Recommender Systems: Advanced

Jaemin Yoo

School of Electrical Engineering

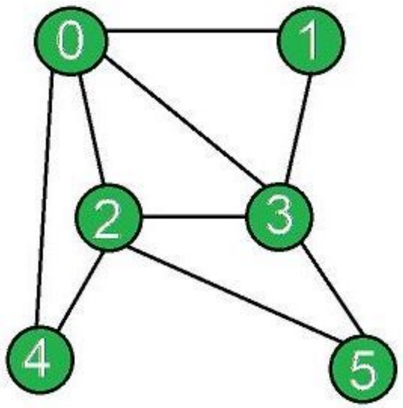
Kim Jaechul Graduate School of AI

Outline

1. **Graph-based recommendation**
2. Graph neural networks
3. Neural graph collaborative filtering
4. Summary

Graphs

- Data structure that represents connections and relationships.
  - A graph consists of **nodes** (e.g., users) and **edges** (e.g., friendship).
  - A graph is represented as a **sparse adjacency matrix**.
    - $a_{ij} = 1$ if nodes i and j are connected; $a_{ij} = 0$ otherwise.

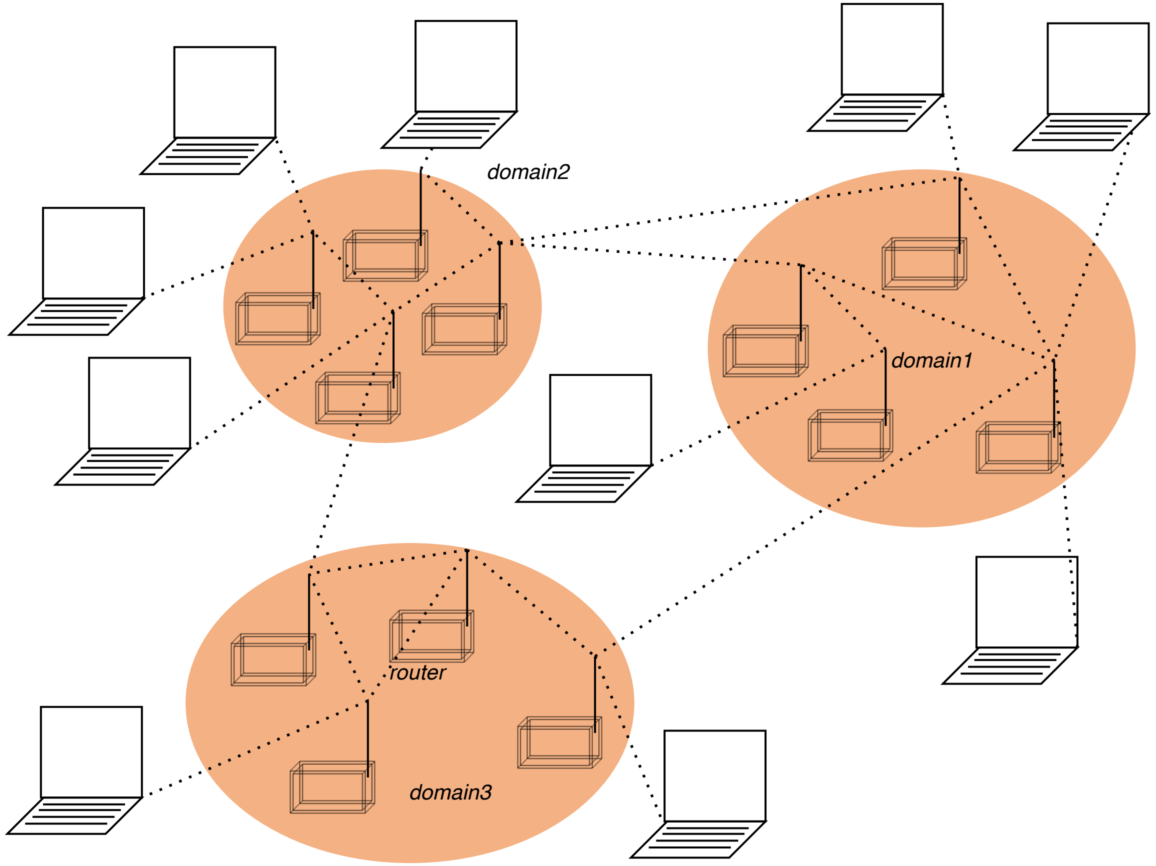


	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	1	1	1
3	1	1	1	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

Graph Data: Social Networks

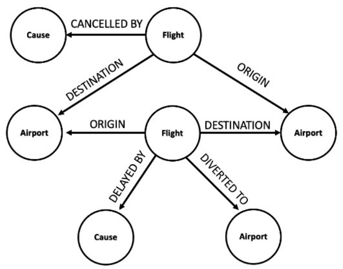
Source: [Backstrom et al., 2011]

Graph Data: Communication



Source: Stanford CS246

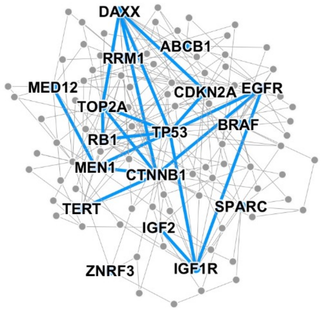
Various Types of Graphs



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

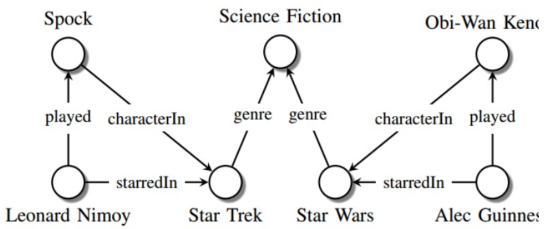


Image credit: Maximilian Nickel et al

Knowledge Graphs

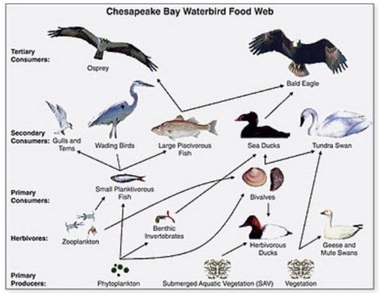


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

Image credit: visitlondon.com

Underground Networks

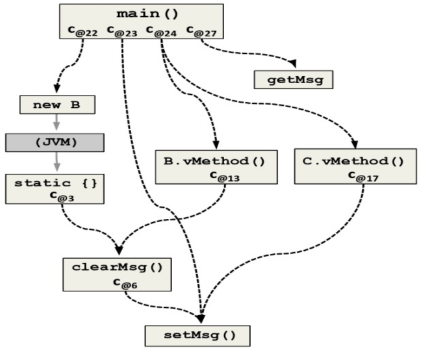


Image credit: ResearchGate

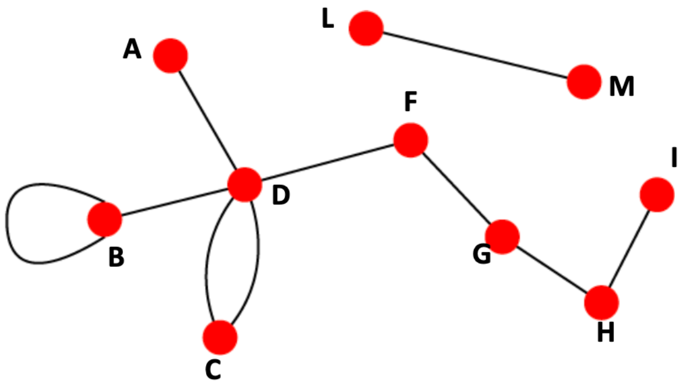
Code Graphs

Directed vs. Undirected Graphs

- We focus on **undirected graphs**, which is a simpler structure.

