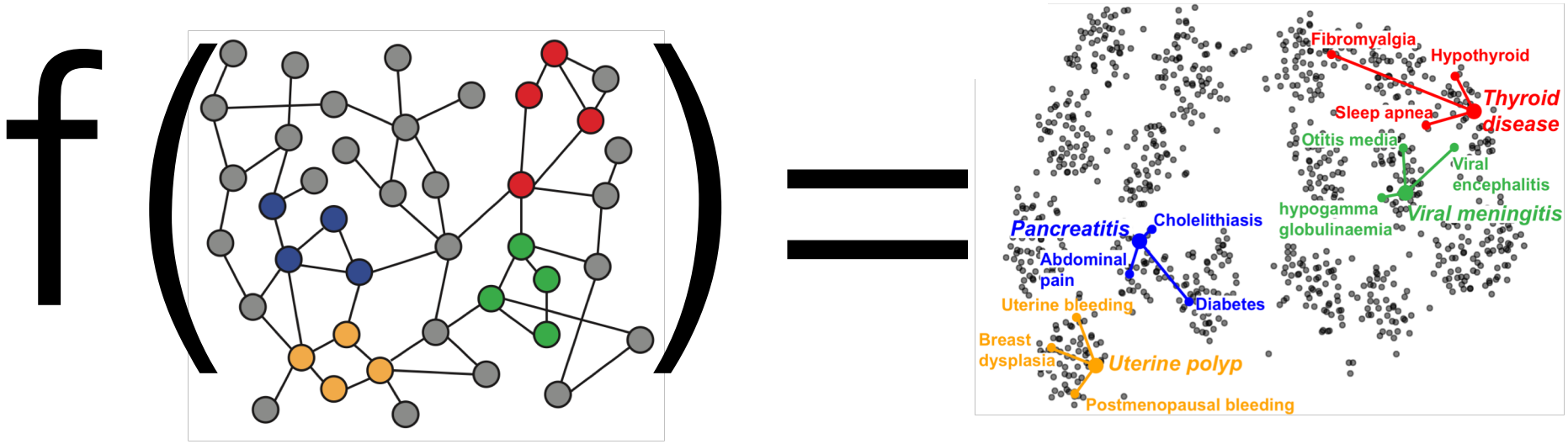
Graph Convolutional Networks

CS224W: Analysis of Networks
Jure Leskovec and Marinka Zitnik, Stanford University
http://cs224w.stanford.edu

Node Embeddings

- **Intuition:** Map nodes to d-dimensional embeddings such that similar nodes in the graph are embedded close together



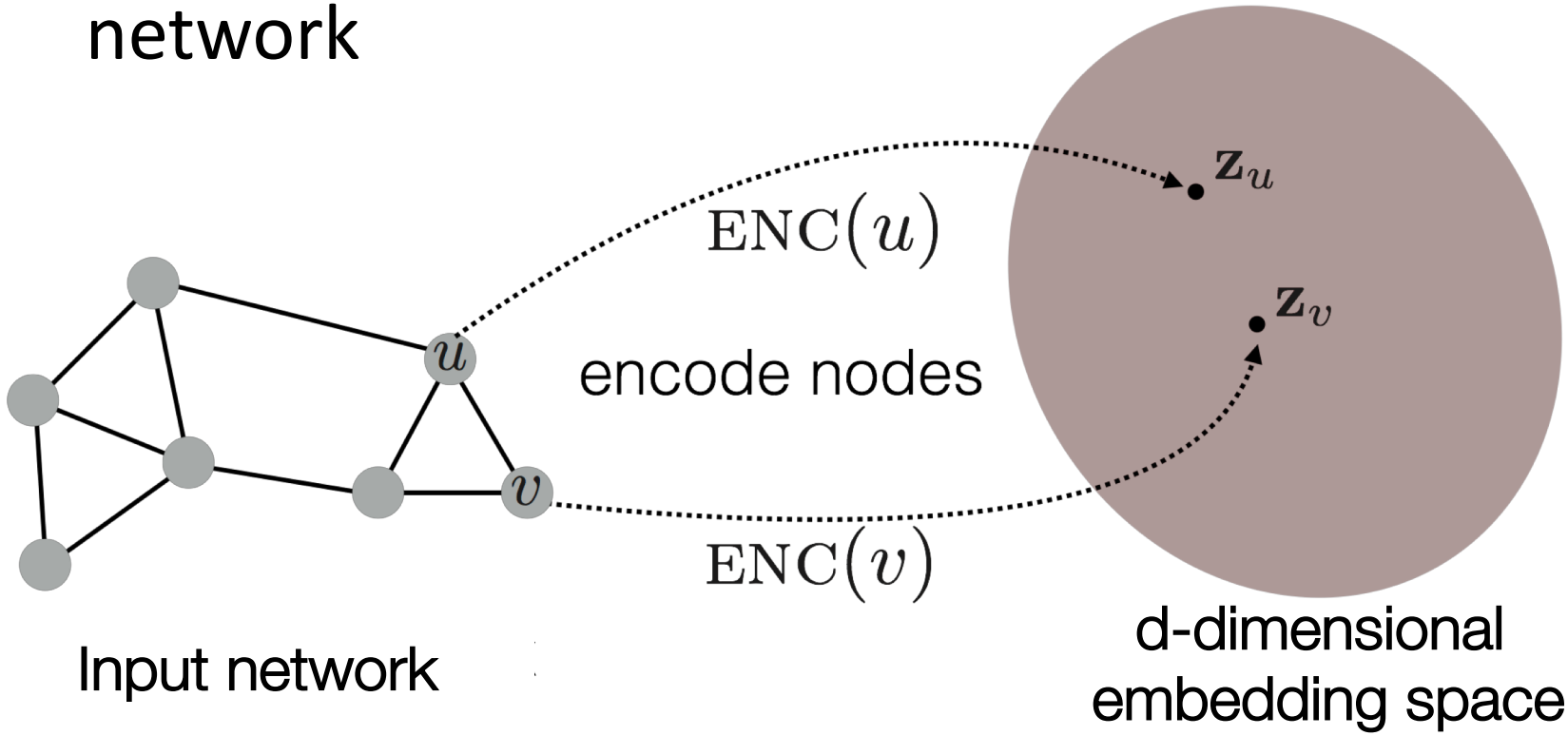
Input graph

2D node embeddings

How to learn mapping function f?

Node Embeddings

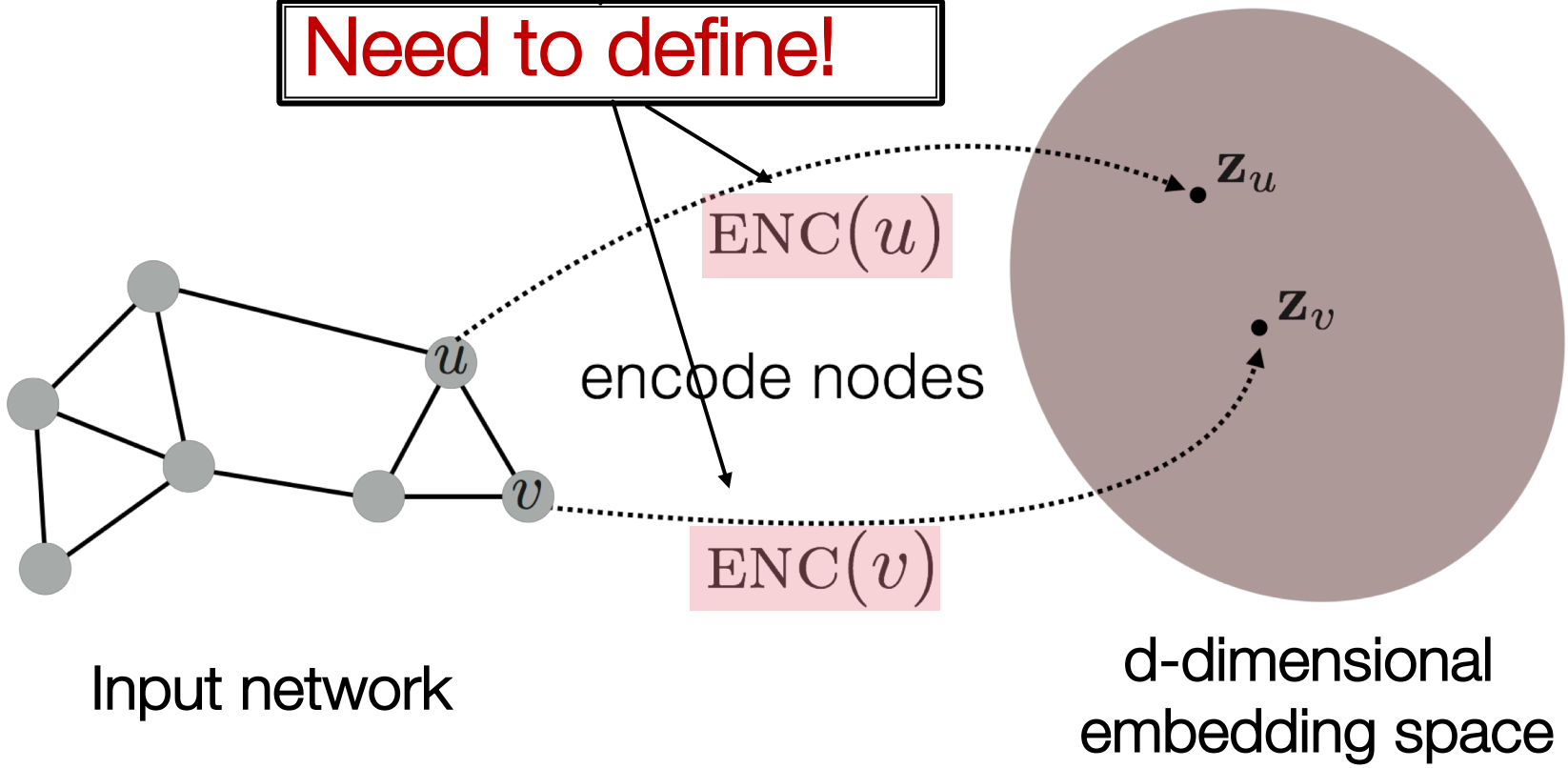
- **Goal:** Map nodes so that similarity in the embedding space (e.g., dot product) approximates similarity (e.g., proximity) in the network



Node Embeddings

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

Need to define!



Two Key Components

- **Encoder:** Map a node to a low-dimensional vector:

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

$\mathbf{z}_v$: d-dimensional embedding

v: node in the input graph

- **Similarity function** defines how relationships in the input network map to relationships in the embedding space:

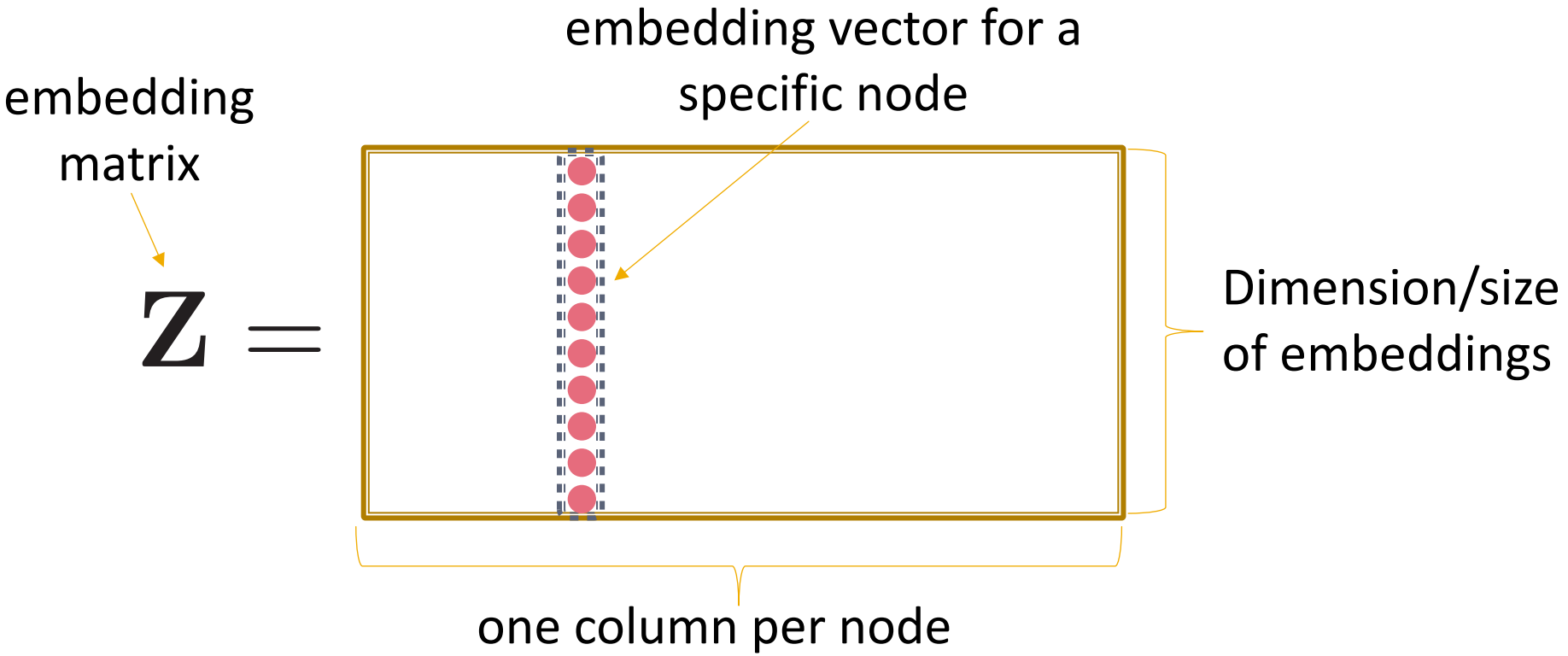
$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the network

dot product between node embeddings

From "Shallow" to "Deep"

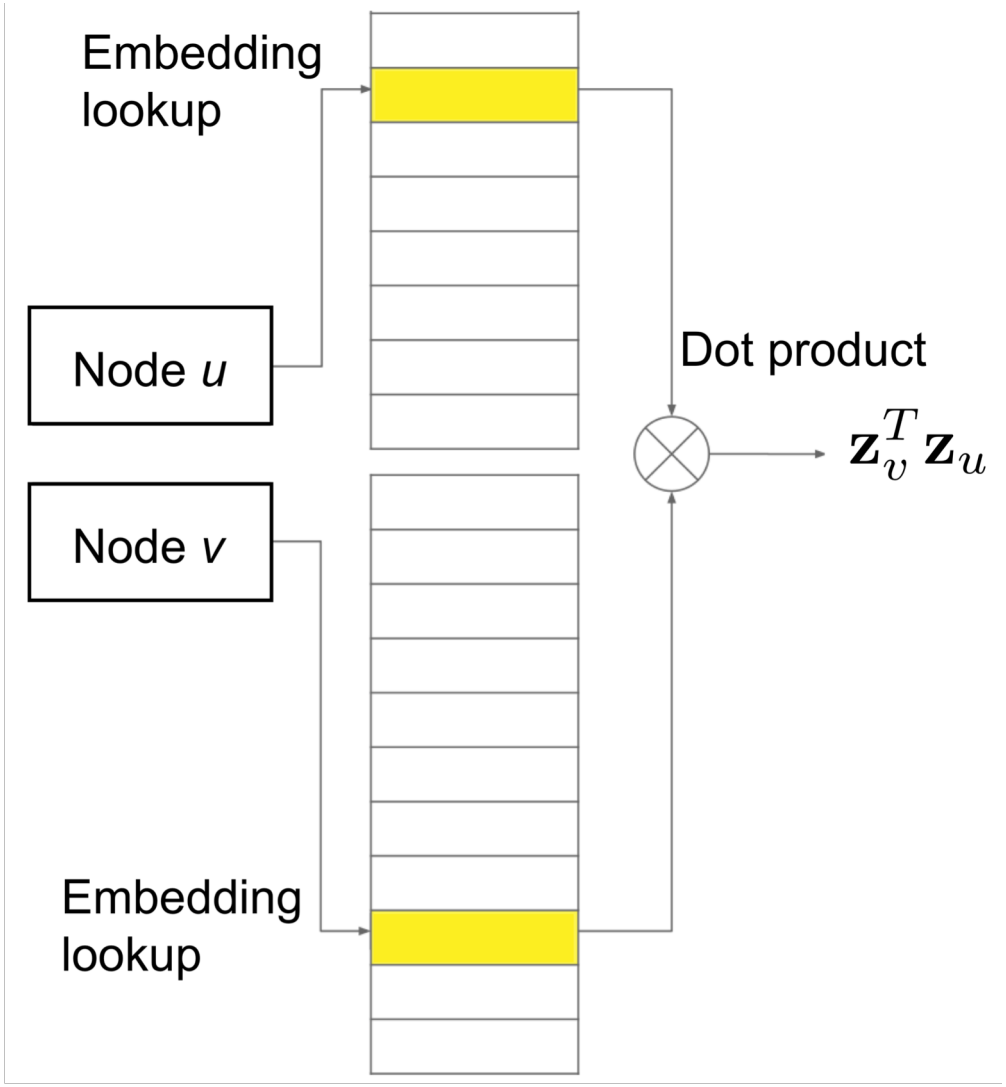
- So far we have focused on **"shallow" encoders**, i.e. embedding lookups:



Shallow Encoders (Lec. 09: 10/23)

Shallow encoders:

- One-layer of data transformation
- A single hidden layer maps node u to embedding $\boldsymbol{z}_u$ via function f, e.g., $\boldsymbol{z}_u = f\left(\boldsymbol{z}_v, v \in N_R(u)\right)$



Shallow Encoders (Lec. 09: 10/23)

- Limitations of shallow embedding methods:
  - **O(|V|) parameters are needed**:
    - No sharing of parameters between nodes
    - Every node has its own unique embedding
  - **Inherently "transductive"**:
    - Cannot generate embeddings for nodes that are not seen during training
  - **Do not incorporate node features**:
    - Many graphs have features that we can and should leverage

Today: Deep Graph Encoders

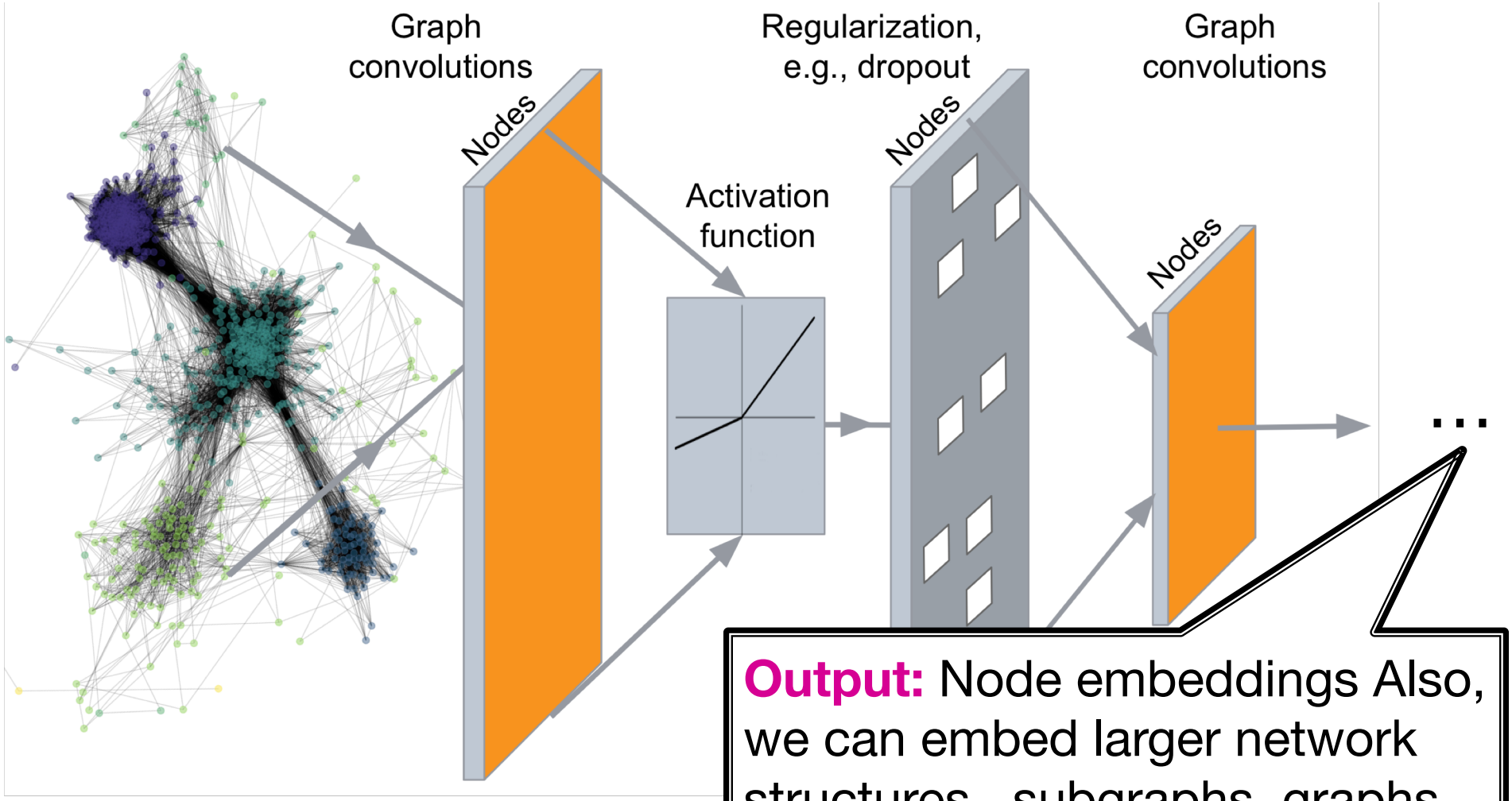
- **Today:** We will now discuss deep methods based on **graph neural networks:**

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

multiple layers of non-linear transformation of graph structure

- **Note:** All these deep encoders can be **combined with node similarity functions** defined in CS224W lecture 09

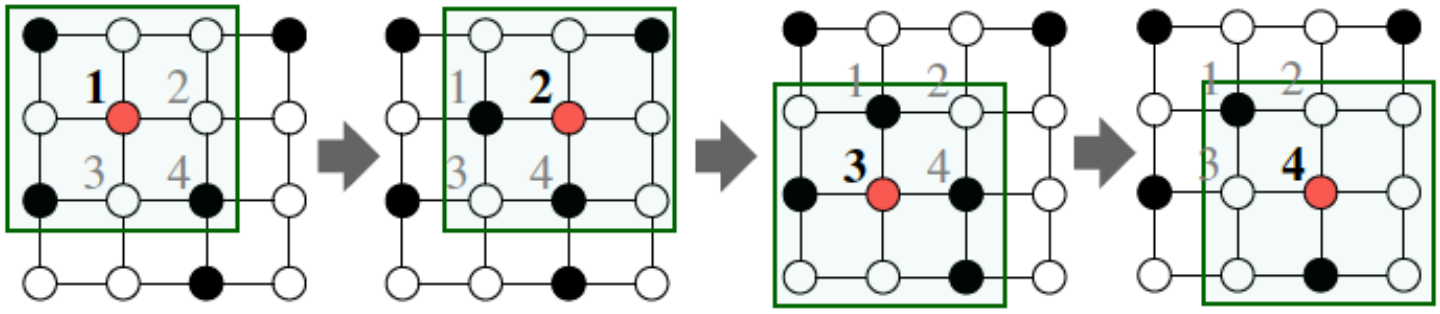
Deep Graph Encoders



Output: Node embeddings Also, we can embed larger network structures, subgraphs, graphs

Idea: Convolutional Networks

CNN on an image:

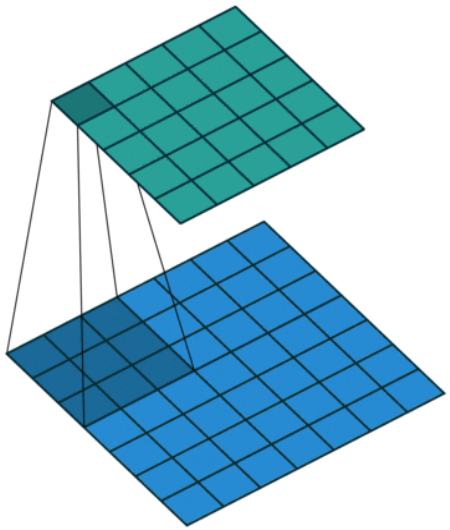


Goal is to generalize convolutions beyond simple lattices

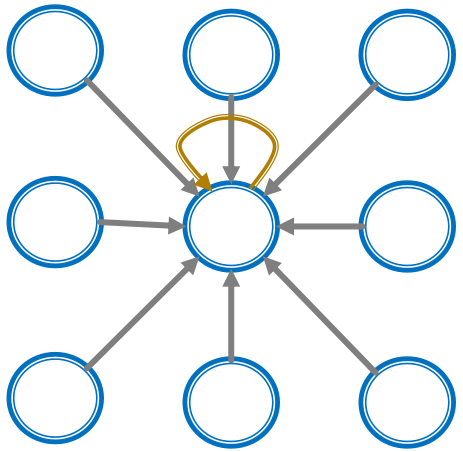
Leverage node features/attributes (e.g., text, images)

From Images to Graphs

Single CNN layer with 3x3 filter:



Image



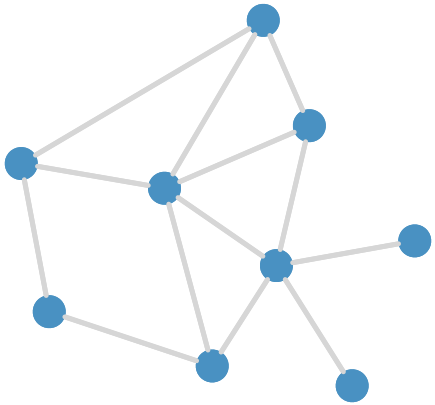
Graph

Transform information at the neighbors and combine it:

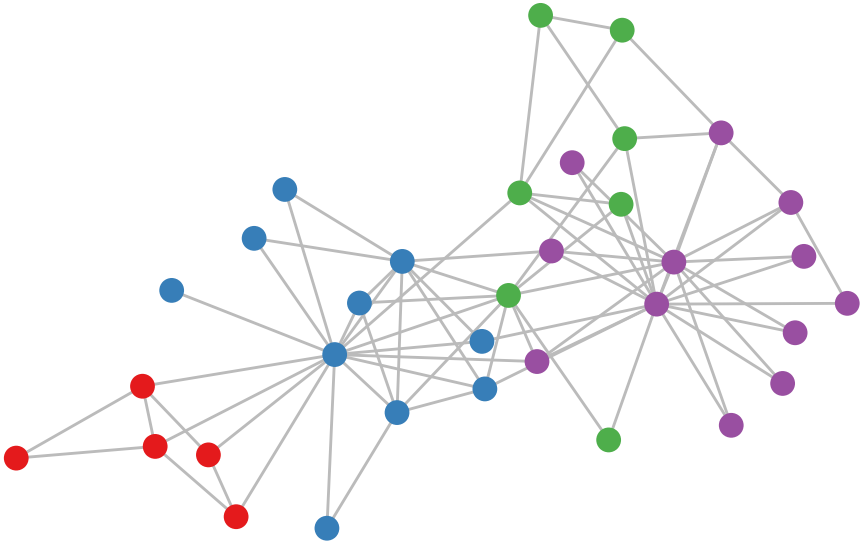
- Transform "messages" h_i from neighbors: $W_i\ h_i$
- Add them up: $\sum_i W_i\ h_i$

Real-World Graphs

But what if your graphs look like this?



or this:

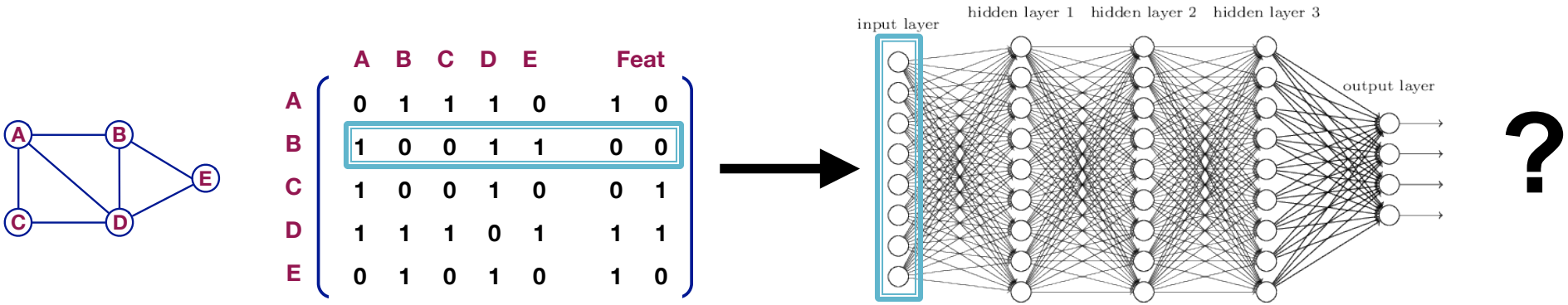


- Examples:

  Biological networks, Medical networks, Social networks, Information networks, Knowledge graphs, Communication networks, Web graph, …

A Naïve Approach

- Join adjacency matrix and features
- Feed them into a deep neural net:



- $O(N)$ parameters
- Not applicable to graphs of different sizes
- Not invariant to node ordering

Outline of Today's Lecture

1. **Basics of deep learning for graphs**

2. **Graph Convolutional Networks**
3. **Graph Attention Networks (GAT)**
4. **Practical tips and demos**

Basics of Deep Learning for Graphs

Content

- Local network neighborhoods:
  - Describe aggregation strategies
  - Define computation graphs

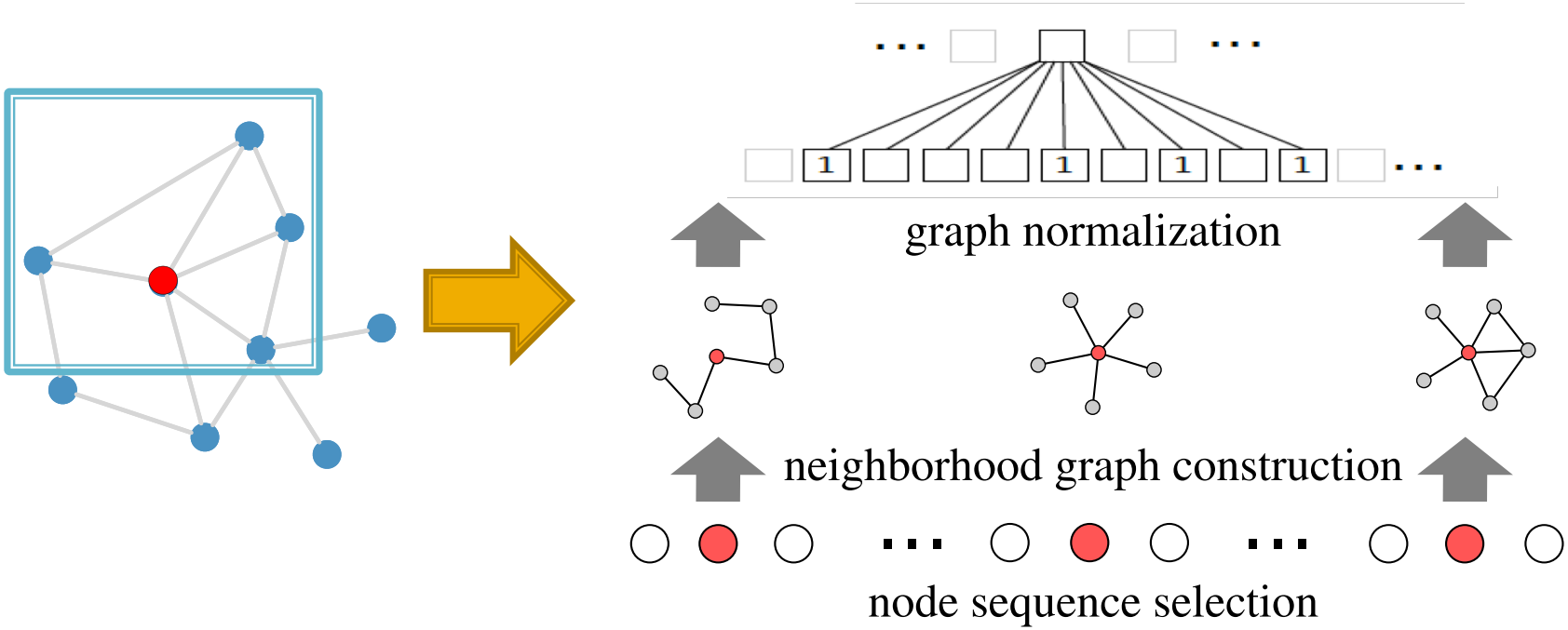
- Stacking multiple layers:
  - Describe the model, parameters, training
  - How to fit the model?
  - Simple example for unsupervised and supervised training

