

# A General Perspective on Graph Neural Networks

CS246: Mining Massive Datasets

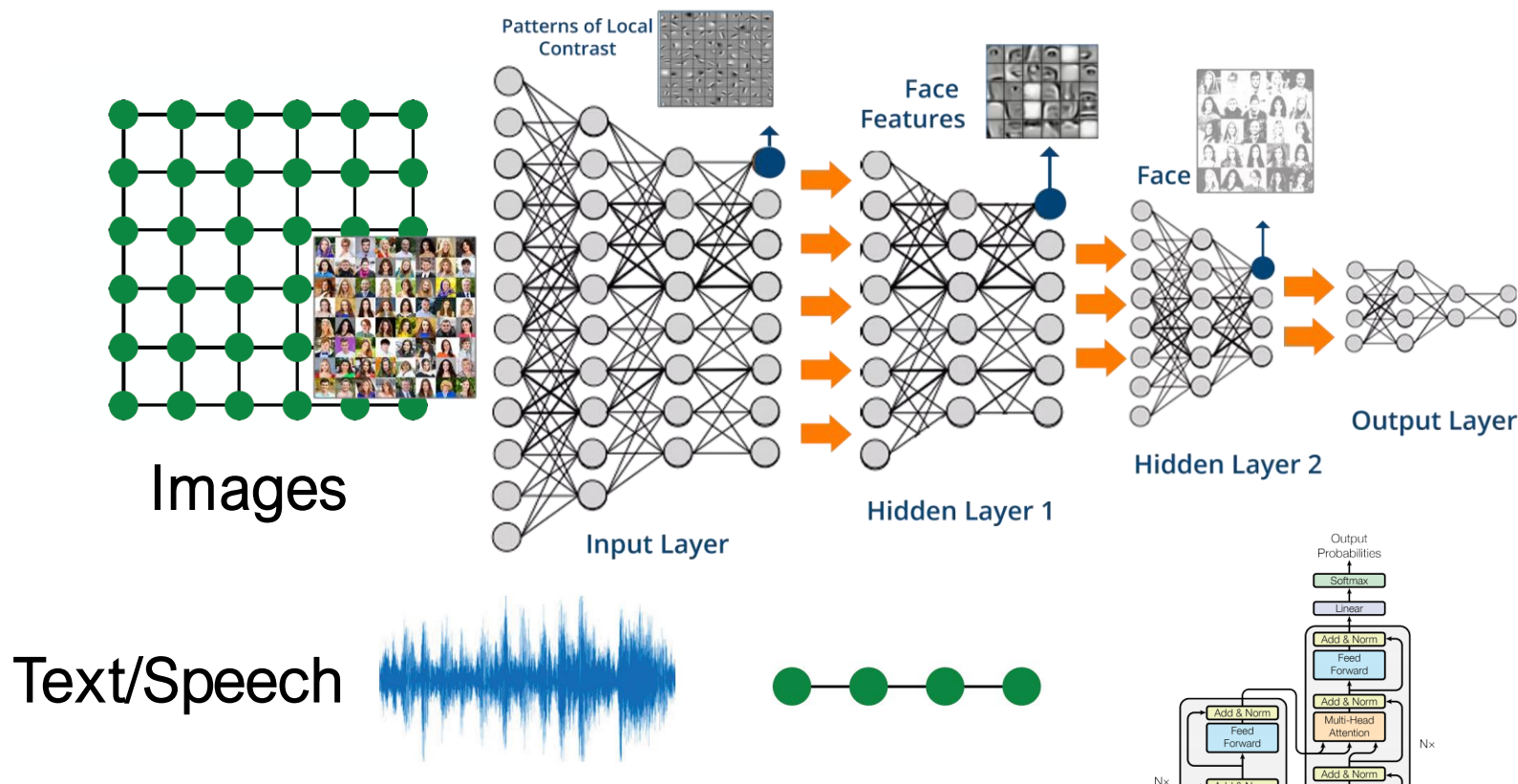
Jure Leskovec, Stanford University

Charilaos Kanatsoulis, Stanford University

<http://cs246.stanford.edu>



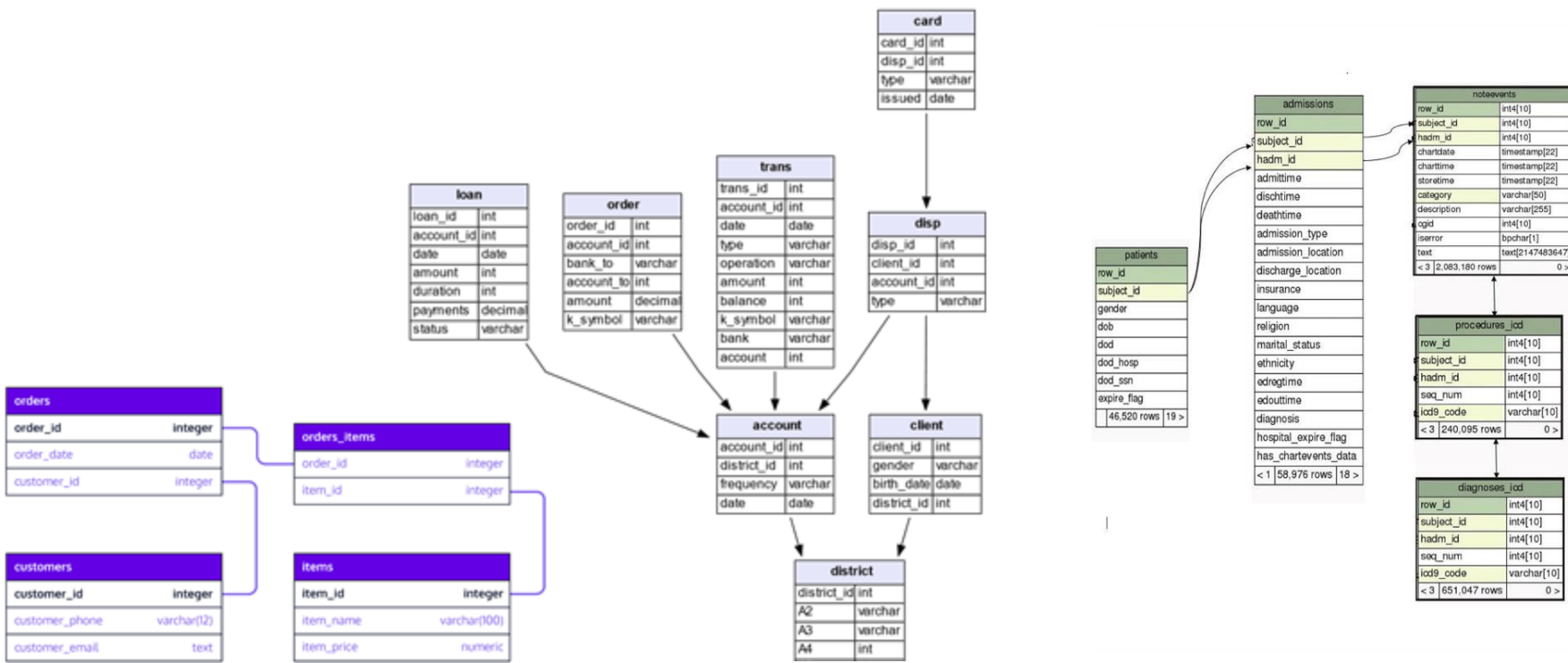
# Modern ML Toolbox



Modern deep learning toolbox is designed for simple sequences & grids

# Relational Deep Learning

Data is stored in **Relational Tables**

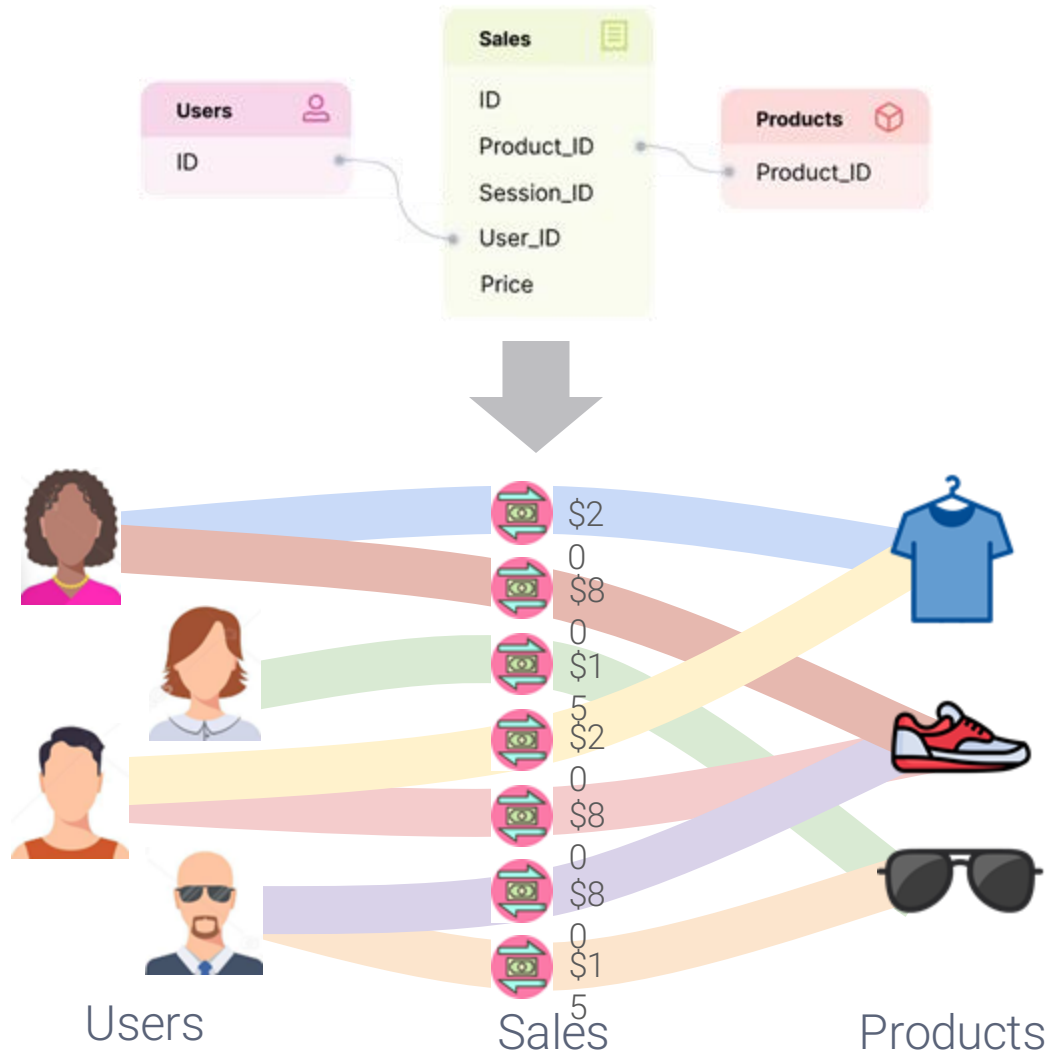


Commerce

Finance

Health care

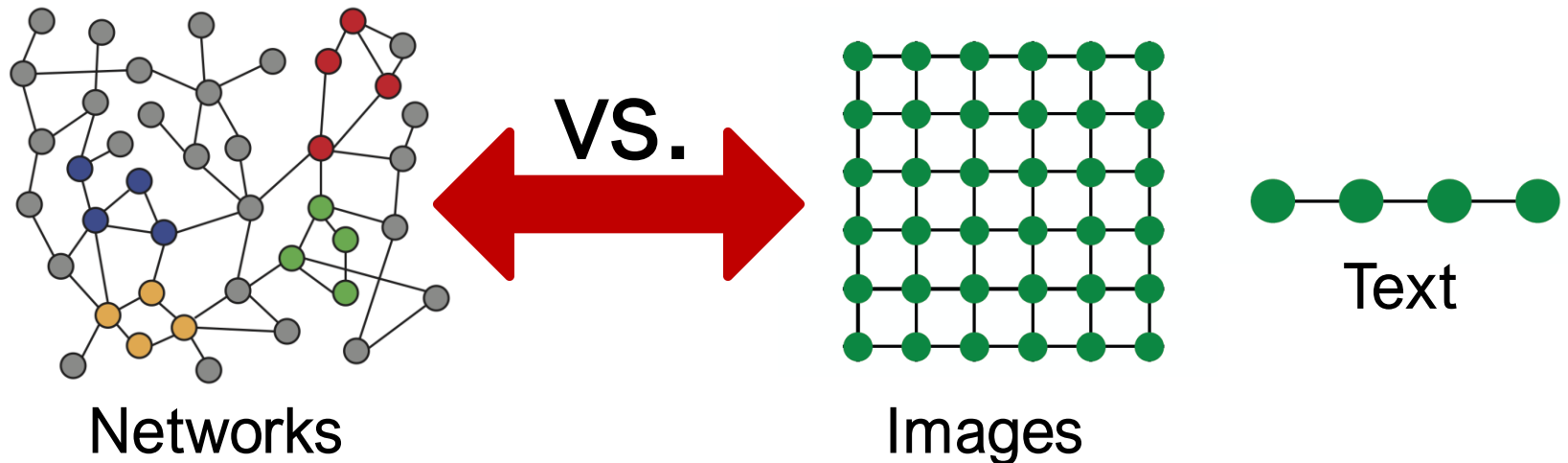
# Relational data is a graph!