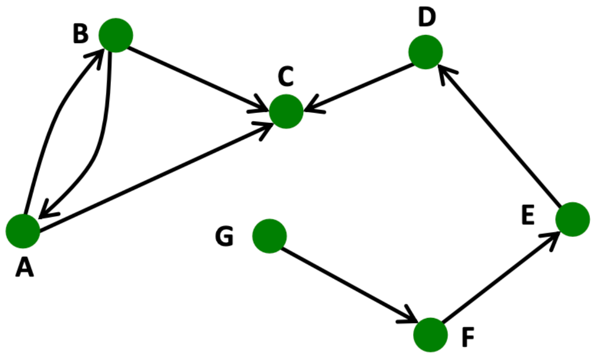
Undirected

- **Links:** undirected (symmetrical, reciprocal)



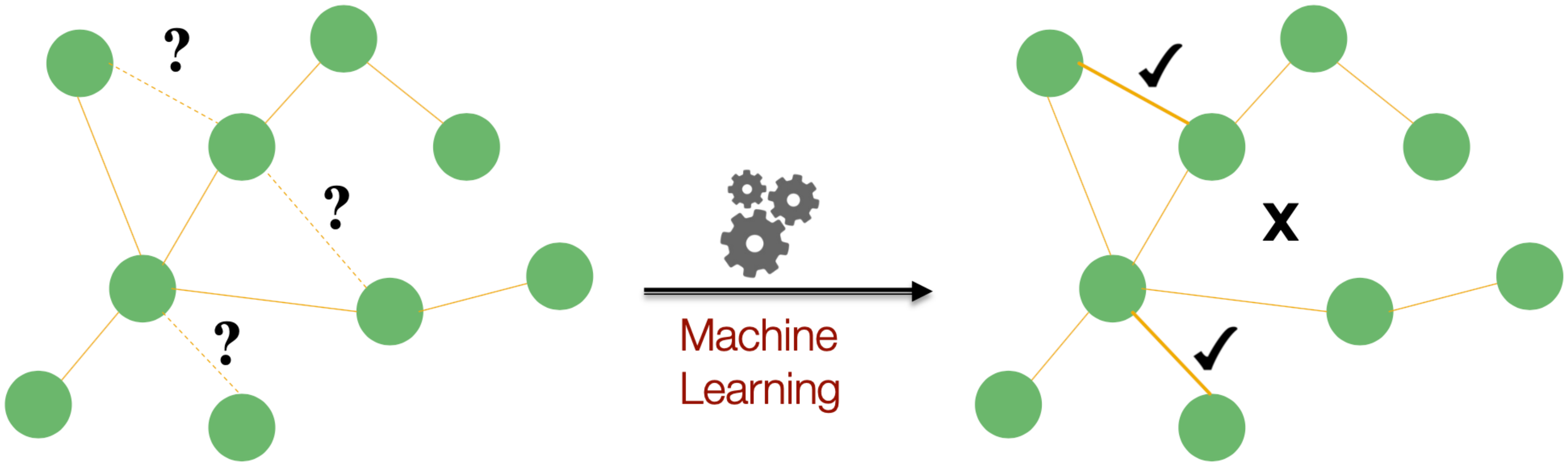
Directed

- **Links:** directed



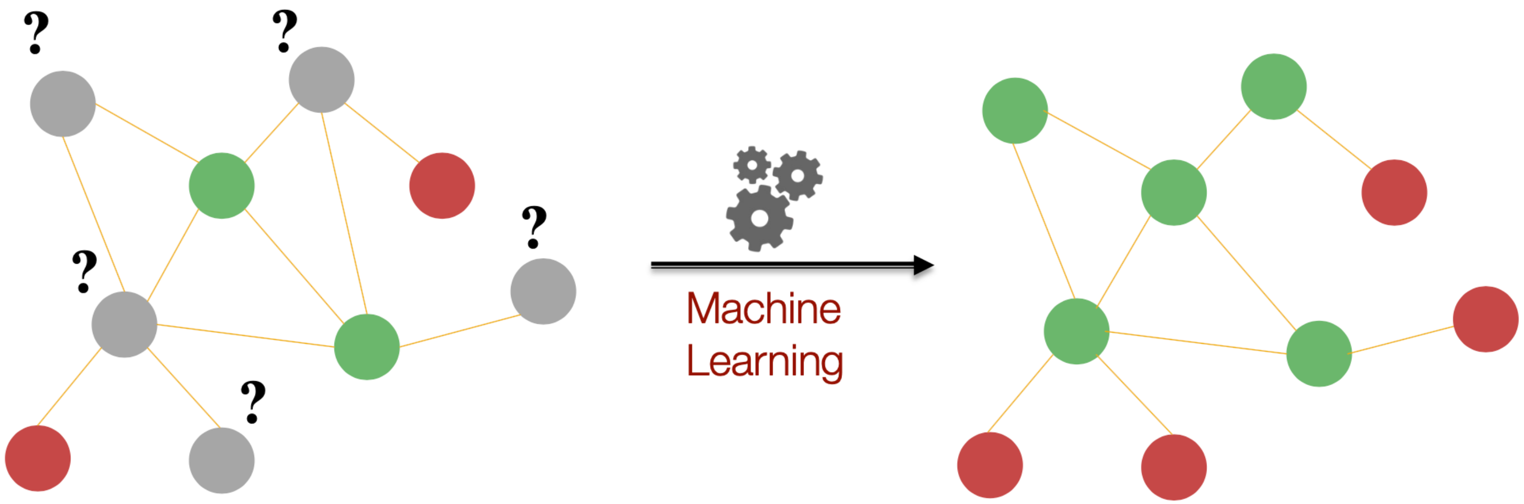
What to Do with Graphs: Link Prediction

- We can solve various tasks defined on graph datasets.
- **Link prediction** is to predict the appearance of new edges.
  - Identifying possible friends in Facebook.
  - Recommending new movies to users in Netflix.



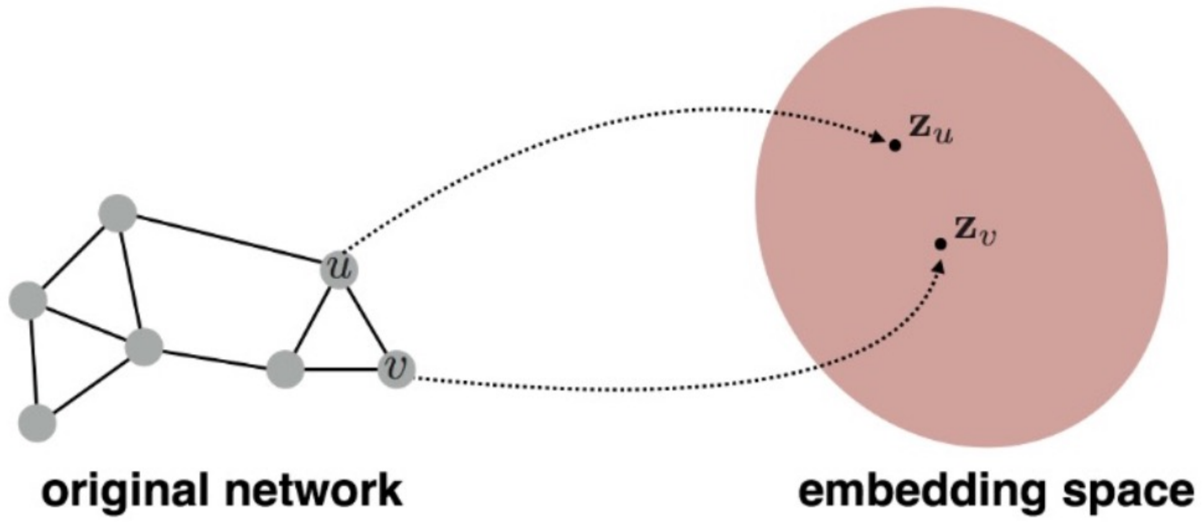
What to Do with Graphs: Node Classification

- **Node classification** is to classify each node in a graph.
  - Identifying political stance of users in Facebook.
  - Categorizing movies in Netflix, going over typical genres.



Representation Learning in Graphs

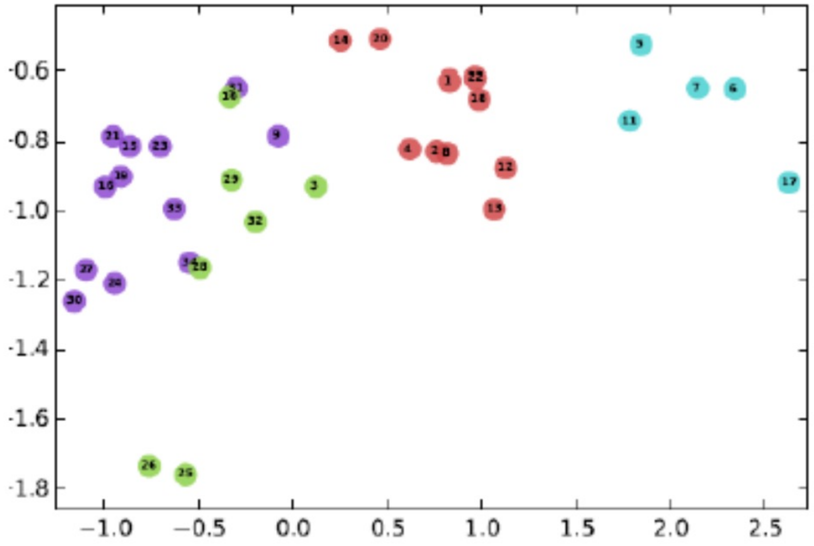
- Both tasks can be solved by **graph representation learning.**
  - **For** (i) nodes, (ii) subgraphs, or the (iii) entire graph.
  - **Learn** low-dimensional embeddings.
  - **Such that** their relationships reflect the graph structure.