Setup

- Assume we have a graph G:
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ is a matrix of **node features**
    - Biologically meaningful node features:
      - E.g., immunological signatures, gene expression profiles, gene functional information
    - No features:
      - Indicator vectors (one-hot encoding of a node)

[Scarselli et al., IEEE TNN 2005, Niepert et al., ICML 2016]

Graph Convolutions

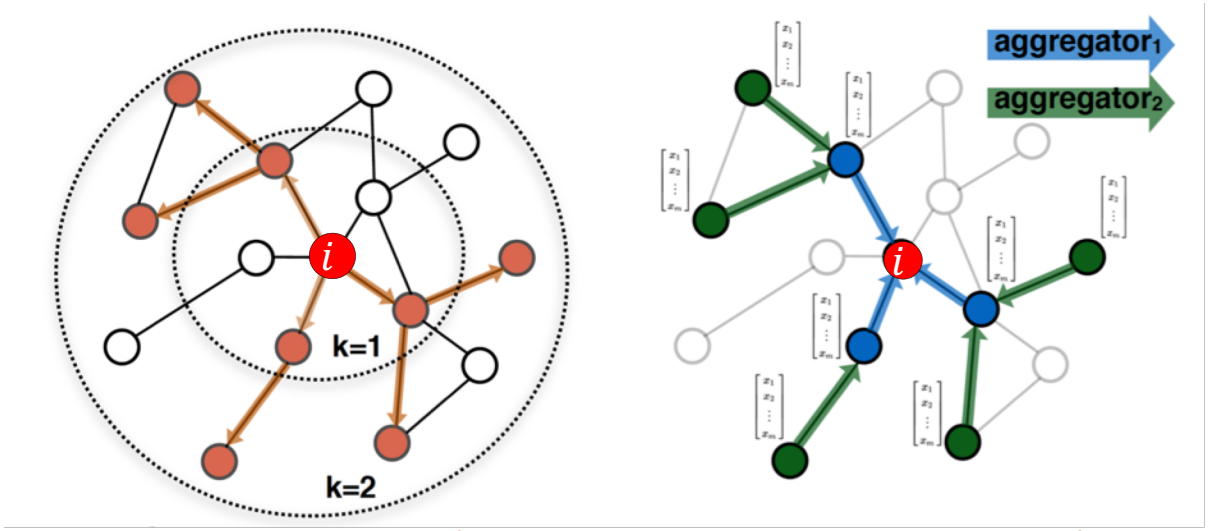


Problem: For a given subgraph how to come with canonical node ordering

[Kipf and Welling, ICLR 2017]

Graph Convolutional Networks

Idea: Node's neighborhood defines a computation graph



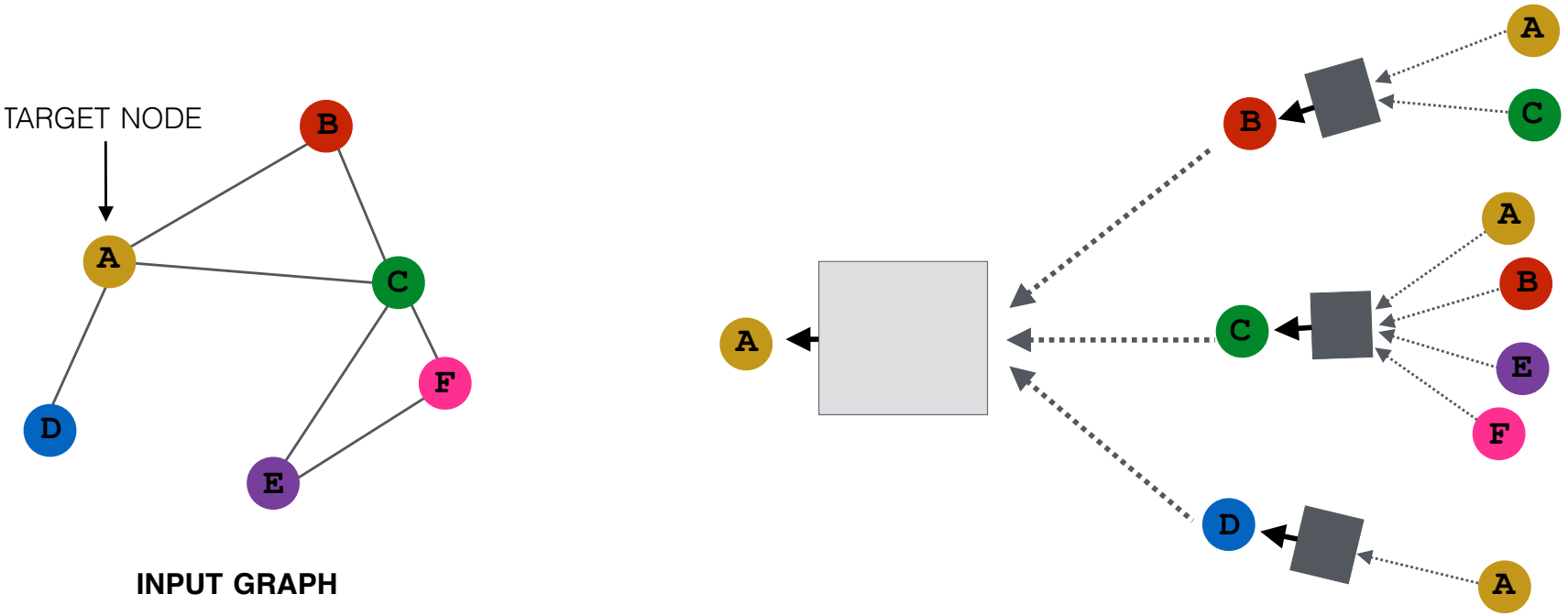
Determine node computation graph

Propagate and transform information

Learn how to propagate information across the graph to compute node features

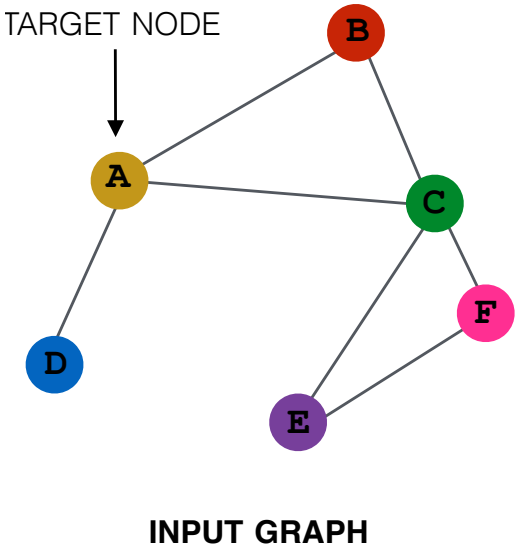
Idea: Aggregate Neighbors

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

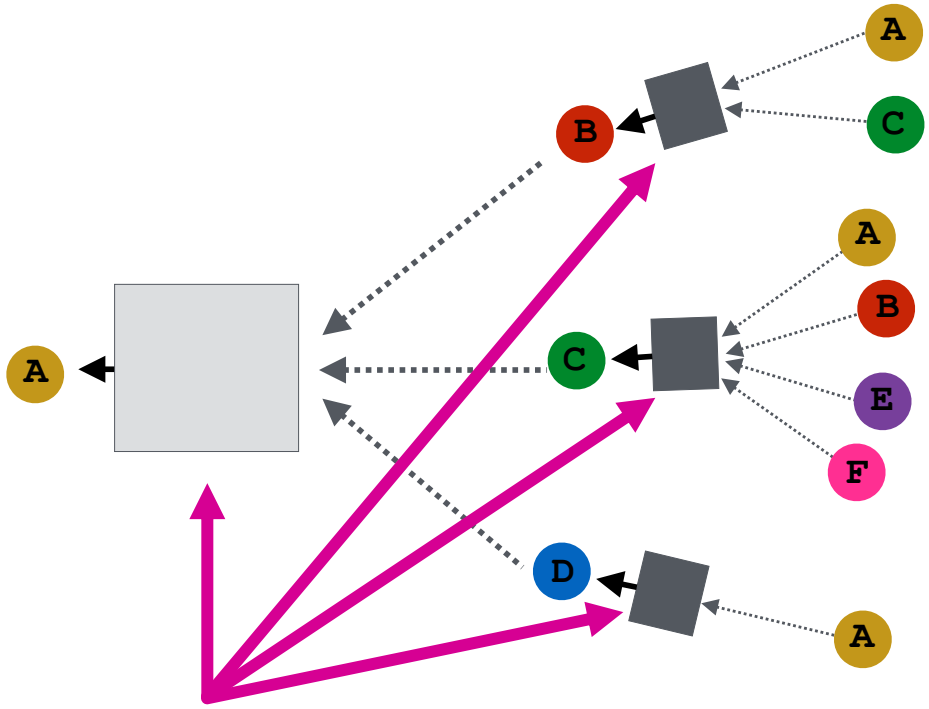


Idea: Aggregate Neighbors

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



INPUT GRAPH

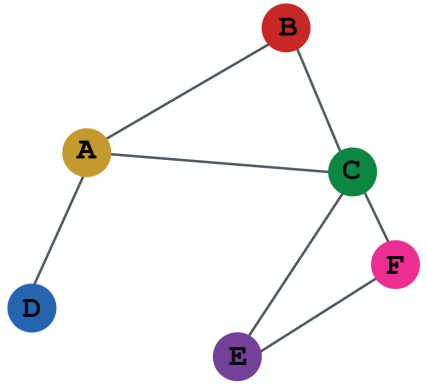


Neural networks

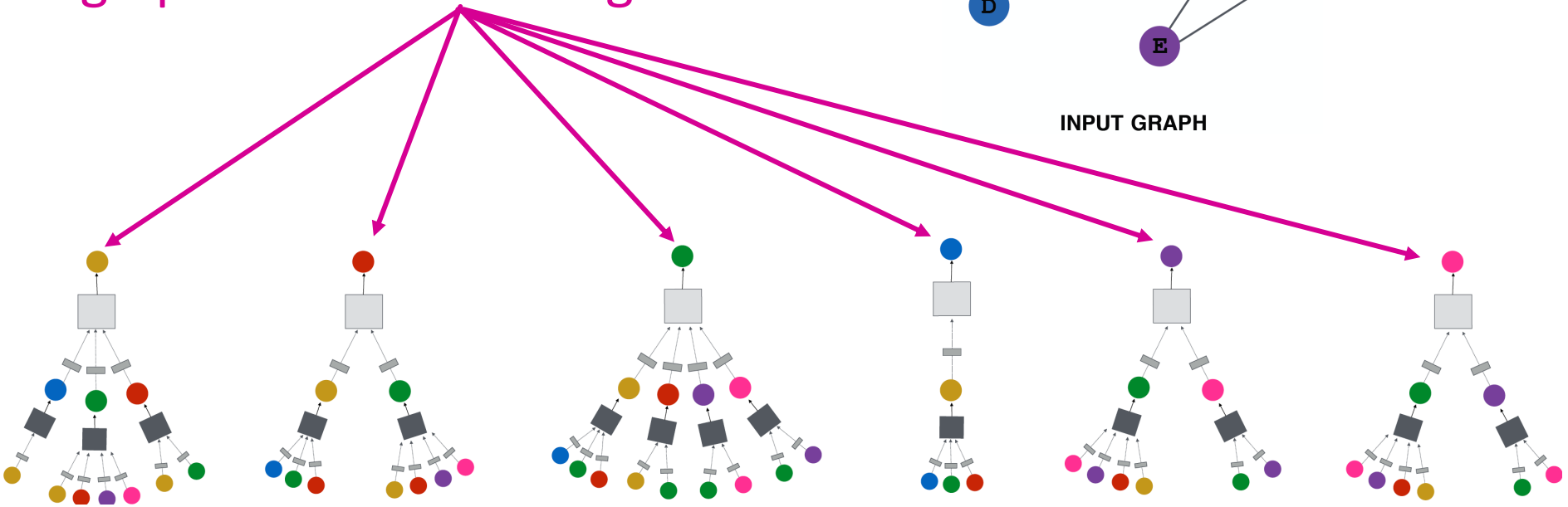
Idea: Aggregate Neighbors

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

Every node defines a computation graph based on its neighborhood!