# Why is it Hard?

## But networks are far more complex!

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



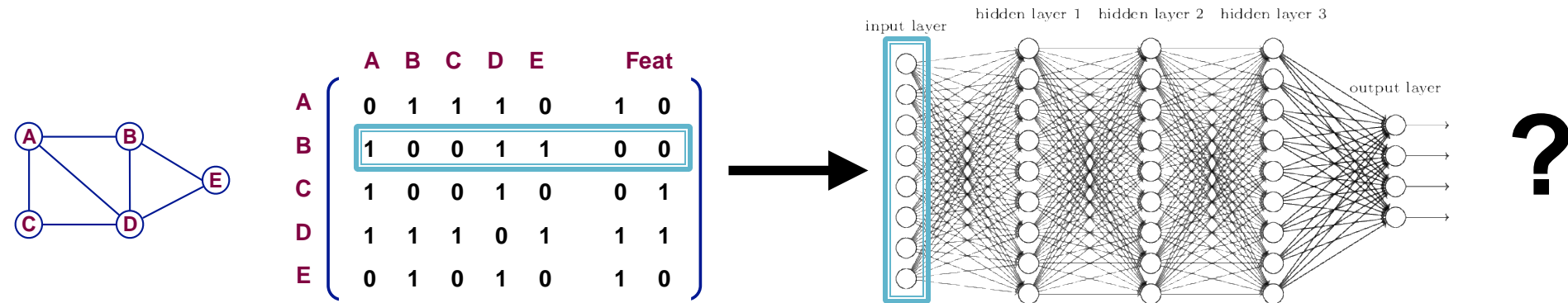
- No fixed node ordering or reference point
- Often dynamic and have multimodal features

# Graph Neural Networks



# A Naïve Approach

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

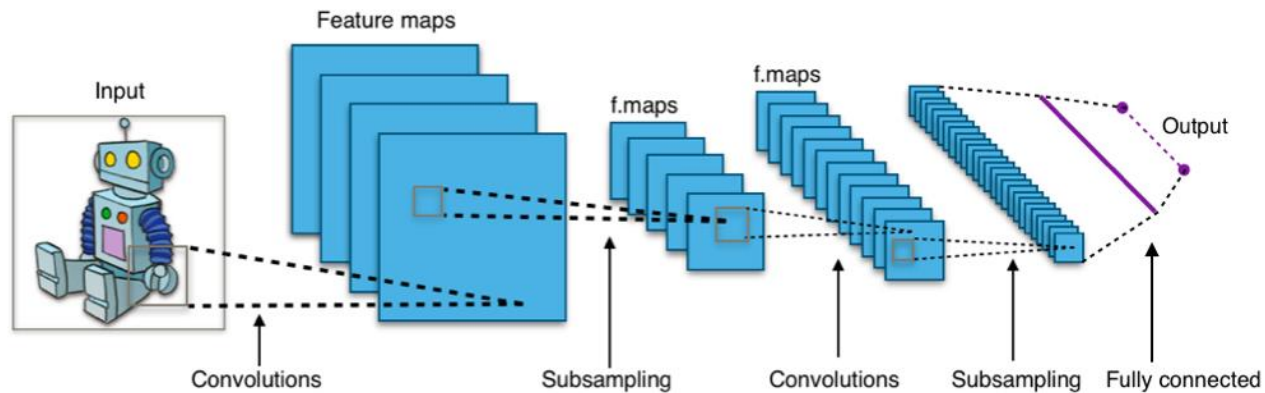
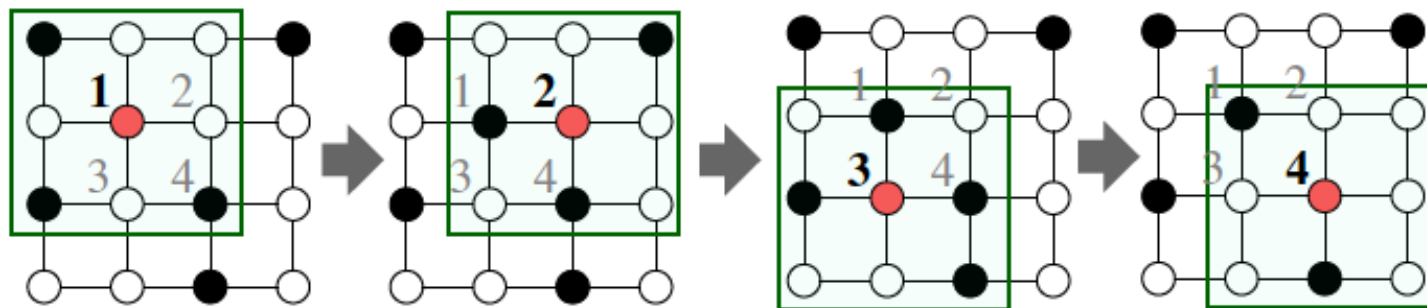


- **Issues with this idea:**

- $O(|V|)$  parameters
- Not applicable to graphs of different sizes
- Sensitive to node ordering

# Idea: Convolutional Networks

## CNN on an image:

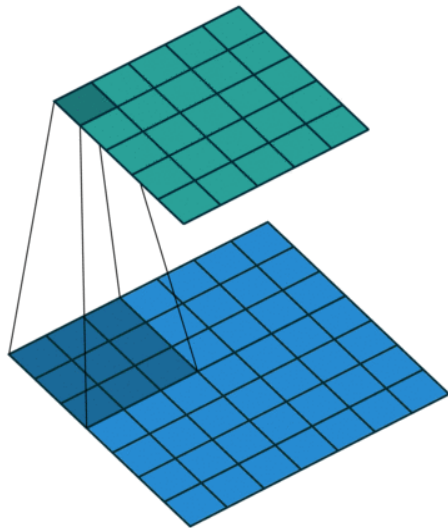


Goal is to generalize convolutions to graphs

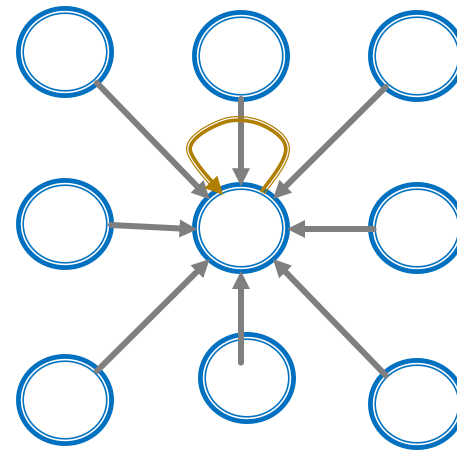


# From Images to Graphs

Single Convolutional neural network (CNN) layer with 3x3 filter:



Image



Graph

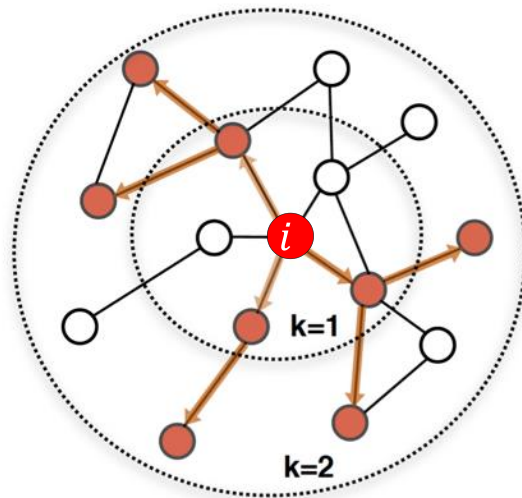
**Idea:** transform information at the neighbors and combine it:

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

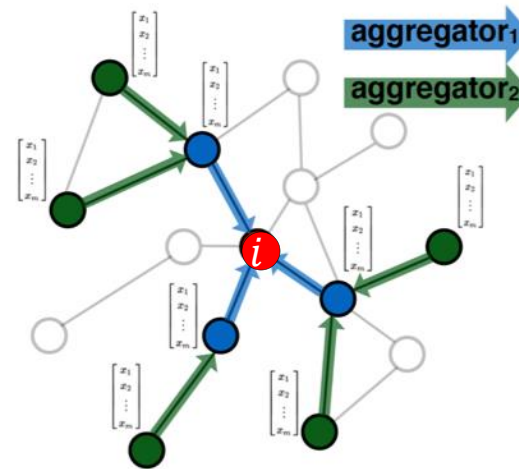
Credit: [Stanford CS224W](#)

# Graph Convolutional Networks

**Idea:** Node's neighborhood defines a computation graph



Determine node  
computation graph



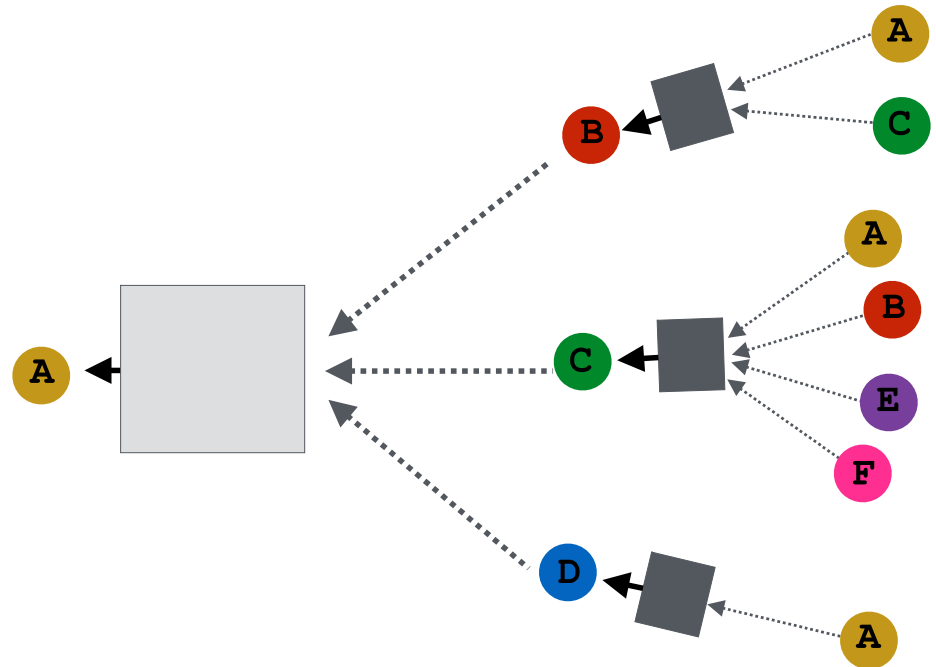
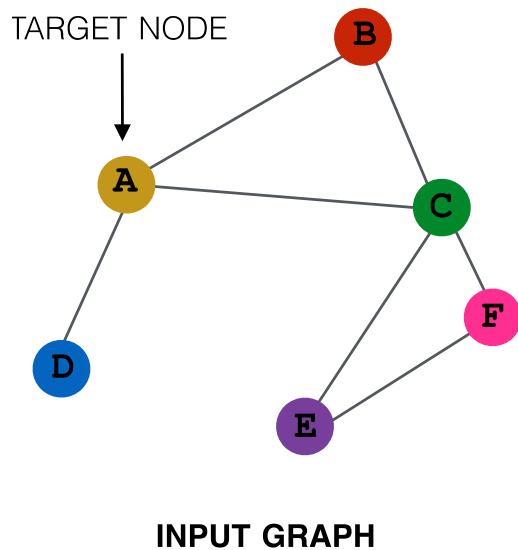
Propagate and  
transform information

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

Credit: [Stanford CS224W](https://stanford.edu/cs224w/)

