

Project Proposal deadline: tonight, 11:59pm

Course Notes: <https://snap-stanford.github.io/cs224w-notes/>
Help us write the course notes – we will give **generous bonuses!**

Graph Neural Networks

CS224W: Machine Learning with Graphs

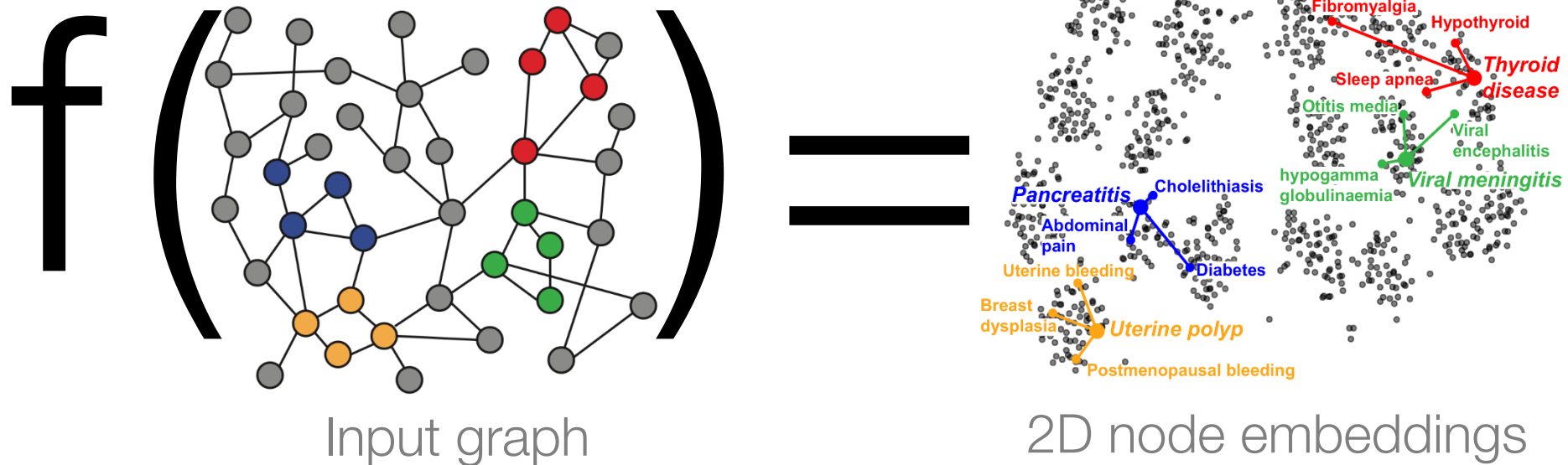
Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



Node Embeddings

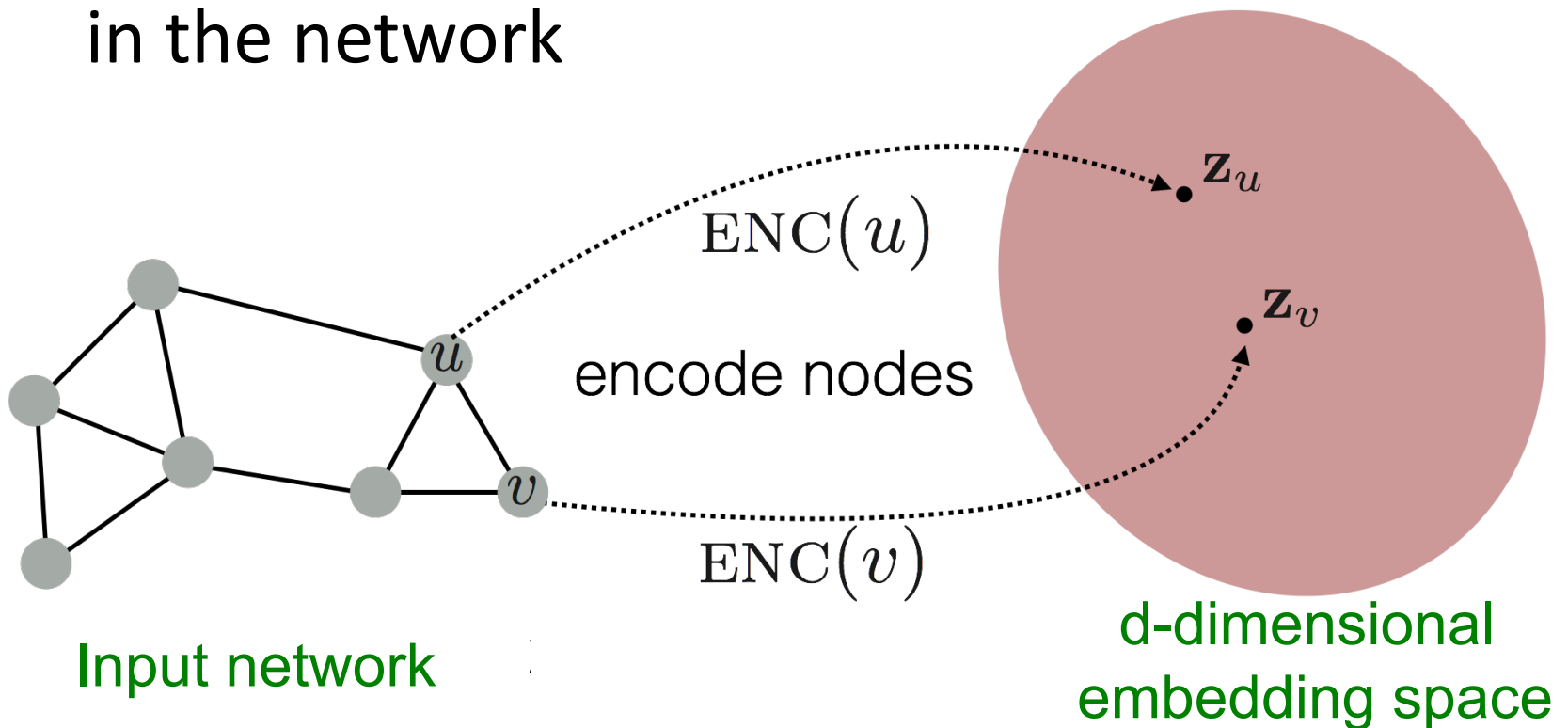
- **Intuition:** Map nodes to d -dimensional embeddings such that similar nodes in the graph are embedded close together



How to learn mapping function f ?

Node Embeddings

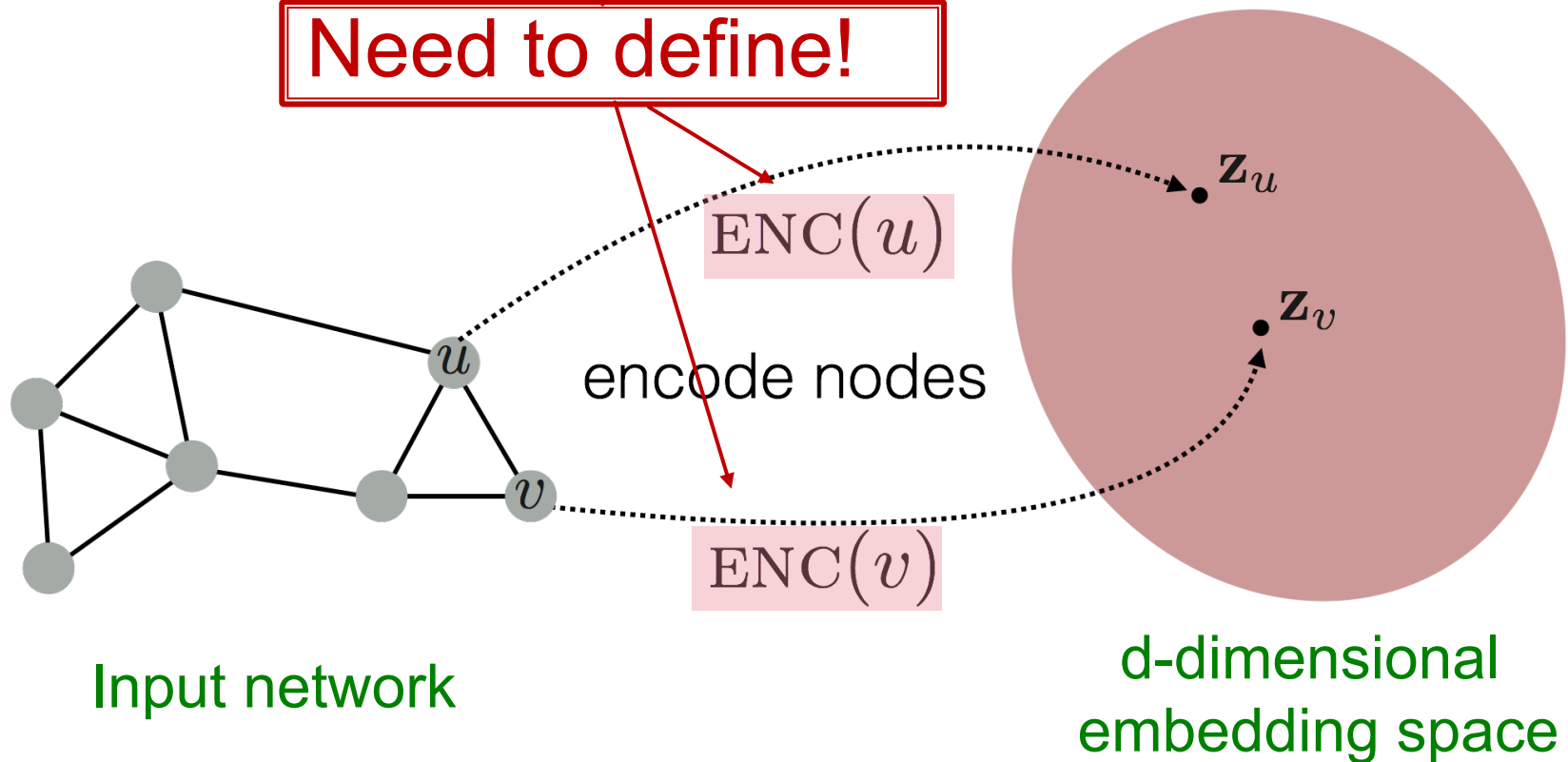
- **Goal:** Map nodes so that similarity in the embedding space (e.g., dot product) approximates similarity (e.g., proximity) in the network



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:

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

node in the input graph

d-dimensional embedding

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

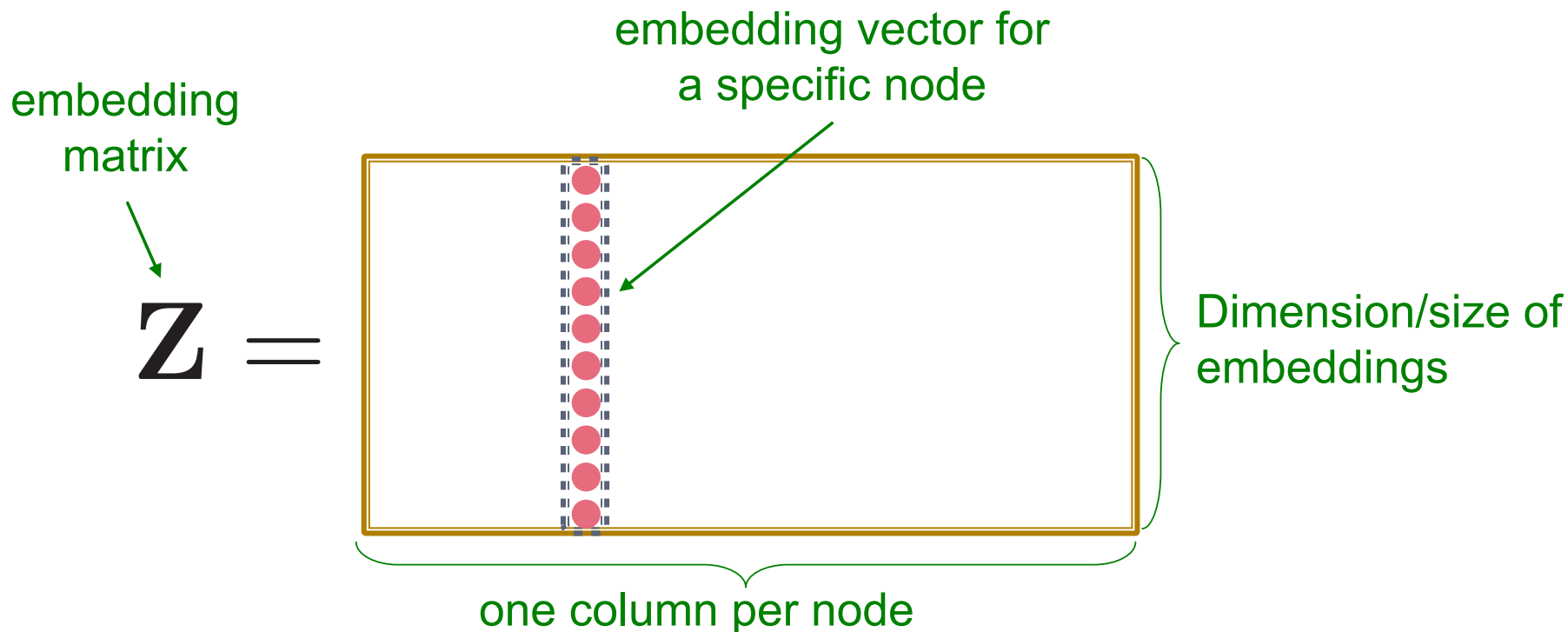
$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

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:

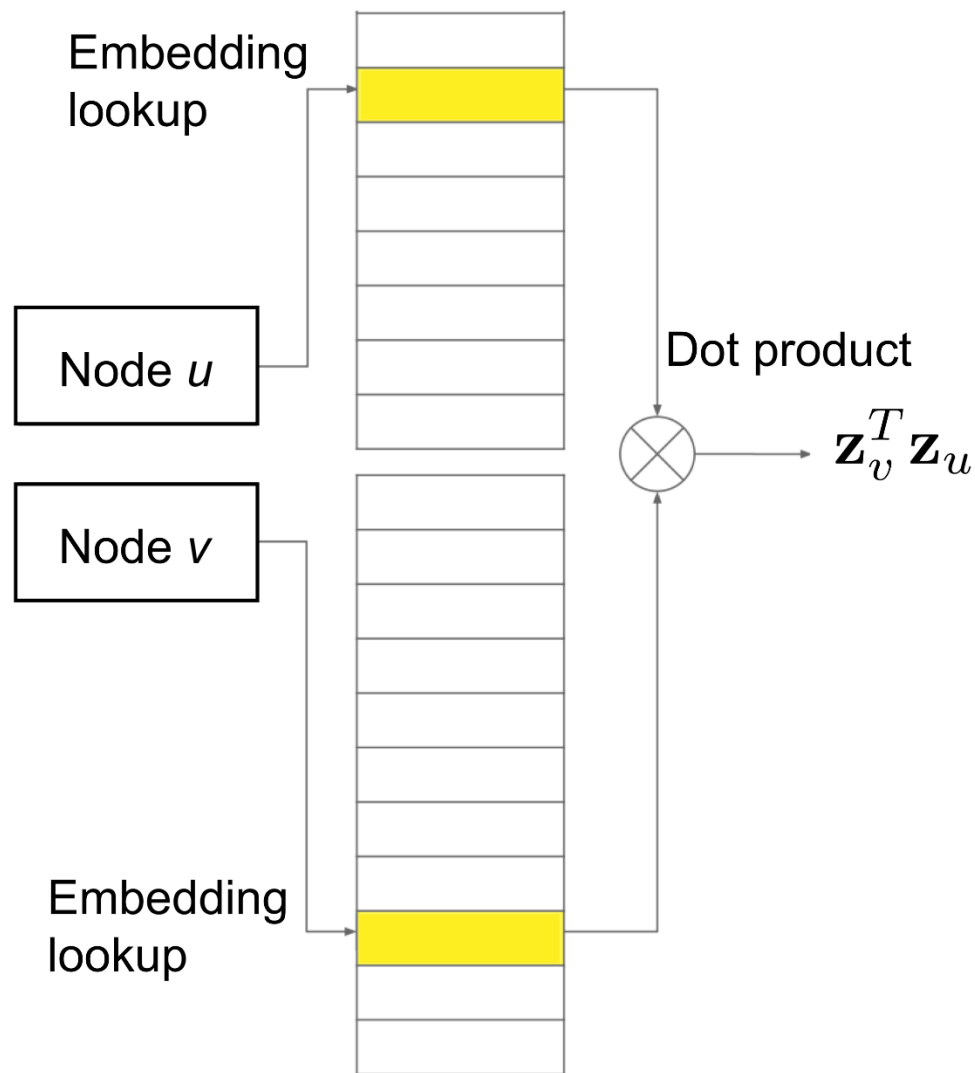


Shallow Encoders (Lec. 09: 10/23)

Shallow encoders:

- One-layer of data transformation
- A single hidden layer maps **node u** to **embedding \mathbf{z}_u** via function $f()$, e.g.

$$\mathbf{z}_u = f(\mathbf{z}_v, v \in N_R(u))$$



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

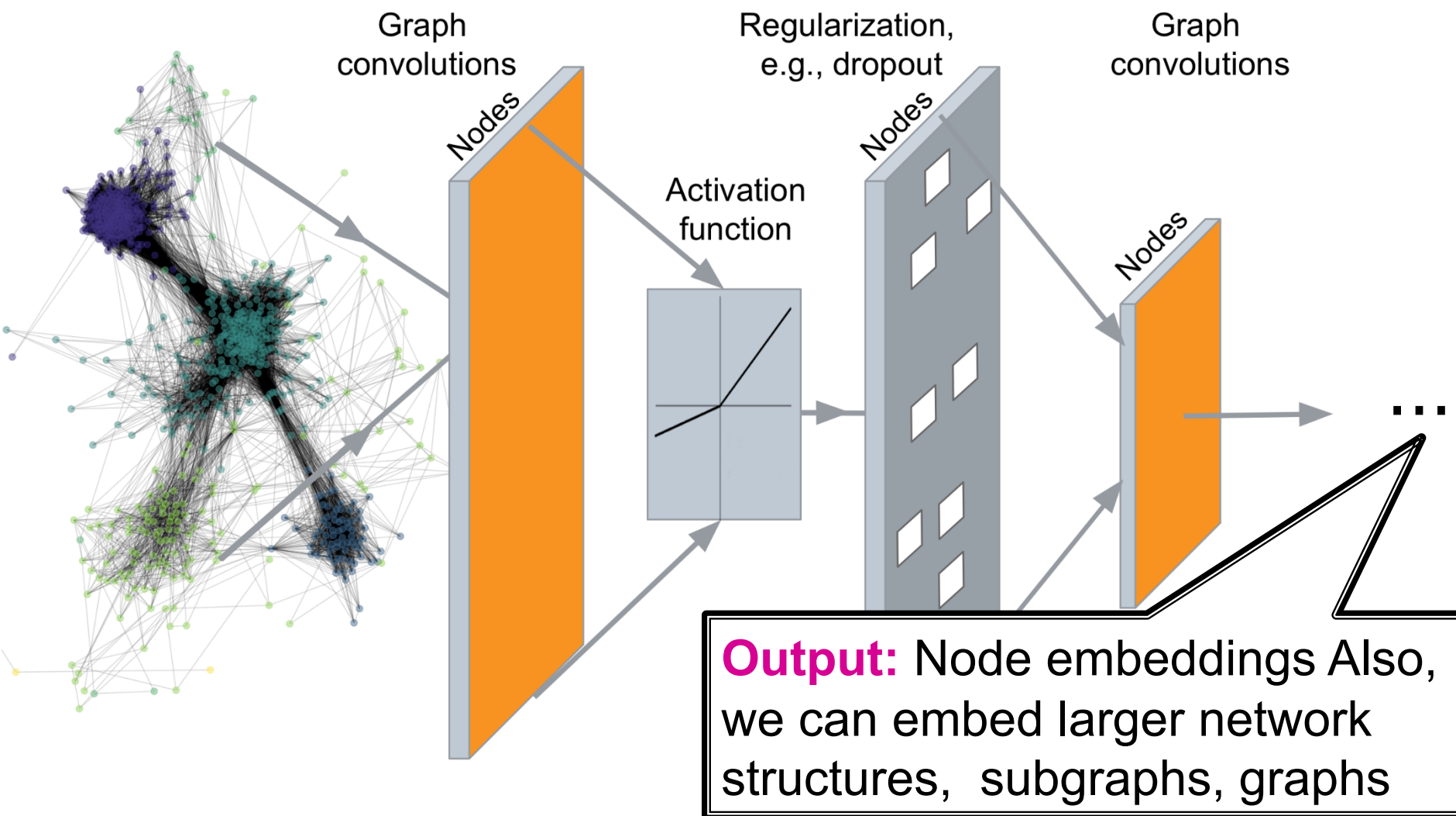
Today: Deep Graph Encoders

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

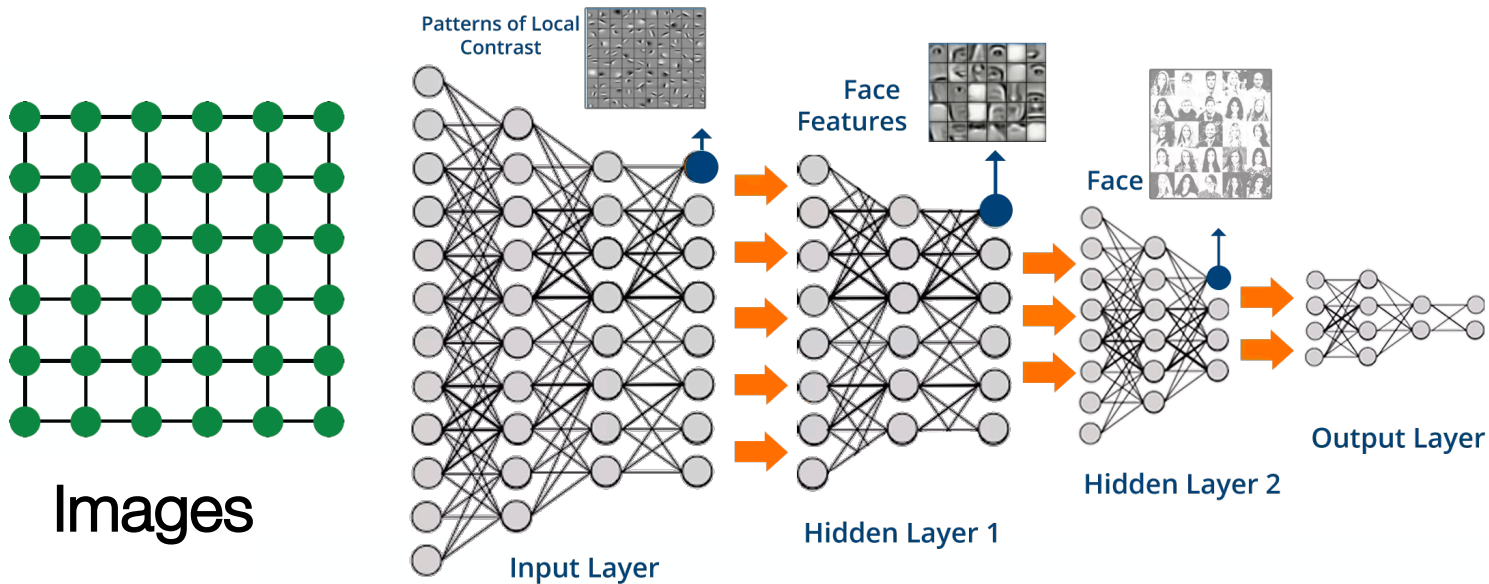
$\text{ENC}(v) =$ **multiple layers of non-linear transformations of graph structure**

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

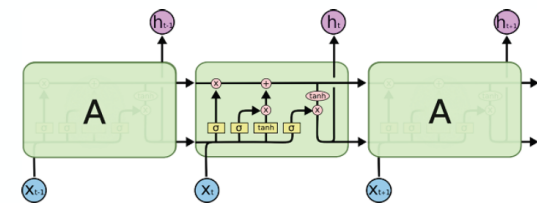
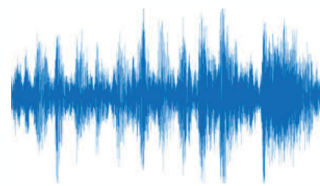
Deep Graph Encoders



Modern ML Toolbox



Text/Speech

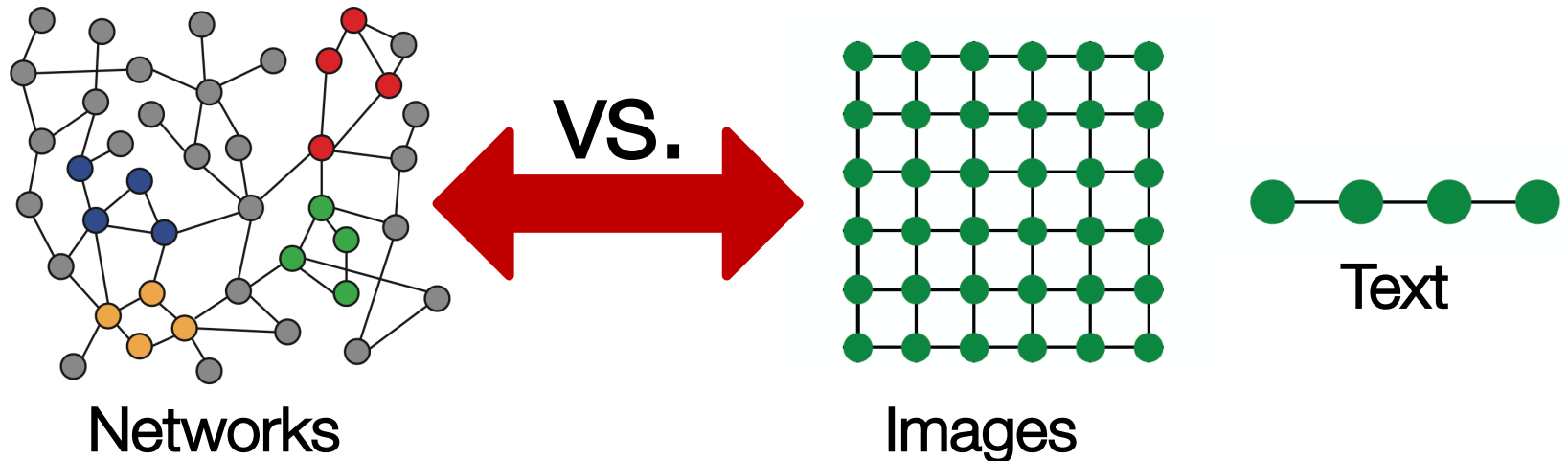


Modern deep learning toolbox is designed for simple sequences & grids

Why is it Hard?

But networks are far more complex!

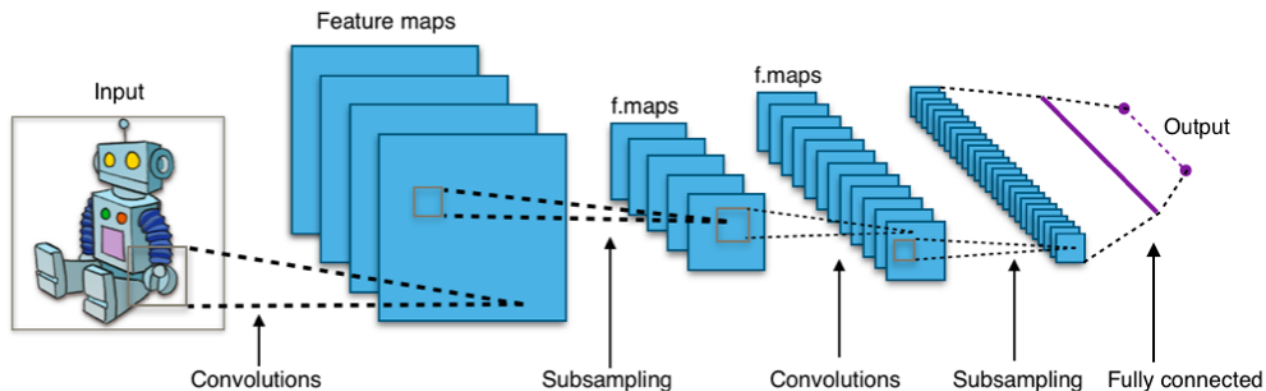
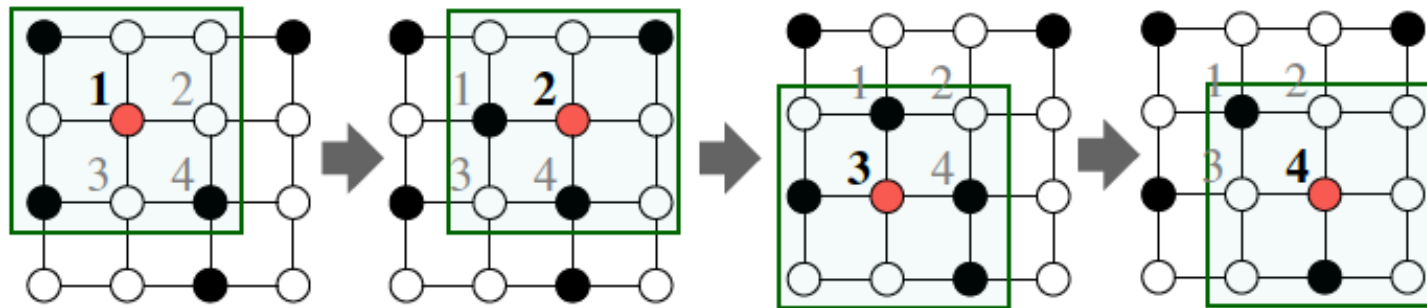
- Arbitrary size and complex topological structure (i.e., no spatial locality like grids)



- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Idea: Convolutional Networks