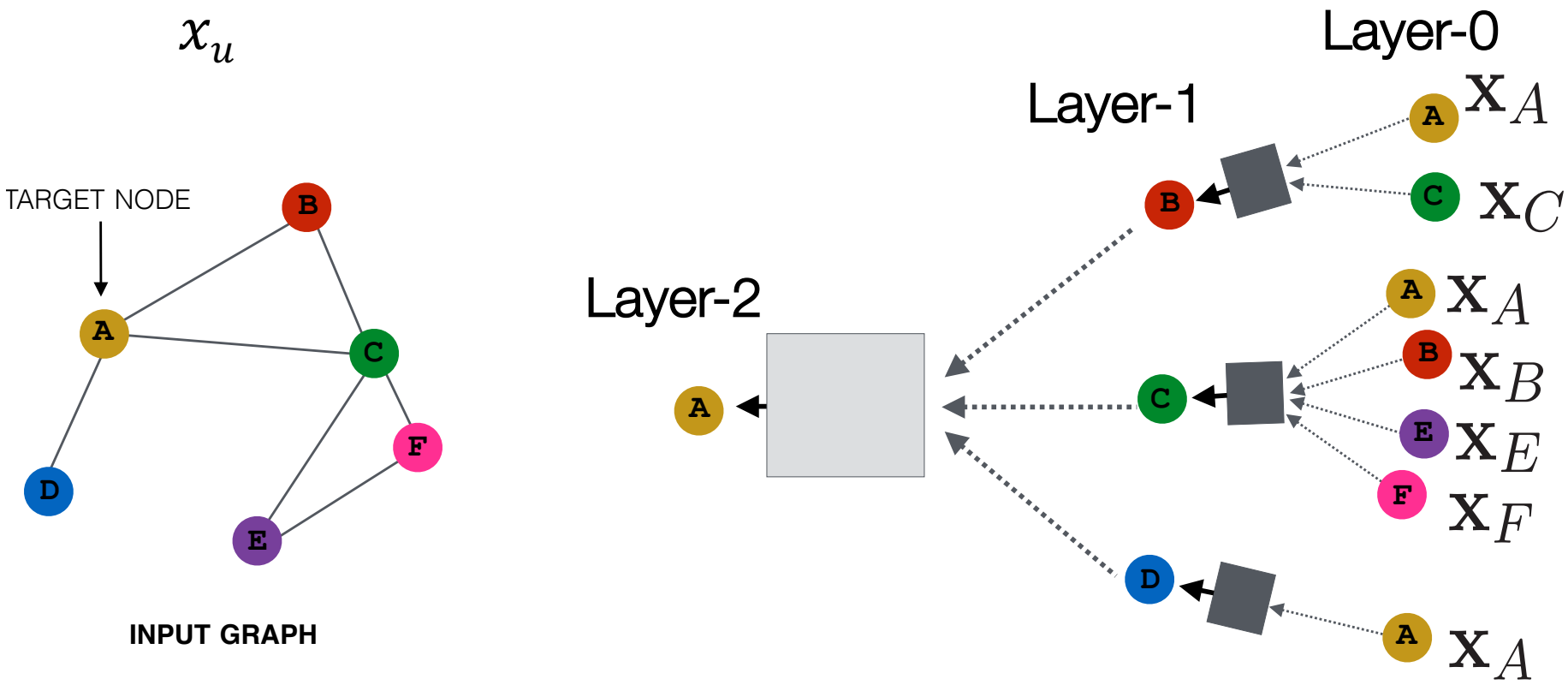


INPUT GRAPH



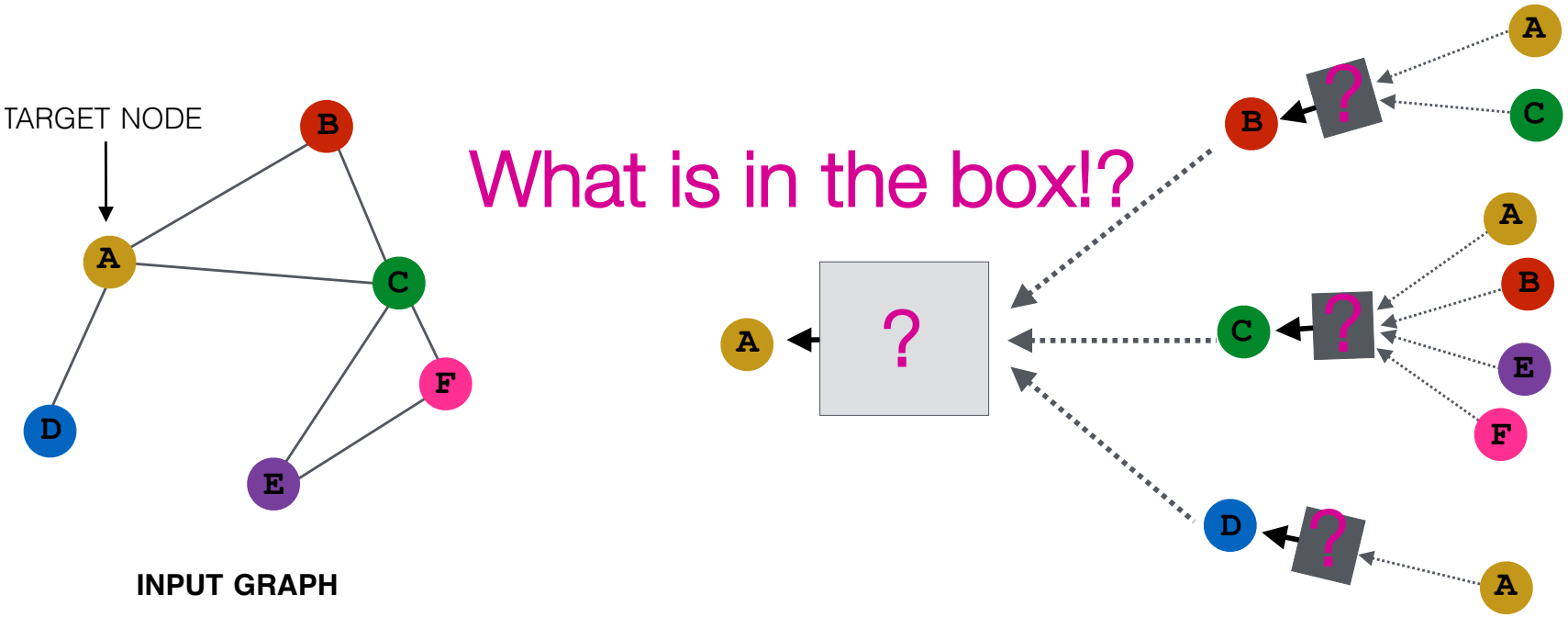
Deep Model: Many Layers

- Model can be of arbitrary depth:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node u is its input feature, i.e. x_u



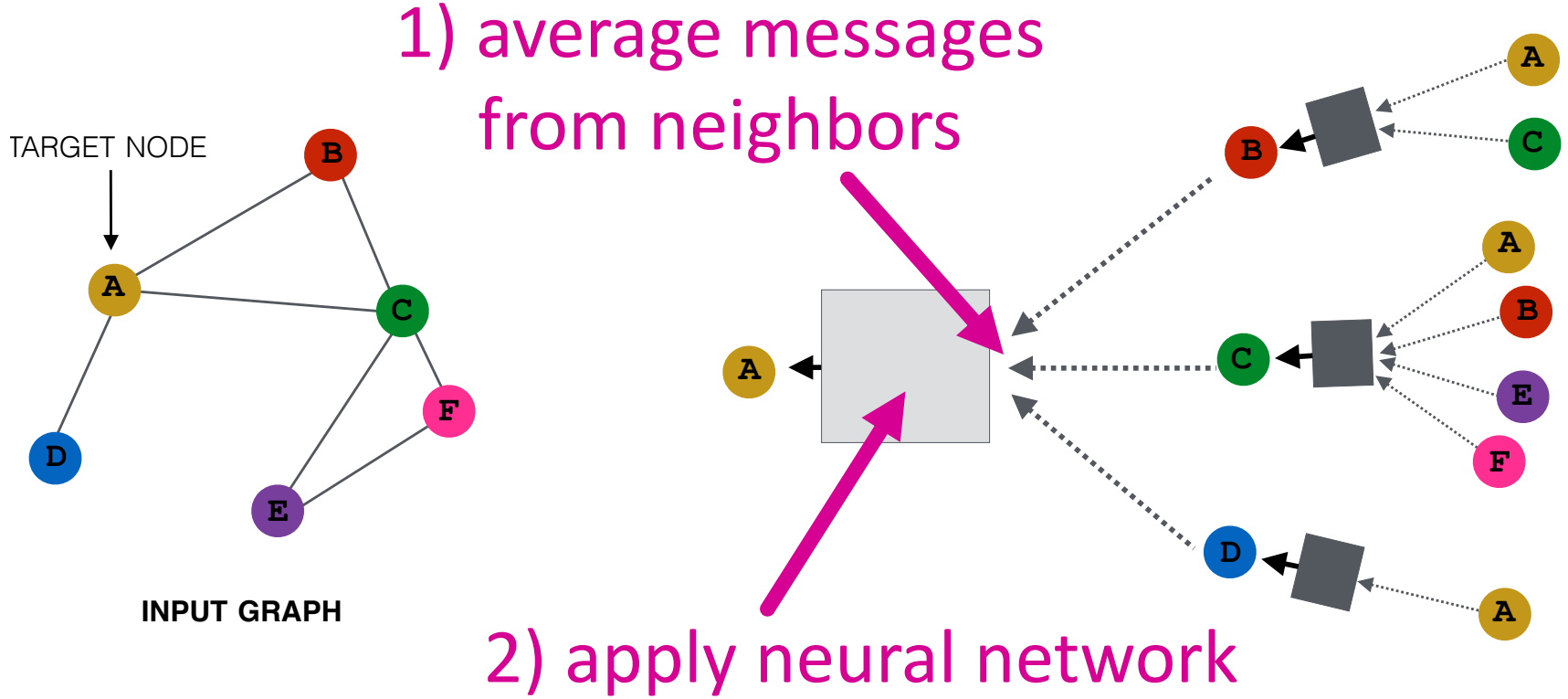
Neighborhood Aggregation

- **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers



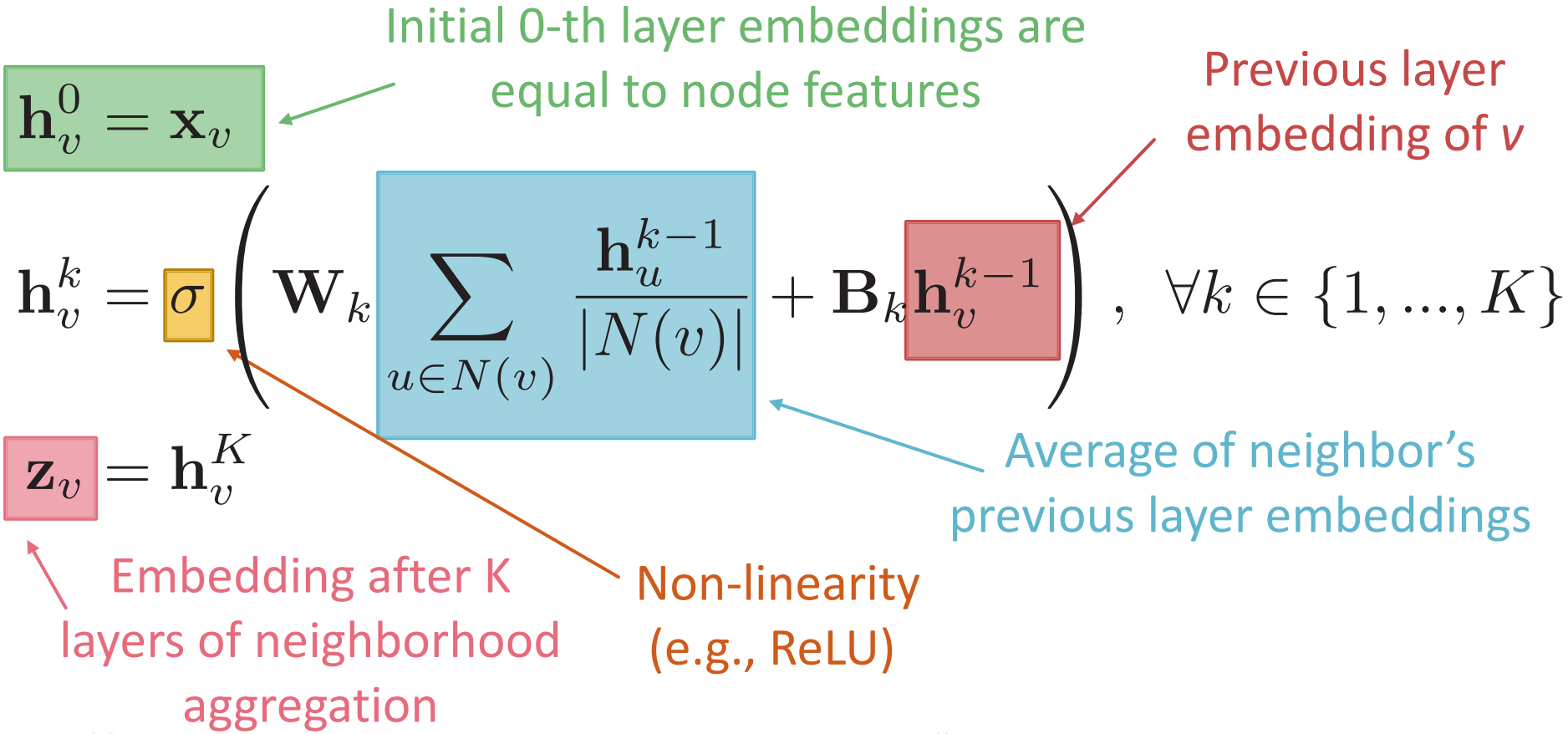
Neighborhood Aggregation

- **Basic approach:** Average information from neighbors and apply a neural network



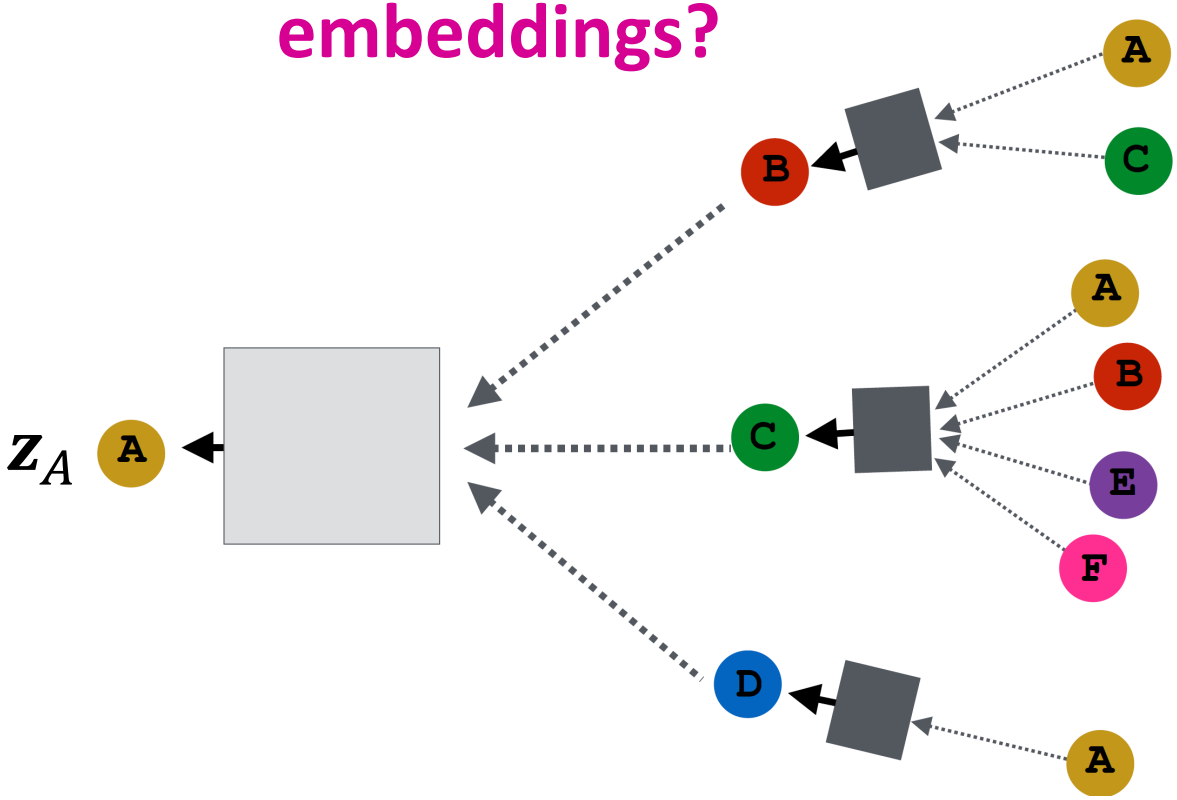
The Math: Deep Encoder

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



Training the Model

How do we train the model to generate embeddings?