Example of Node Embeddings

- 2D embedding of nodes of the Zachary’s Karate Club network
  - Each node is originally a $|V|(=34)$-dimensional sparse vector

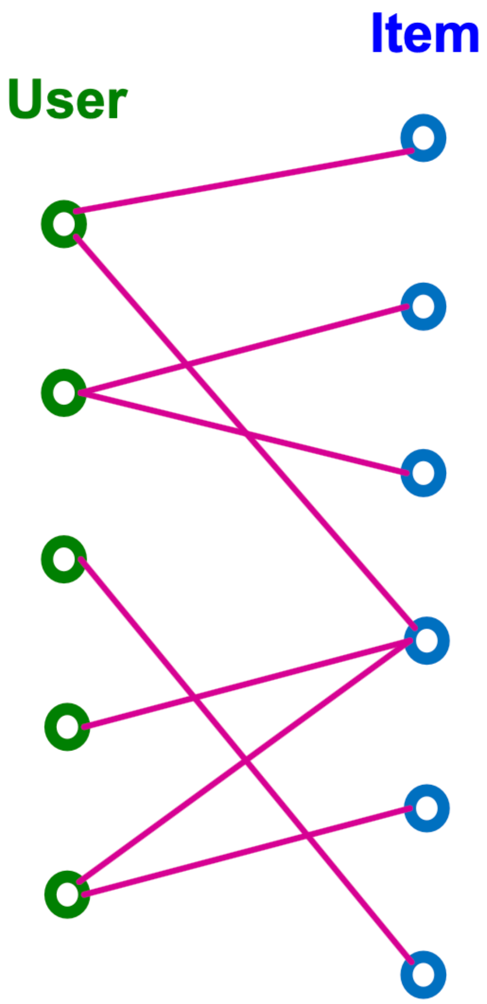
Input



Output

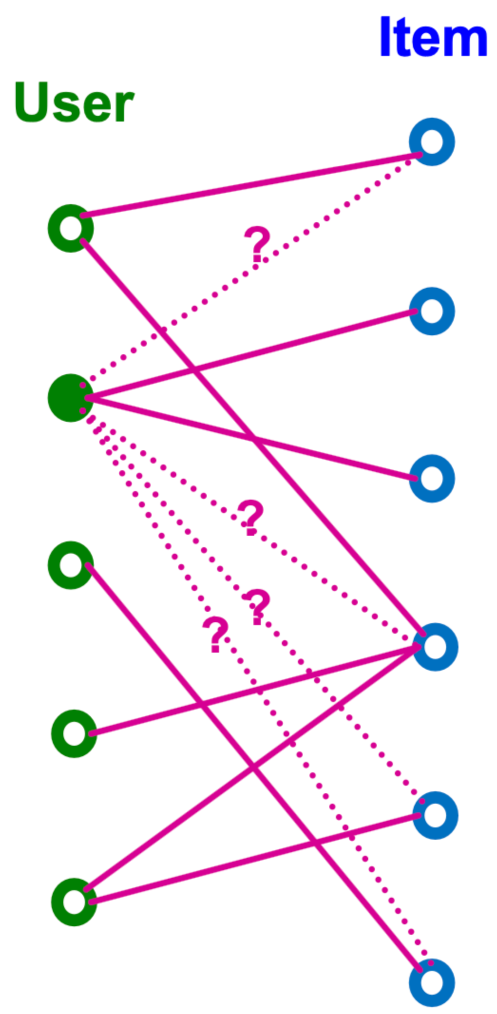
Recommender System as a Graph

- Recommender system is modeled as a **bipartite graph**.
- Edges connect **users** and **items:**
  - User-item interaction (e.g., click, purchase, review etc.).
  - Often associated with timestamp (timing of the interaction).



Recommendation Task

- **Given** past user-item interactions (as a graph).
  - Implicit feedback is assumed for simplicity.
- **Predict** items each user will interact in the future.
  - Can be cast as a **link prediction** problem.
    - Predict new user-item edges given the past edges
  - Aim to learn a real-valued score function $f(u, v)$.
    - Between a user u and an item v.
    - Recommend items with the highest $f(u, v)$.

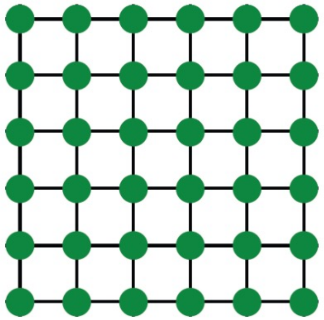


Outline

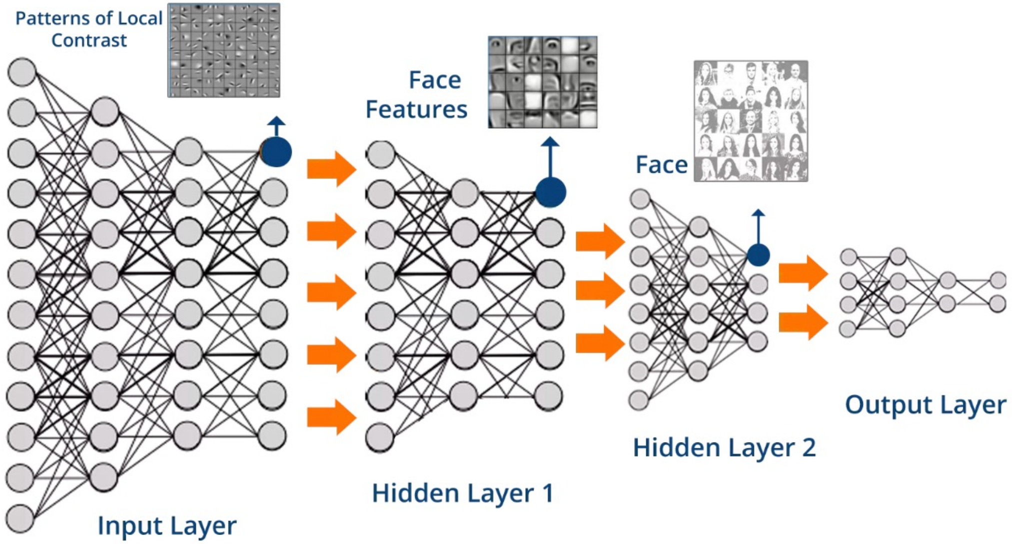
1. Graph-based recommendation
2. **<u>Graph neural networks</u>**
3. Neural graph collaborative filtering
4. Summary

Deep Learning on Graphs

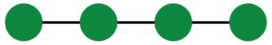
- Modern deep learning focuses on simple **sequences** and **grids**.

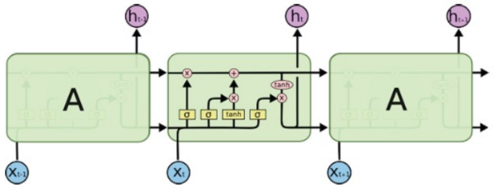


Images



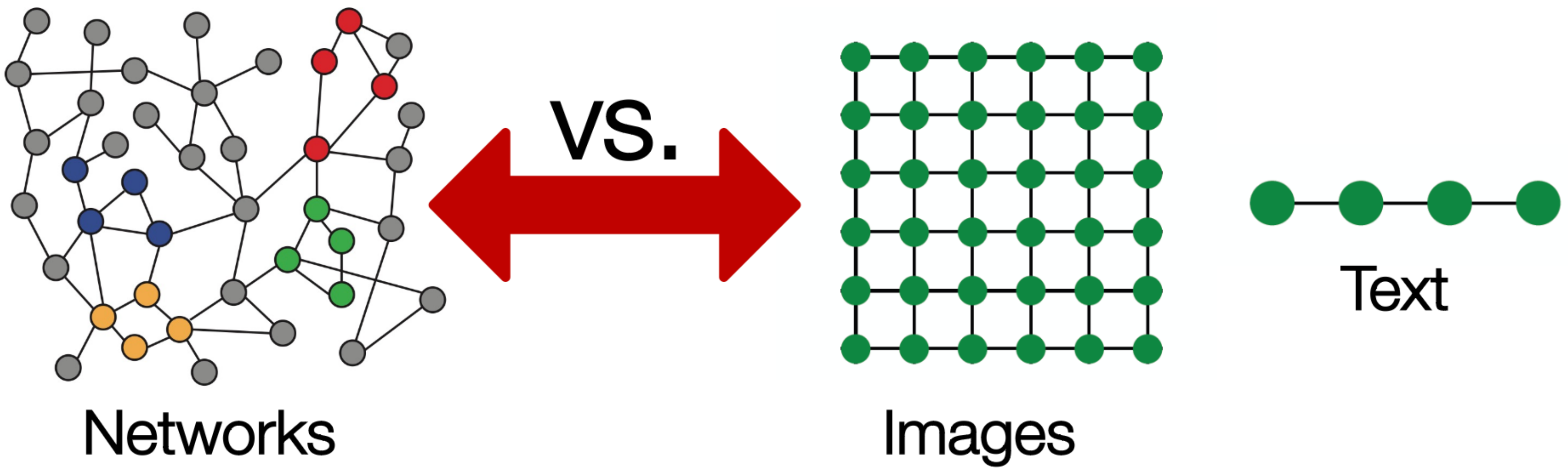
Text/Speech





Deep Learning on Graphs

- Graph-structured data are far more complex.
  - Graphs have arbitrary size and complex topological structure.
  - No fixed node ordering or reference point.
    - Graph G with nodes $(1, 2, 3)$ is the same as G' with nodes $(1, 3, 2)$?