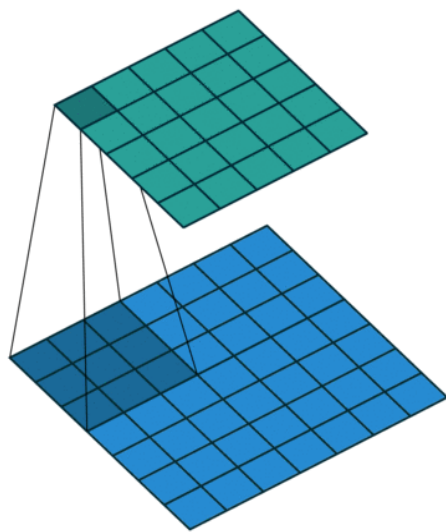
CNN on an image:



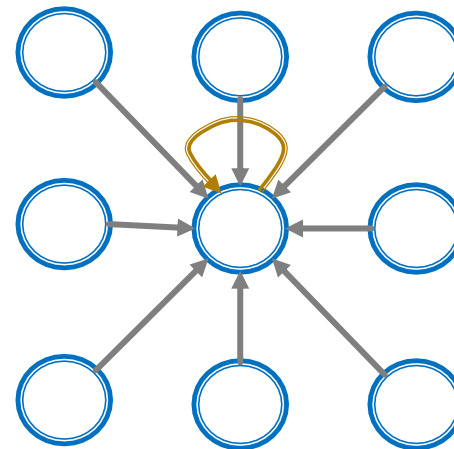
Goal is to generalize convolutions beyond simple lattices
Leverage node features/attributes (e.g., text, images)

From Images to Graphs

Single CNN layer with 3x3 filter:



Image



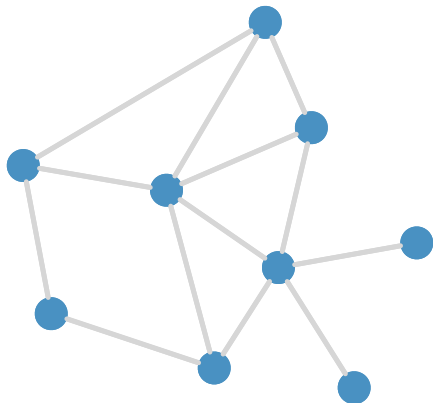
Graph

Transform information at the neighbors and combine it:

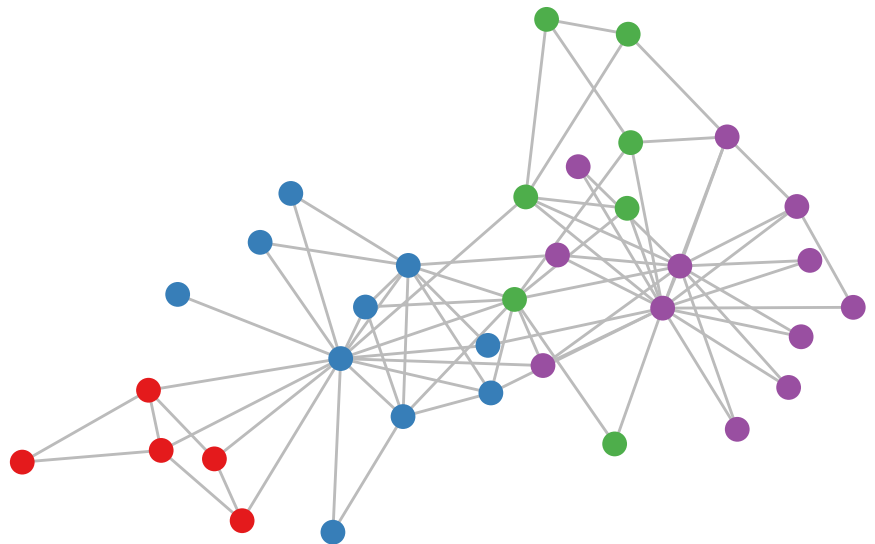
- Transform “messages” h_i from neighbors: $W_i h_i$
- Add them up: $\sum_i W_i h_i$

Real-World Graphs

But what if your graphs look like this?



or this:

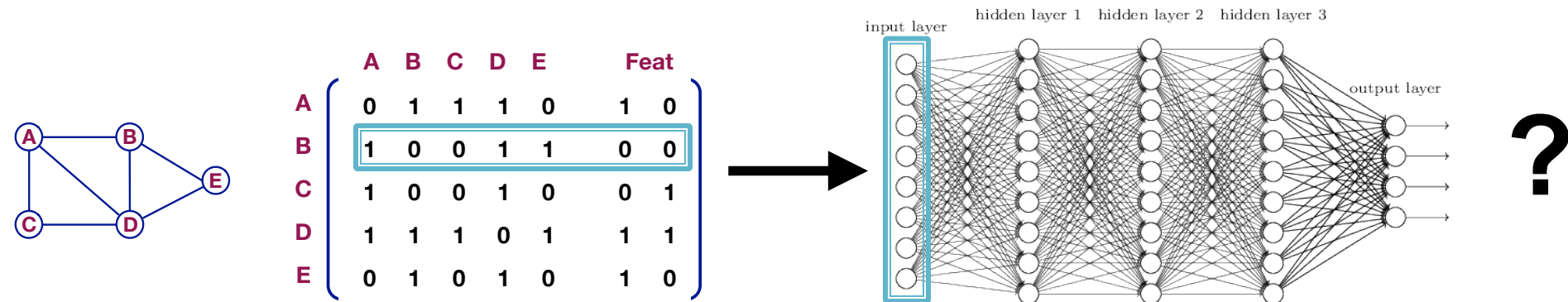


- Examples:

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:



- **Issues with this idea:**
 - $O(N)$ 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

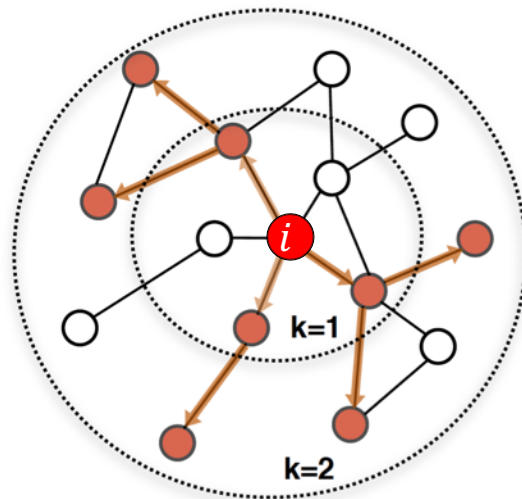
- **Local network neighborhoods:**
 - Describe aggregation strategies
 - Define computation graphs
- **Stacking multiple layers:**
 - Describe the model, parameters, training
 - How to fit the model?
 - Simple example for unsupervised and supervised training

Setup

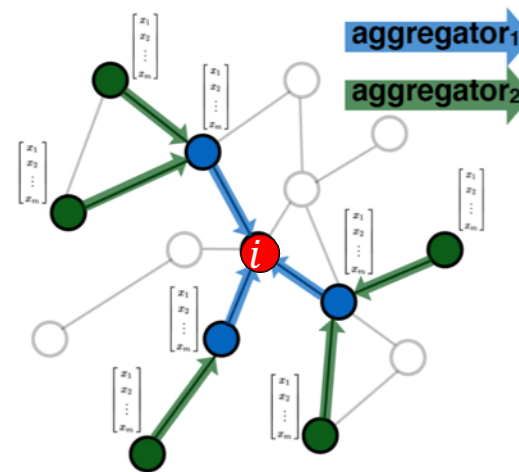
- Assume we have a graph G :
 - V is the **vertex set**
 - A is the **adjacency matrix** (assume binary)
 - $X \in \mathbb{R}^{m \times |V|}$ is a matrix of **node features**
 - Node features:
 - Social networks: User profile, User image
 - Biological networks: Gene expression profiles, gene functional information
 - No features:
 - Indicator vectors (one-hot encoding of a node)
 - Vector of constant 1: $[1, 1, \dots, 1]$