Need to define a loss function on the embeddings

Model Parameters

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

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

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

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

We can feed these **embeddings into any loss function** and run stochastic gradient descent to **train the weight parameters**

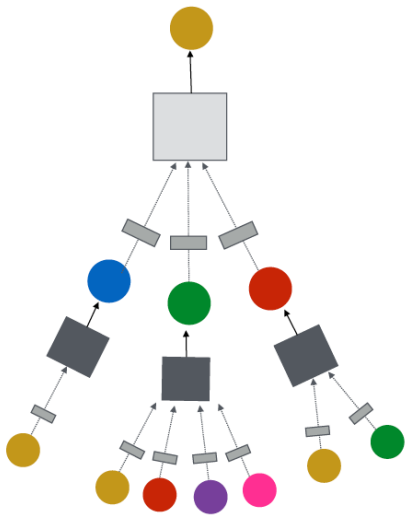
Unsupervised Training

- Train in an unsupervised manner:
  - Use only the graph structure
  - **"Similar" nodes have similar embeddings**
- Unsupervised loss function can be anything from the last section, e.g., a loss based on
  - Random walks (node2vec, DeepWalk, struc2vec)
  - Graph factorization
  - Node proximity in the graph

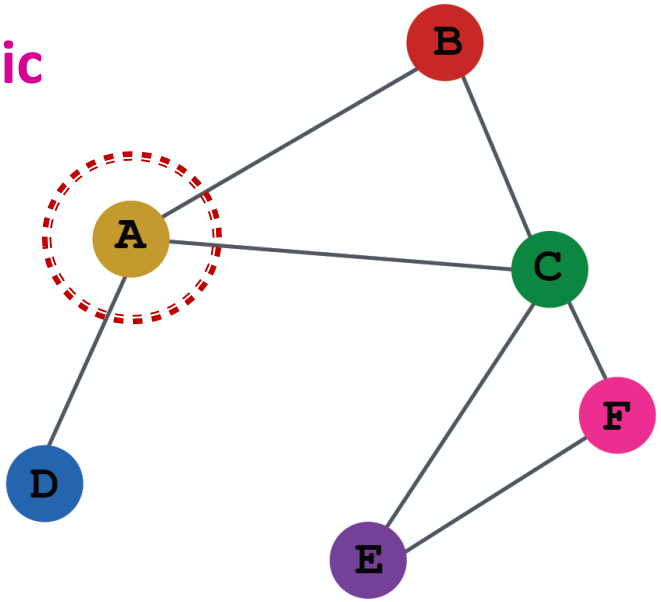
Supervised Training

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

Safe or toxic drug?



Safe or toxic drug?



E.g., a drug-drug interaction network

Supervised Training

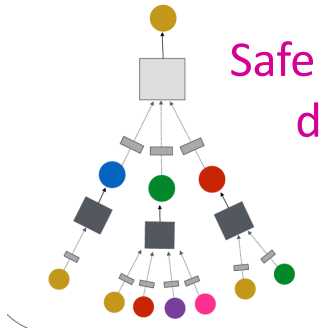
Directly train the model for a supervised task (e.g., **node classification**)

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

Encoder output: node embedding

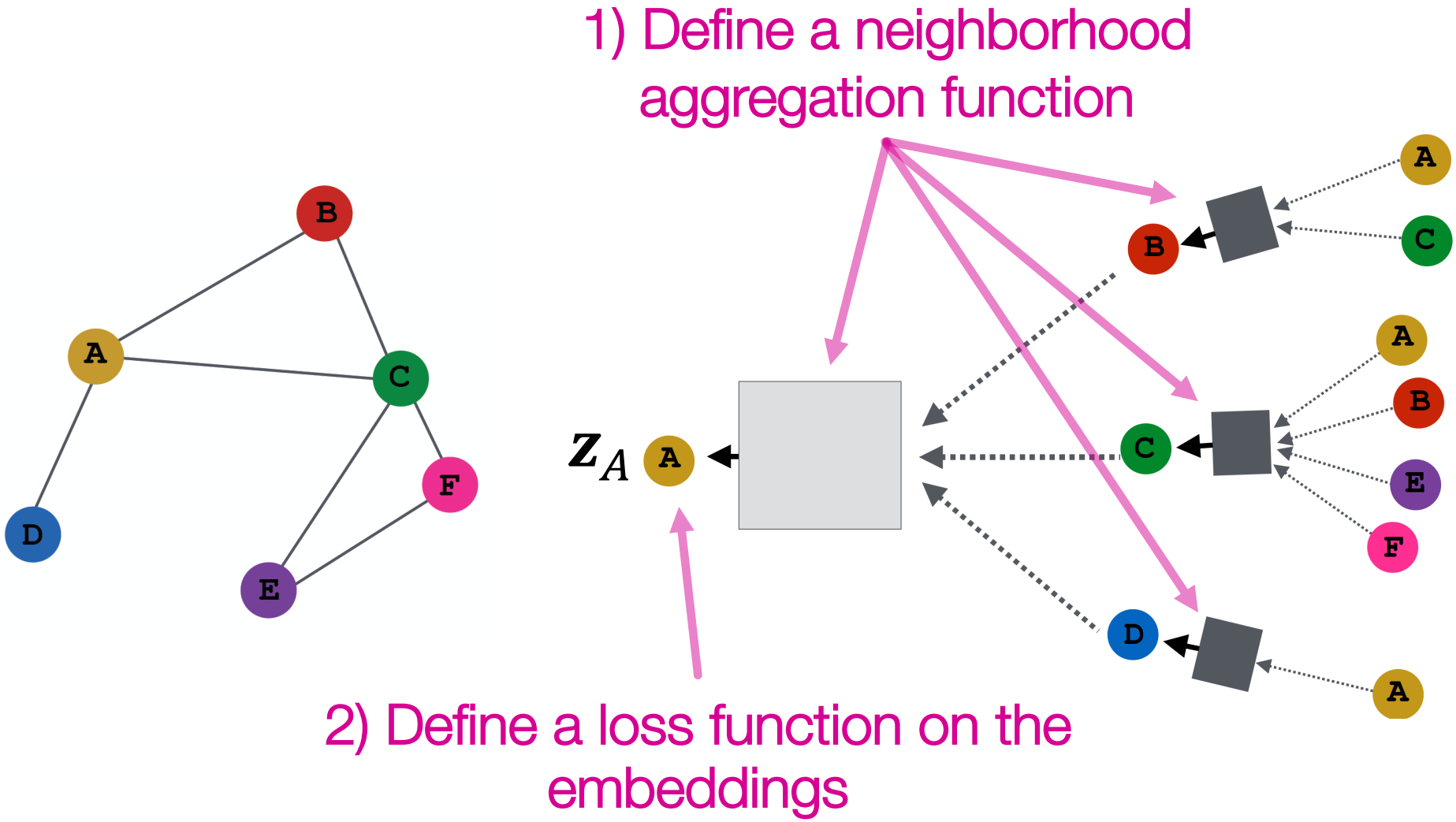
Classification weights

Node class label

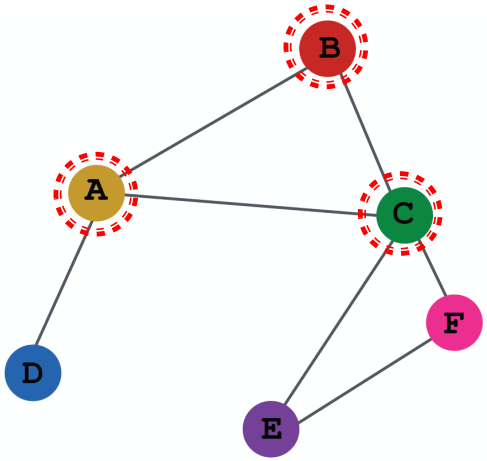


Safe or toxic drug?

Model Design: Overview

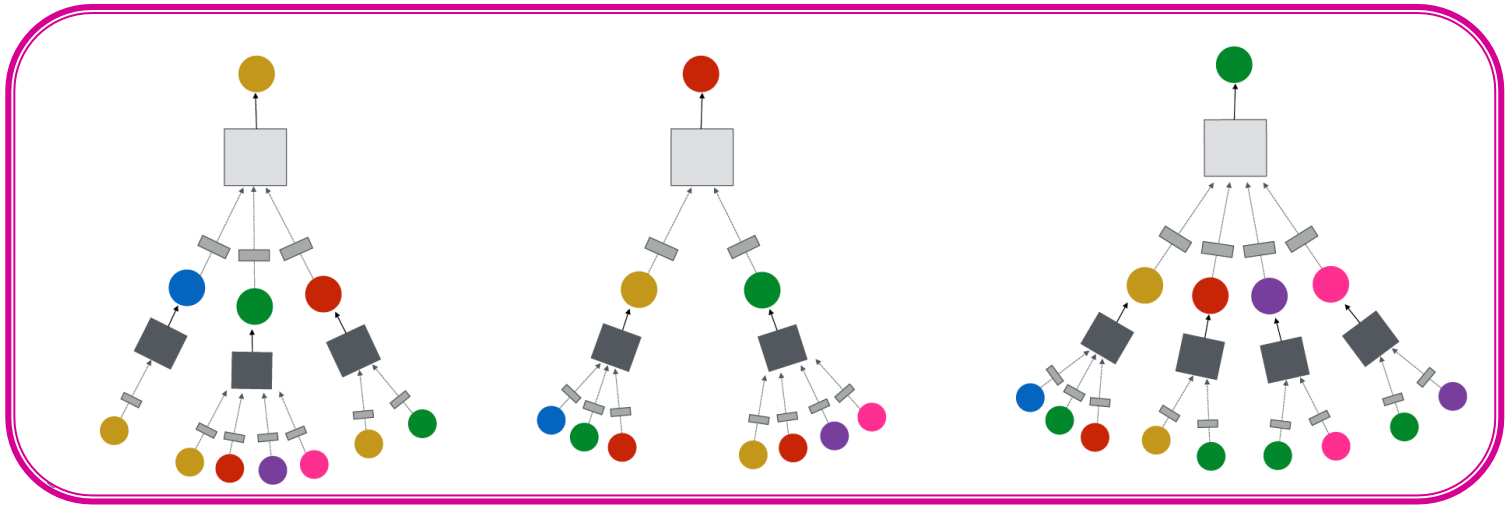


Model Design: Overview

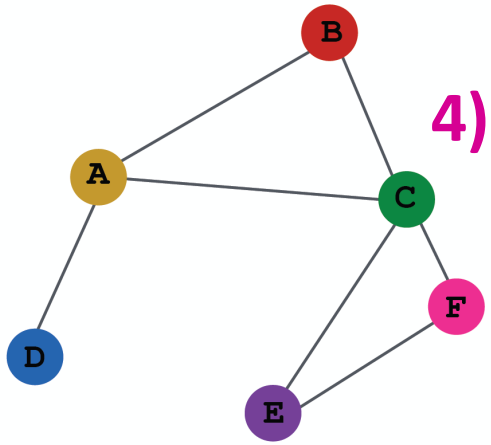


INPUT GRAPH

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



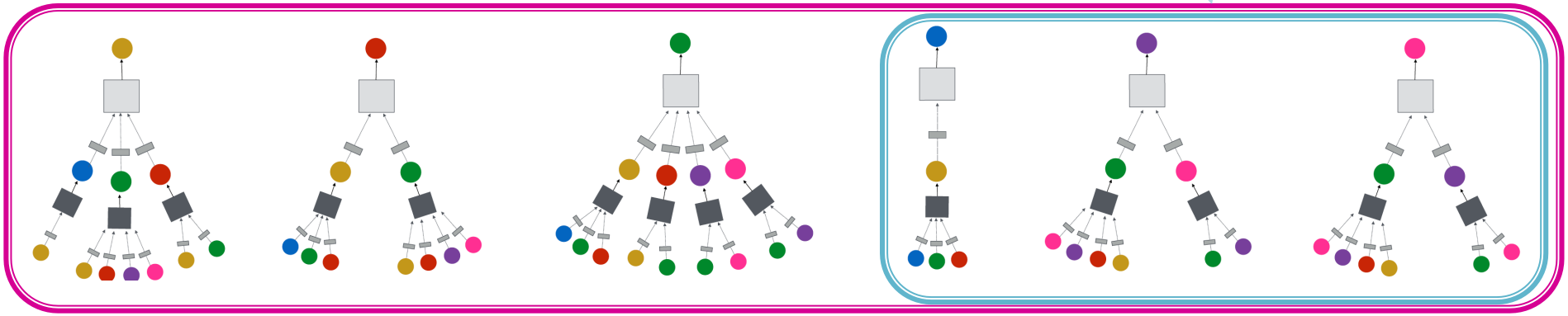
Model Design: Overview



INPUT GRAPH

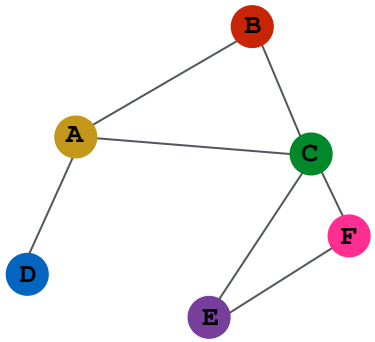
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

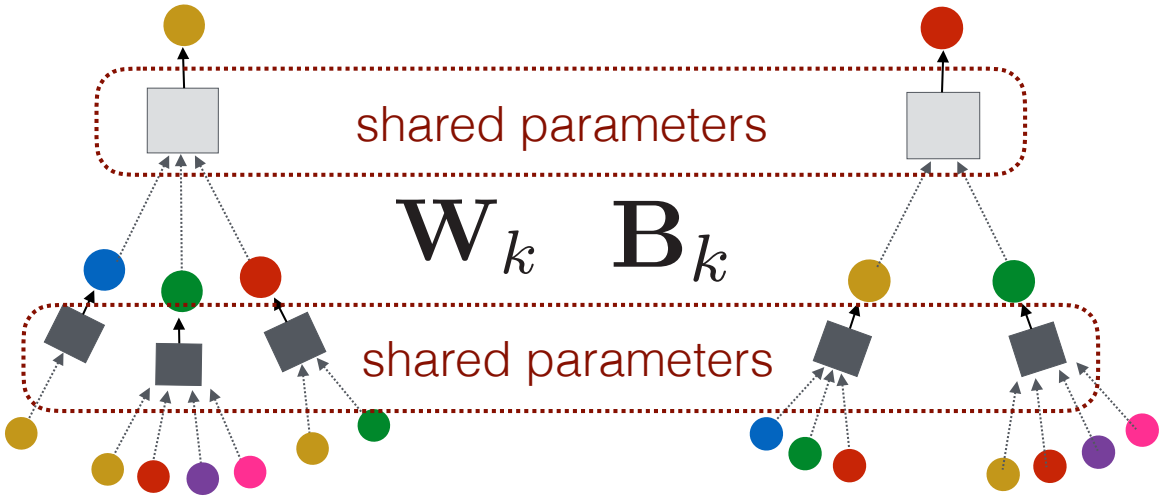


Inductive Capability

- 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**!