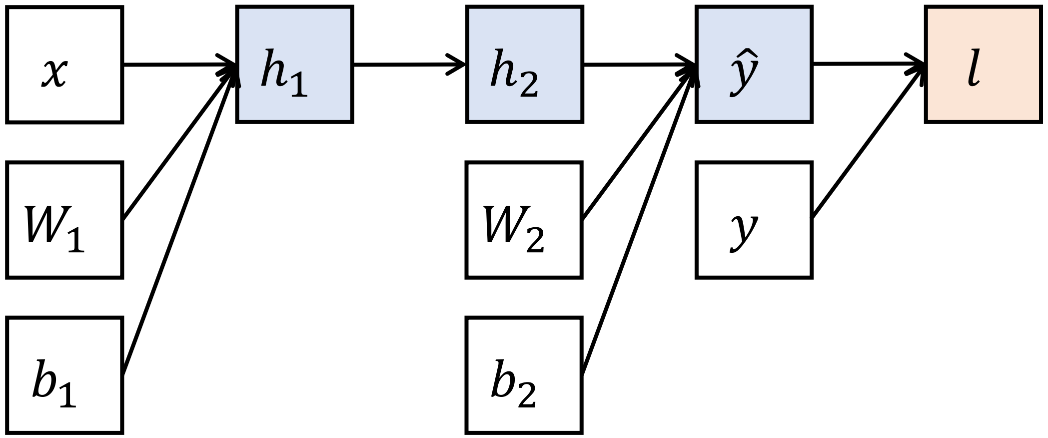


Problem Definition

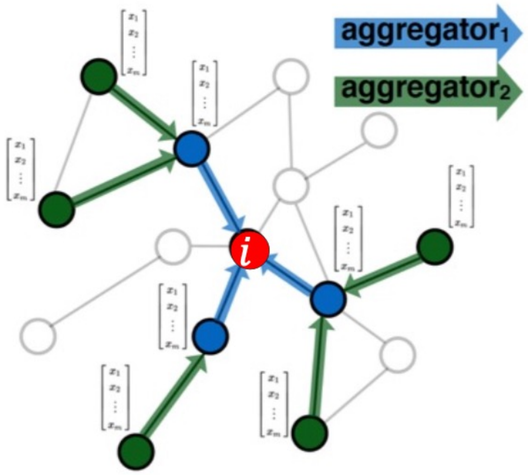
- **Given** a graph $G = (\boldsymbol{A}, \boldsymbol{X})$.
  - $\boldsymbol{A} \in \{0,1\}^{|V|\times|V|}$ is a (symmetric) adjacency matrix.
  - $\boldsymbol{X} \in \mathbb{R}^{|V|\times d}$ is a node feature matrix.
    - It can be a learnable matrix.
- **Goal:** Train a neural network for useful graph tasks.
  - E.g., node classification or link prediction.
  - Different types of **labels** will be given based on the task.

Graph Neural Networks

- **Graph neural networks (GNNs)** are neural nets for graphs.
- **Idea:** Node’s neighborhood defines a **computation graph**.
  - Generalize the chain graph of an MLP.



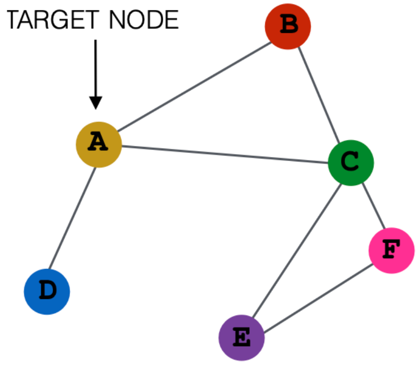
$\approx$



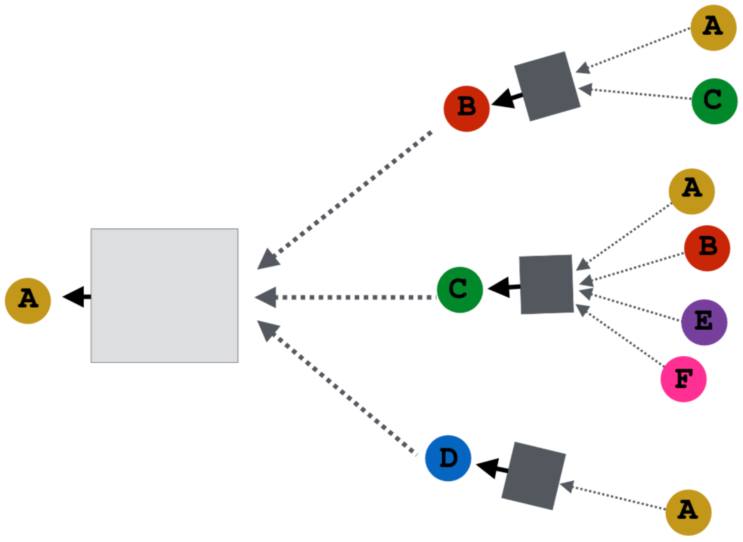
Propagate and transform information

Computation Graph

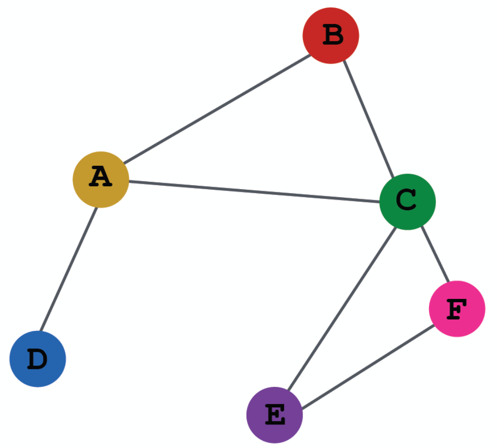
- Computation graph creates a **tree structure** through layers.
  - Let $h_A^{(l)}$ be the hidden representation of node A at layer l.
  - $h_A^{(l)}$ is computed from $h_A^{(l-1)}$, $h_B^{(l-1)}$, $h_C^{(l-1)}$, and $h_D^{(l-1)}$ at layer $l-1$.