Graph Convolutional Networks

Idea: Node's neighborhood defines a computation graph



Determine node
computation graph

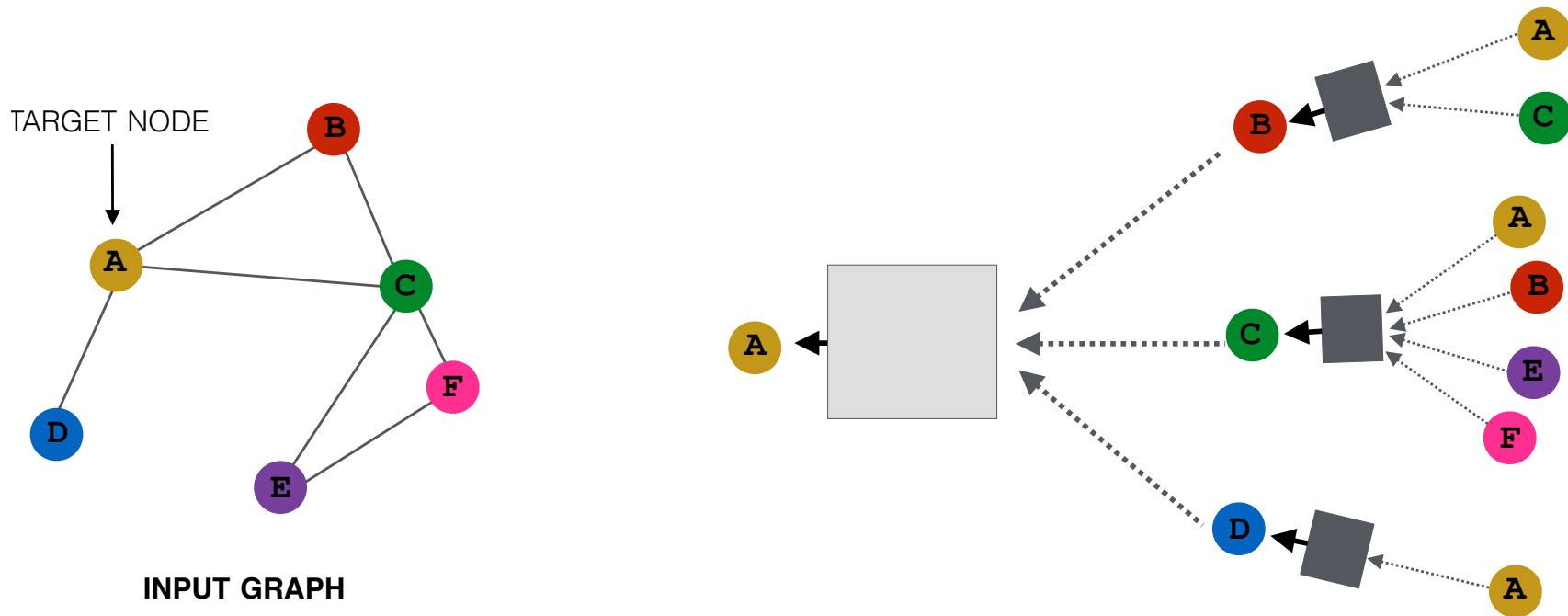


Propagate and
transform information

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

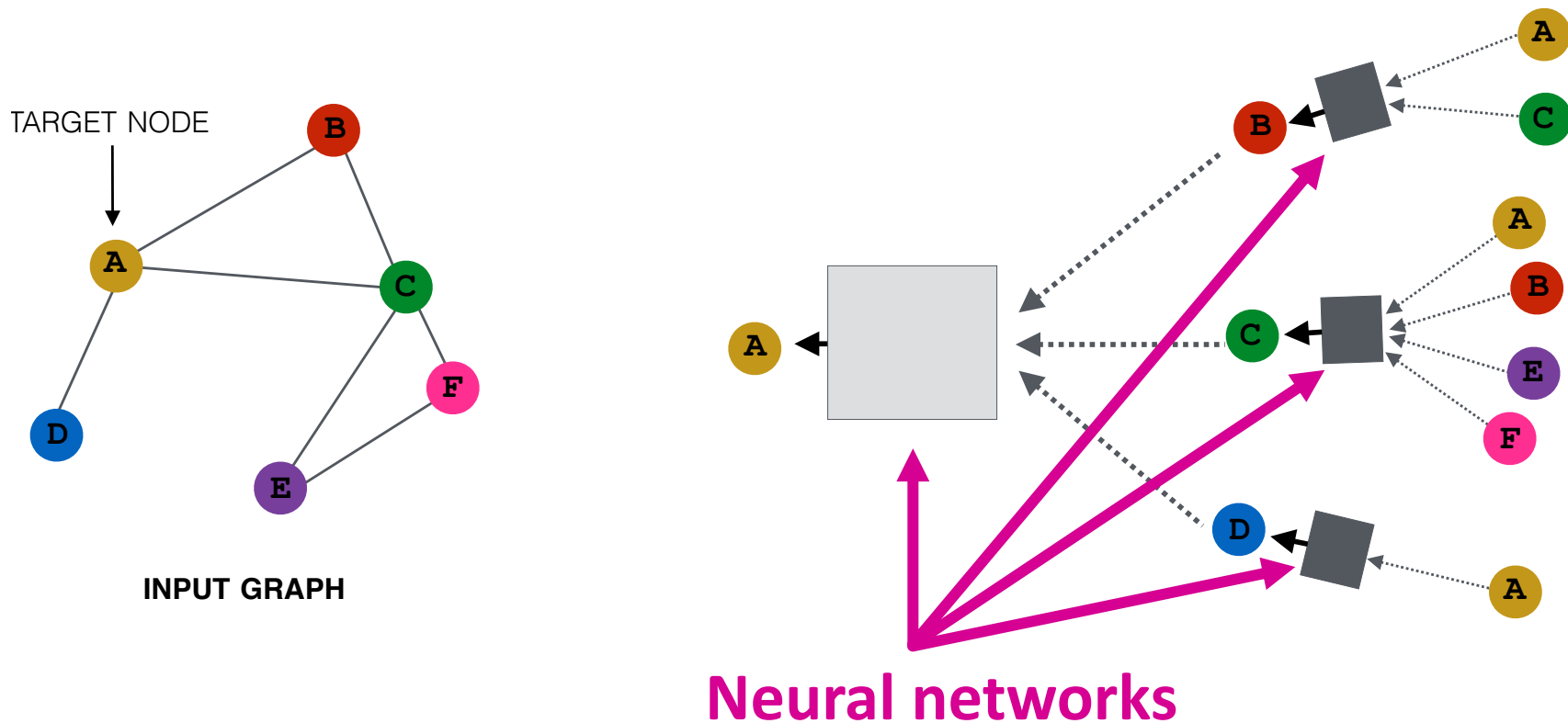
Idea: Aggregate Neighbors

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



Idea: Aggregate Neighbors

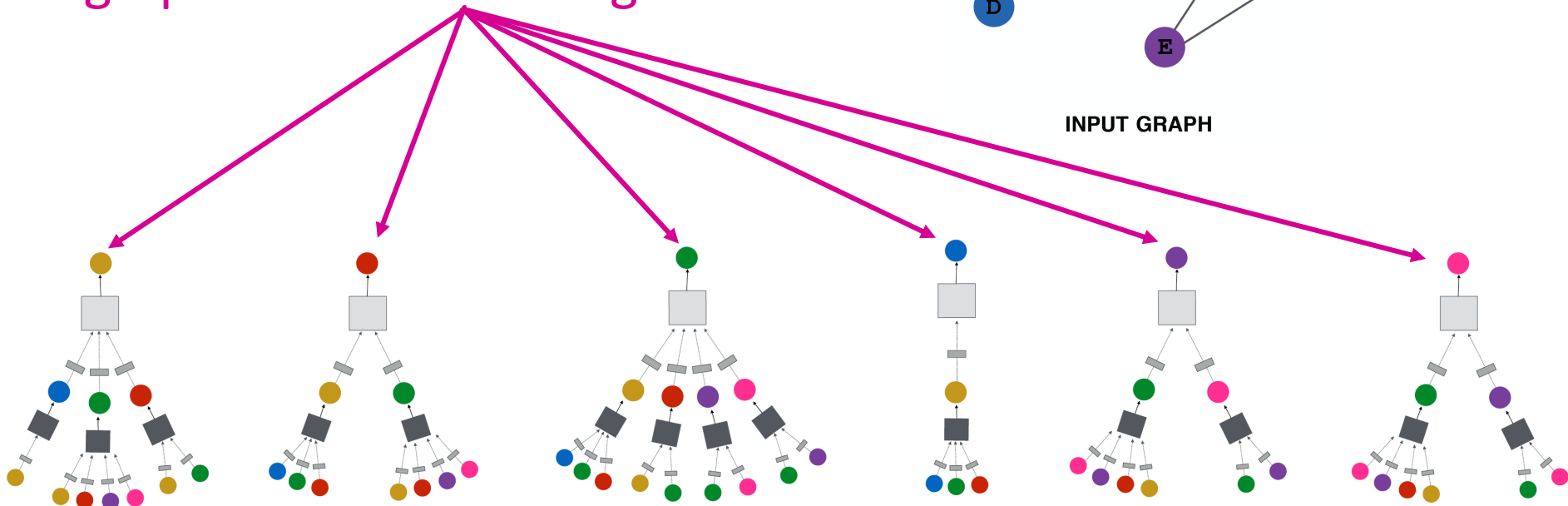
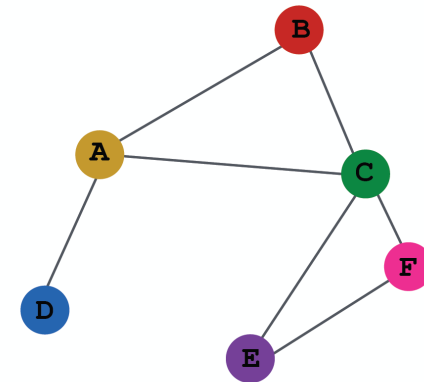
- **Intuition:** Nodes aggregate information from their neighbors using neural networks



Idea: Aggregate Neighbors

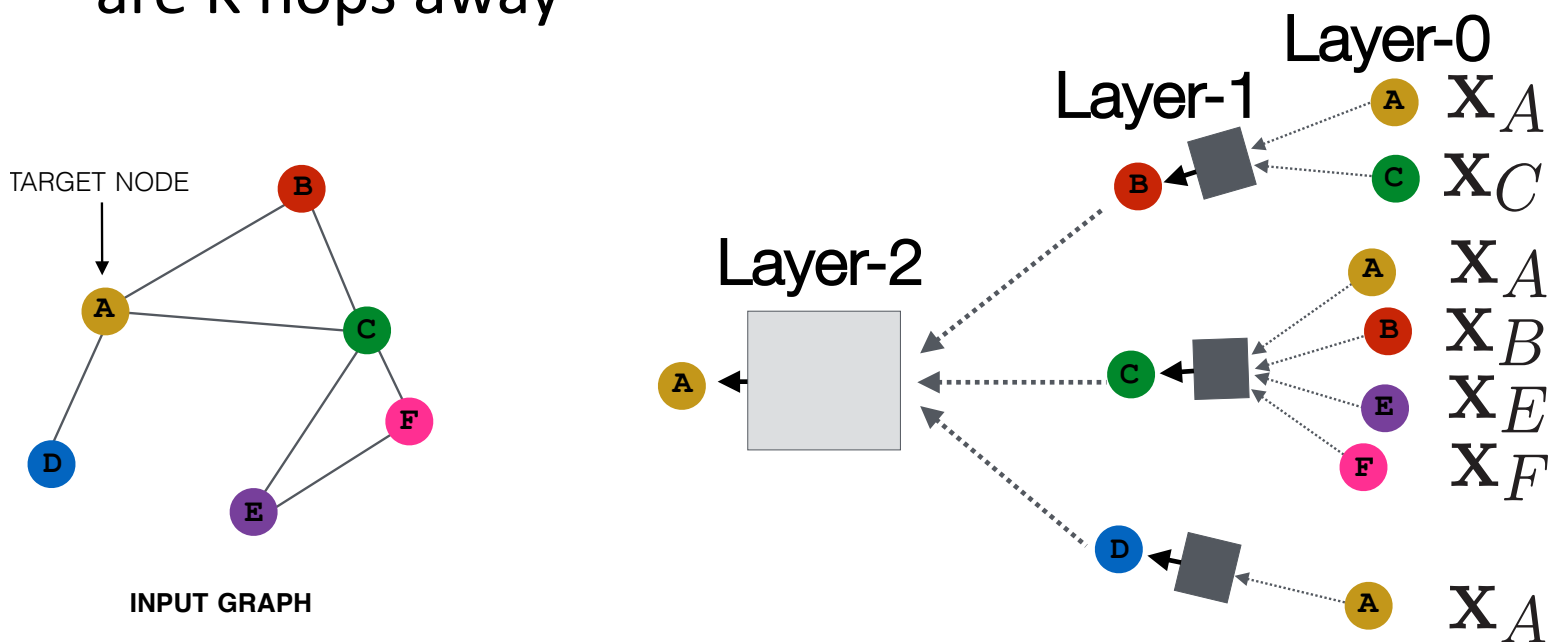
- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



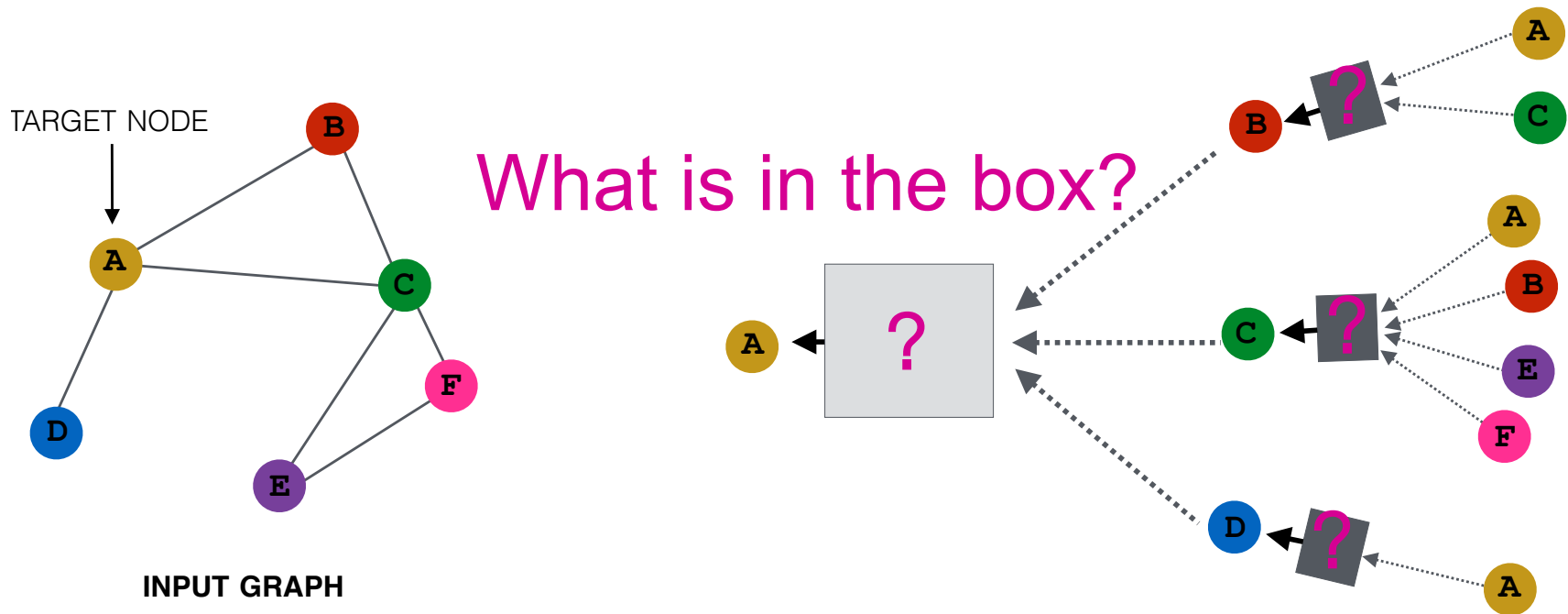
Deep Model: Many Layers

- Model can be of arbitrary depth:
 - Nodes have embeddings at each layer
 - Layer-0 embedding of node u is its input feature, x_u
 - Layer-K embedding gets information from nodes that are K hops away