# Idea: Aggregate Neighbors

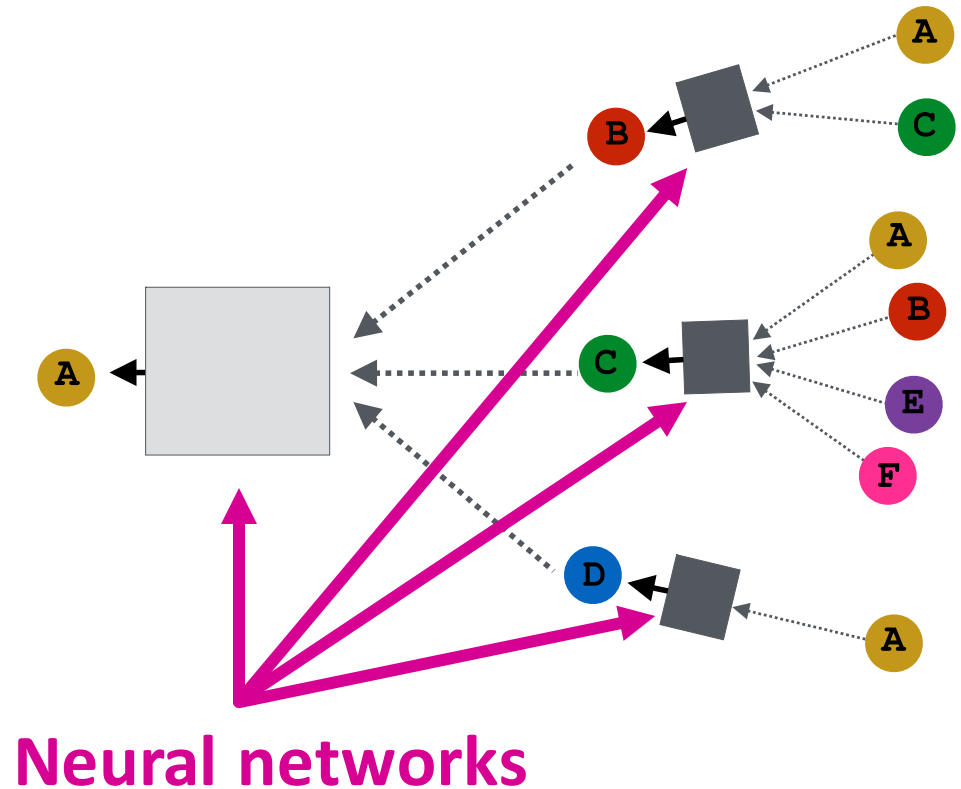
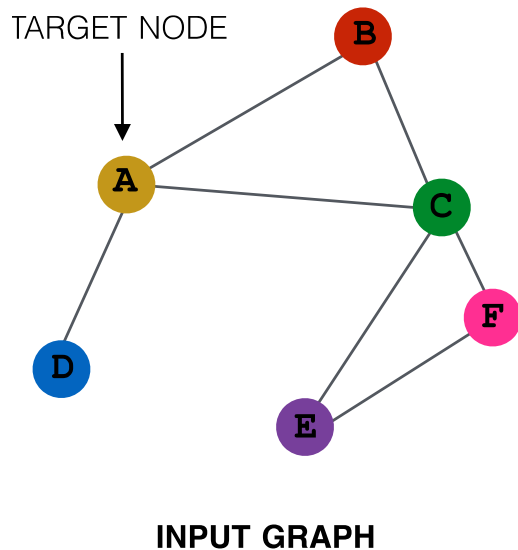
- **Key idea:** Generate node embeddings based on **local network neighborhoods**



Credit: [Stanford CS224W](#)

# Idea: Aggregate Neighbors

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

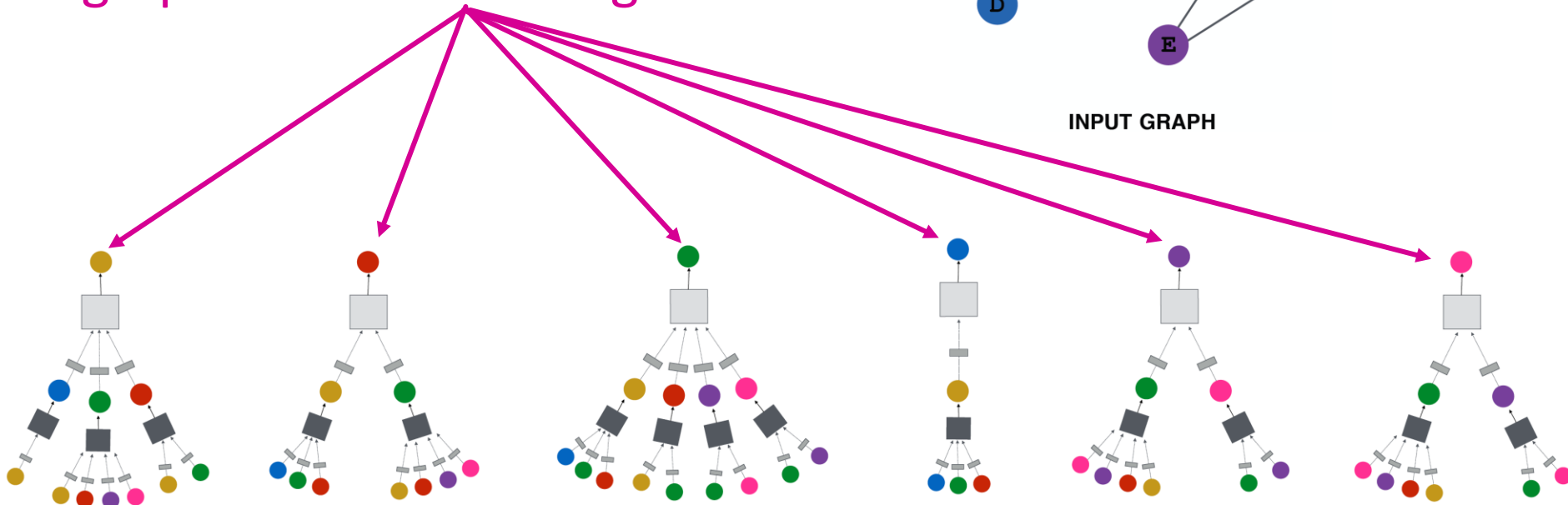
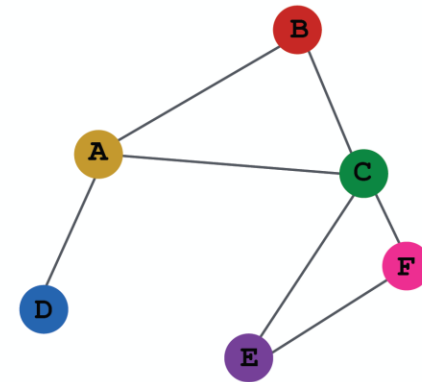


Credit: [Stanford CS224W](#)

# Idea: Aggregate Neighbors

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

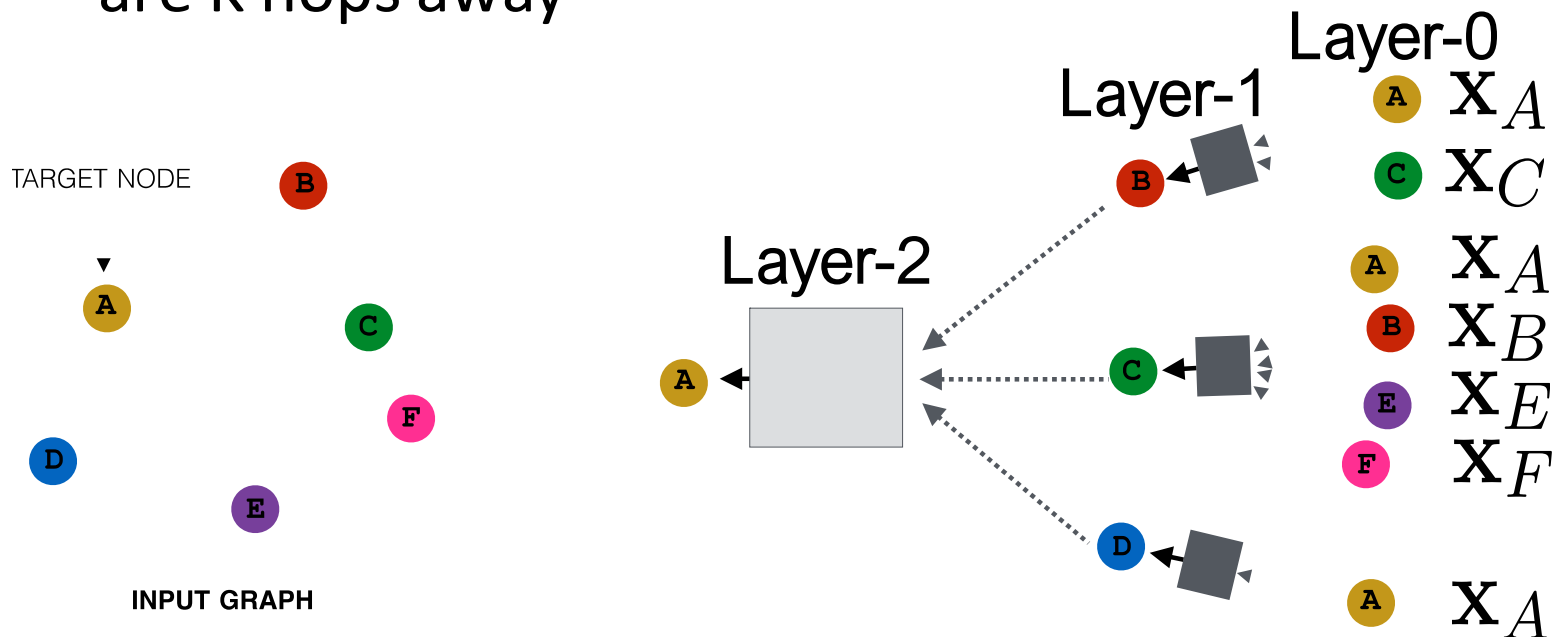
Every node defines a computation graph based on its neighborhood!



Credit: [Stanford CS224W](#)