INPUT GRAPH

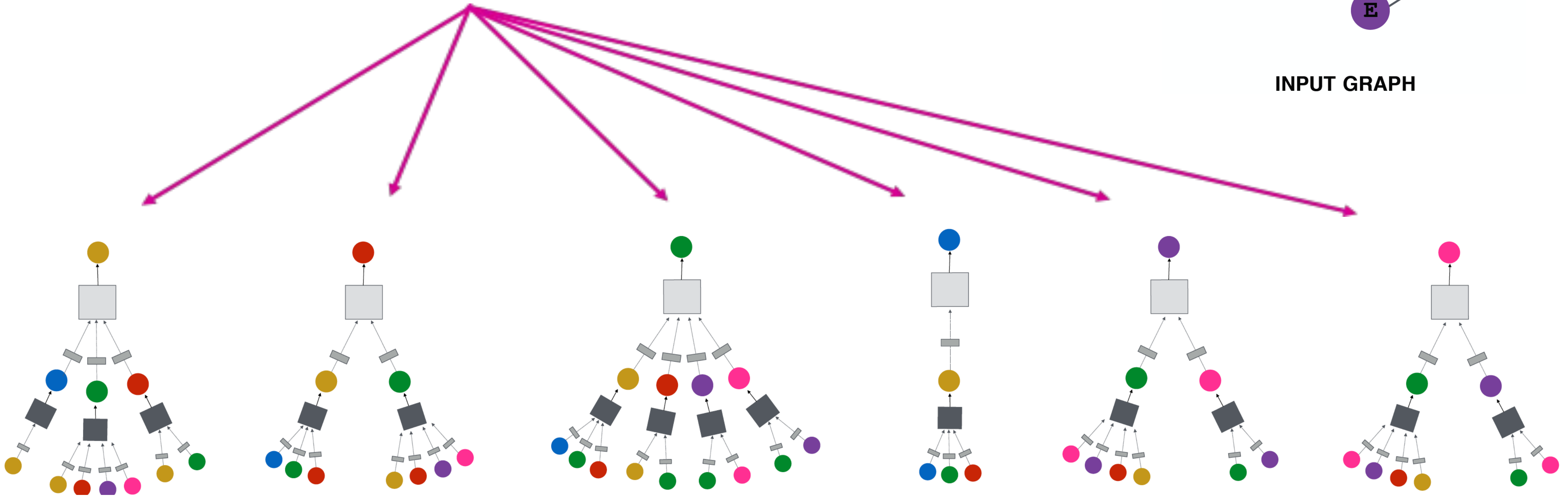


Computation Graph



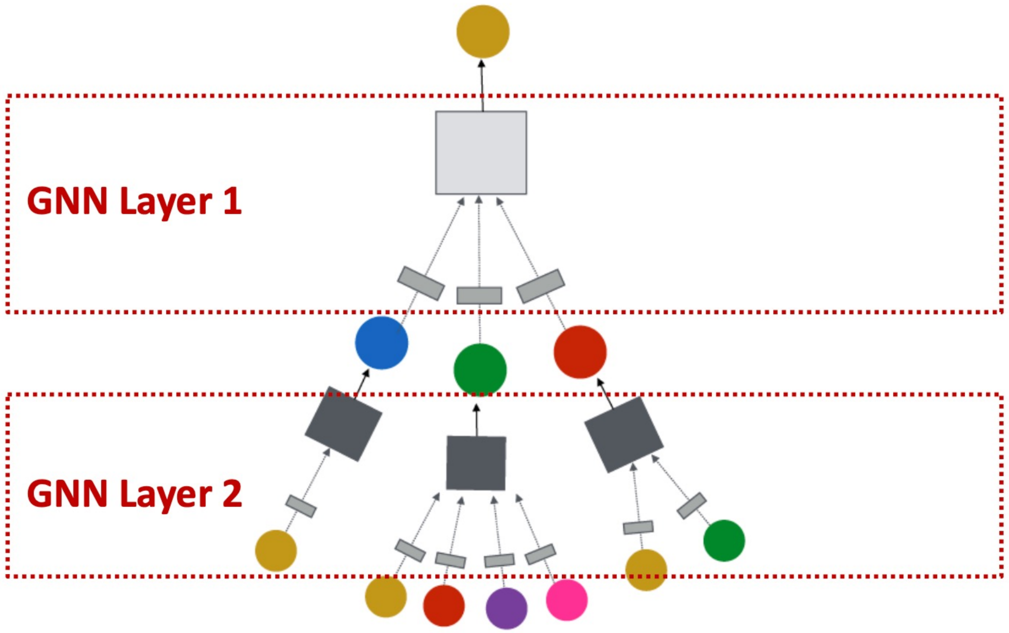
INPUT GRAPH

- **Every node** defines a computation graph based on its neighborhood, in parallel.



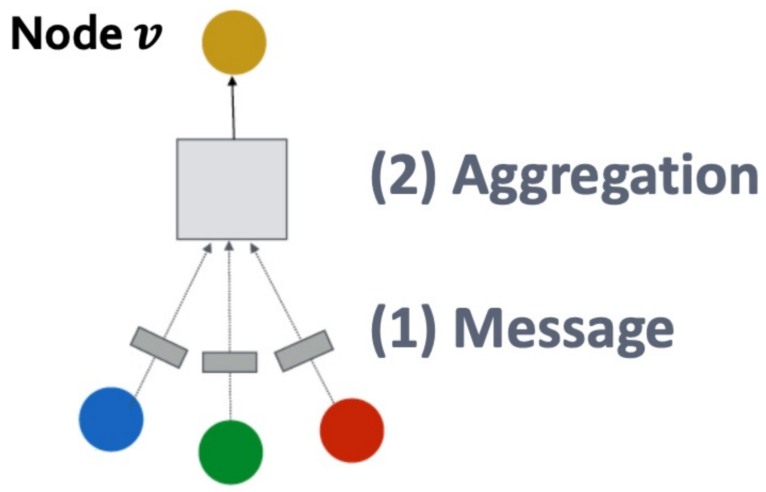
GNN Layers

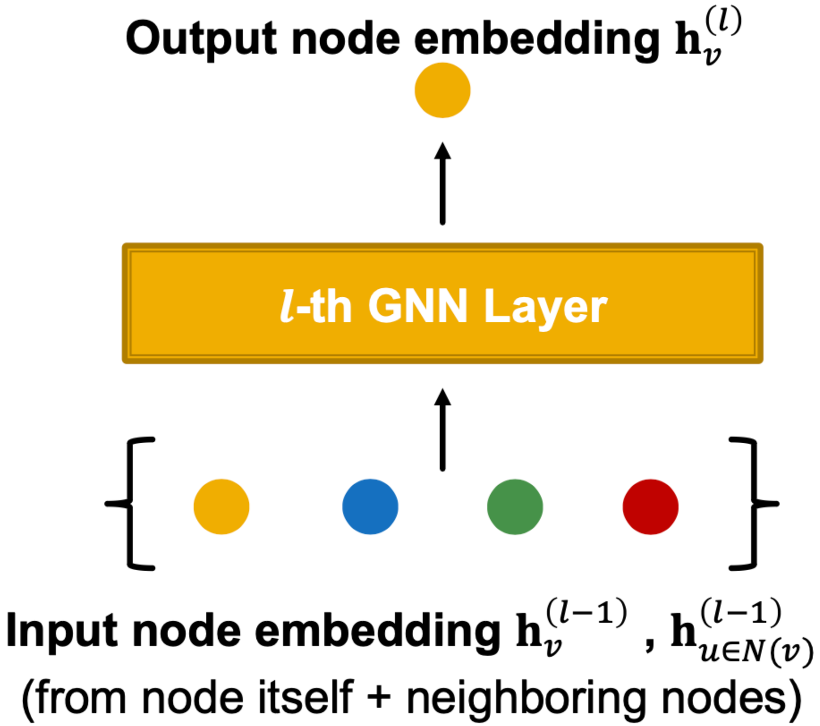
- **GNN layers** are a core building block of GNNs.
  - The number of layers determines the expressiveness of GNNs.
  - L-layer GNN considers the L-hop neighborhood of each node.



Components of Each Layer

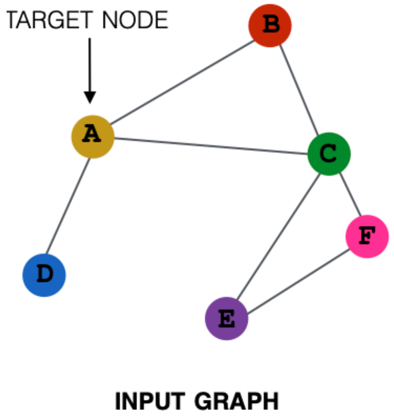
- GNN layer is a function from a set of vectors into a single vector.
  1. **Message:** Transform each vector.
  2. **Aggregation:** Aggregate the messages.

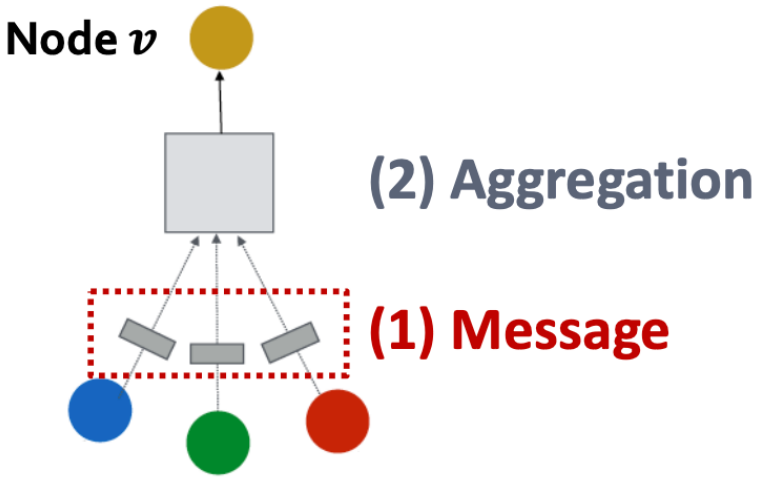




Message Computation

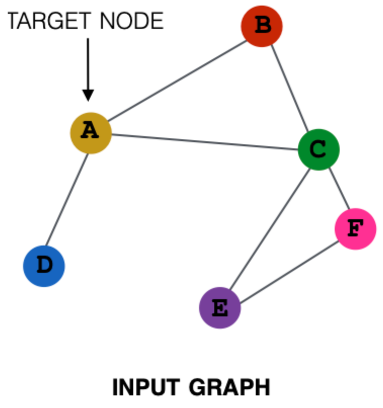
- **Message** is defined as $m_u^{(l)} = \mathrm{MSG}^{(l)}(h_u^{(l-1)})$.
  - **Intuition:** Each node creates a message, which is sent to other nodes.
  - **Example:** Linear layer $m_u^{(l)} = W^{(l)} h_u^{(l-1)}$.