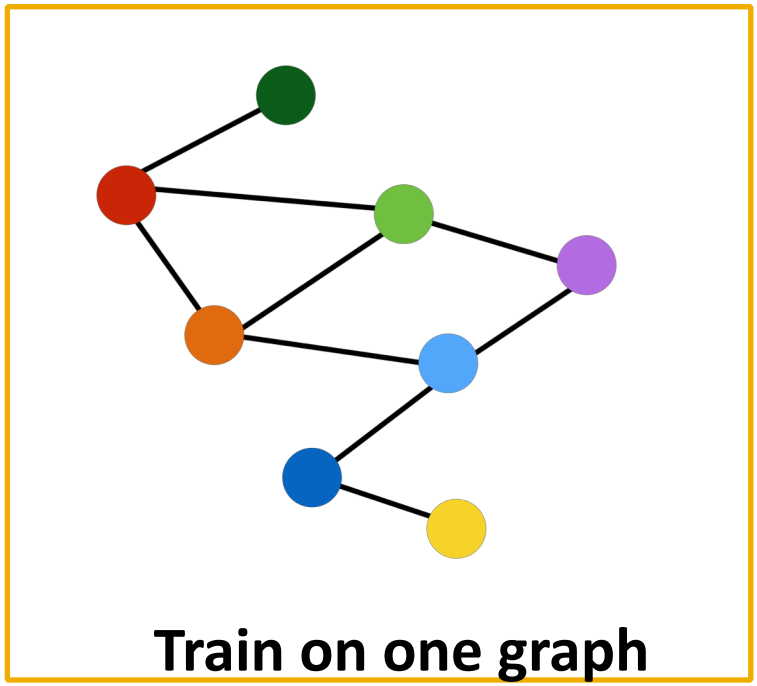


INPUT GRAPH

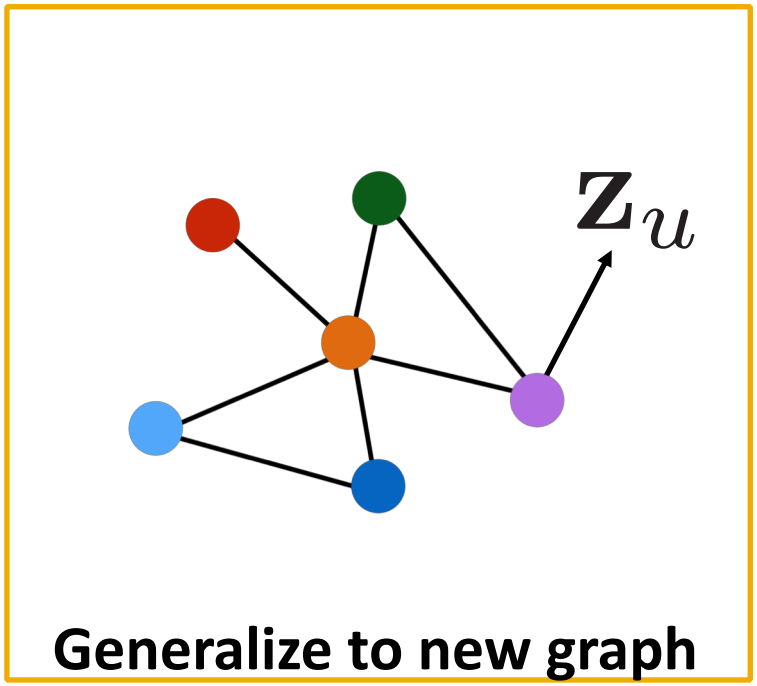


Compute graph for node A **Compute graph for node B**

Inductive Capability: New Graphs



Train on one graph

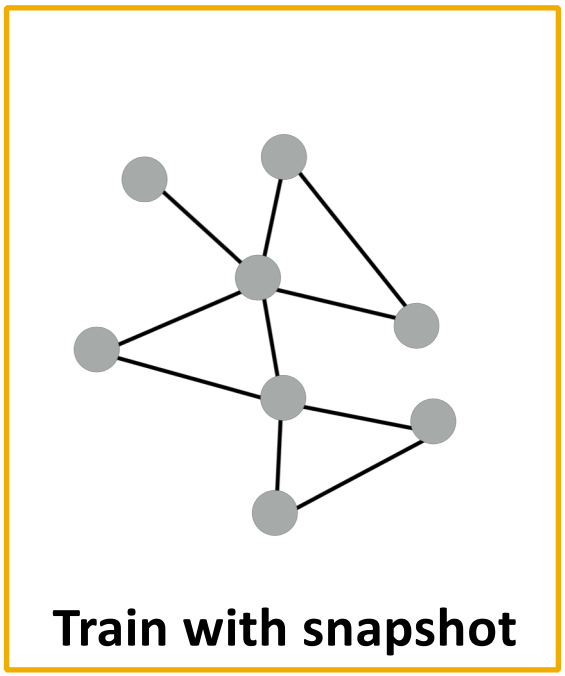


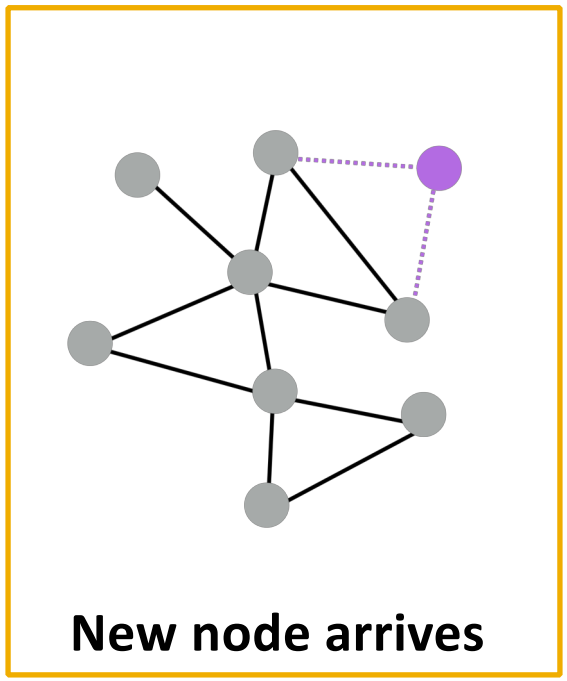
Generalize to new graph

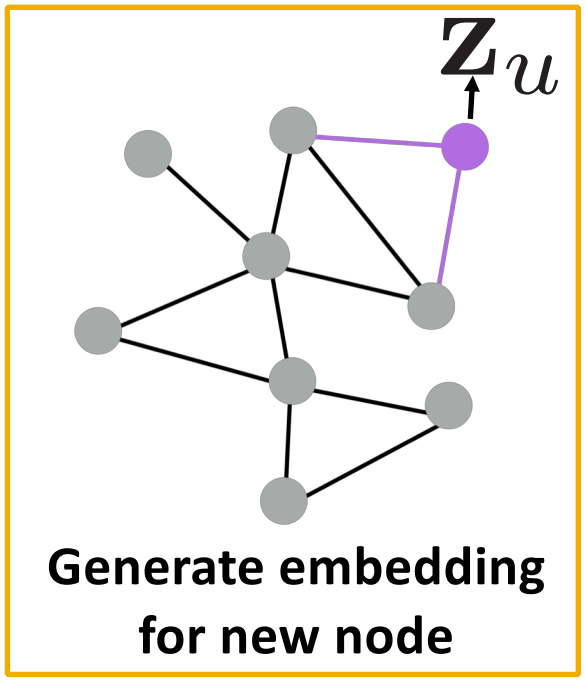
Inductive node embedding → Generalize to entirely unseen graphs

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

Inductive Capability: New Nodes







- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

Summary So Far

- **Recap:** Generate node embeddings by aggregating neighborhood information
  - We saw a **basic variant of this idea**
  - Key distinctions are in how different approaches aggregate information across the layers

- **Next:** Describe GraphSAGE graph neural network

Outline of Today's Lecture

1. **Basics of deep learning for graphs** ✓
2. **Graph Convolutional Networks**
3. **Graph Attention Networks (GAT)**
4. **Practical tips and demos**

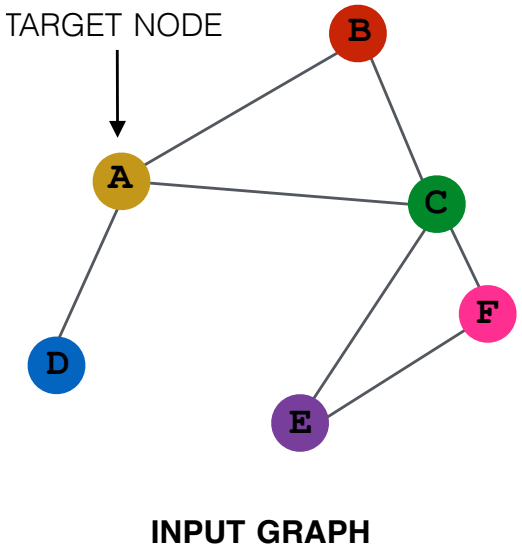
Graph Convolutional Networks and GraphSAGE

GraphSAGE Idea

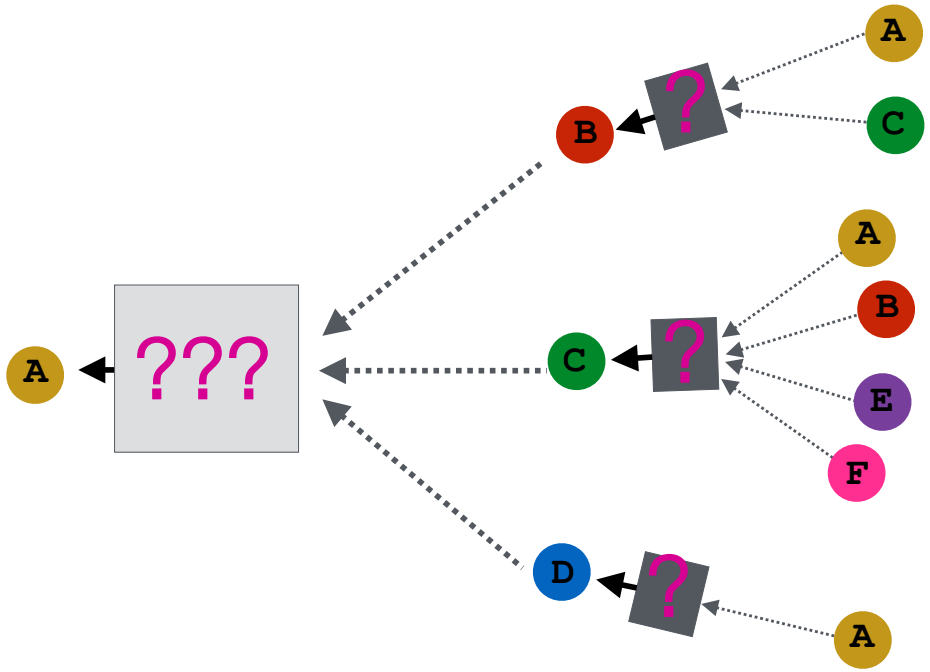
[Hamilton et al., NIPS 2017]

So far we have aggregated the neighbor messages by taking their (weighted) average

Can we do better?

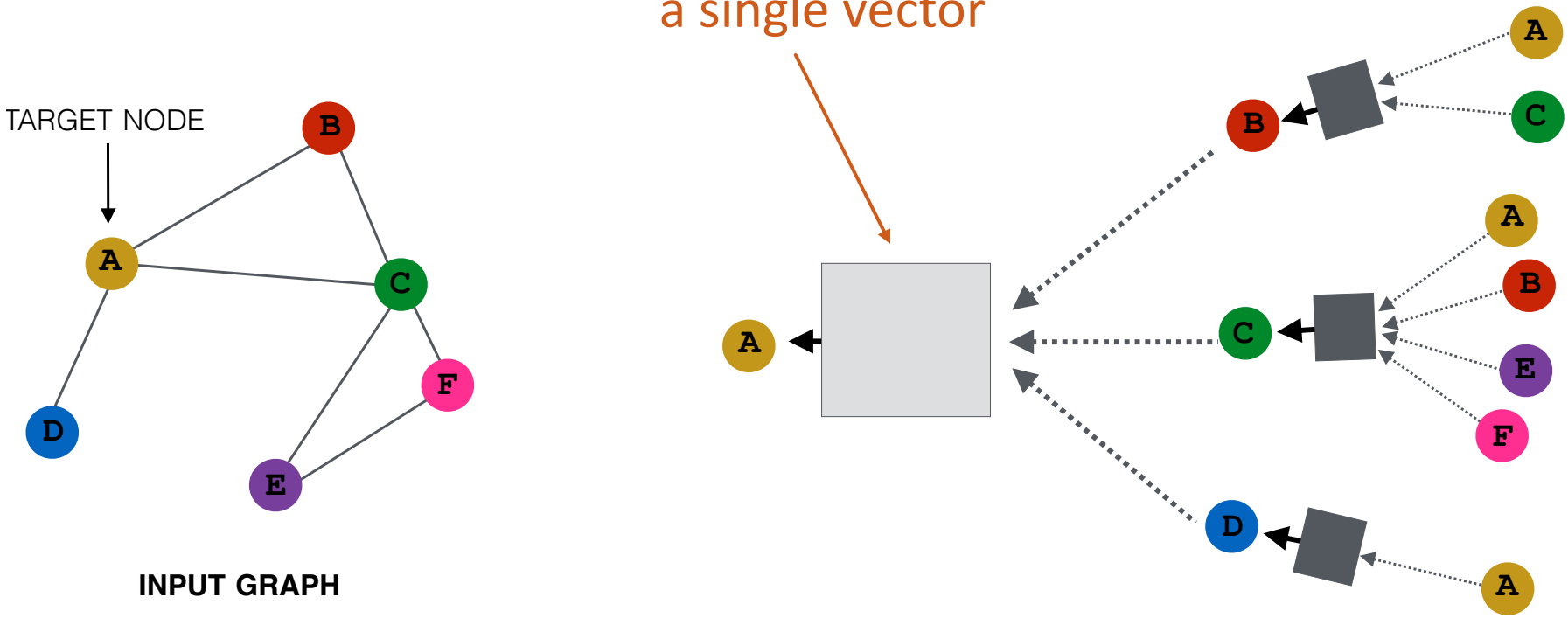


INPUT GRAPH



GraphSAGE Idea

Any differentiable function that maps set of vectors in $N(u)$ to a single vector



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

Neighborhood Aggregation

- **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

Neighbor Aggregation: Variants

- **Mean:** Take a weighted average of neighbors

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

- **Pool:** Transform neighbor vectors and apply symmetric vector function

Element-wise mean/max

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

- **LSTM:** Apply LSTM to reshuffled of neighbors

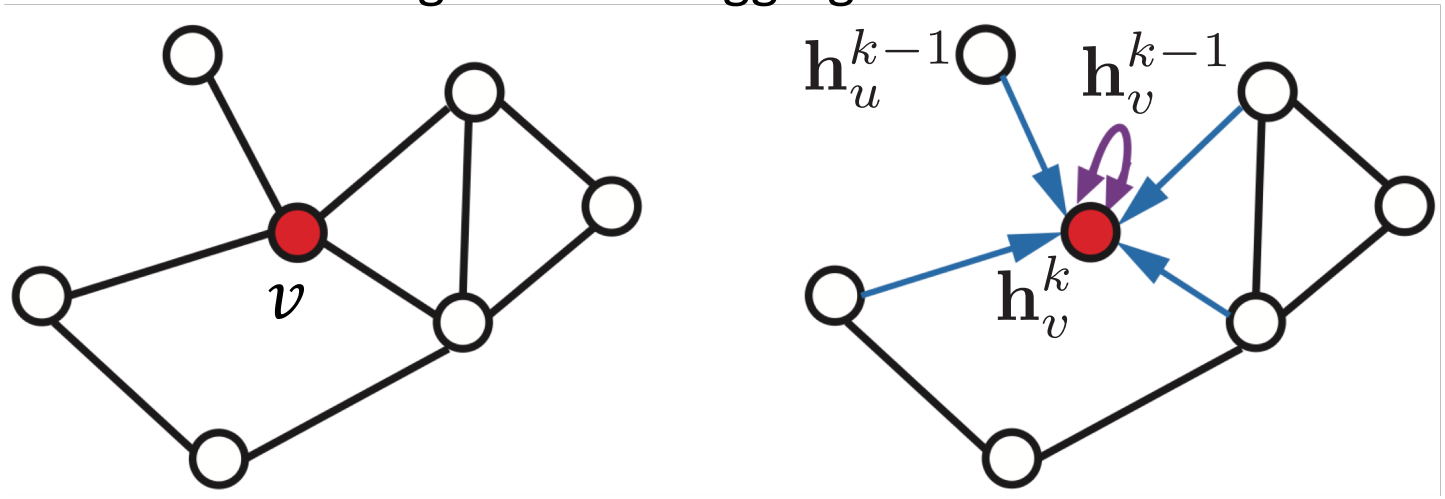
$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

Recap

Key idea: Generate node embeddings based on local neighborhoods

- Nodes aggregate "messages" from their neighbors using neural networks

- **Graph convolutional networks:**
  - **Basic variant:** Average neighborhood information and stack neural networks.
- **GraphSAGE:**
  - Generalized neighborhood aggregation



More on Graph Neural Networks

Relational inductive biases and graph networks (Battaglia, 2018)