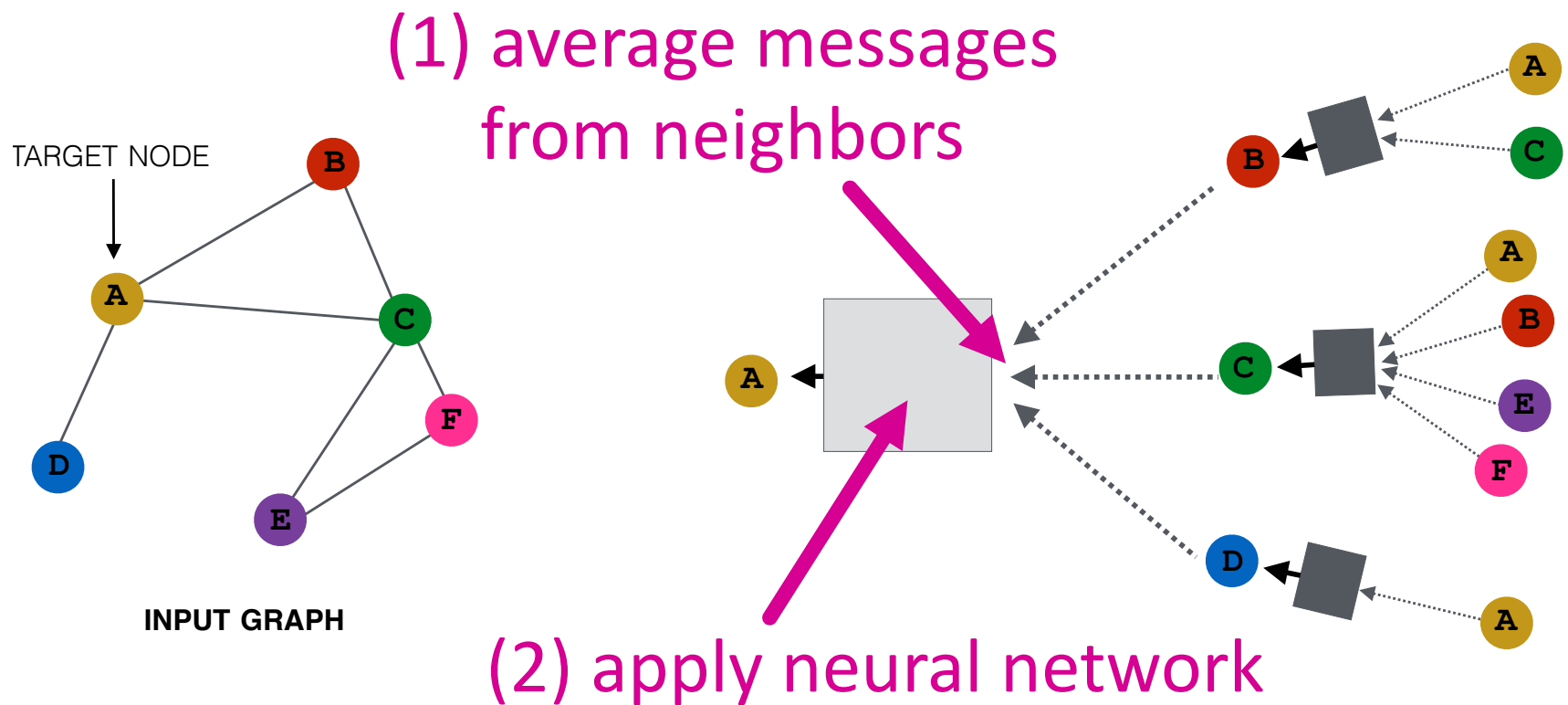
Neighborhood Aggregation

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



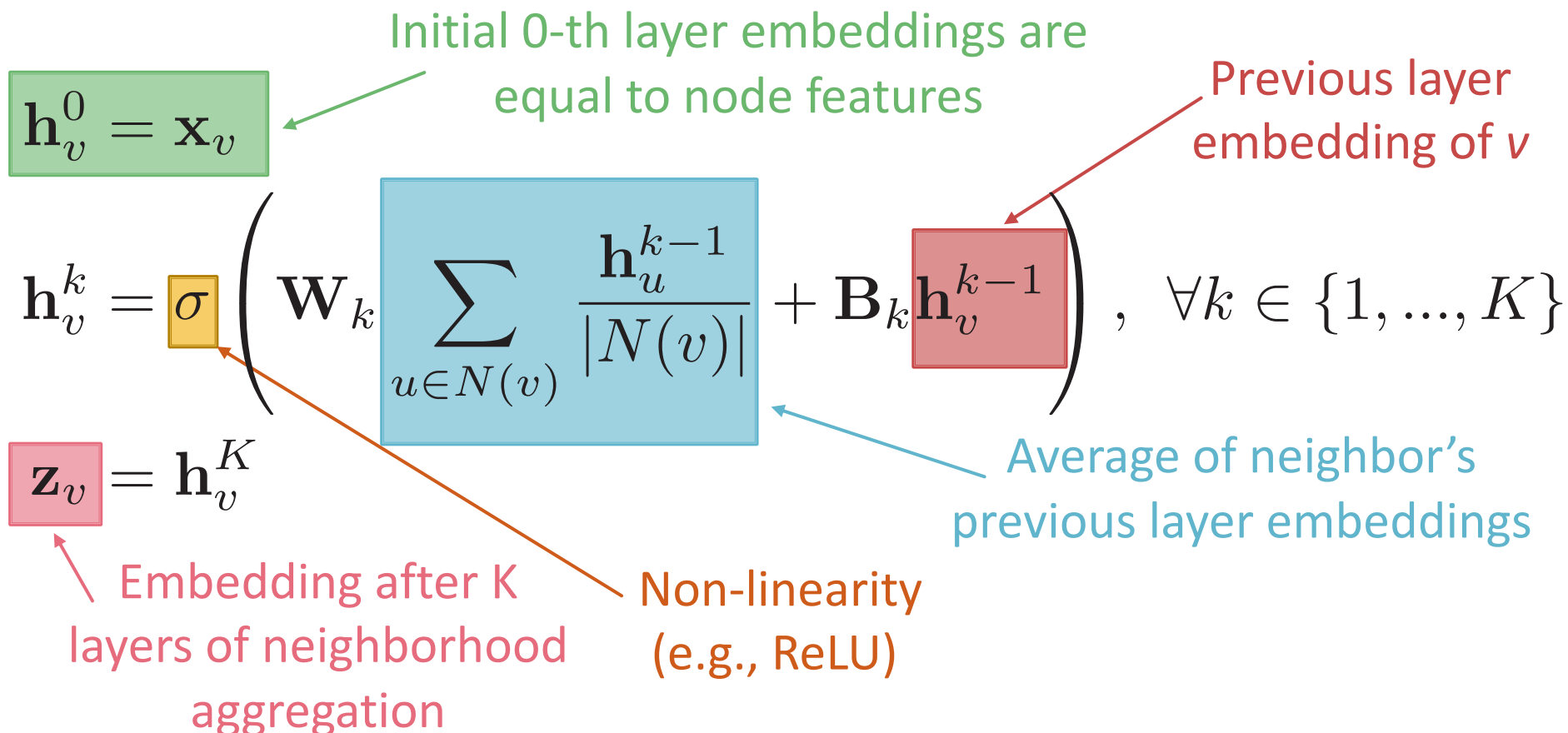
Neighborhood Aggregation

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



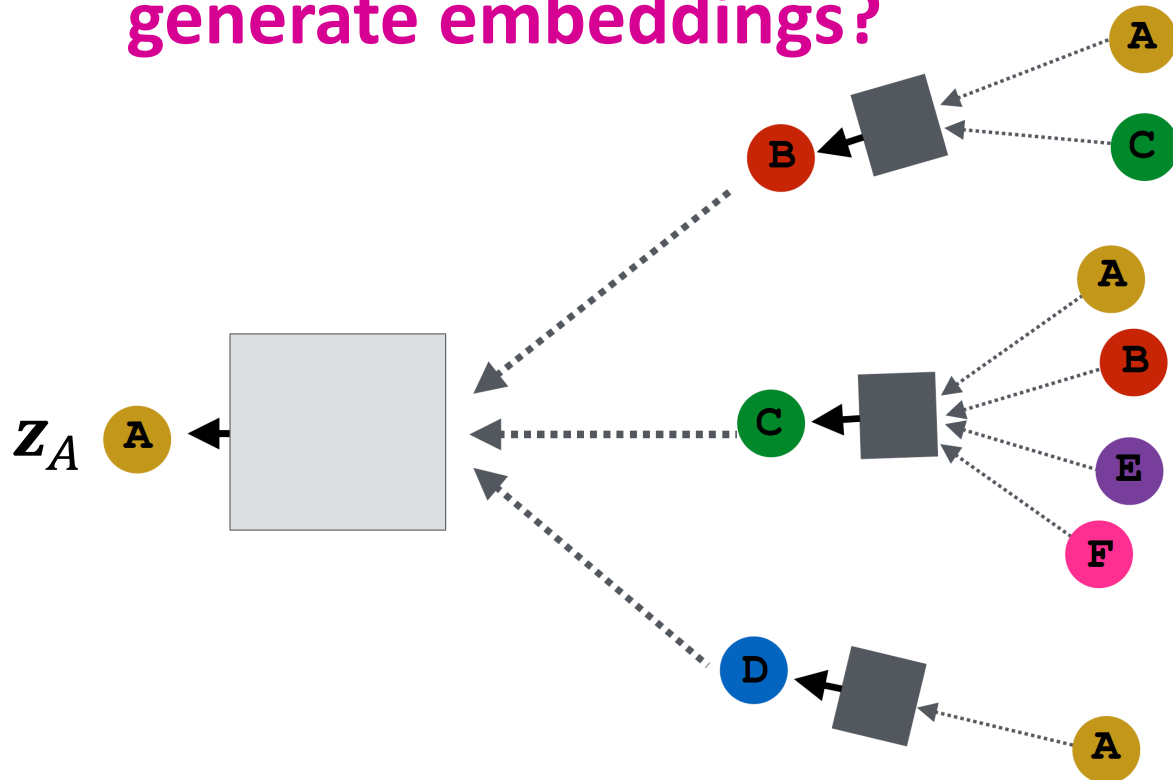
The Math: Deep Encoder

- **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)

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

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

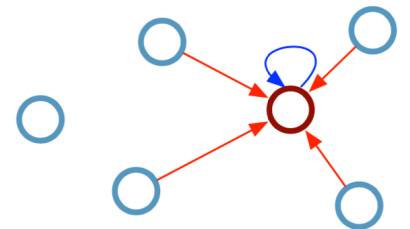
$$\mathbf{z}_v = \mathbf{h}_v^K$$

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

Equivalently
rewritten in
vector form:

$$\mathbf{H}^{(l+1)} = \sigma \left(\mathbf{H}^{(l)} \mathbf{W}_0^{(l)} + \tilde{\mathbf{A}} \mathbf{H}^{(l)} \mathbf{W}_1^{(l)} \right)$$

$$\text{with } \tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$$



$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)T}, \dots, \mathbf{h}_N^{(l)T}]^T$$

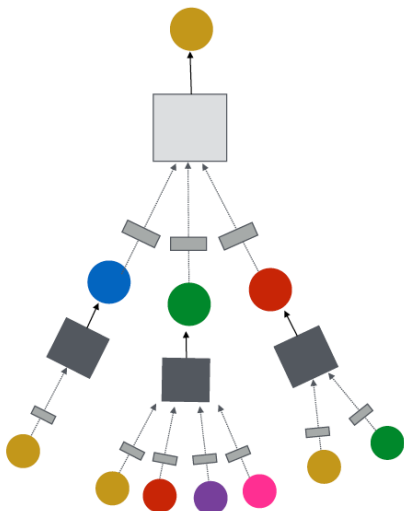
Unsupervised Training

- 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**

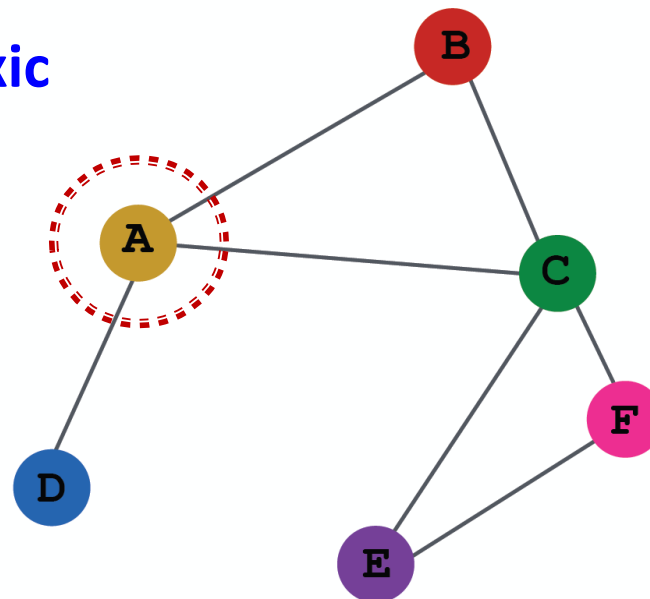
Supervised Training

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**)

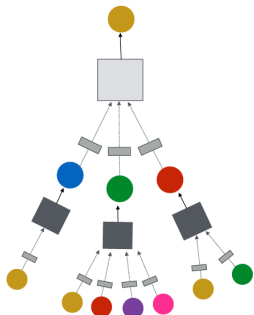
$$\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(\mathbf{z}_v^\top \boldsymbol{\theta})) + (1 - y_v) \log(1 - \sigma(\mathbf{z}_v^\top \boldsymbol{\theta}))$$

Encoder output:
node embedding

Classification
weights

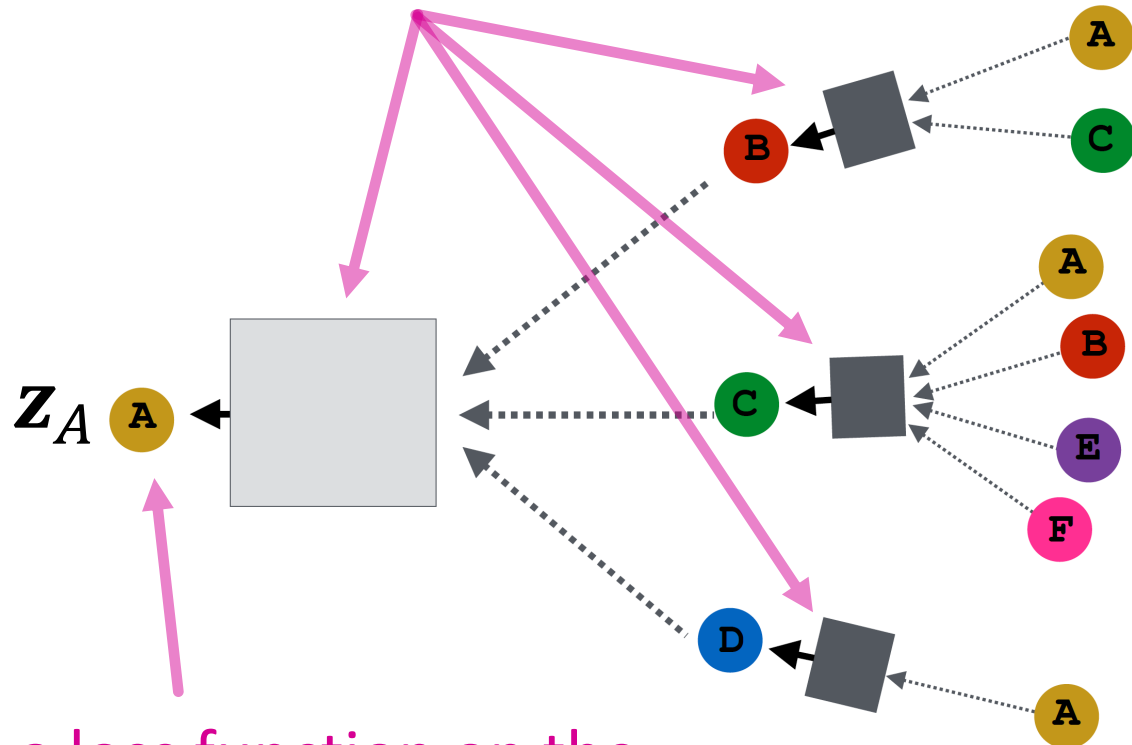
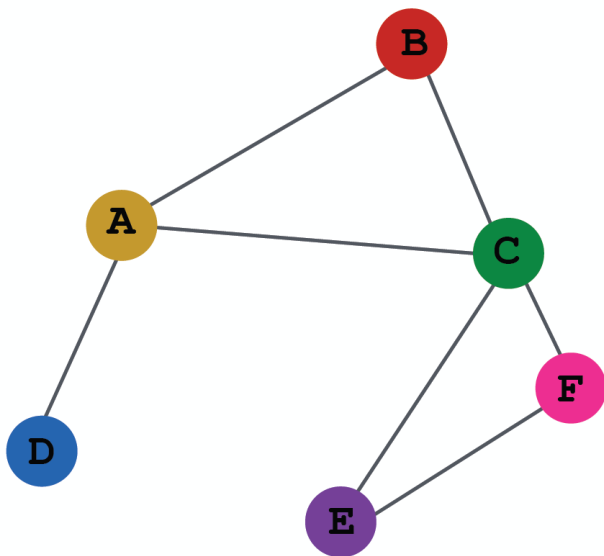
Node class
label

Safe or toxic drug?



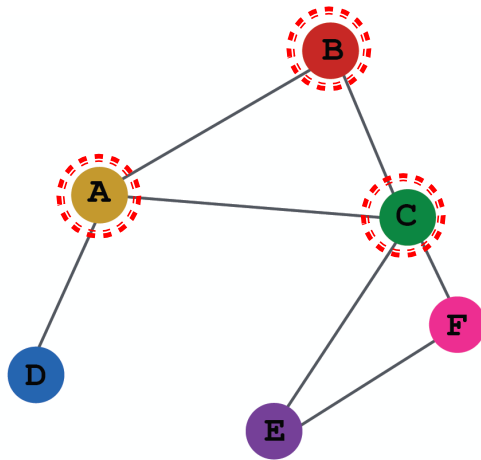
Model Design: Overview

(1) Define a neighborhood aggregation function



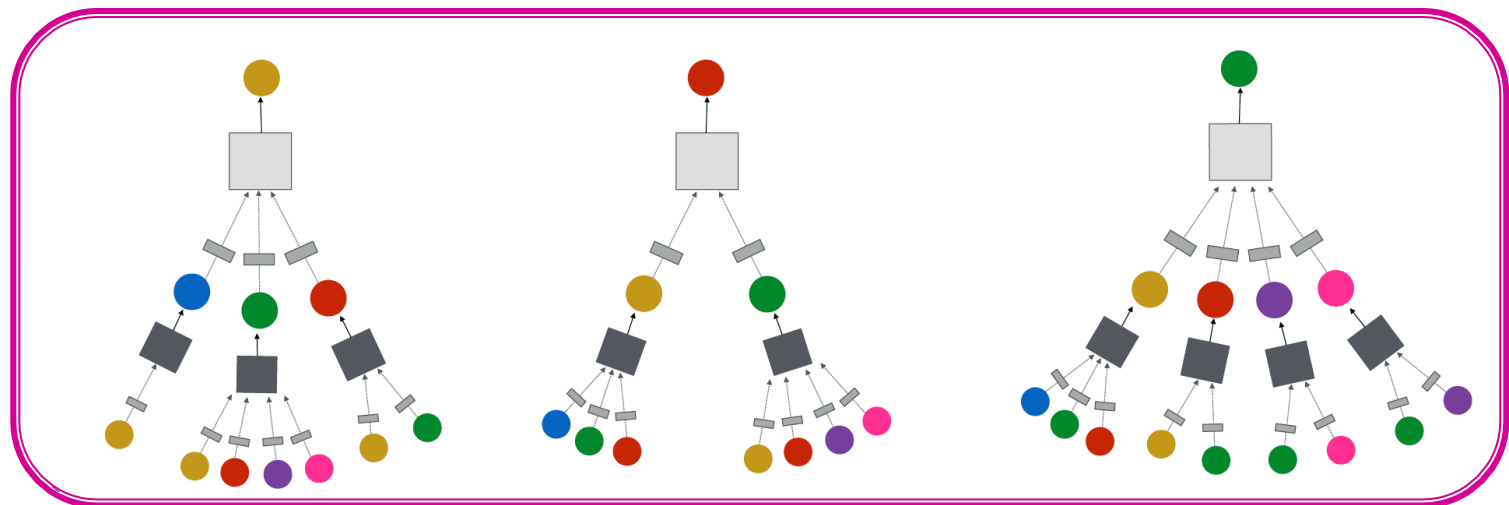
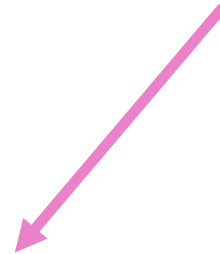
(2) Define a loss function on the embeddings