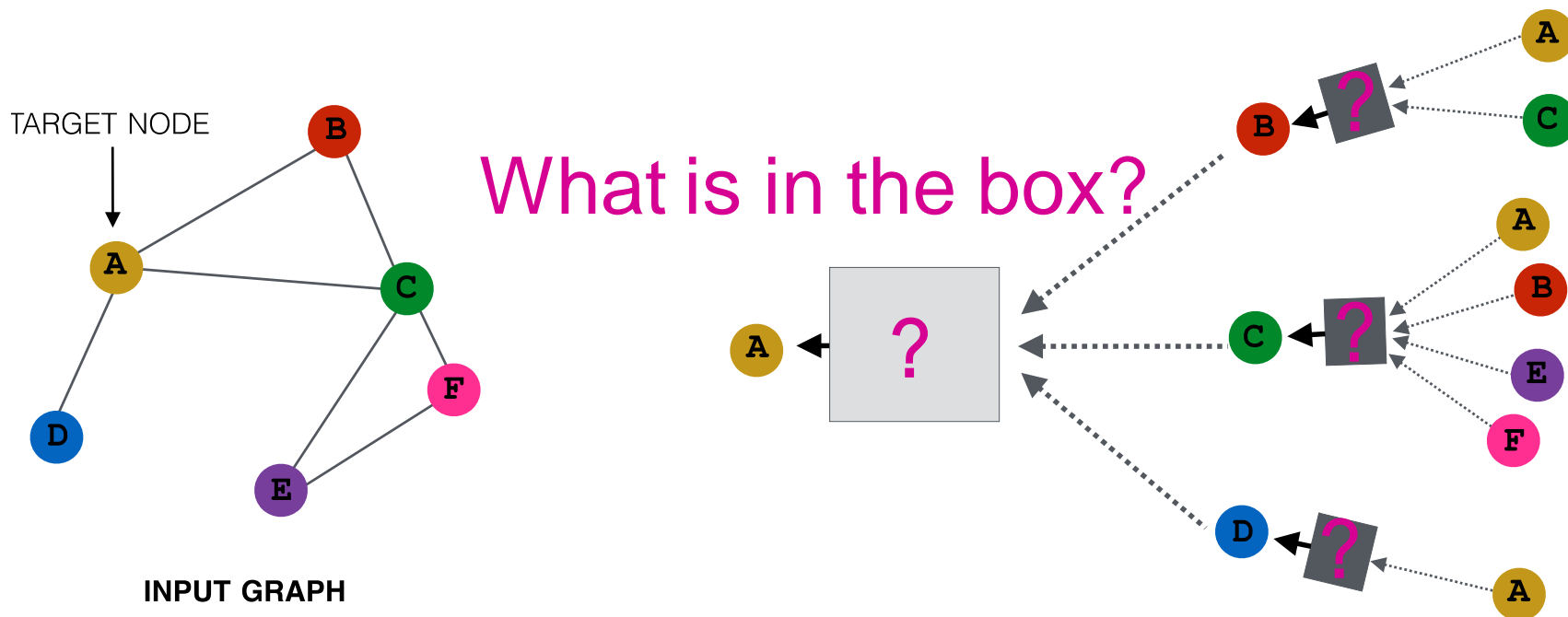
# Deep Model: Many Layers

- 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

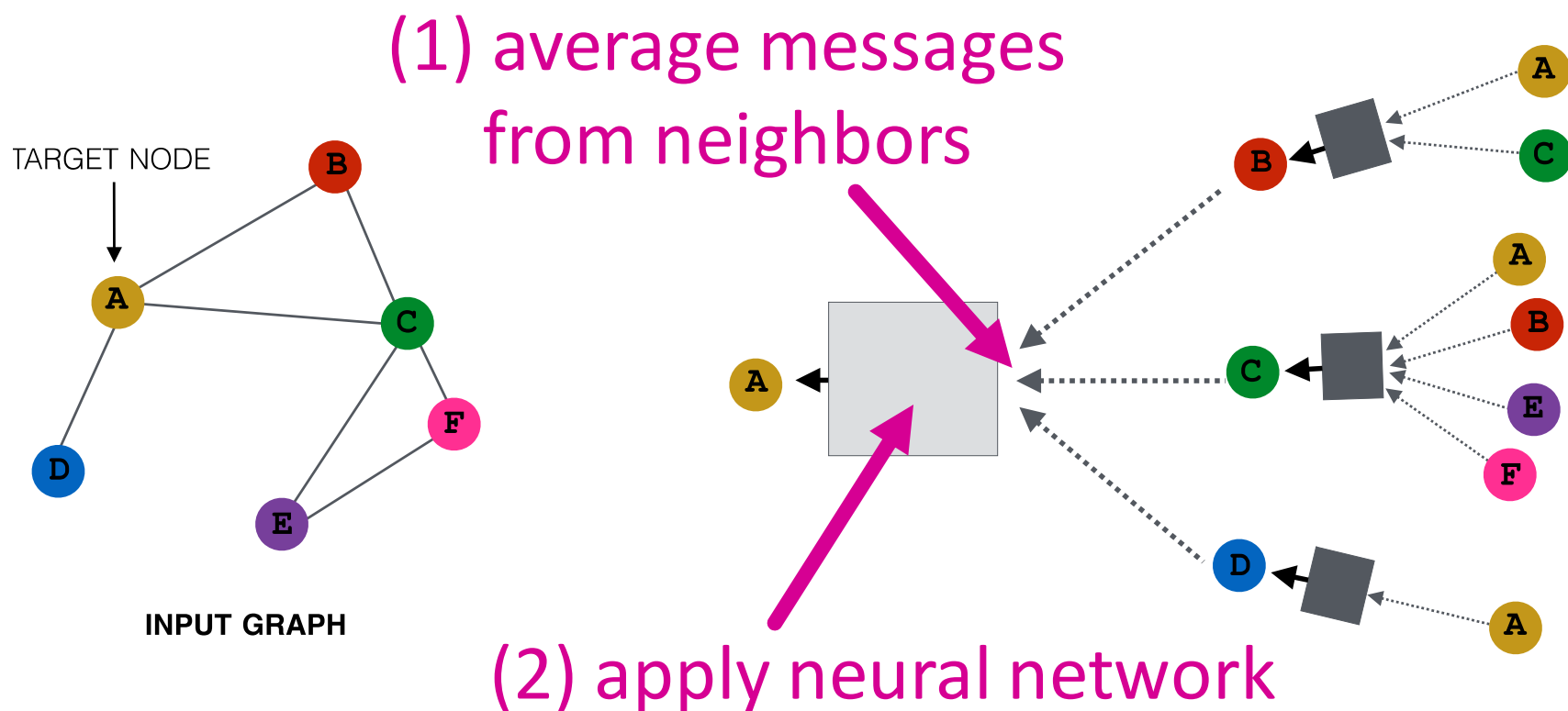
- **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers



Credit: [Stanford CS224W](#)

# Neighborhood Aggregation

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



Credit: [Stanford CS224W](#)

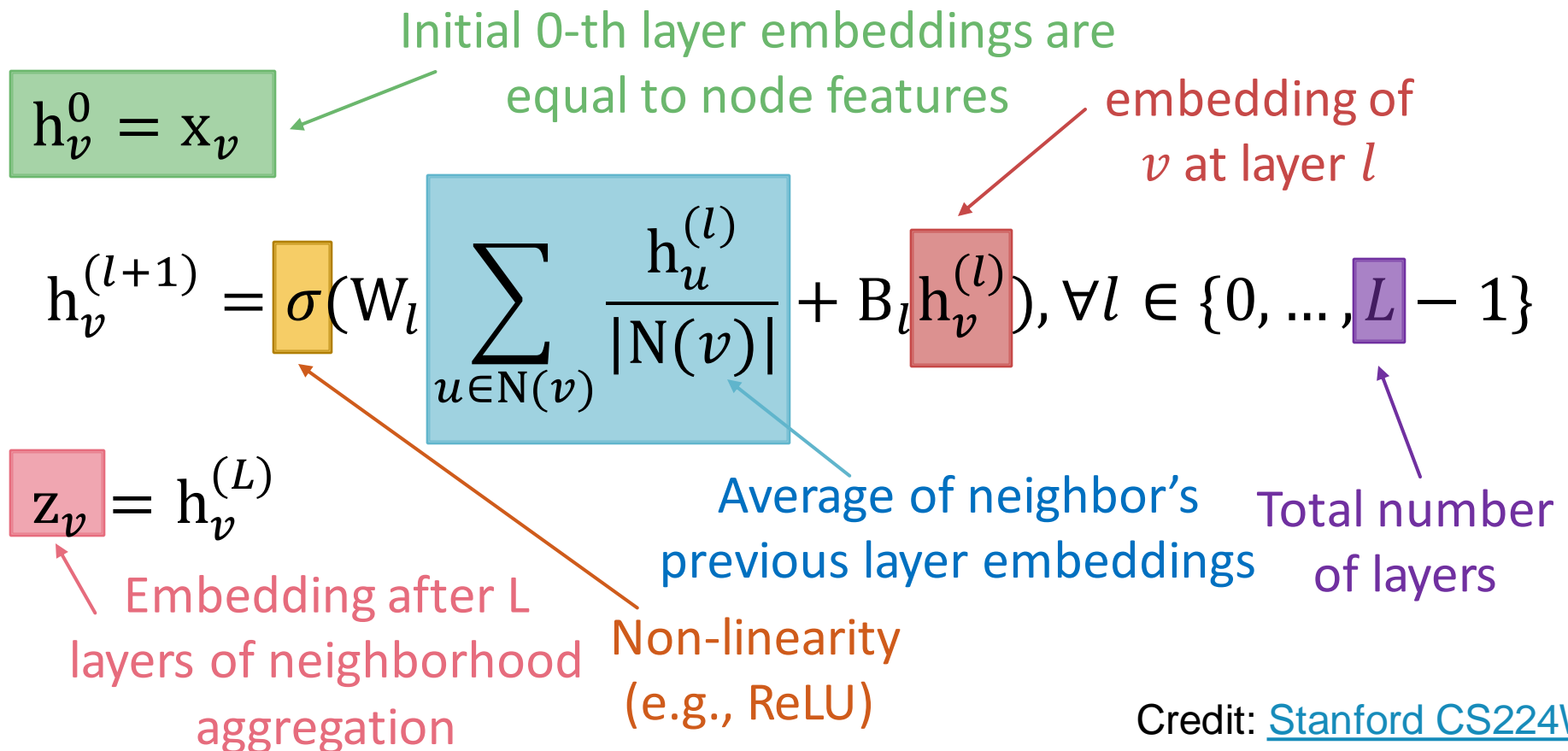


# Setup: Learning from Graphs

- 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**
  - $v$ : a node in  $V$ ;  $N(v)$ : the set of neighbors of  $v$ .
  - **Node features:**
    - Relational data: User/item descriptions, categories
    - Social networks: User profile, User image
    - Biological networks: Gene expression profiles, gene functional information
    - What if there is no node feature in the graph dataset?

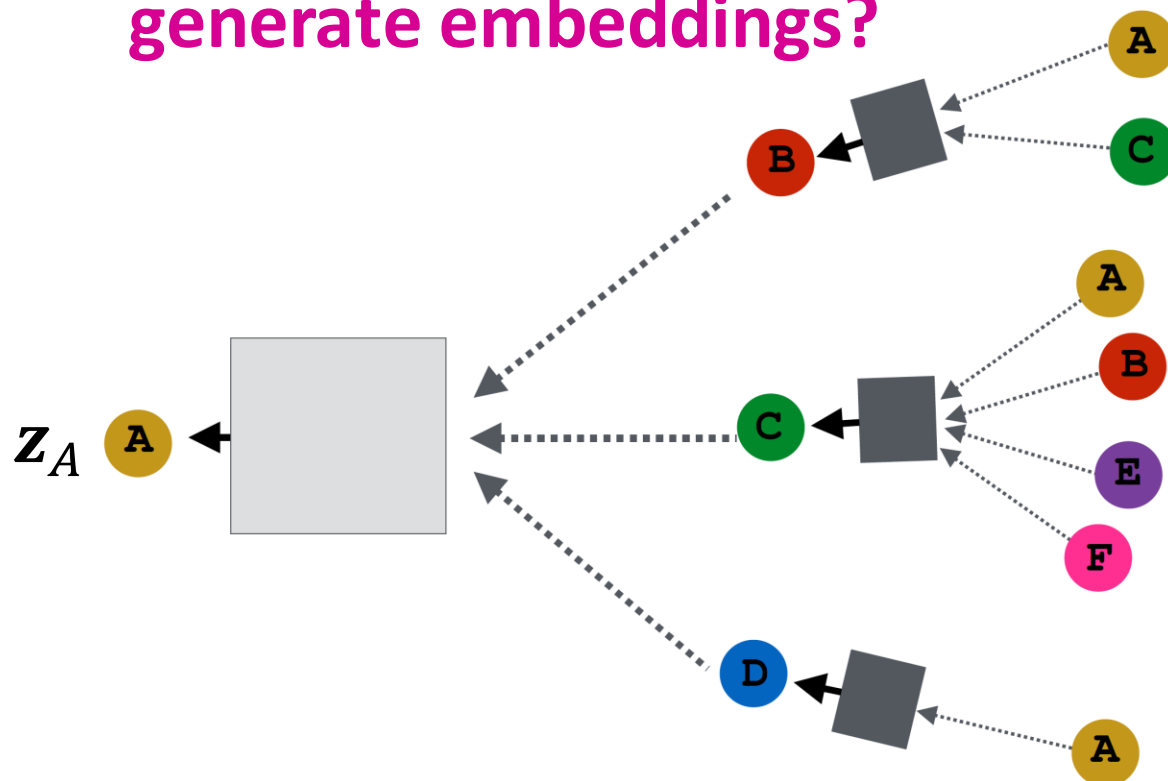
# The Math: Deep Encoder

- **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

Credit: [Stanford CS224W](#)

# How to train a GNN

- Node embedding  $\mathbf{z}_v$  is a function of input graph
- **Supervised setting**: we want to minimize the loss  $\mathcal{L}$ :

$$\min_{\Theta} \mathcal{L}(\mathbf{y}, f(\mathbf{z}_v))$$

- $\mathbf{y}$ : node label
- $\mathcal{L}$  could be L2 if  $\mathbf{y}$  is real number, or cross entropy if  $\mathbf{y}$  is categorical

# Model Parameters

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

$$\begin{aligned}h_v^{(0)} &= x_v \\h_v^{(l+1)} &= \sigma\left(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}\right), \forall l \in \{0, \dots, L-1\} \\z_v &= h_v^{(L)}\end{aligned}$$

Final node embedding

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

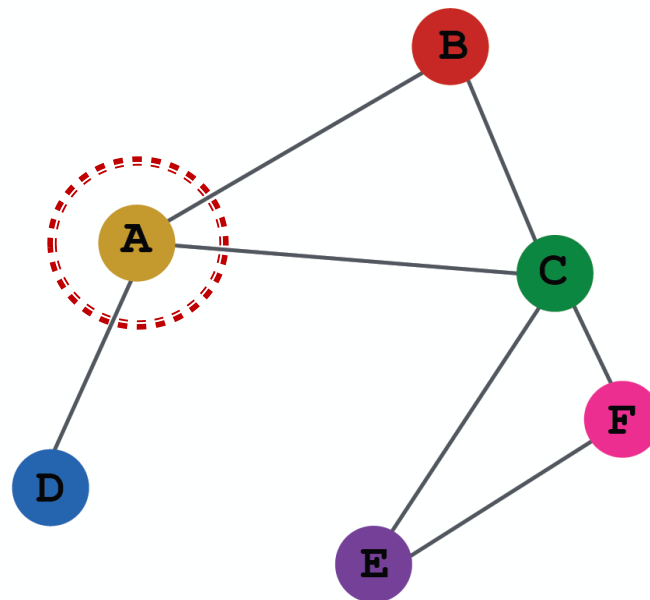
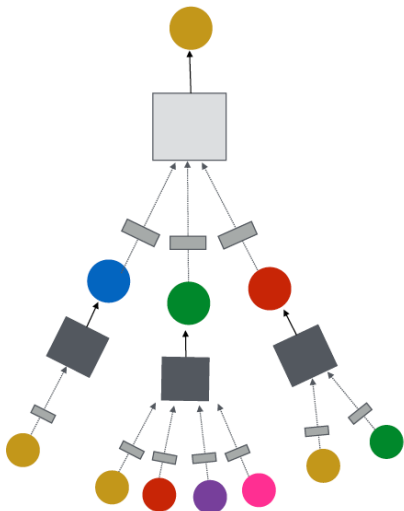
- $h_v^l$ : the hidden representation of node  $v$  at layer  $l$
- $W_l$ : weight matrix for neighborhood aggregation
  - $B_l$ : weight matrix for transforming hidden vector of self

Credit: [Stanford CS224W](#)

# Supervised Training

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

Is user going to churn in the next week?



E.g., a relational graph of users, sales, products

# Supervised Training

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

- Use cross entropy loss

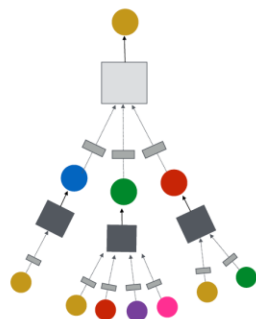
$$\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta))$$

Encoder output:  
node embedding

Classification  
weights

Node class  
label

Is user going to churn  
in the next week?



