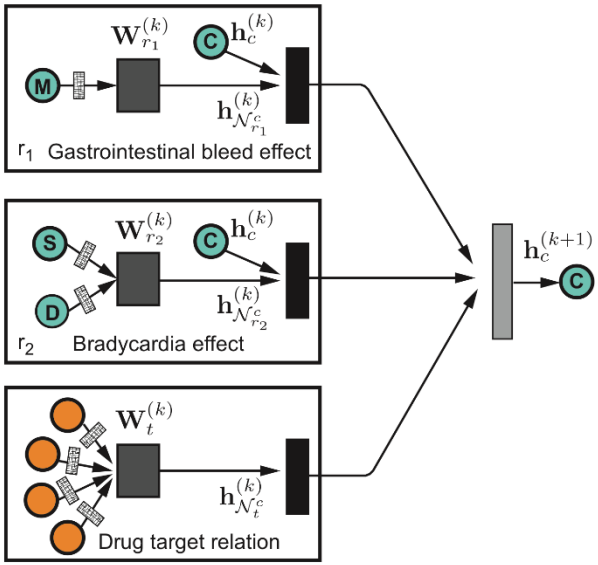
Task Description

- Predict labeled edges between drugs
  - i.e., predict the likelihood that an edge (c, r_2, s) exists
- Meaning: Drug combination (c, s) leads to polypharmacy side effect r_2



Neural Architecture: Encoder

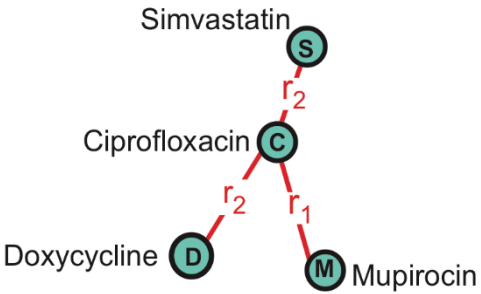


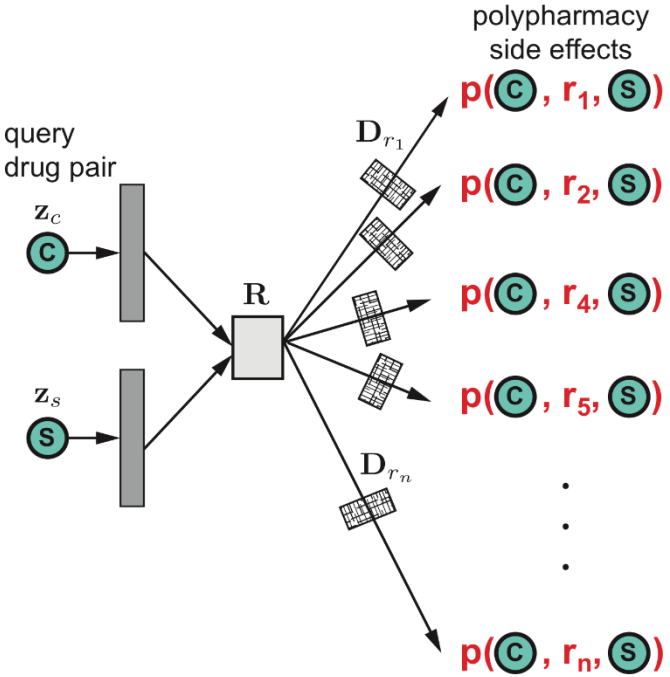


- Input: graph, additional node features
- Output: node embeddings

Drug Gene
r_1 Gastrointestinal bleed effect
r_2 Bradycardia effect
Feature vector
Drug target interaction
Physical protein binding

Making Edge Predictions





- Input: Query drug pairs and their embeddings
- Output: predicted edges

Drug Gene

r_1 Gastrointestinal bleed effect

r_2 Bradycardia effect

Feature vector

Drug target interaction

Physical protein binding

Experimental Setup

- **Data:**
  - Molecular: protein-protein interactions and drug target relationships
  - Patient data: Side effects of individual drugs, polypharmacy side effects of drug combinations
- **Setup:**
  - Construct a heterogeneous graph of all the data
  - Train: Fit a model to predict known associations of drug pairs and polypharmacy side effects
  - Test: Given a query drug pair, predict candidate polypharmacy side effects

Prediction Performance

	AUROC	AUPRC	AP@50
Decagon (3-layer)	**0.834**	**0.776**	**0.731**
Decagon (2-layer)	0.809	0.762	0.713
RESCAL	0.693	0.613	0.476
Node2vec	0.725	0.708	0.643
Drug features	0.736	0.722	0.679

- Up to 54% improvement over baselines
- First opportunity to computationally flag polypharmacy side effects for follow-up analyses