Model Design: Overview

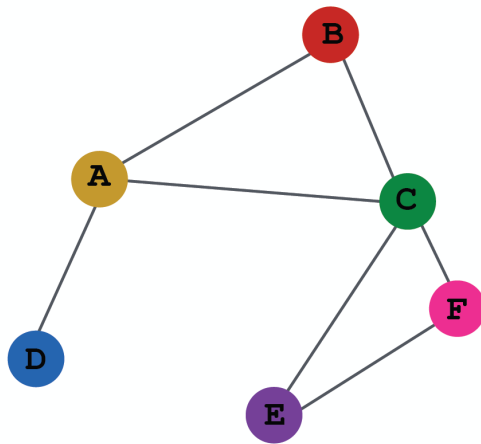


INPUT GRAPH

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



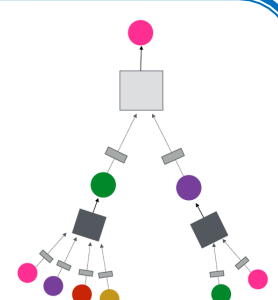
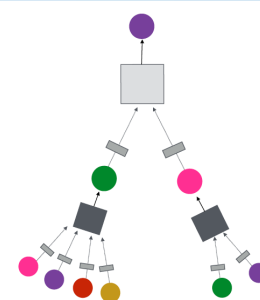
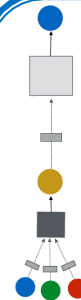
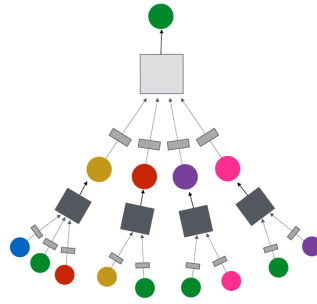
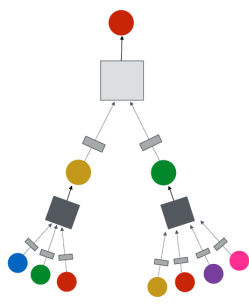
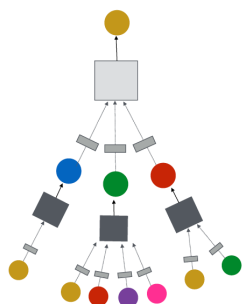
Model Design: Overview



INPUT GRAPH

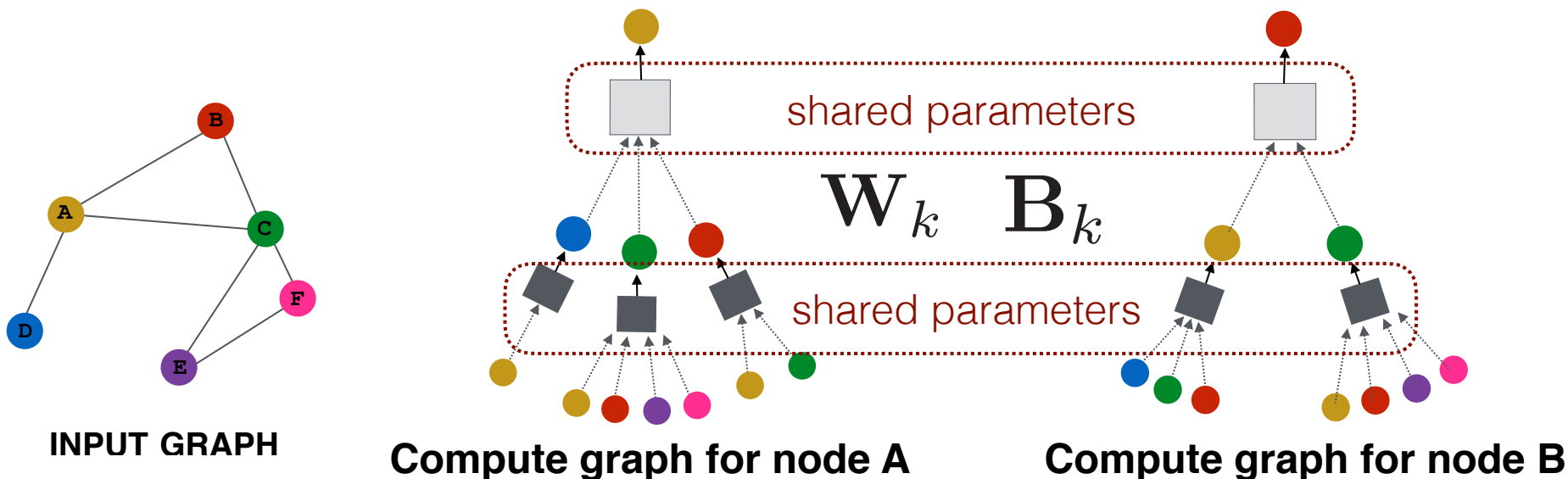
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

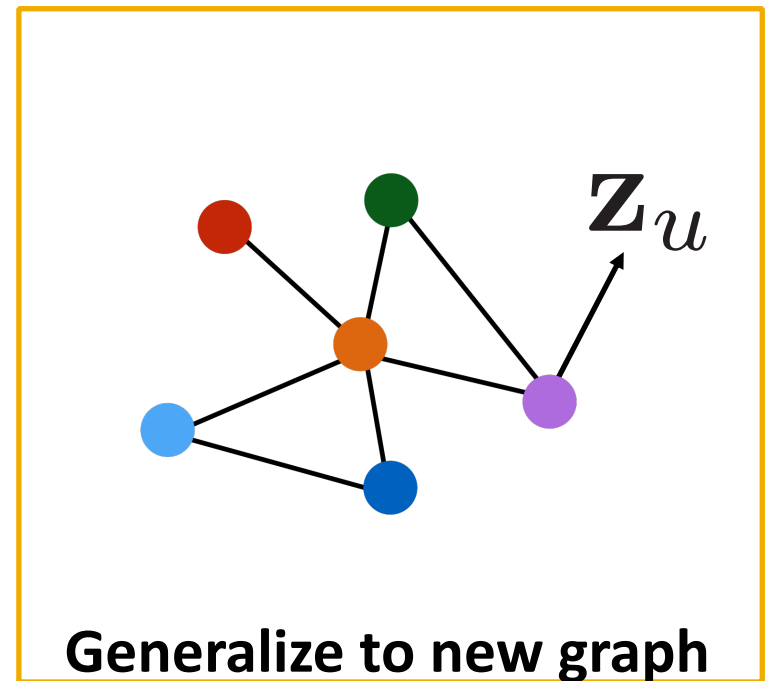
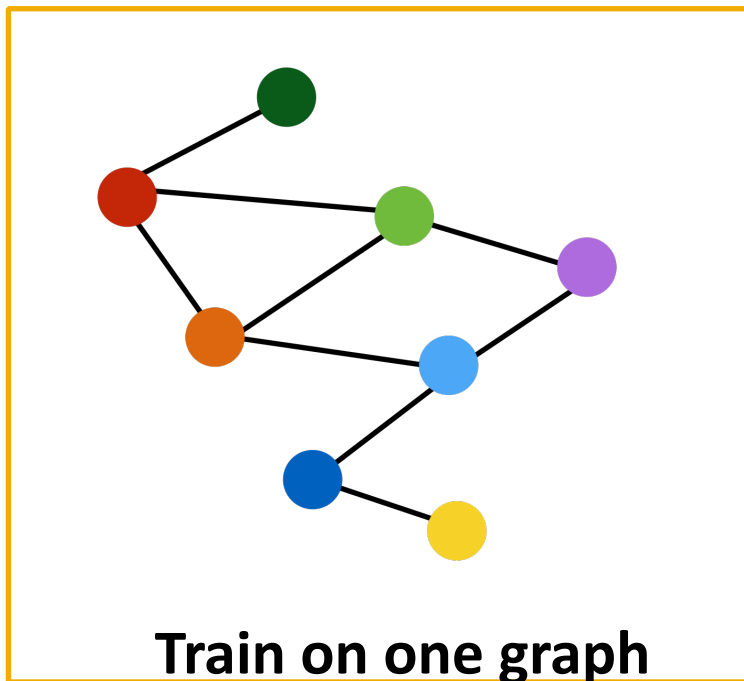


Inductive Capability

- 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!**



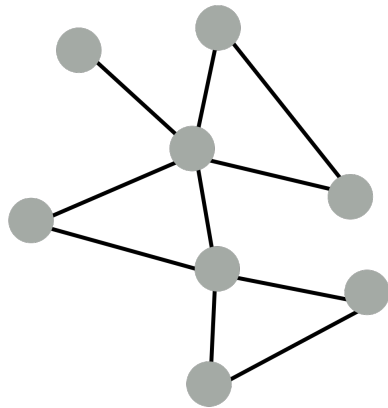
Inductive Capability: New Graphs



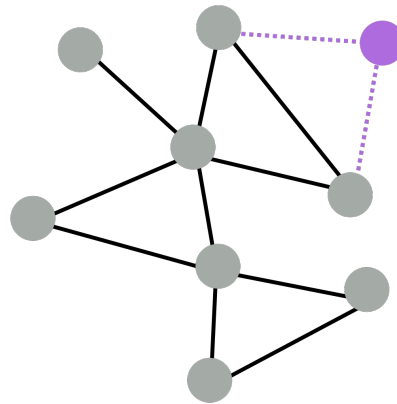
Inductive node embedding \rightarrow Generalize to entirely unseen graphs

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

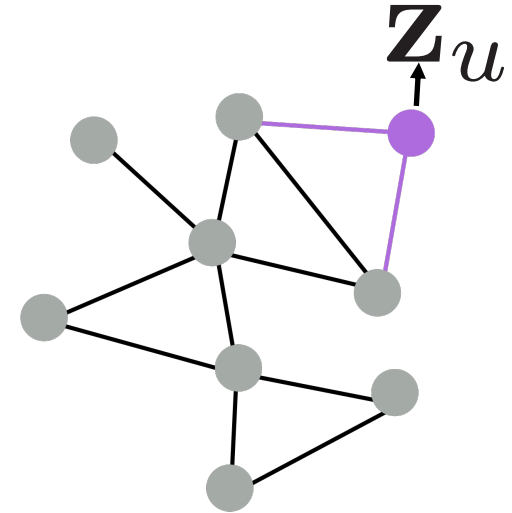
Inductive Capability: New Nodes



Train with snapshot



New node arrives




Generate embedding
for new node

- 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 architecture

Outline of Today's Lecture

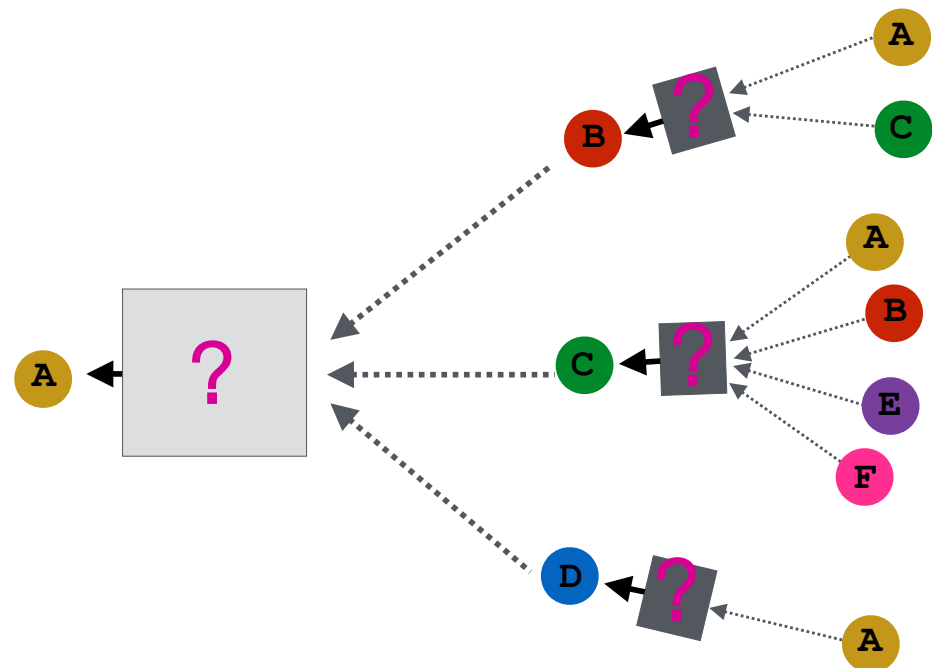
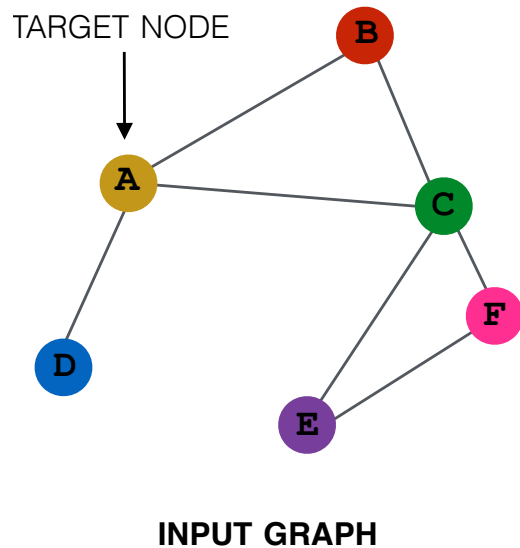
1. Basics of deep learning for graphs ✓
2. Graph Convolutional Networks
3. Graph Attention Networks (GAT) 
4. Practical tips and demos

Graph Convolutional Networks and GraphSAGE

GraphSAGE Idea

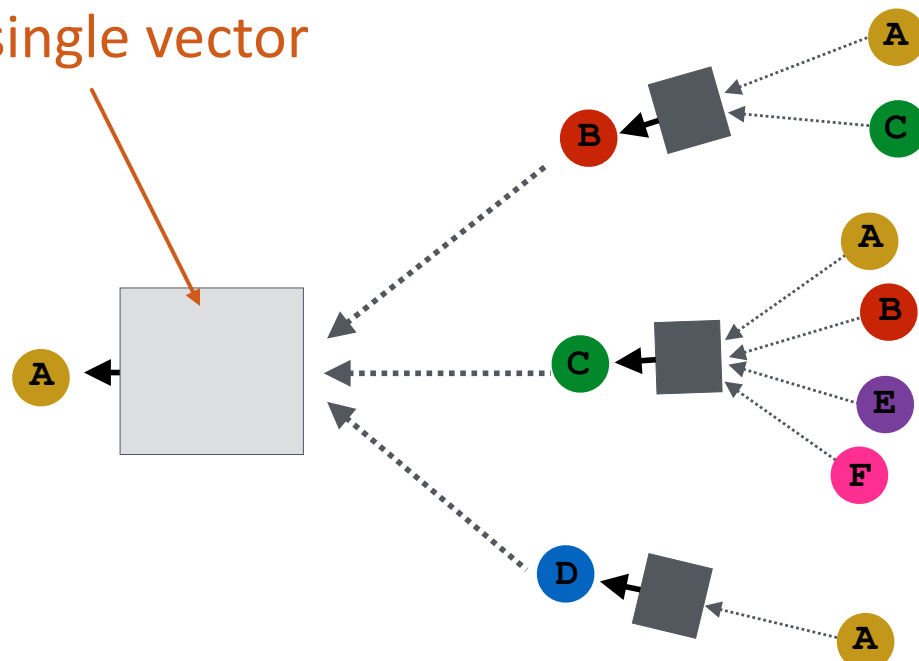
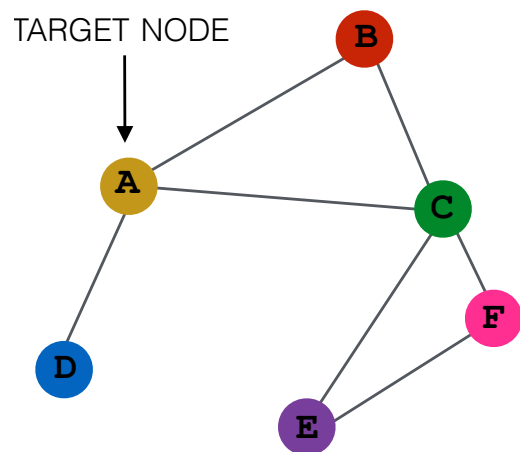
So far we have aggregated the neighbor messages by taking their (weighted) average

Can we do better?