Credit: [Stanford CS224W](#)

# Designing a GNN

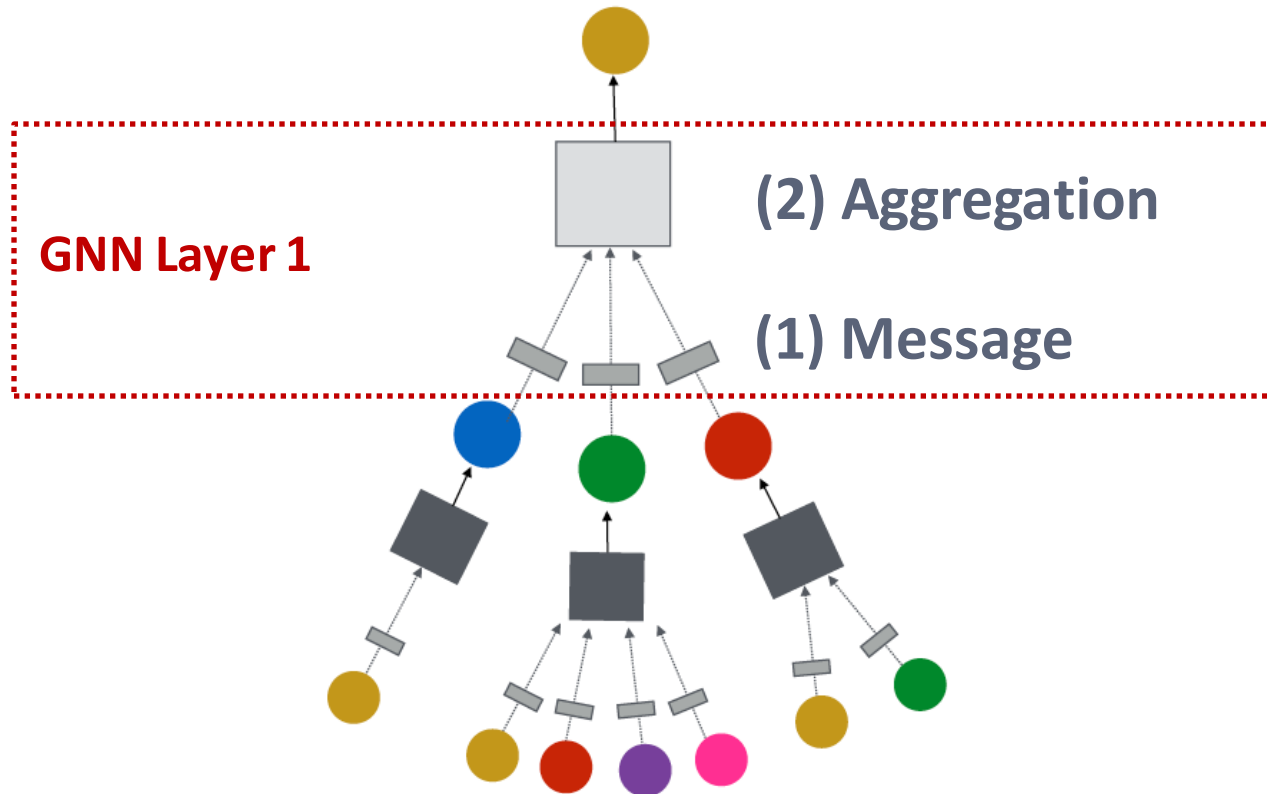
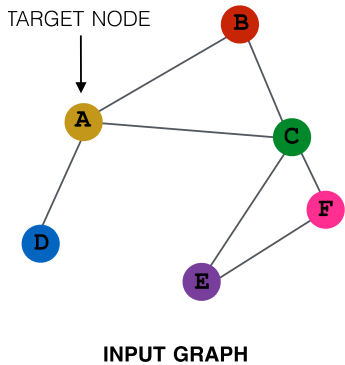




# A General GNN Framework (1)

## GNN Layer = Message + Aggregation

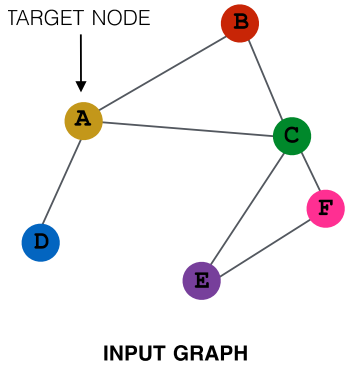
- Different instantiations under this perspective
- GCN, GraphSAGE, GAT, ...



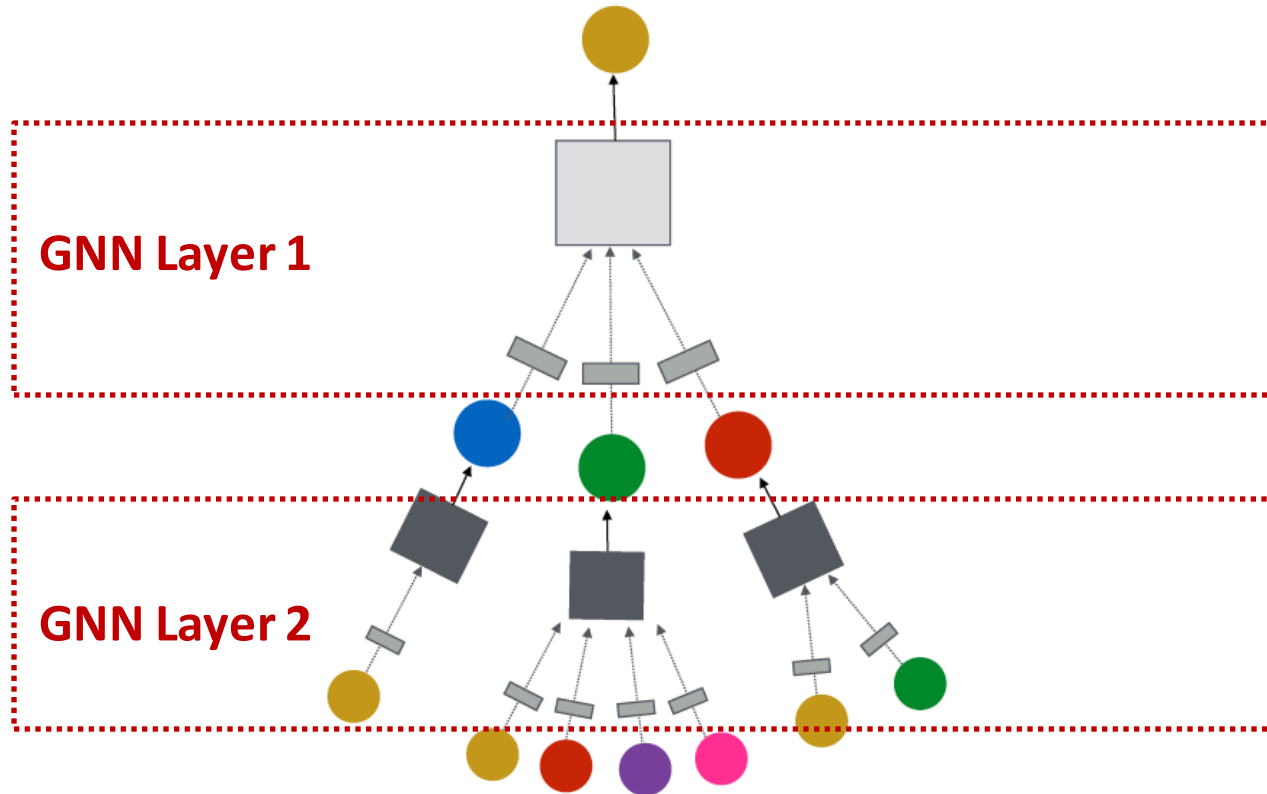
# A General GNN Framework (2)

## Connect GNN layers into a GNN

- Stack layers sequentially
- Ways of adding skip connections



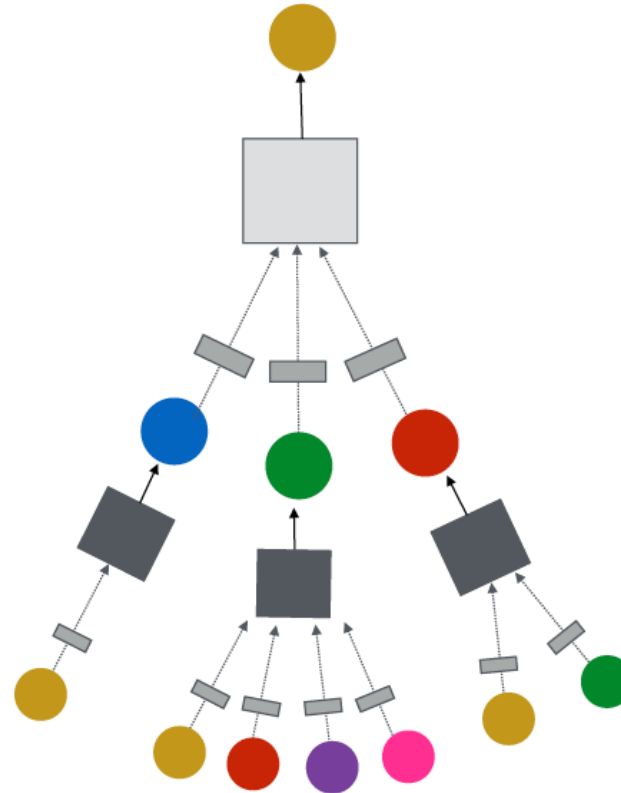
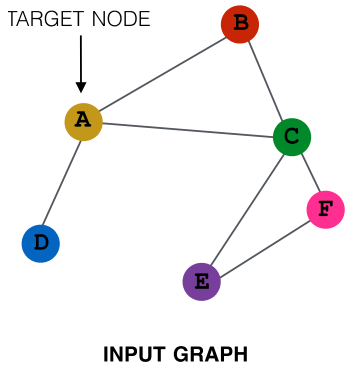
### (3) Layer connectivity



# A General GNN Framework (3)

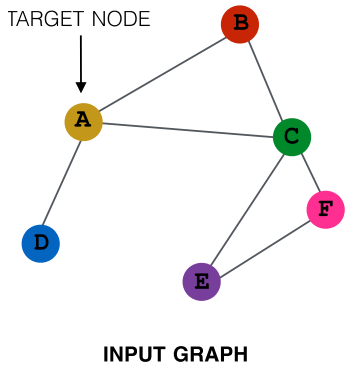
**Idea: Raw input graph  $\neq$  computational graph**

- Graph feature augmentation
- Graph structure augmentation



**(4) Graph augmentation**

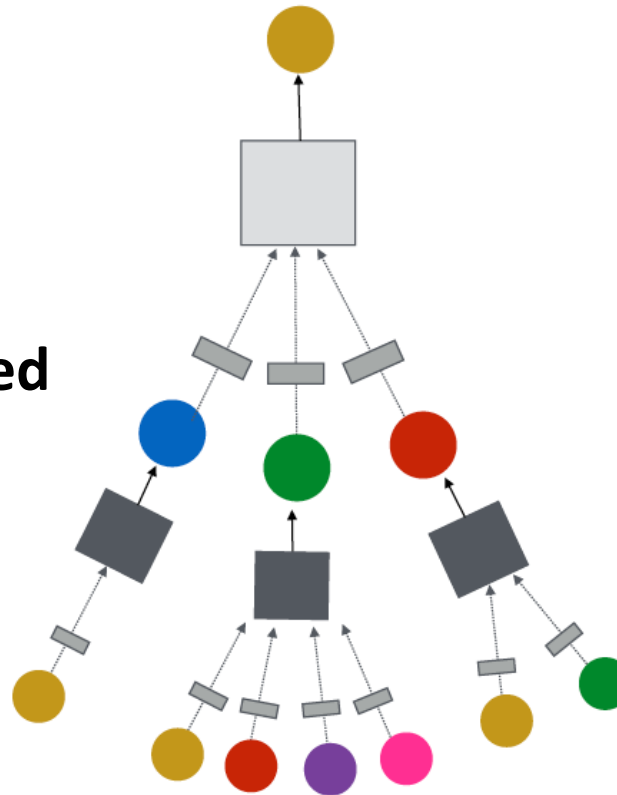
# A General GNN Framework (4)