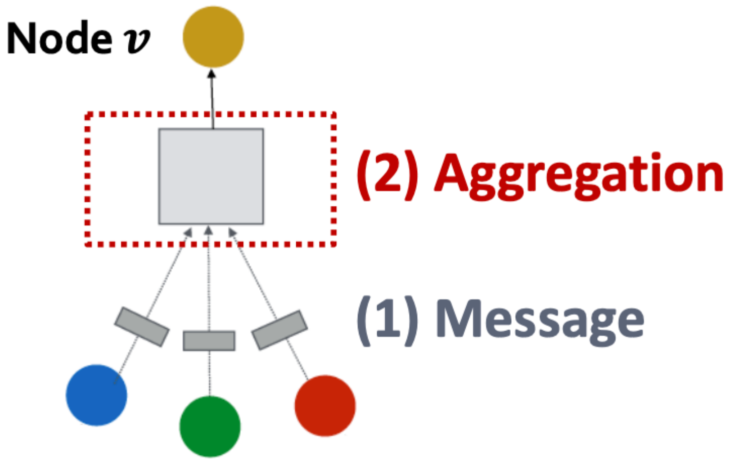




Aggregation

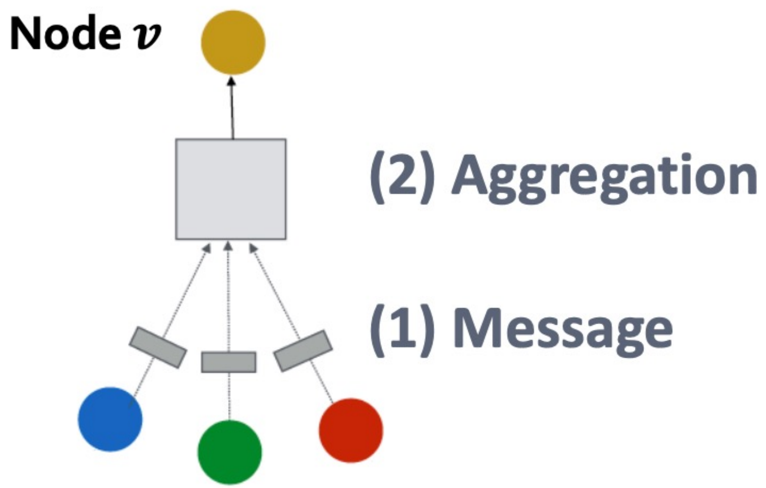
- **Aggregation** is defined as $h_v^{(l)} = \mathrm{AGG}^{(l)}(\{m_u^{(l)} \mid u \in N(v)\})$.
  - **Intuition:** Aggregate the messages from node v's neighbors.
  - **Example:** Elementwise $\mathrm{sum}(\cdot)$, $\mathrm{mean}(\cdot)$, or $\max(\cdot)$ operator.
    - Any many-to-one function is okay.





Self-Connection

- We should also include a **self-connection** at each layer.
  - We don't want to lose information from node v.
  - Use $N(v) \cup \{v\}$ instead of $N(v)$.



General Framework

- Putting things together, we have a **GNN layer** defined as
  - **Message:** $m_u^{(l)} = \mathrm{MSG}^{(l)}(h_u^{(l-1)})$ where $u \in \{N(v) \cup \{v\}\}$.
  - **Aggregation:** $\hat{h}_v^{(l)} = \mathrm{AGG}^{(l)}(\{m_u^{(l)} \mid u \in N(v)\}, m_v^{(l)})$.
  - **Activation function:** $h_v^{(l)} = \sigma(h_v^{(l)})$.
    - The function σ can be ReLU, Sigmoid, etc., and is used for nonlinearity.
- There are many GNNs with different choices of components.
  - GCN, GraphSAGE, GAT, GIN, etc.

Graph Convolutional Network

- **Graph convolutional network (GCN)** is defined as

$$\hat{h}_v^{(l)} = \sum_{u \in N(v) \cup \{v\}} \frac{W^{(l)} h_u^{(l-1)}}{\sqrt{(d_v + 1)(d_u + 1)}}$$

  - where d_v is the degree of node v, i.e., $d_v = |N(v)|$.
  - The actual representation is $h_v^{(l)} = \sigma(\hat{h}_v^{(l)})$ with nonlinearity.

GCN in the Matrix Form

- We implement a GNN with **matrix-vector operations**.
- For example, the following two are equivalent:

$$h_u^{(l)} = \sum_{u \in N(v) \cup \{v\}} \frac{W^{(l)} h_u^{(l-1)}}{\sqrt{(d_v+1)(d_u+1)}}, \qquad H^{(l)} = \widetilde{D}^{-1/2} \tilde{A} \widetilde{D}^{-1/2} H^{(l-1)} W^{(l)}.$$

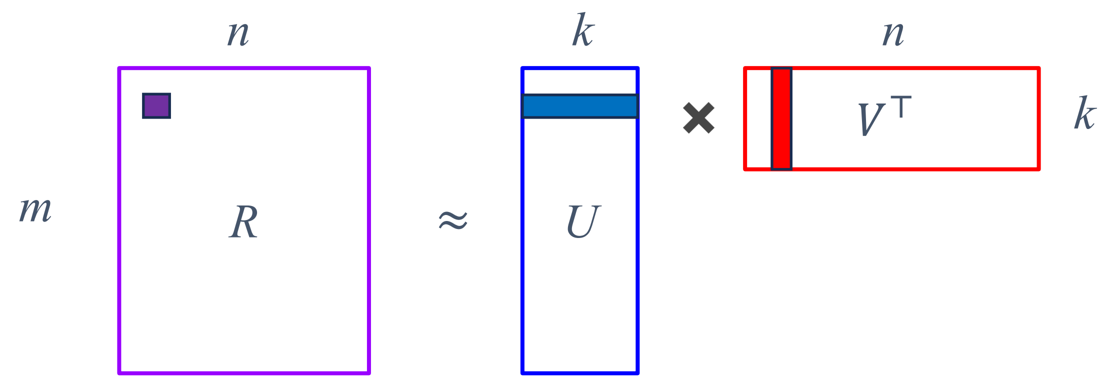
  - $\tilde{A} = A + I$ is the adjacency matrix with self-loops.
  - $\widetilde{D}$ is the degree matrix of $\tilde{A}$, such that $\tilde{d}_{ii} = \sum_k \tilde{a}_{ik}$.

Outline

1. Graph-based recommendation
2. Graph neural networks
3. **Neural graph collaborative filtering**
4. Summary

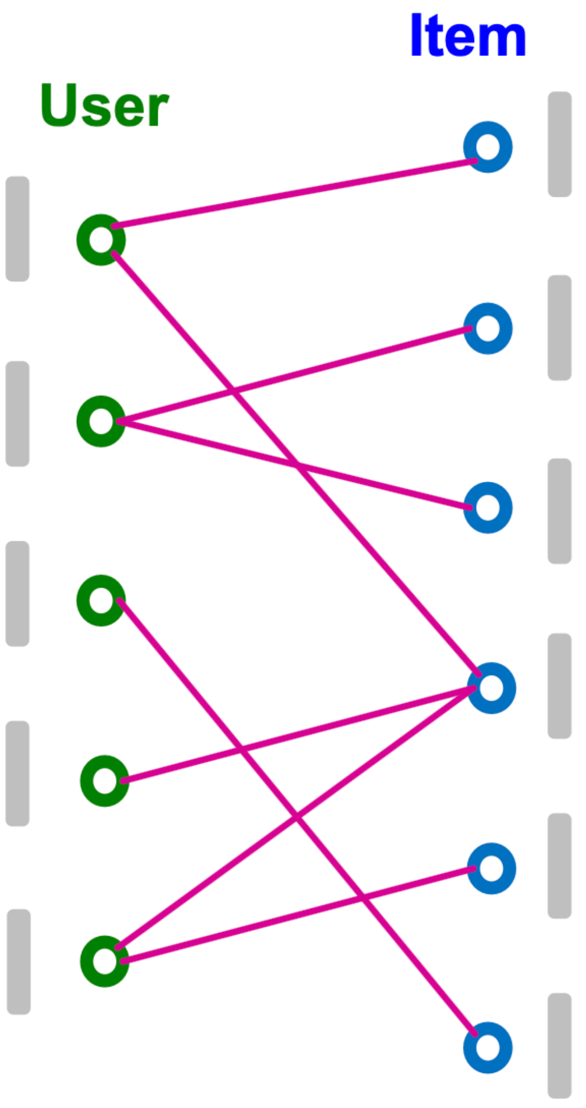
Recap: Latent Factor Models

- **Latent factor models:**
  - Learn an embedding $\boldsymbol{z}_u$ and $\boldsymbol{z}_v$ for every user u and item v, resp.
  - Given a user u, find an item v with a high score $f(u, v) = \boldsymbol{z}_u^\top \boldsymbol{z}_v$.
- The score function can be modeled as a neural network.
  - Use $f_\theta(u, v)$ with learnable parameters θ as a score function.