Attention-based neighborhood aggregation:

- Graph attention networks (Hoshen, 2017; Velickovic et al., 2018; Liu et al., 2018)

Embedding entire graphs:

- Graph neural nets with edge embeddings (Battaglia et al., 2016; Gilmer et. al., 2017)
- Embedding entire graphs (Duvenaud et al., 2015; Dai et al., 2016; Li et al., 2018) and graph pooling (Ying et al., 2018)
- Graph generation and relational inference (You et al., 2018; Kipf et al., 2018

Spectral approaches to graph neural networks:

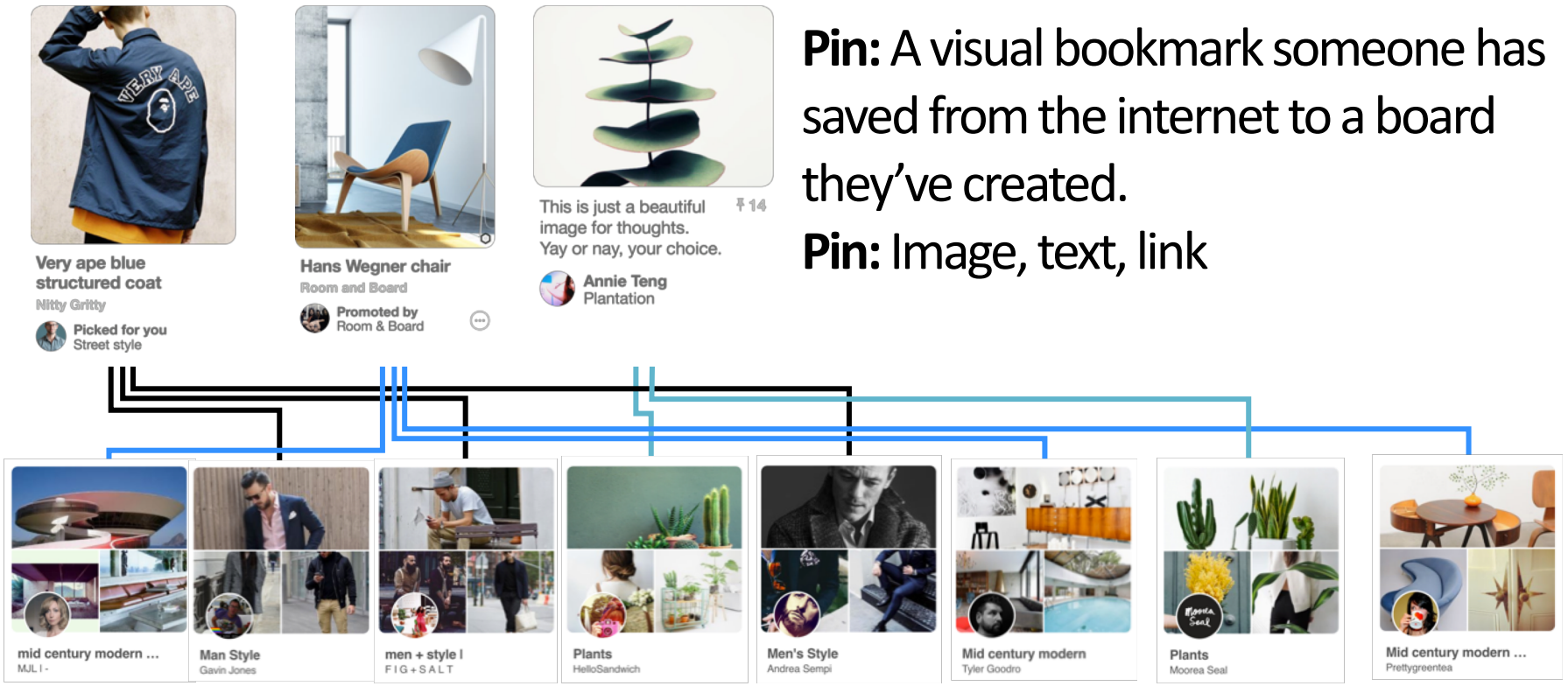
- Spectral graph CNN & ChebNet (Bruna et al., 2015; Defferrard et al., 2016)
- Geometric deep learning (Bronstein et al., 2017; Monti et al., 2017)

Hyperbolic geometry and hierarchical embeddings:

- Poincare embeddings and hierarchical relations (Nickel et al., 2017; Nickel et al., 2018)
- Graph representation learning tradeoffs (De Sa et al., 2018)

Application: Pinterest

Human curated collection of pins

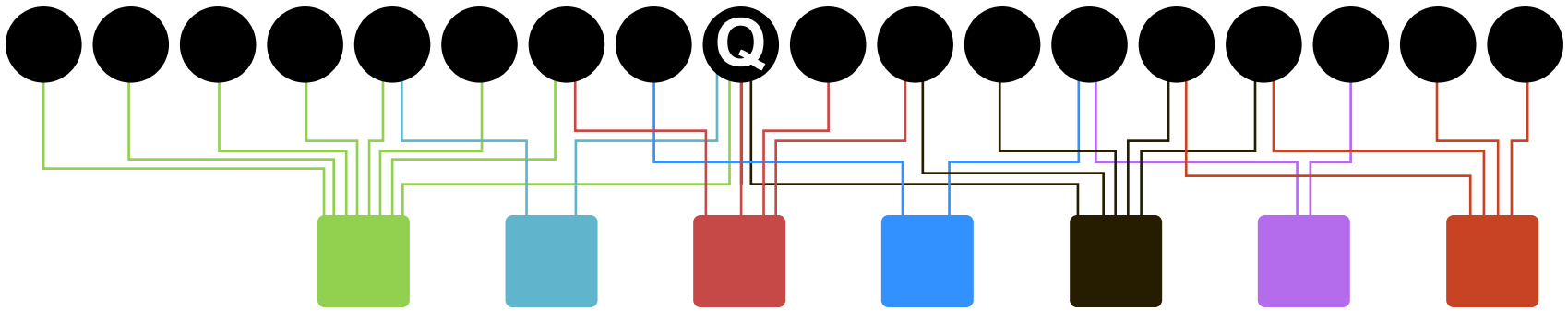


Pin: A visual bookmark someone has saved from the internet to a board they've created.

Pin: Image, text, link

Board: A collection of ideas (pins having something in common)

Pinterest Graph



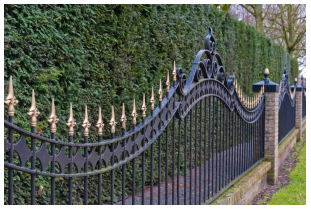
Graph: 2B pins, 1B boards, 20B edges

- **Graph is dynamic:** Need to apply to new nodes without model retraining
- **Rich node features:** Content, images

PinSage: Overview

[Ying et al., WWW 2018]

- **PinSage** graph convolutional network:
  - **Goal:** Generate embeddings for nodes (e.g., Pins/images) in a web-scale Pinterest graph containing billions of objects
  - **Key Idea:** Borrow information from nearby nodes
    - E.g., bed rail Pin might look like a garden fence, but gates and beds are rarely adjacent in the graph



    - Pin embeddings are essential to various tasks like recommendation of Pins, classification, clustering, ranking
      - Services like “Related Pins”, “Search”, “Shopping”, “Ads”

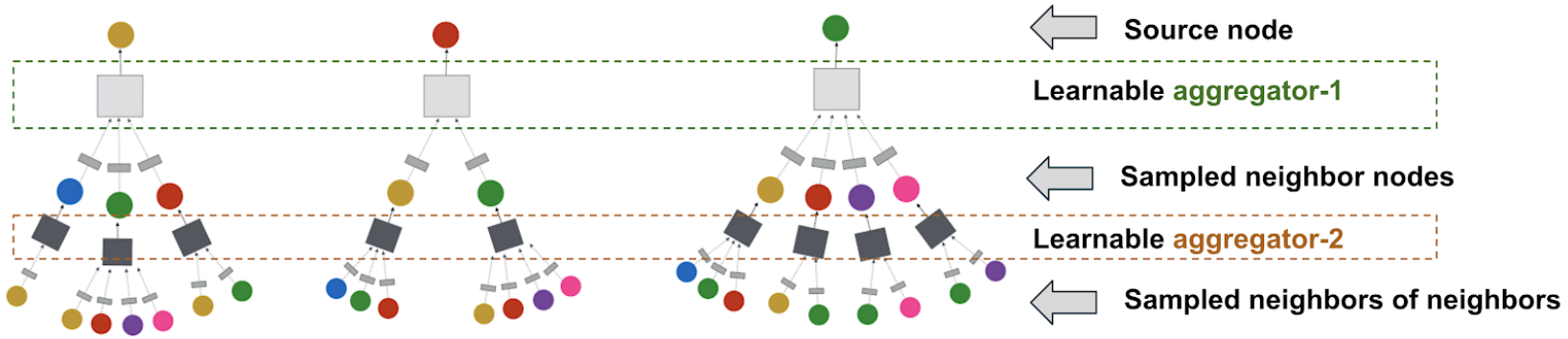
PinSage: Key Challenge

How to scale the training as well as inference of node embeddings to graphs with billions of nodes and tens of billions of edges?

- Scaling up is difficult:
  - Existing GCN-based recommender systems required the **full graph Laplacian during training**
    - Infeasible when the underlying graph has billions of nodes and whose structure is constantly evolving

PinSage: Key Innovations (1)

- Three key innovations:
  1. **On-the-fly graph convolutions**
     - Sample the neighborhood around a node and dynamically construct **a computation graph**
     - Perform a **localized graph convolution** around a particular node
     - Does not need the entire graph during training



PinSage: Key Innovations (2)

- Three key innovations:
  1. **On-the-fly graph convolutions**
  2. **Constructing convolutions via random walks**
     - Performing convolutions on full neighborhoods is infeasible:
       - How to select the set of neighbors of a node to convolve over?
     - **Importance pooling:** Define importance-based neighborhoods by simulating random walks and selecting the neighbors with the highest visit counts
  3. **Efficient MapReduce inference**
     - Bottom-up aggregation of node embeddings lends itself to MapReduce
       - Decompose each aggregation step across all nodes into three operations in MapReduce, i.e., *map*, *join*, and *reduce*

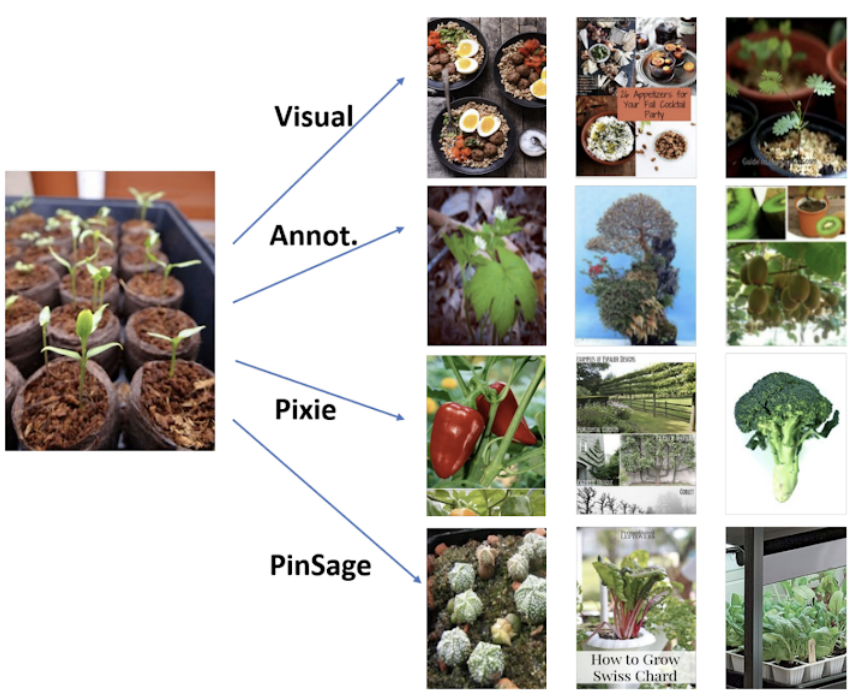
PinSage: Experiments

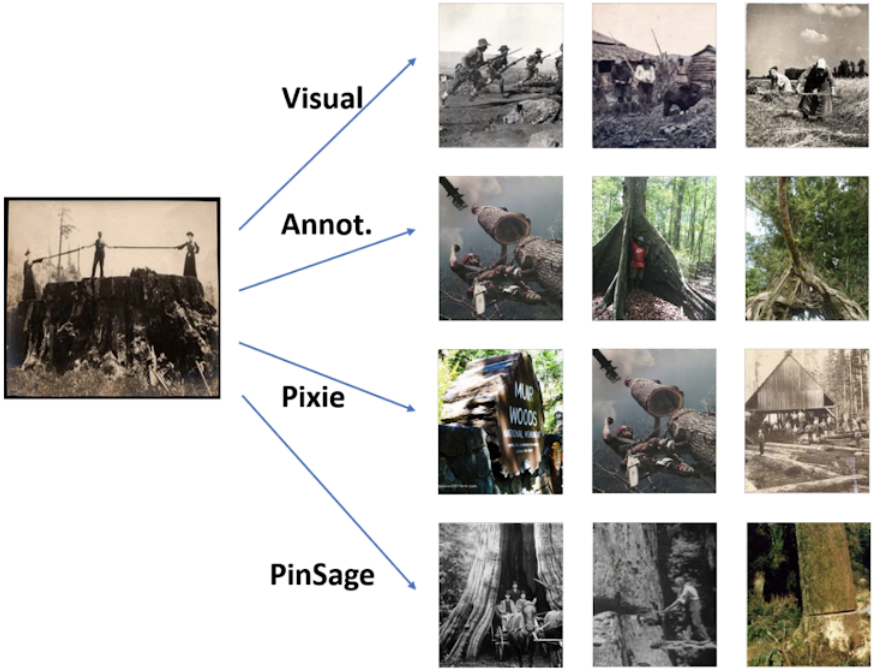
- Baselines:
  - **Visual**: Nearest neighbors of CNN visual embeddings for recommendations
  - **Annotation**: Nearest neighbors in terms of Word2vec embeddings
  - **Combined**: Concatenate embeddings:
    - Uses exact same data and loss function as PinSage

Method	Hit-rate	MRR
Visual	17%	0.23
Annotation	14%	0.19
Combined	27%	0.37
max-pooling	39%	0.37
mean-pooling	41%	0.51
mean-pooling-xent	29%	0.35
mean-pooling-hard	46%	0.56
PinSage	**67%**	**0.59**

PinSage gives 150% improvement in hit rate and 60% improvement in MRR over the best baseline

Example Pin Recommendations





Pixie is a purely graph-based method that uses biased random walks to generate ranking scores by simulating random walks starting at query Pin. Items with top scores are retrieved as recommendations [Eksombatchai et al., 2018]

Outline of Today's Lecture

1. **Basics of deep learning for graphs** ✓
2. **Graph Convolutional Networks** ✓
3. **Graph Attention Networks (GAT)** 
4. **Practical tips and demos**

Graph Attention Networks

Simple Neighborhood Aggregation

- **Recap: Simple neighborhood aggregation:**

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

- Graph convolutional operator:
  - Aggregates messages across neighborhoods, $N(v)$
  - $\alpha_{vu} = 1/|N(v)|$ is the **weighting factor (importance)** of node u's message to node v
  - $\Longrightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

[Velickovic et al., ICLR 2018; Vaswani et al., NIPS 2017]

Graph Attention Networks

Can we do better than simple neighborhood aggregation?

Can we let weighting factors α_{vu} to be implicitly defined?