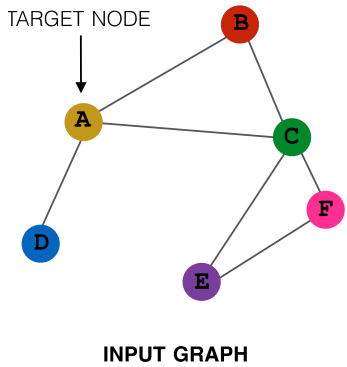
## (5) Learning objective

### How do we train a GNN

- Supervised/Unsupervised objectives
- Node/Edge/Graph level objectives

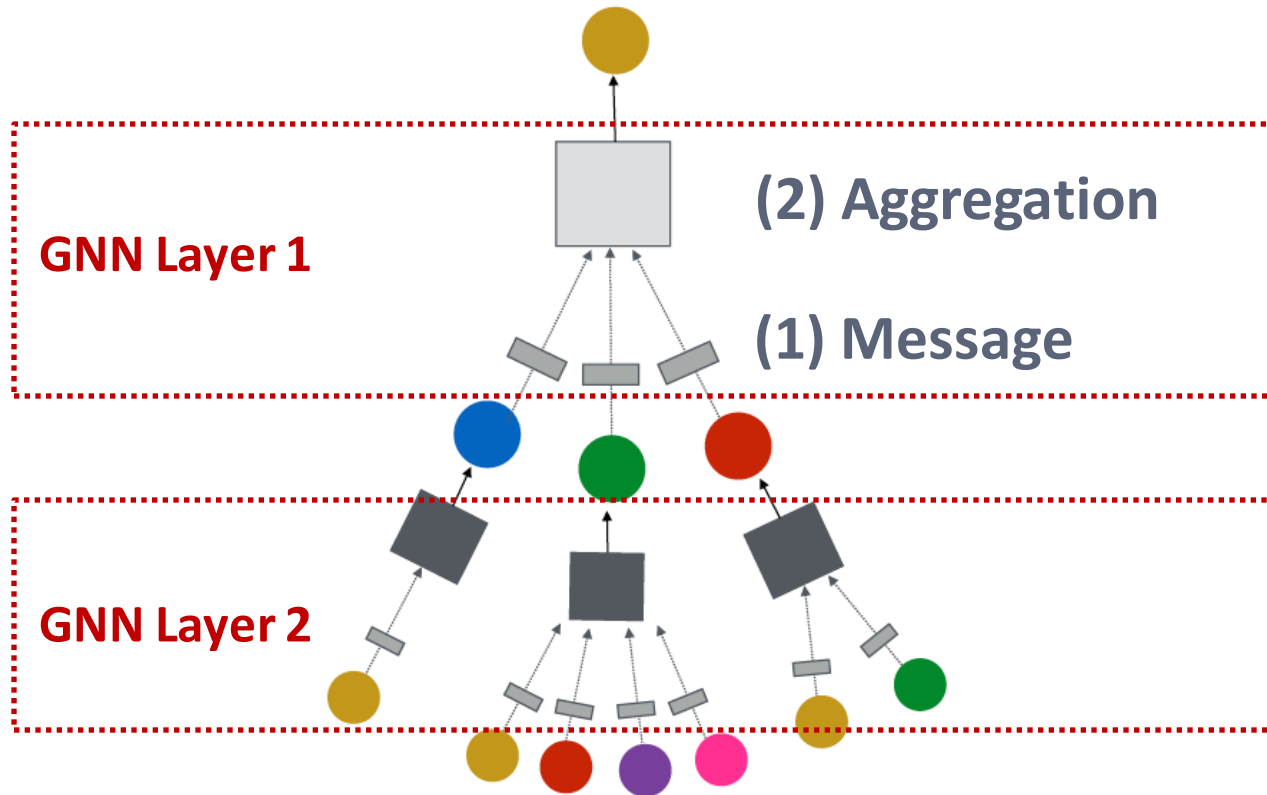


# A General GNN Framework (5)



(5) Learning objective

(3) Layer connectivity



(4) Graph augmentation

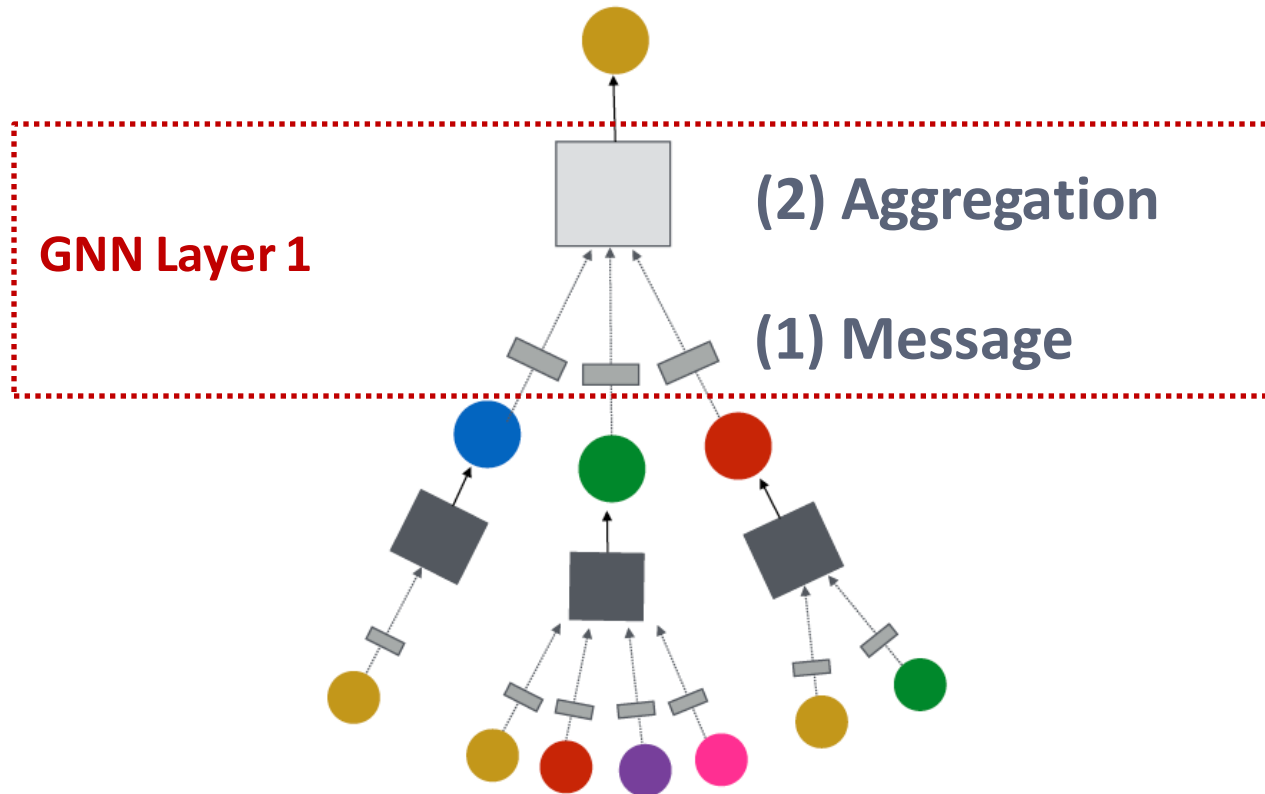
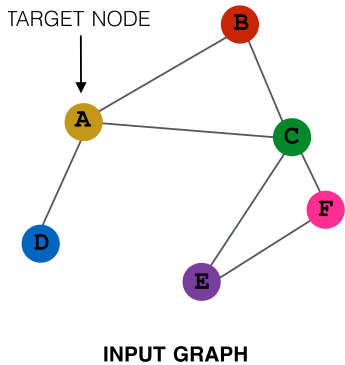
# A Single Layer of a GNN



# A GNN Layer

## GNN Layer = Message + Aggregation

- Different instantiations under this perspective
- GCN, GraphSAGE, GAT, ...



# A Single GNN Layer

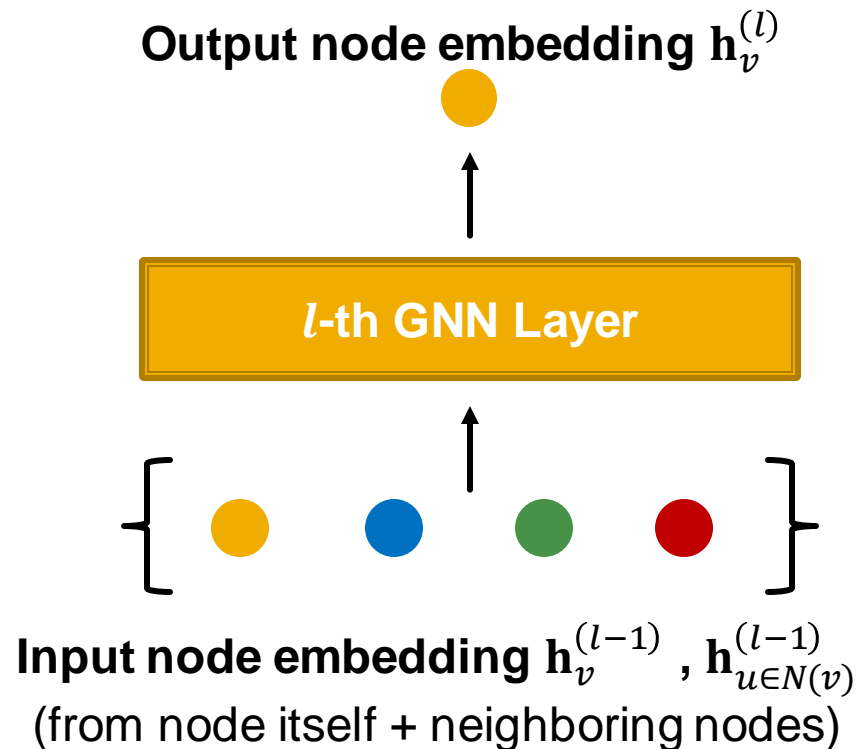
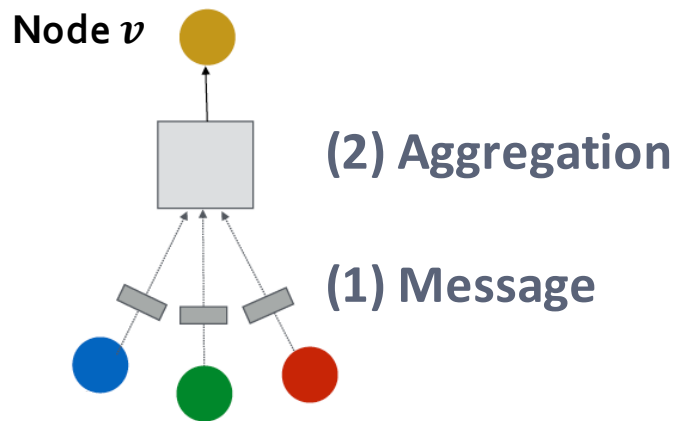
- **Idea of a GNN Layer:**

- Compress a set of vectors into a single vector

- **Two step process:**

- (1) Message

- (2) Aggregation





# Message Computation

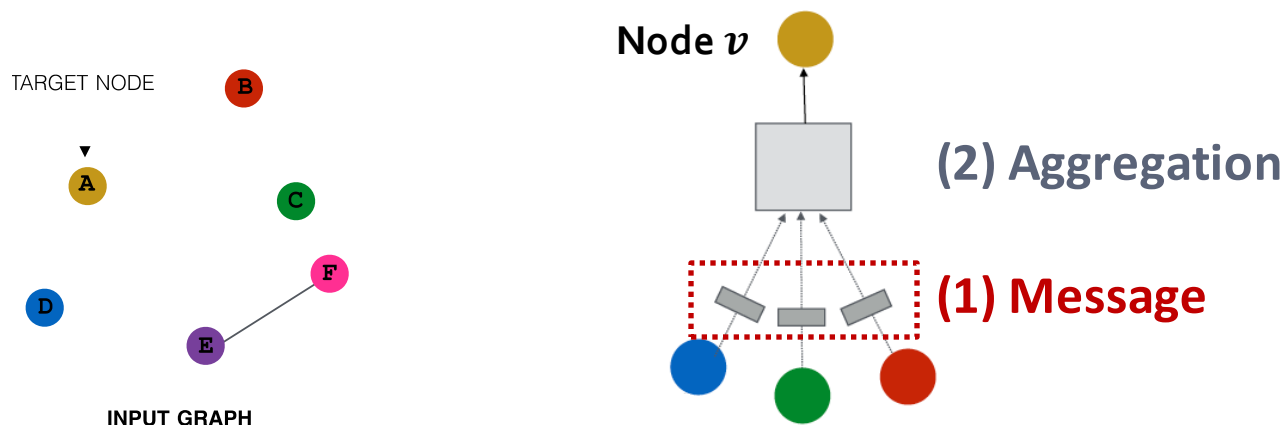
## ■ (1) Message computation

■ **Message function:**  $\mathbf{m}_u^{(l)} = \text{MSG}^{(l)} \left( \mathbf{h}_u^{(l-1)} \right)$

■ **Intuition:** Each node will create a message, which will be sent to other nodes later

■ **Example:** A Linear layer  $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

■ Multiply node features with weight matrix  $\mathbf{W}^{(l)}$



# Message Aggregation

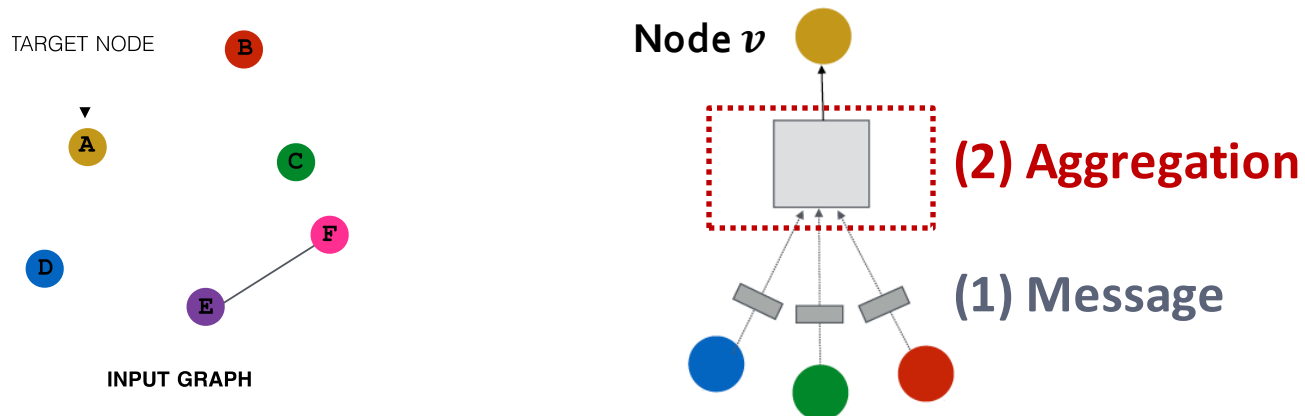
## ■ (2) Aggregation

- **Intuition:** Each node will aggregate the messages from node  $v$ 's neighbors

$$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)} \left( \left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right)$$