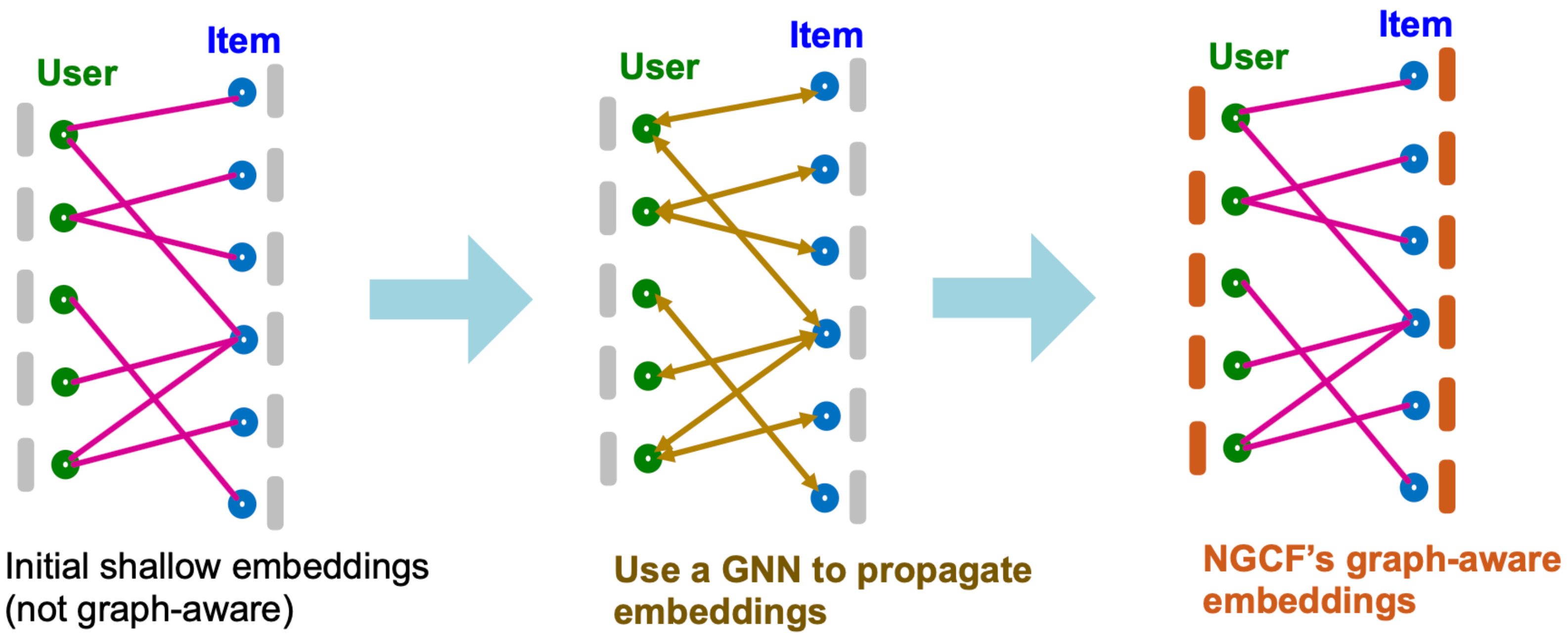
Limitations of LFMs

- CF captures only the **first-order** structure.
  - Only u and v participate in computing $f(u, v)$.
  - High-order graph structure (e.g. K-hop paths between u and v) is not explicitly captured.
- **Example:**
  - $K = 2$: Users u_1 and u_2 bought the same item.
  - $K = 4$: Users u_1 and u_2 bought items v_1 and v_2 that are bought by the same user u_3.
    - Represents high-order similarity.



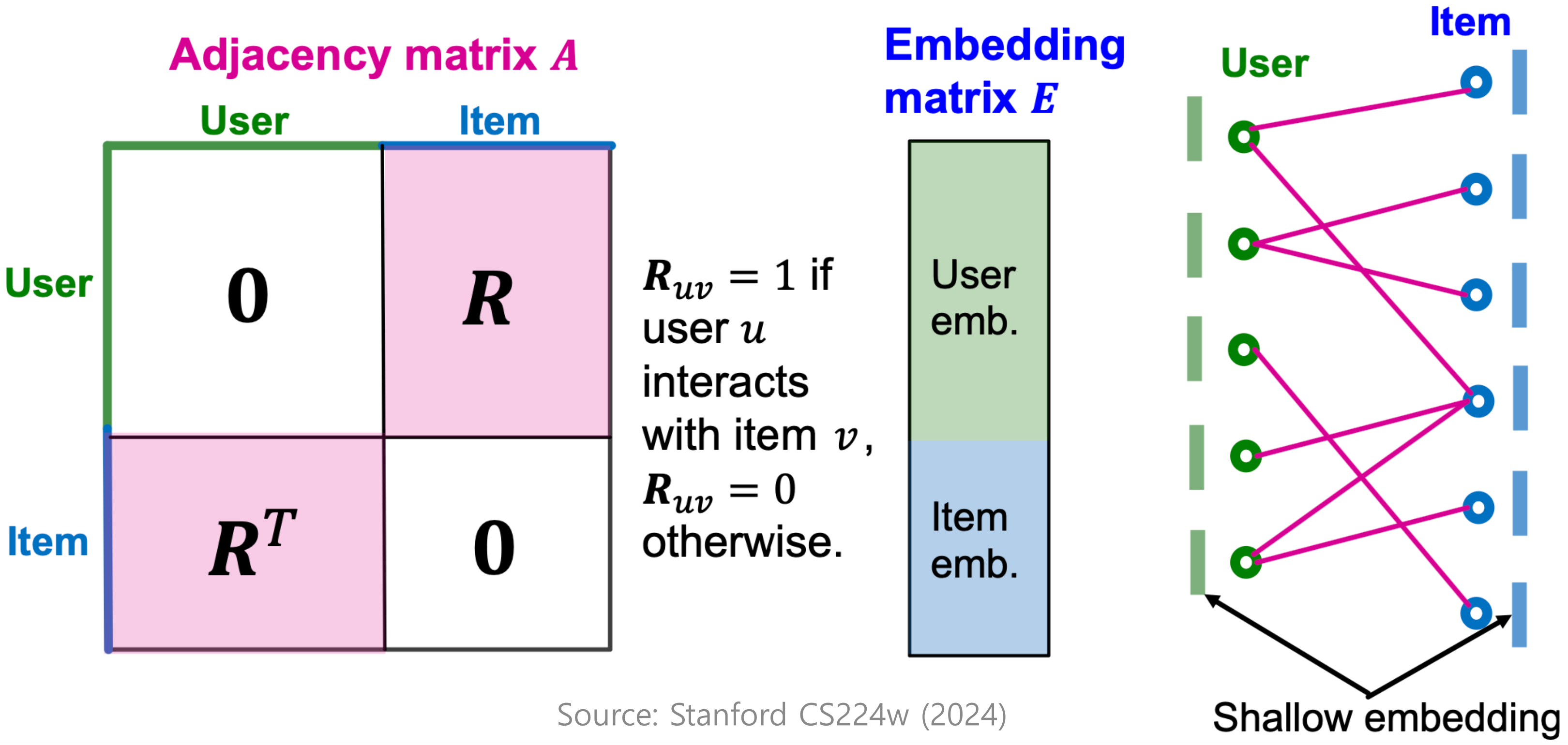
Neural Graph Collaborative Filtering

- **Neural Graph Collaborative Filtering (NGCF):**
  - Explicitly incorporates the high-order graph structure for embeddings.
  - **Key idea:** Use a GNN to generate graph-aware final embeddings.