- **Goal:** Specify **arbitrary importances** to different neighbors of each node in the graph
- **Idea:** Compute embedding $\boldsymbol{h}_v^k$ of each node in the graph following an **attention strategy:**
  - Nodes attend over their neighborhoods' message
  - Implicitly specifying different weights to different nodes in a neighborhood

Attention Mechanism (1)

- Let α_{vu} be computed as a byproduct of an **attention mechanism a:**
  - Let a compute **attention coefficients e_{vu}** across pairs of nodes u, v based on their messages:

$$e_{vu} = a(\boldsymbol{W}_k \boldsymbol{h}_u^{k-1}, \boldsymbol{W}_k \boldsymbol{h}_v^{k-1})$$

    - **e_{vu} indicates the importance of node u's message to node v**
  - **Normalize coefficients** using the softmax function in order to be comparable across different neighborhoods:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

$$\boldsymbol{h}_v^k = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \boldsymbol{W}_k \boldsymbol{h}_u^{k-1}\right)$$

Next: What is the form of attention mechanism a?

Attention Mechanism (2)

- Attention mechanism a:
  - The approach is agnostic to the choice of a
    - E.g., use a simple single-layer neural network
    - a can have parameters, which need to be estimates
  - Parameters of a are trained jointly:
    - Learn the parameters together with weight matrices (i.e., other parameter of the neural net) in an end-to-end fashion

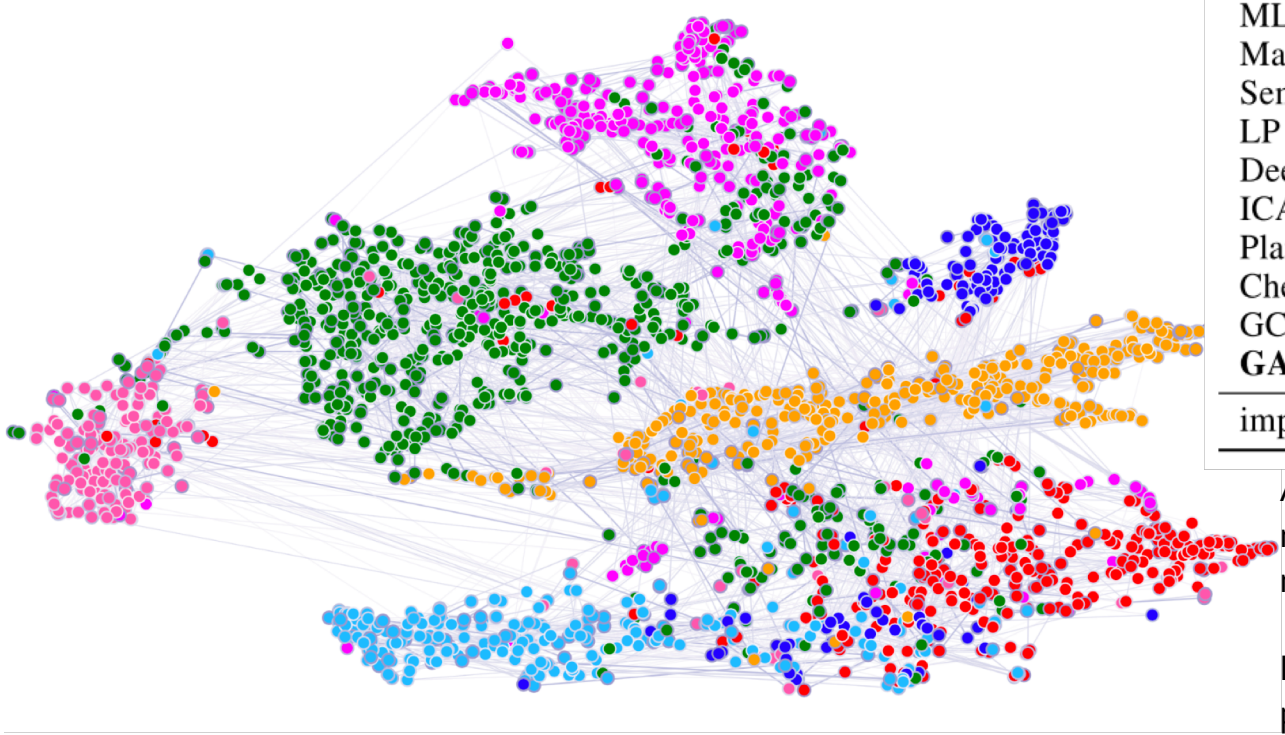
- **Multi-head attention:** Stabilize the learning process of attention mechanism [Velickovic et al., ICLR 2018]:
  - Attention operations in a given layer are independently replicated R times (each replica with different parameters)
  - Outputs are aggregated (by concatenating or adding)

Properties of Attentional Mechanism

- **Key benefit:** Allows for (implicitly) specifying **different importance values (α_{vu}) to different neighbors**

- **Computationally efficient**:
  - Computation of attentional coefficients can be parallelized across all edges of the graph
  - Aggregation may be parallelized across all nodes
- **Storage efficient**:
  - Sparse matrix operations do not require more than O(V+E) entries to be stored
  - **Fixed** number of parameters, irrespective of graph size
- Trivially **localized**:
  - Only **attends over local network neighborhoods**
- **Inductive capability**:
  - It is a shared *edge-wise* mechanism
  - It does not depend on the global graph structure

GAT Example: Cora Citation Net



Method	Cora
MLP	55.1%
ManiReg (Belkin et al., 2006)	59.5%
SemiEmb (Weston et al., 2012)	59.0%
LP (Zhu et al., 2003)	68.0%
DeepWalk (Perozzi et al., 2014)	67.2%
ICA (Lu & Getoor, 2003)	75.1%
Planetoid (Yang et al., 2016)	75.7%
Chebyshev (Defferrard et al., 2016)	81.2%
GCN (Kipf & Welling, 2017)	81.5%
GAT	**83.3%**
improvement w.r.t GCN	1.8%

Attention mechanism can be used with many different graph neural network models

In many cases, attention leads to performance gains

- t-SNE plot of GAT-based node embeddings:
  - Node color: 7 publication classes
  - Edge thickness: Normalized attention coefficients between nodes i and j, across eight attention heads, $\sum_k (\alpha_{ij}^k + \alpha_{ji}^k)$

Outline of Today's Lecture

1. **Basics of deep learning for graphs** ✓
2. **Graph Convolutional Networks (GCN)** ✓
3. **Graph Attention Networks (GAT)** ✓
4. **Practical tips and demos**

General Tips and Practical Demos

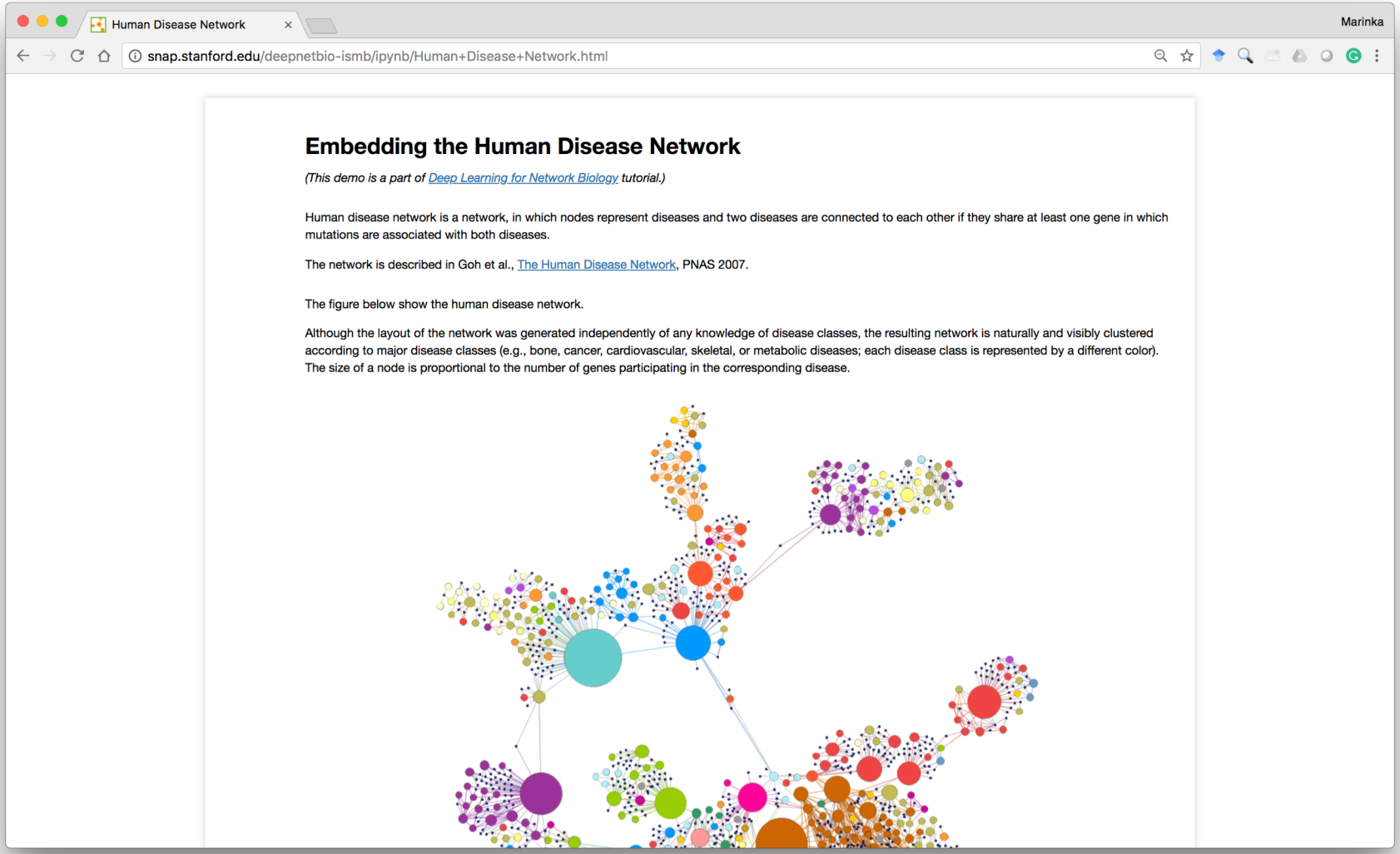
General Tips

- Data preprocessing is important:
  - Use renormalization tricks
  - Variance-scaled initialization
  - Network data whitening
- ADAM optimizer:
  - ADAM naturally takes care of decaying the learning rate
- ReLU (activation function) often works really well
- No activation function at your output layer:
  - Easy mistake if you build layers with a shared function
- Include bias term in every layer
- GCN layer of size 64 or 128 is already plenty

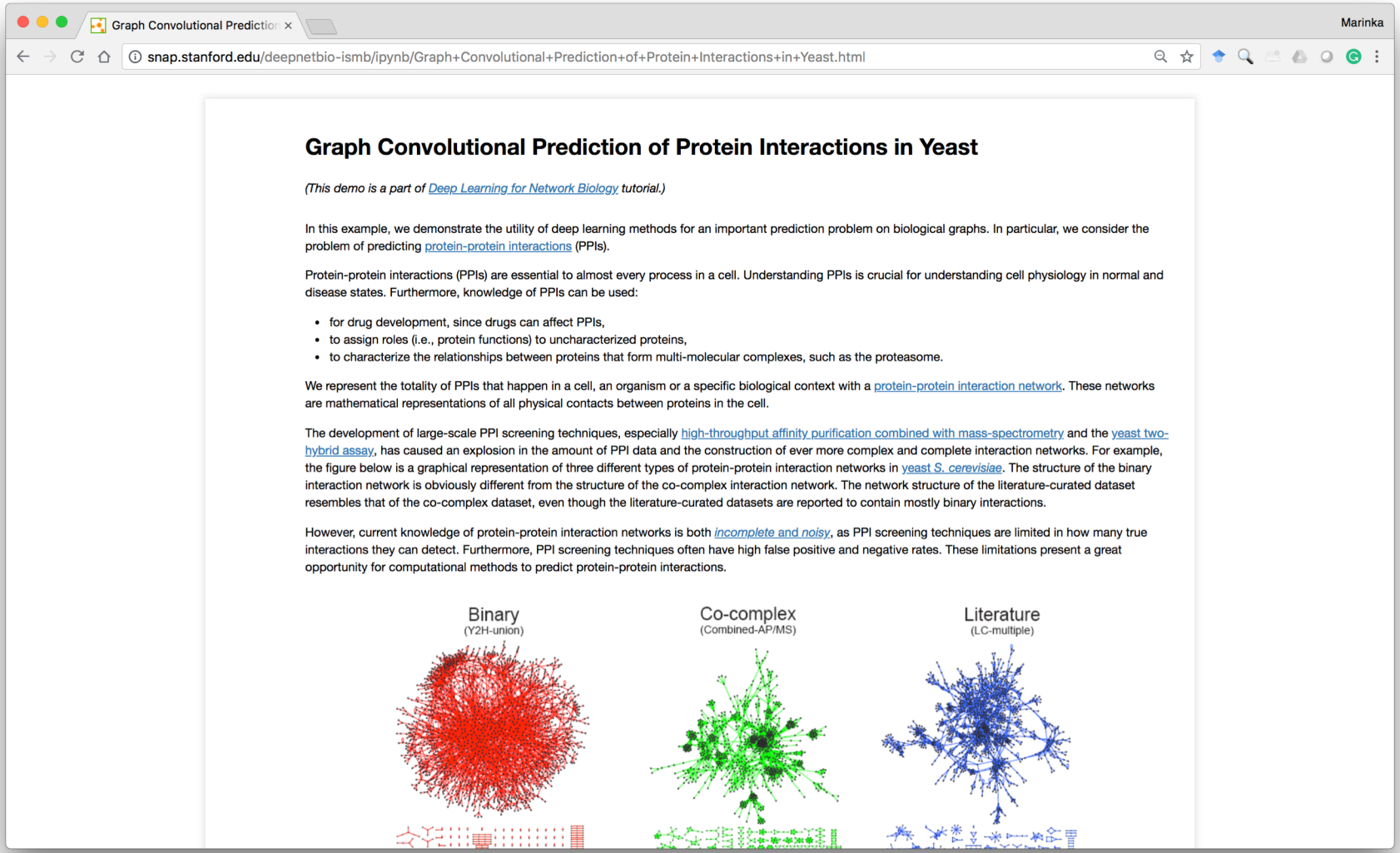
Debugging Deep Networks

- Debug?!:
  - Loss/accuracy not converging during training
- Important for model development:
  - **Overfit on training data:**
    - Accuracy should be essentially 100% or error close to 0
    - If neural network cannot overfit a single data point, something is wrong
  - **Scrutinize your loss function!**
  - **Scrutinize your visualizations!**

Demo: Human Disease Network



Demo: Protein Interaction Prediction



Outline of Today's Lecture

1. **Basics of deep learning for graphs** ✓
2. **Graph Convolutional Networks** ✓
3. **Graph Attention Networks (GAT)** ✓
4. **Practical tips and demos** ✓

What Next?

- **Project write-ups:**
  - Sun Dec 9 Midnight (11:59pm) Pacific Time:
    - 1 team member uploads PDF to Gradescope
    - See course website for more info
- **Poster session:**
  - Tue Dec 11 at 3:30pm Pacific Time
  - All groups with at least one non-SCPD member must present
  - There should be 1 person at the poster at all times
  - **Prepare a 2-minute elevator pitch of your poster**
  - More instructions to follow

What Next? Courses

- **CS246: Mining Massive Datasets (Winter 2019)**
  - Data Mining & Machine Learning for big data
    - (big==does' fit in memory/single machine), SPARK
- **CS341: Project in Data Mining (Spring 2019)**
  - Groups do a research project on big data
  - We provide interesting data, projects and access to the Amazon computing infrastructure
  - Nice way to finish up CS224W project & publish it!

In Closing…

- **You Have Done a Lot!**
- **And (hopefully) learned a lot!**
  - Answered questions and proved many interesting results
  - Implemented a number of methods
  - And are doing excellently on the class project!

Thank You for the Hard Work!!!