- **Example:** Sum( $\cdot$ ), Mean( $\cdot$ ) or Max( $\cdot$ ) aggregator

- $\mathbf{h}_v^{(l)} = \text{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$



# Message Aggregation: Issue

- **Issue:** Information from node  $v$  itself **could get lost**

- Computation of  $\mathbf{h}_v^{(l)}$  does not directly depend on  $\mathbf{h}_v^{(l-1)}$

- **Solution:** Include  $\mathbf{h}_v^{(l-1)}$  when computing  $\mathbf{h}_v^{(l)}$

- **(1) Message:** compute message from node  $v$  itself

- Usually, a **different message computation** will be performed

$$\bullet \bullet \bullet \quad \mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)} \qquad \bullet \quad \mathbf{m}_v^{(l)} = \mathbf{B}^{(l)} \mathbf{h}_v^{(l-1)}$$

- **(2) Aggregation:** After aggregating from neighbors, we can **aggregate the message from node  $v$  itself**

- Via **concatenation** or **summation**

**Then aggregate from node itself**

$$\mathbf{h}_v^{(l)} = \text{CONCAT} \left( \text{AGG} \left( \left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right), \mathbf{m}_v^{(l)} \right)$$

**First aggregate from neighbors**

# A Single GNN Layer

- **Putting things together:**

- **(1) Message:** each node computes a message

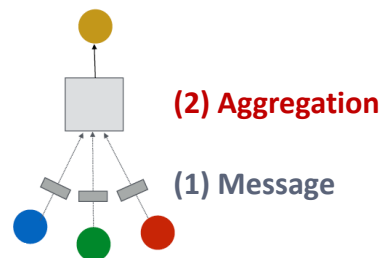
$$\mathbf{m}_u^{(l)} = \text{MSG}^{(l)} \left( \mathbf{h}_u^{(l-1)} \right), u \in \{N(v) \cup v\}$$

- **(2) Aggregation:** aggregate messages from neighbors

$$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)} \left( \left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\}, \mathbf{m}_v^{(l)} \right)$$

- **Nonlinearity (activation):** Adds expressiveness

- Often written as  $\sigma(\cdot)$ :  $\text{ReLU}(\cdot)$ ,  $\text{Sigmoid}(\cdot)$ , ...
- Can be added to **message or aggregation**



# Classical GNN Layers: GCN (1)

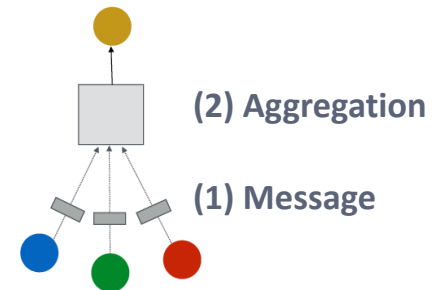
## ■ (1) Graph Convolutional Networks (GCN)

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

## ■ How to write this as Message + Aggregation?

$$\mathbf{h}_v^{(l)} = \sigma \left( \underbrace{\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}_{\text{Aggregation}} \right)$$

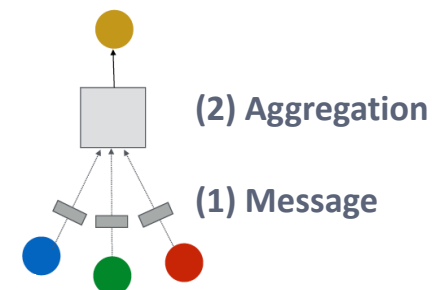
Message



# Classical GNN Layers: GCN (2)

## ■ (1) Graph Convolutional Networks (GCN)

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



### ■ Message:

- Each Neighbor:  $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

**Normalized by node degree**  
(In the GCN paper they use a slightly different normalization)

### ■ Aggregation:

- **Sum** over messages from neighbors, then apply activation

- $\mathbf{h}_v^{(l)} = \sigma \left( \text{Sum} \left( \left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right) \right)$

# Classical GNN Layers: GraphSAGE

## ■ (2) GraphSAGE

$$\mathbf{h}_v^{(l)} = \sigma \left( \mathbf{W}^{(l)} \cdot \text{CONCAT} \left( \mathbf{h}_v^{(l-1)}, \text{AGG} \left( \left\{ \mathbf{h}_u^{(l-1)}, \forall u \in N(v) \right\} \right) \right) \right)$$

## ■ How to write this as Message + Aggregation?

■ **Message** is computed within the  $\text{AGG}(\cdot)$

■ **Two-stage aggregation**

■ **Stage 1:** Aggregate from node neighbors

$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \text{AGG} \left( \left\{ \mathbf{h}_u^{(l-1)}, \forall u \in N(v) \right\} \right)$$

■ **Stage 2:** Further aggregate over the node itself

$$\mathbf{h}_v^{(l)} \leftarrow \sigma \left( \mathbf{W}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)}) \right)$$

# Classical GNN Layers: GAT (1)

## ■ (3) Graph Attention Networks

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

Attention weights

## ■ In GCN / GraphSAGE

- $\alpha_{vu} = \frac{1}{|N(v)|}$  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 (node degree)
- $\Rightarrow$  **All neighbors  $u \in N(v)$  are equally important to node  $v$**



# Classical GNN Layers: GAT (2)

Can we do better than simple neighborhood aggregation?

Can we let weighting factors  $\alpha_{vu}$  to be learned?

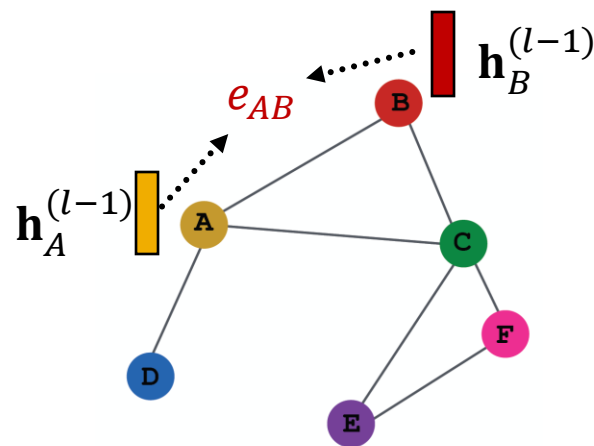
- **Goal:** Specify **arbitrary importance** to different neighbors of each node in the graph
- **Idea:** Compute embedding  $\mathbf{h}_v^{(l)}$  of each node in the graph following an **attention strategy**:
  - Implicitly specify different weights to different nodes in a neighborhood

# Attention Mechanism (1)

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

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

- $e_{vu}$  indicates the importance of  $u$ 's message to node  $v$



$$e_{AB} = a(\mathbf{W}^{(l)} \mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)} \mathbf{h}_B^{(l-1)})$$

# Attention Mechanism (2)

- **Normalize**  $e_{vu}$  into the **final attention weight**  $\alpha_{vu}$ 
  - Use the **softmax** function, so that  $\sum_{u \in N(v)} \alpha_{vu} = 1$ :

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

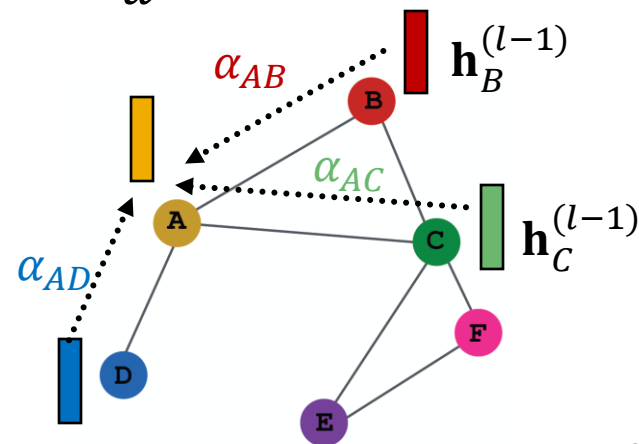
- **Weighted sum** based on the **final attention weight**

$\alpha_{vu}$

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

Weighted sum using  $\alpha_{AB}$ ,  $\alpha_{AC}$ ,  $\alpha_{AD}$ :

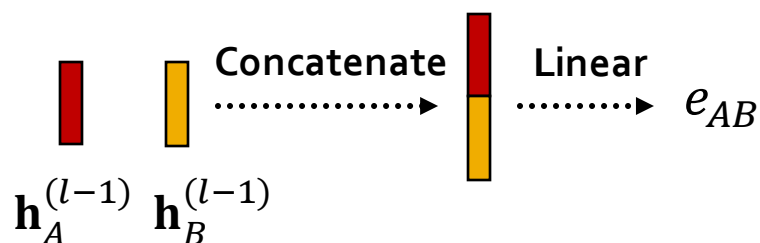
$$\mathbf{h}_A^{(l)} = \sigma\left(\alpha_{AB} \mathbf{W}^{(l)} \mathbf{h}_B^{(l-1)} + \alpha_{AC} \mathbf{W}^{(l)} \mathbf{h}_C^{(l-1)} + \alpha_{AD} \mathbf{W}^{(l)} \mathbf{h}_D^{(l-1)}\right)$$



# Attention Mechanism (3)

- **What is the form of attention mechanism  $a$ ?**

- E.g., use a simple single-layer neural network
  - $a$  have trainable parameters (weights in the Linear layer)



$$\begin{aligned} e_{AB} &= a\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right) \\ &= \text{Linear}\left(\text{Concat}\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)\right) \end{aligned}$$

- **Parameters of  $a$  are trained jointly:**
  - Learn the parameters together with weight matrices (i.e., other parameter of the neural net  $\mathbf{W}^{(l)}$ ) in an end-to-end fashion

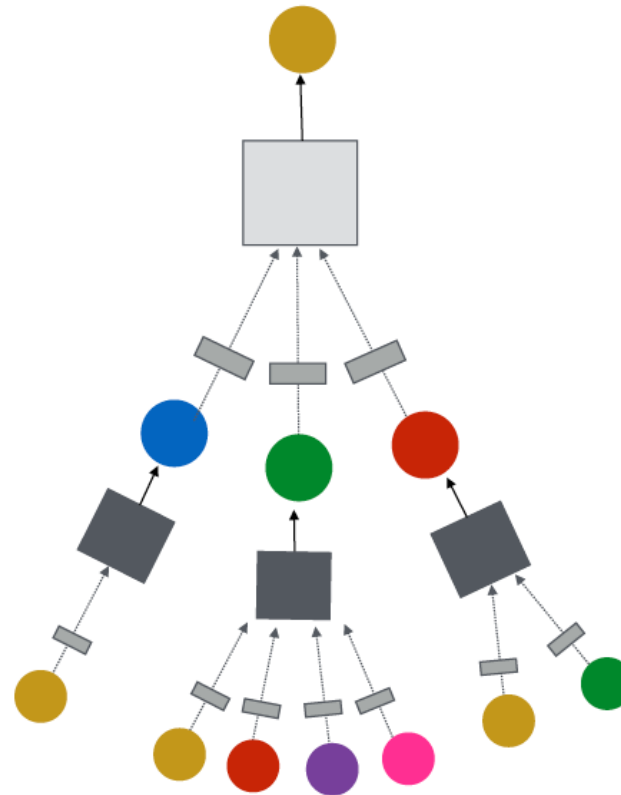
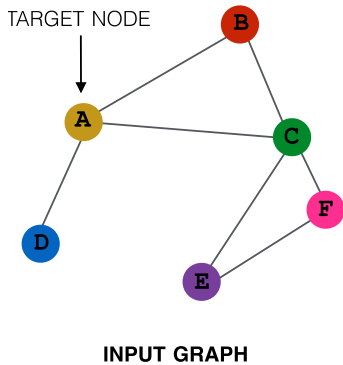
# Graph Manipulation in GNNs



# General GNN Framework

**Idea: Raw input graph  $\neq$  computational graph**

- Graph feature augmentation
- Graph structure manipulation



**(4) Graph manipulation**

# Why Manipulate Graphs

Our assumption so far has been

■ **Raw input graph = computational graph**

**Reasons for breaking this assumption**

■ **Feature level:**

■ The input graph **lacks features** → feature augmentation

■ **Structure level:**

■ The graph is **too sparse** → inefficient message passing

■ The graph is **too dense** → message passing is too costly

■ The graph is **too large** → cannot fit the computational graph into a GPU

■ It's just **unlikely that the input graph happens to be the optimal computation graph** for embeddings