NGCF Visualization

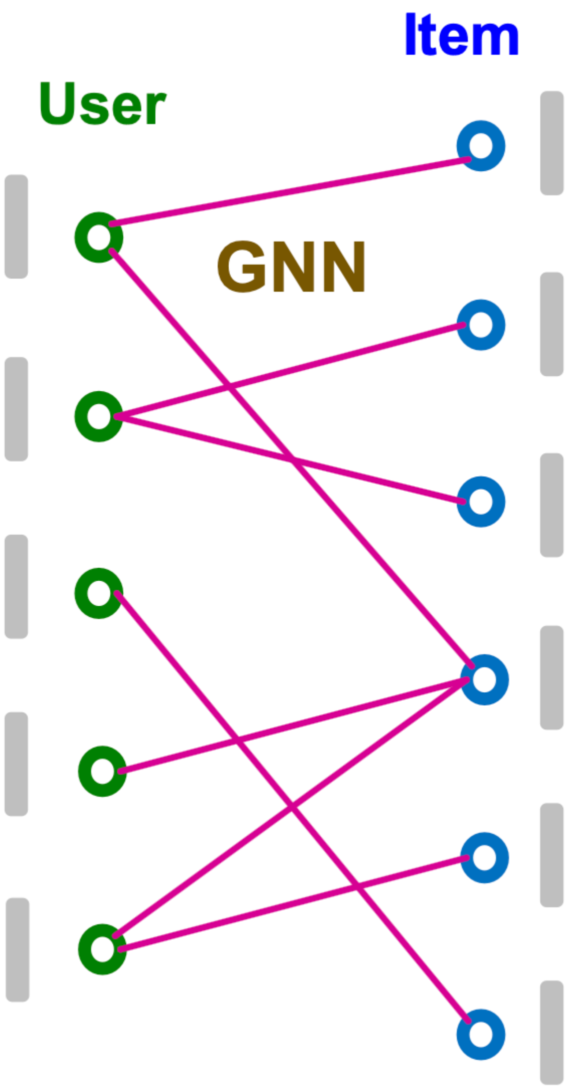
- A graph (with the matrix $\boldsymbol{A}$) is created from the utility matrix $\boldsymbol{R}$.



Source: Stanford CS224w (2024)

NGCF Framework

- **Given:** User-item bipartite graph G.
- **NGCF framework:**
  - Initialize learnable node embeddings $\boldsymbol{E}$.
  - Use a GNN to propagate $\boldsymbol{E}$ through G.
- Contains two kinds of parameters:
  - Shallow user/item embeddings: $O(D|V|)$.
    - $|V|$ is the number of nodes.
    - D is the embedding dimension.
  - GNN's parameters: $O(LD^2)$.
    - L is the number of GNN layers.

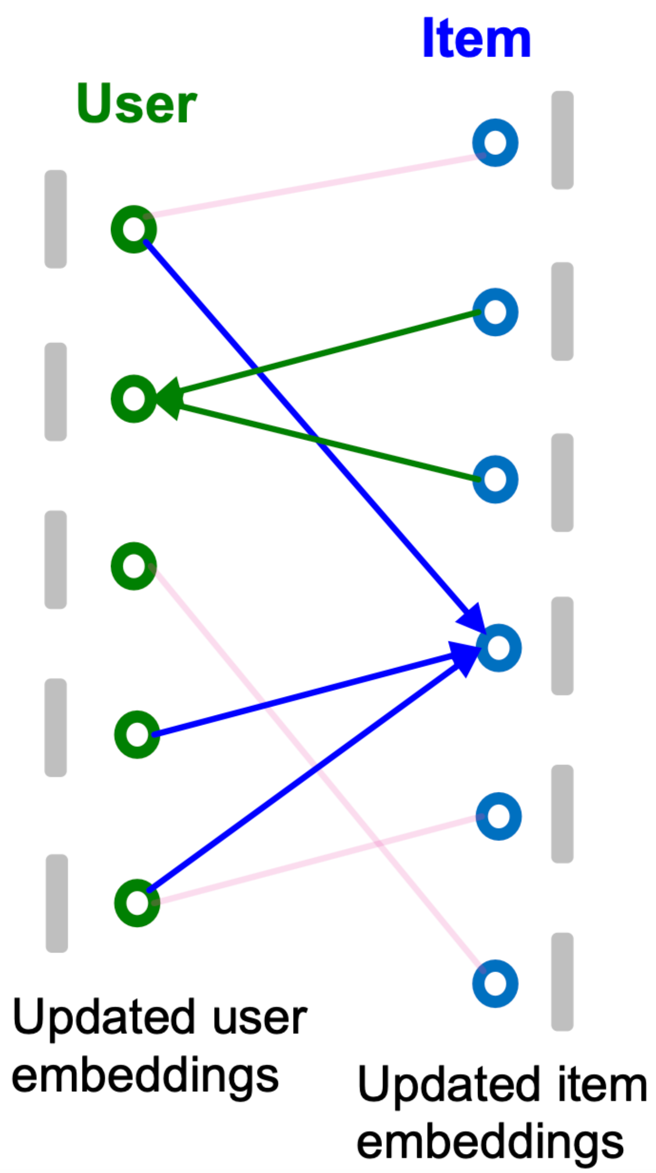


Neighborhood Aggregation

- **Step 1:** Set embeddings as initial features.
  - Each node u has a learnable embedding $\boldsymbol{h}_u^{(0)}$.
- **Step 2:** Update the embeddings through layers.

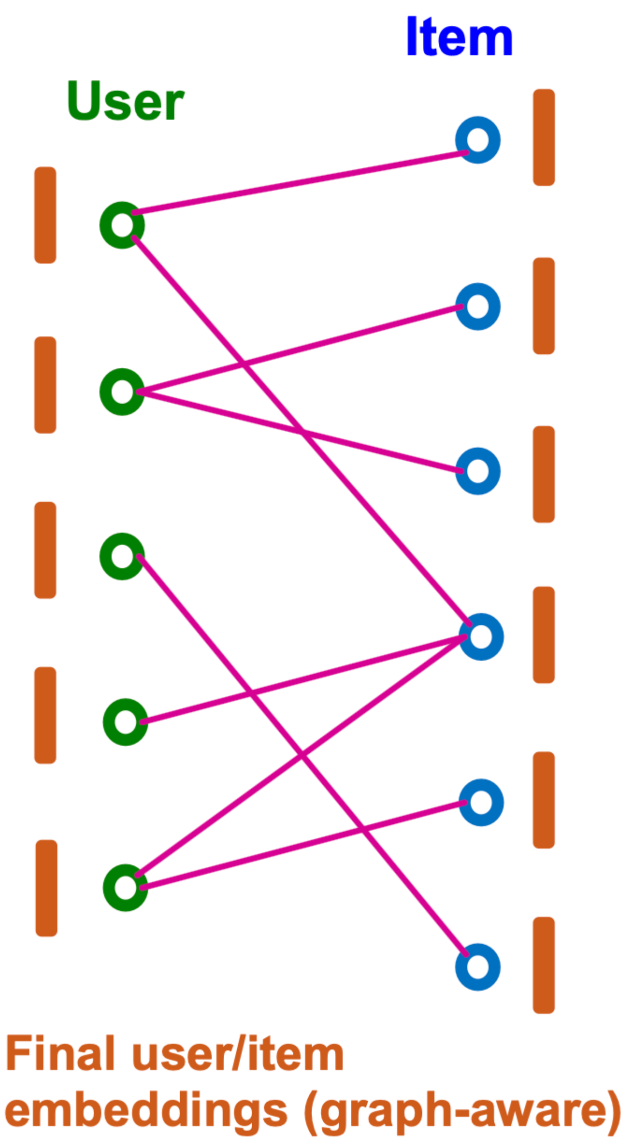
$$\boldsymbol{h}_u^{(l+1)} = \mathrm{COMBINE}\left(\boldsymbol{h}_u^{(l)}, \mathrm{AGG}\left(\left\{\boldsymbol{h}_v^{(l)} \middle| v \in N(u)\right\}\right)\right).$$

  - Done for all users/items simultaneously.
  - $\mathrm{COMBINE}$ and AGG functions can be anything.



Score Function

- Get final embeddings $\boldsymbol{z}_u = \boldsymbol{h}_u^{(L)}$ and $\boldsymbol{z}_v = \boldsymbol{h}_v^{(L)}$.
- Use the inner product $\boldsymbol{z}_u^{\top}\boldsymbol{z}_v$ as a score function.
- Training done by (stochastic) gradient descent.
- Any loss function for implicit feedback is used.
  - E.g., BPR loss.



Outline

1. Graph-based recommendation
2. Graph neural networks
3. Neural graph collaborative filtering
4. **Summary**

Summary

- Graphs can represent various datasets and tasks.
- GNNs are a core deep learning architecture for graphs.
  - The main idea is to apply convolution to graphs.
- Graph-based recommendation utilize graph information.
  - We can capture high-order relationships of users/items.