GraphSAGE Idea

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



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

Apply L2 normalization for each node embedding at every layer

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 neighbor embedding
and self 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

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

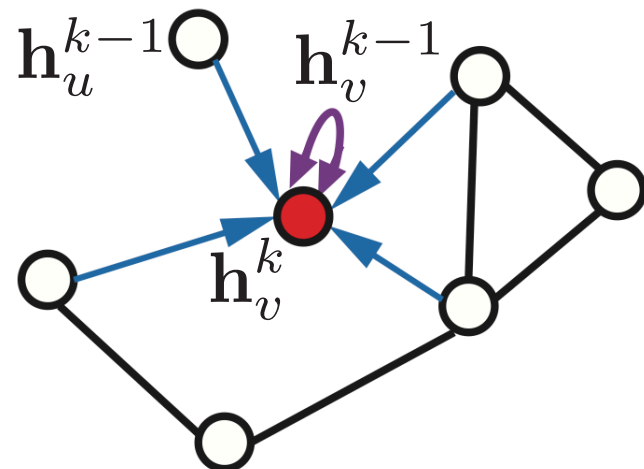
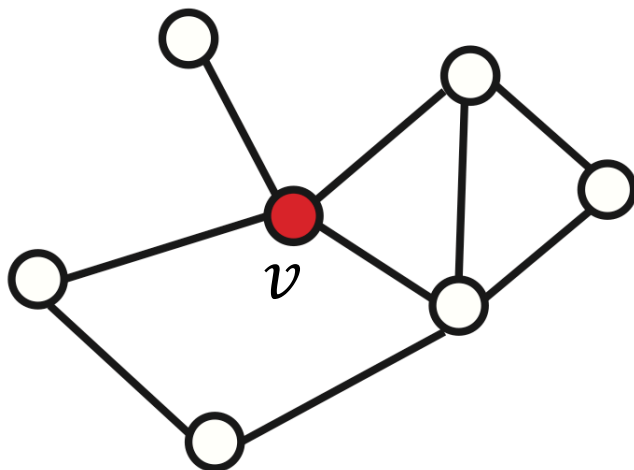
- **LSTM:** Apply LSTM to reshuffled of neighbors

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

Recap: GCN, GraphSAGE

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



Efficient Implementation

- Many aggregations can be performed efficiently by (sparse) matrix operations
- Let $H^{k-1} = [\mathbf{h}_1^{k-1} \dots \mathbf{h}_n^{k-1}]$

$$\sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} \quad \Longrightarrow \quad H^k = D^{-1} A H^{k-1}$$

- Another example: GCN (Kipf *et al.* 2017)

$$H^k = D^{-1/2} A D^{1/2} H^{k-1}$$

More on Graph Neural Networks

Tutorials and overviews:

- Relational inductive biases and graph networks (Battaglia et al., 2018)
- Representation learning on graphs: Methods and applications (Hamilton et al., 2017)

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, Zhang et al., 2018)
- Graph generation and relational inference (You et al., 2018; Kipf et al., 2018)
- How powerful are graph neural networks (Xu et al., 2017)

Embedding nodes:

- Varying neighborhood: Jumping knowledge networks (Xu et al., 2018), GeniePath (Liu et al., 2018)
- Position-aware GNN (You et al. 2019)


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)

Other GNN techniques:

- Pre-training Graph Neural Networks (Hu et al., 2019)
- GNNExplainer: Generating Explanations for Graph Neural Networks (Ying et al., 2019)

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
 - $\Rightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph
 - \Rightarrow All neighbors $u \in N(v)$ are equally important to node v

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

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

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

Next: What is the form of attention mechanism a ?

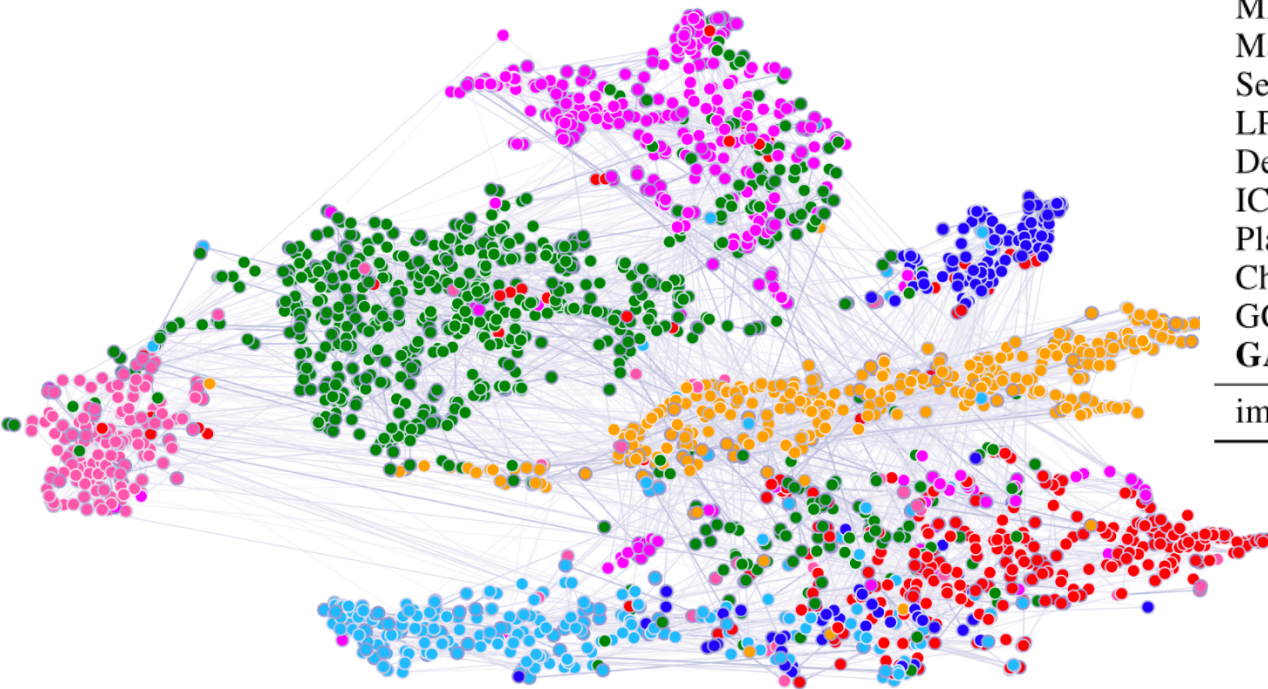
Attention Mechanism (2)

- 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 estimated
 - 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

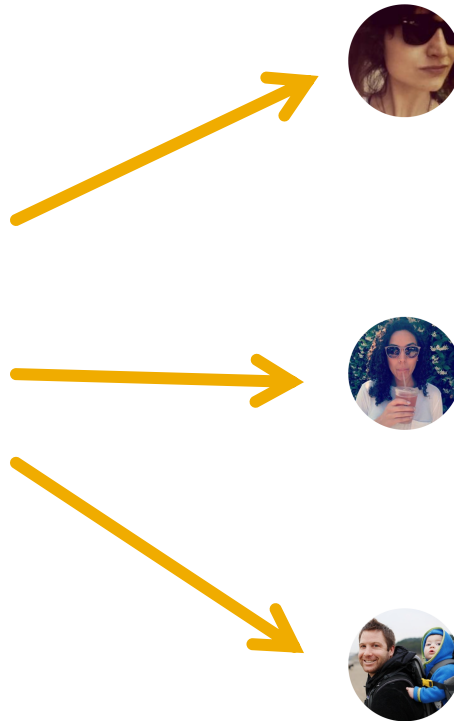
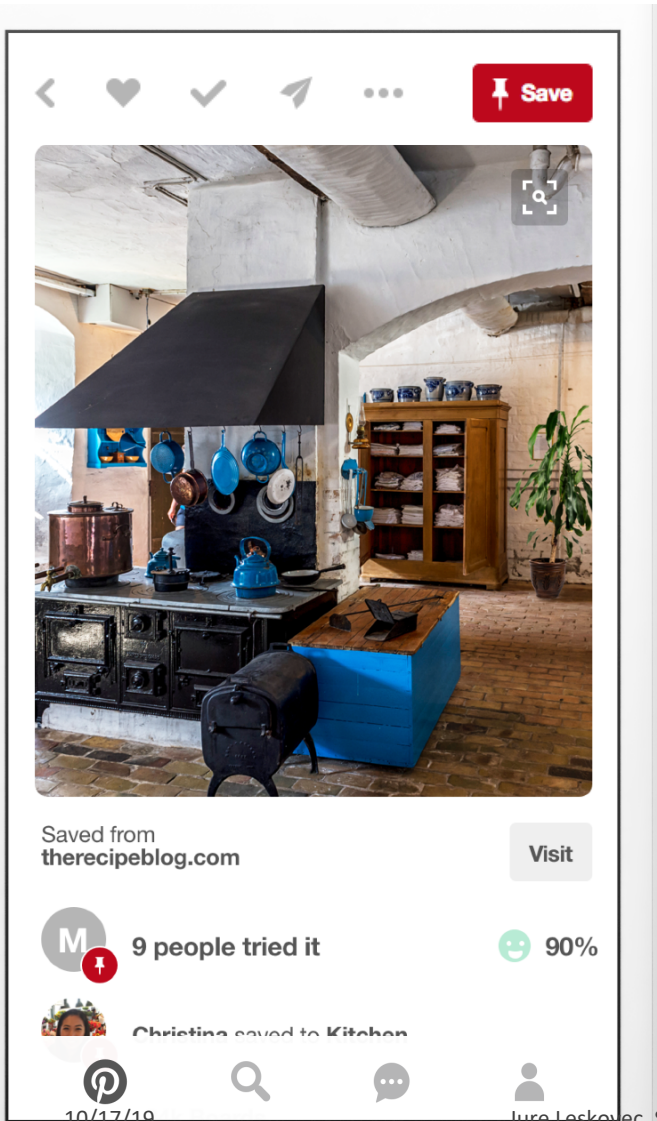
- t-SNE plot of GAT-based node embeddings:
 - Node color: 7 publication classes
 - Edge thickness: Normalized attention coefficients between nodes i and j , across eight attention heads, $\sum_k (\alpha_{ij}^k + \alpha_{ji}^k)$

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 

Example Application

Application: Pinterest



Blue accents
219 Pins



Vintage kitchen
377 Pins



- 300M users
- 4+B pins, 2+B boards

Application: Pinterest

Human curated collection of pins



Very ape blue structured coat
Nitty Gritty

Picked for you
Street style



Hans Wegner chair
Room and Board

Promoted by
Room & Board

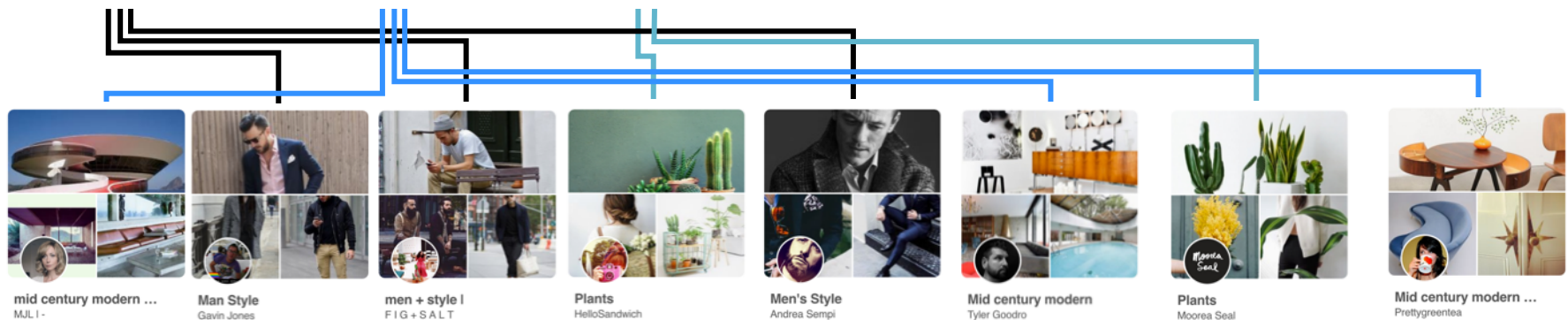


This is just a beautiful image for thoughts. Yay or nay, your choice. 14

Annie Teng
Plantation

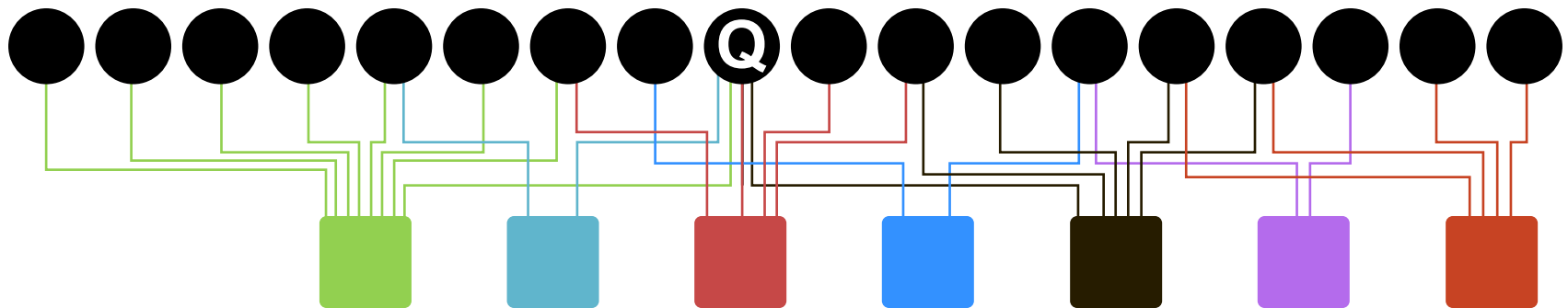
Pin: A visual bookmark someone has saved from the internet to a board they've created.