# Graph Manipulation Approaches

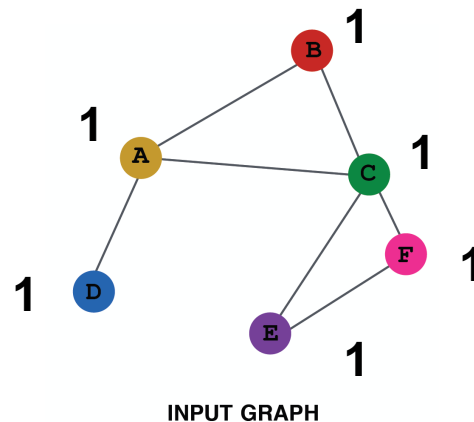
- **Graph Feature manipulation**
  - The input graph **lacks features** → **feature augmentation**
- **Graph Structure manipulation**
  - The graph is **too sparse** → **Add virtual nodes / edges**
  - The graph is **too dense** → **Sample neighbors when doing message passing**
  - The graph is **too large** → **Sample subgraphs to compute embeddings**
    - Will cover later in lecture: Scaling up GNNs



# Feature Augmentation on Graphs

## Why do we need feature augmentation?

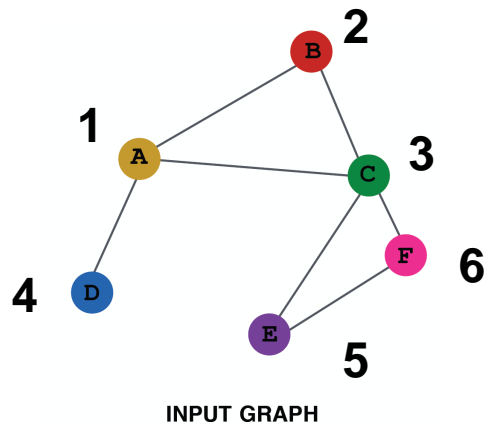
- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **a) Assign constant values to nodes**



# Feature Augmentation on Graphs

## Why do we need feature augmentation?

- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **b) Assign unique IDs to nodes**
  - These IDs are converted into **one-hot vectors**



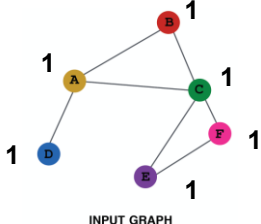
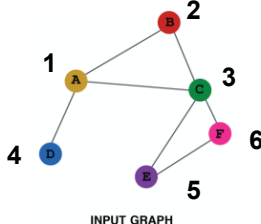
One-hot vector for node with ID=5

ID = 5  
↓  
[0, 0, 0, 0, 1, 0]

└──────────────────┘  
Total number of IDs = 6

# Feature Augmentation on Graphs

## ■ Feature augmentation: constant vs. one-hot

	Constant node feature	One-hot node feature
	 <p style="text-align: center;">INPUT GRAPH</p>	 <p style="text-align: center;">INPUT GRAPH</p>
<b>Expressive power</b>	<b>Medium.</b> All the nodes are identical, but <b>GNN can still learn from the graph structure</b>	<b>High.</b> Each node has a unique ID, so <b>node-specific information can be stored</b>
<b>Inductive learning (Generalize to unseen nodes)</b>	<b>High.</b> Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	<b>Low.</b> Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
<b>Computational cost</b>	<b>Low.</b> Only 1 dimensional feature	<b>High.</b> $O( V )$ dimensional feature, cannot apply to large graphs
<b>Use cases</b>	<b>Any graph, inductive settings (generalize to new nodes)</b>	<b>Small graph, transductive settings (no new nodes)</b>

# Feature Augmentation on Graphs

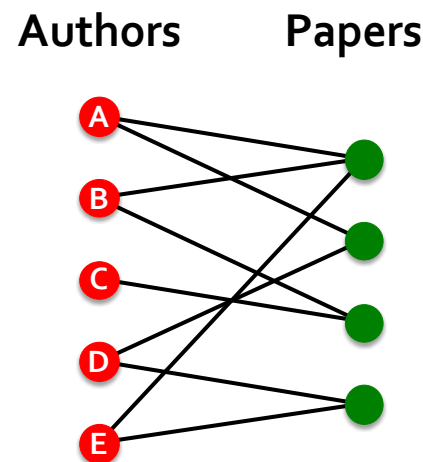
## Why do we need feature augmentation?

- **(2) Certain features can help GNN learning**
- Other commonly used augmented features:
  - Node degree
  - PageRank
  - Clustering coefficient
  - ...
- **Any useful graph statistics can be used!**

# Add Virtual Nodes / Edges

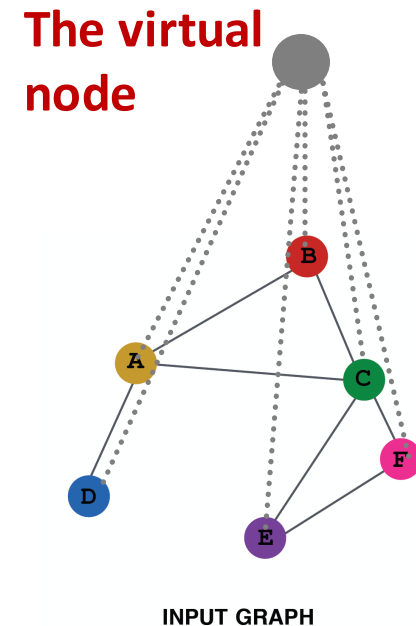
- **Motivation:** Augment sparse graphs
- **(1) Add virtual edges**
  - **Common approach:** Connect 2-hop neighbors via virtual edges
  - **Intuition:** Instead of using adj. matrix  $A$  for GNN computation, use  $A + A^2$

- **Use cases:** Bipartite graphs
  - Author-to-papers (they authored)
  - 2-hop virtual edges make an author-author collaboration graph



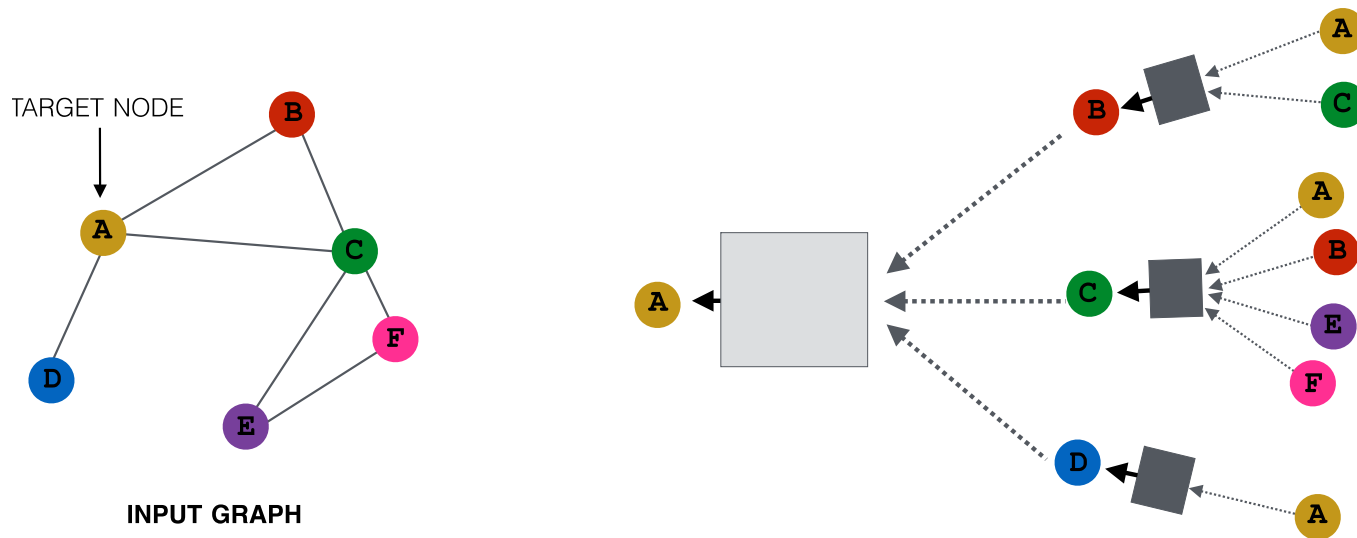
# Add Virtual Nodes / Edges

- **Motivation:** Augment sparse graphs
- **(2) Add virtual nodes**
  - The virtual node will connect to all the nodes in the graph
    - Suppose in a sparse graph, two nodes have shortest path distance of 10
    - After adding the virtual node, **all the nodes will have a distance of 2**
      - Node A – Virtual node – Node B
  - **Benefits:** Greatly **improves message passing in sparse graphs**



# Node Neighborhood Sampling

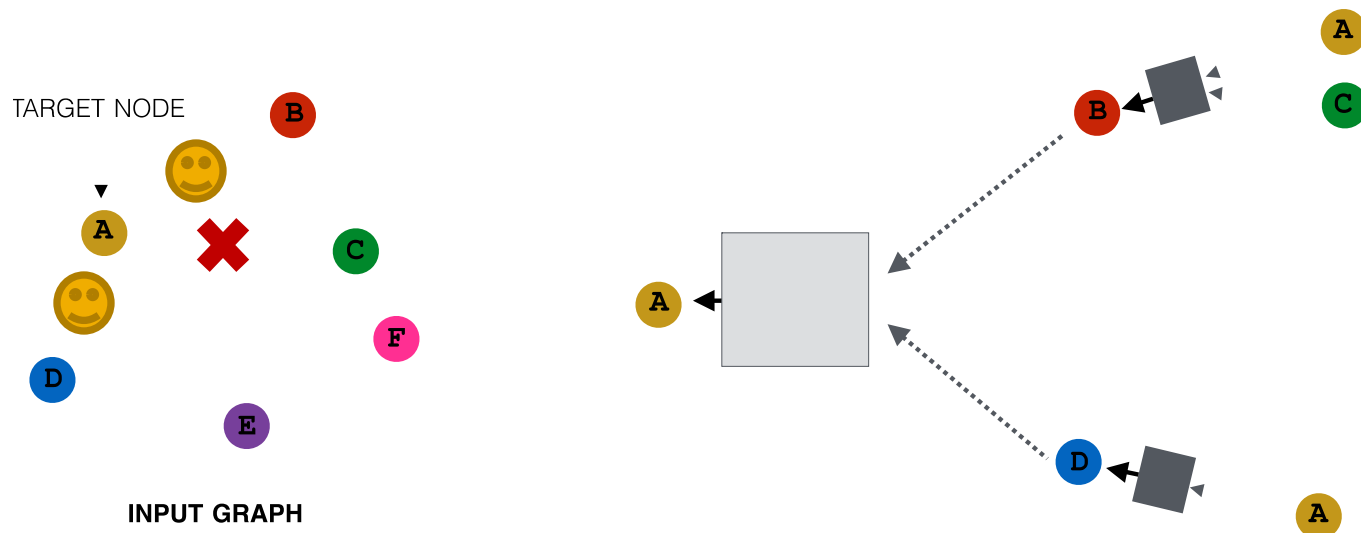
- **Previously:**
  - All the nodes are used for message passing



- **New idea:** (Randomly) sample a node's neighborhood for message passing

# Neighborhood Sampling Example

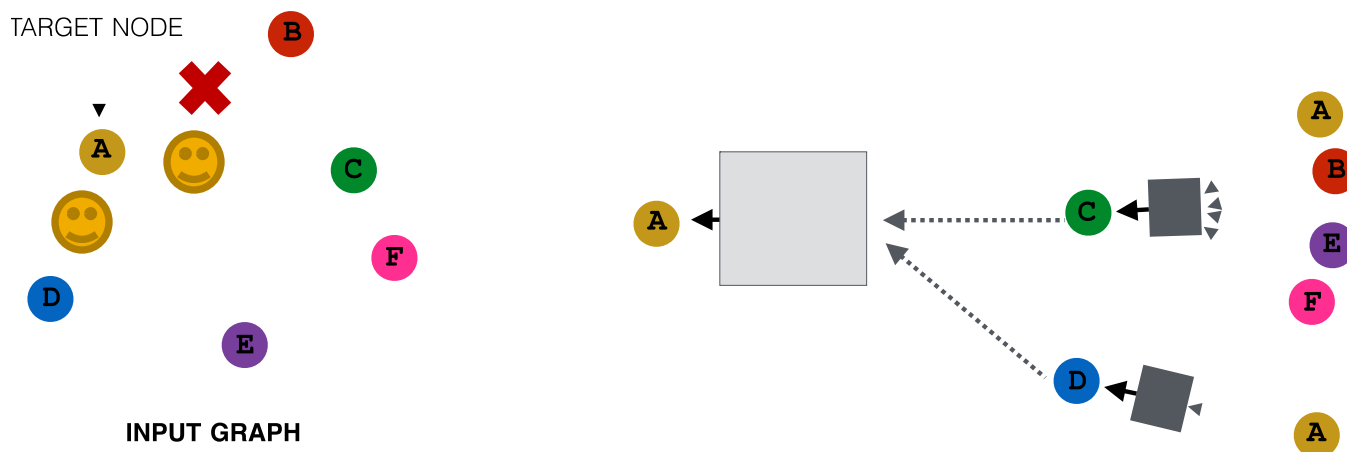
- For example, we can randomly choose 2 neighbors to pass messages
  - Only nodes  $B$  and  $D$  will pass message to  $A$





# Neighborhood Sampling Example

- Next time when we compute the embeddings, we can sample different neighbors
  - Only nodes *C* and *D* will pass message to *A*



# Neighborhood Sampling Example

- In expectation, we can get embeddings similar to the case where all the neighbors are used
  - **Benefits:** greatly reduce computational cost
  - And in practice it works great!