Pin: Image, text, link



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

Pinterest Graph

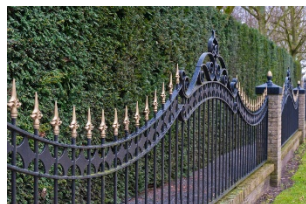


Graph: 2B pins, 1B boards, 20B edges

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

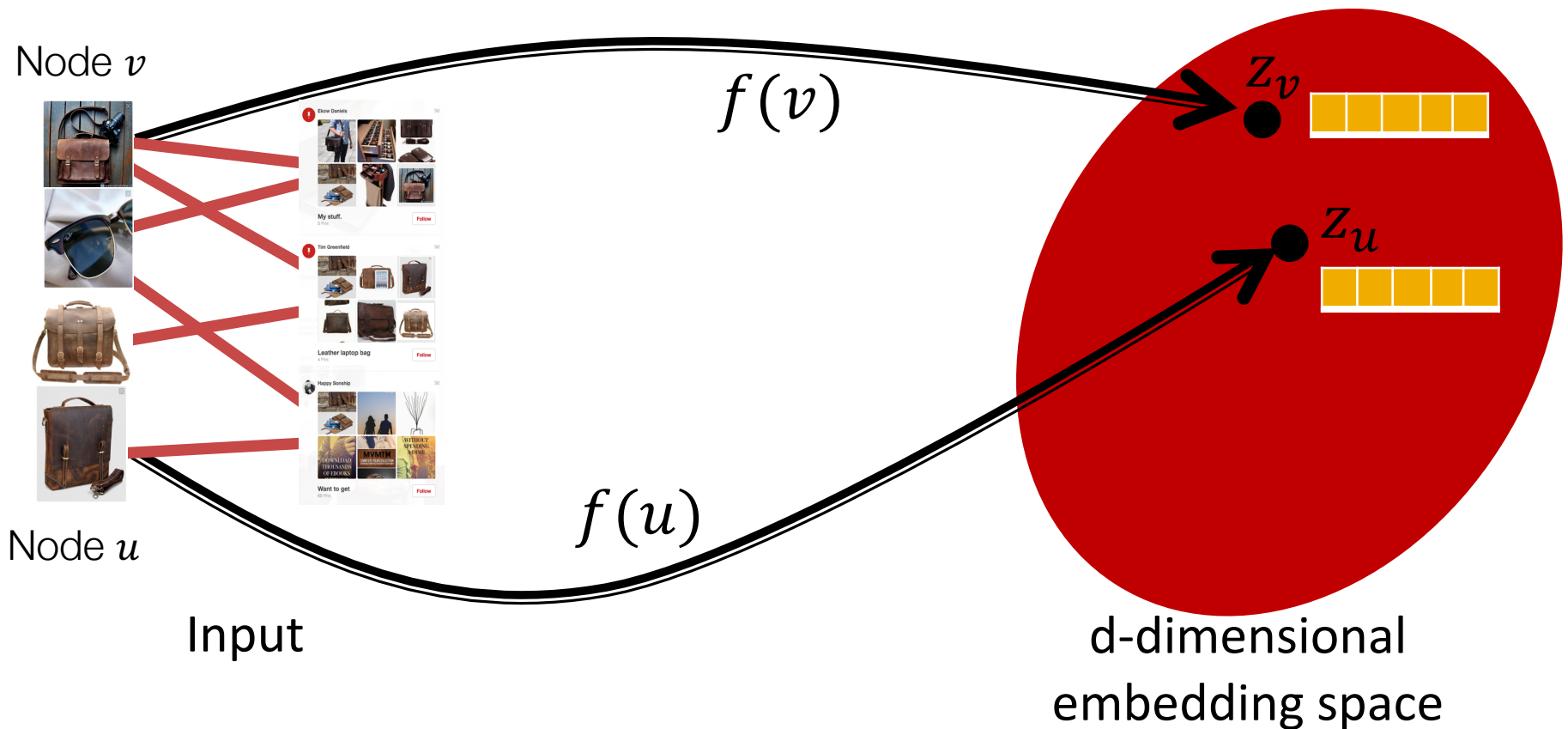
PinSage: Overview

- **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”

Embedding Nodes



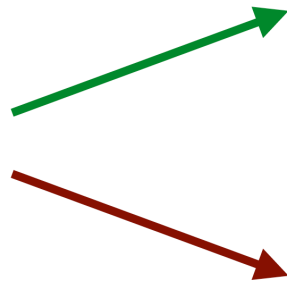
Goal: Map nodes to d-dimensional embeddings such that **nodes that are related are embedded close together**

Task Overview

Task: Recommend related pins to users



Source pin



SUCCESSFUL
RECOMMENDATION



BAD RECOMMENDATION

Task: Learn node embeddings z_i such that

$$d(z_{cake1}, z_{cake2}) < d(z_{cake1}, z_{sweater})$$

■ Challenges:

- **Massive size:** 3 billion nodes, 20 billion edges
- **Heterogeneous data:** Rich image and text features

PinSAGE Training

Goal: Identify target pin among 3B pins

- **Issue:** Need to learn with resolution of 100 vs. 3B
- **Idea:** Use harder and harder negative samples
- Include more and more hard negative samples for each epoch



Source pin



Positive



Easy negative



Hard negative

PinSAGE Efficiency

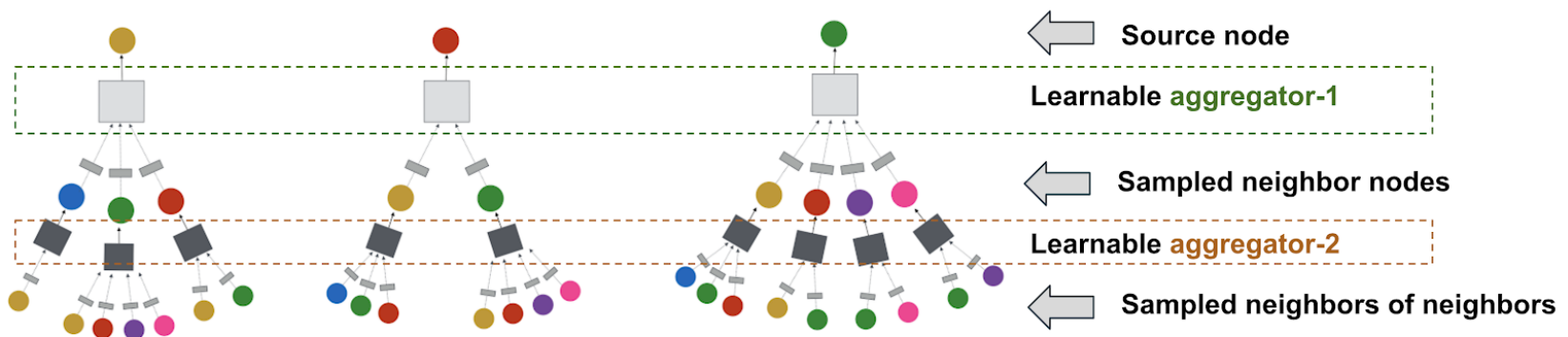
- **How to scale the training as well as inference of node embeddings to graphs with billions of nodes and tens of billions of edges?**
 - 10,000X larger dataset than any previous graph neural network application
- **Key innovations:**
 - Sub-sample neighborhoods for efficient GPU batching
 - Producer-consumer CPU-GPU training pipeline
 - Curriculum learning for negative samples
 - MapReduce for efficient inference

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



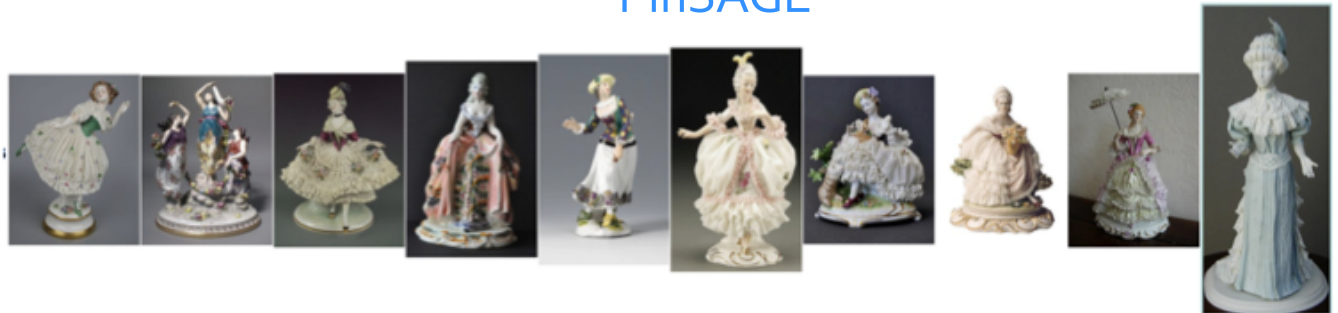
Pixie is a purely graph-based method that uses biased random walks to generate ranking scores by simulating random walks starting at query Pin. Items with top scores are retrieved as recommendations [Eksombatchai et al., 2018]

PinSAGE Recommendations

Query

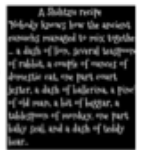
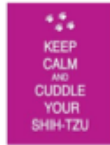
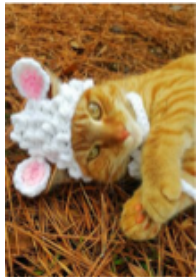


PinSAGE

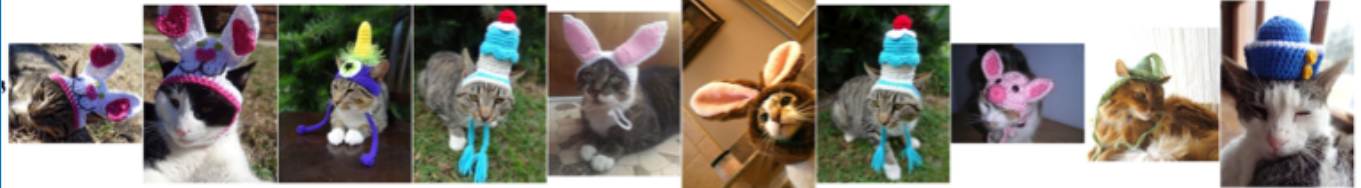


PinSAGE Recommendations

Query



PinSAGE



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

Human Disease Network x Marinka

← → ↻ 🏠 ⓘ snap.stanford.edu/deepnetbio-ismb/ipynb/Human+Disease+Network.html 🔍 ☆ 📄 🗑️ 🔄 🌐 ⋮

Embedding the Human Disease Network

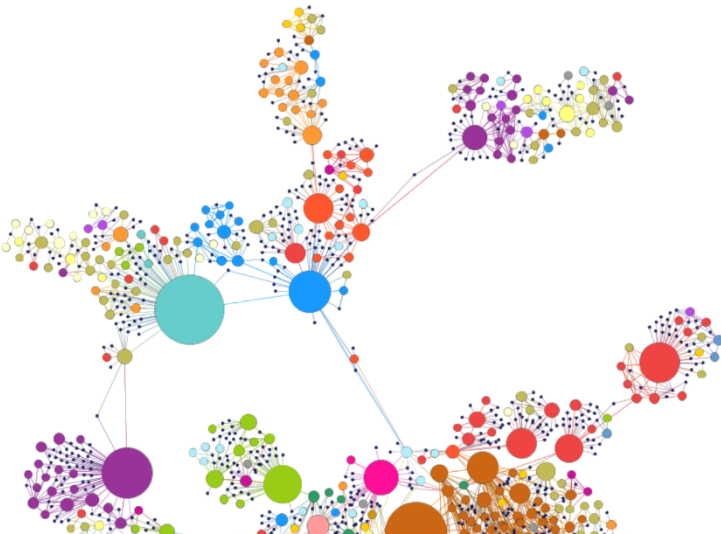
(This demo is a part of [Deep Learning for Network Biology](#) tutorial.)

Human disease network is a network, in which nodes represent diseases and two diseases are connected to each other if they share at least one gene in which mutations are associated with both diseases.

The network is described in Goh et al., [The Human Disease Network](#), PNAS 2007.

The figure below show the human disease network.

Although the layout of the network was generated independently of any knowledge of disease classes, the resulting network is naturally and visibly clustered according to major disease classes (e.g., bone, cancer, cardiovascular, skeletal, or metabolic diseases; each disease class is represented by a different color). The size of a node is proportional to the number of genes participating in the corresponding disease.



Demo: Protein Interaction Prediction

Graph Convolutional Prediction of Protein Interactions in Yeast

(This demo is a part of [Deep Learning for Network Biology](#) tutorial.)

In this example, we demonstrate the utility of deep learning methods for an important prediction problem on biological graphs. In particular, we consider the problem of predicting [protein-protein interactions](#) (PPIs).

Protein-protein interactions (PPIs) are essential to almost every process in a cell. Understanding PPIs is crucial for understanding cell physiology in normal and disease states. Furthermore, knowledge of PPIs can be used:

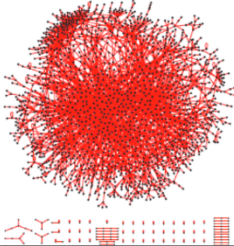
- for drug development, since drugs can affect PPIs,
- to assign roles (i.e., protein functions) to uncharacterized proteins,
- to characterize the relationships between proteins that form multi-molecular complexes, such as the proteasome.

We represent the totality of PPIs that happen in a cell, an organism or a specific biological context with a [protein-protein interaction network](#). These networks are mathematical representations of all physical contacts between proteins in the cell.

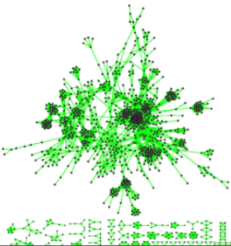
The development of large-scale PPI screening techniques, especially [high-throughput affinity purification combined with mass-spectrometry](#) and the [yeast two-hybrid assay](#), has caused an explosion in the amount of PPI data and the construction of ever more complex and complete interaction networks. For example, the figure below is a graphical representation of three different types of protein-protein interaction networks in yeast *S. cerevisiae*. The structure of the binary interaction network is obviously different from the structure of the co-complex interaction network. The network structure of the literature-curated dataset resembles that of the co-complex dataset, even though the literature-curated datasets are reported to contain mostly binary interactions.

However, current knowledge of protein-protein interaction networks is both [incomplete and noisy](#), as PPI screening techniques are limited in how many true interactions they can detect. Furthermore, PPI screening techniques often have high false positive and negative rates. These limitations present a great opportunity for computational methods to predict protein-protein interactions.

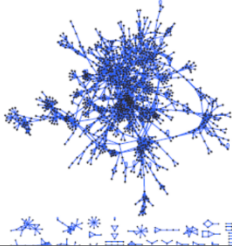
Binary
(Y2H-union)



Co-complex
(Combined-AP/MS)



Literature
(LC-multiple)



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 ✓