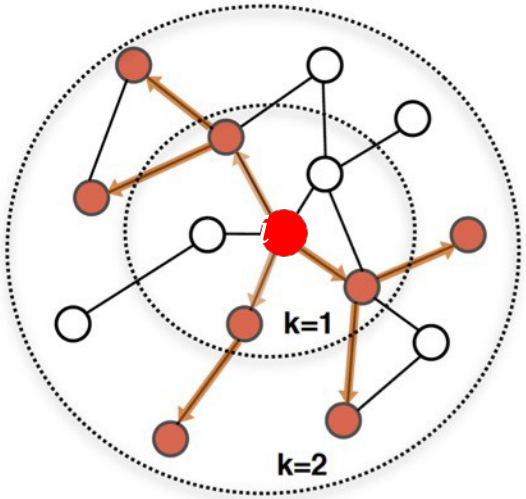
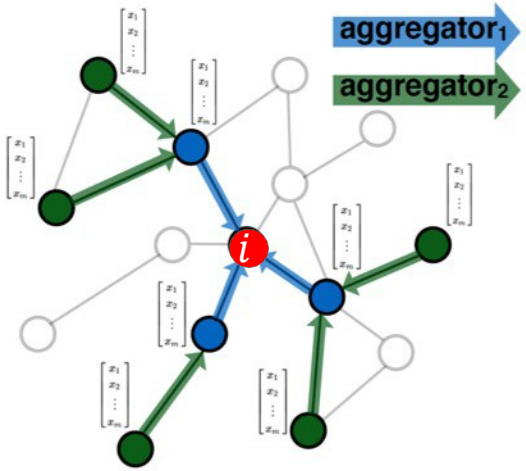
Graph Neural Networks

上海交通大学
约翰·霍普克罗夫特
计算机科学中心
John Hopcroft Center for Computer Science

Core of GNN: propagate information across the graph



Determine node computation graph



Propagate and transform information

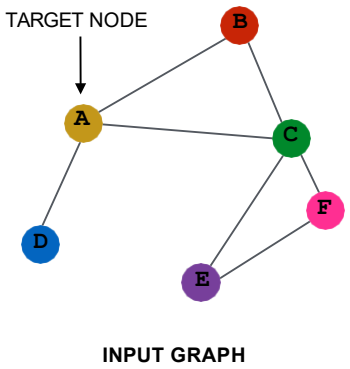
Adapted from CS224W Machine Learning with Graphs https://web.stanford.edu/class/cs224w/

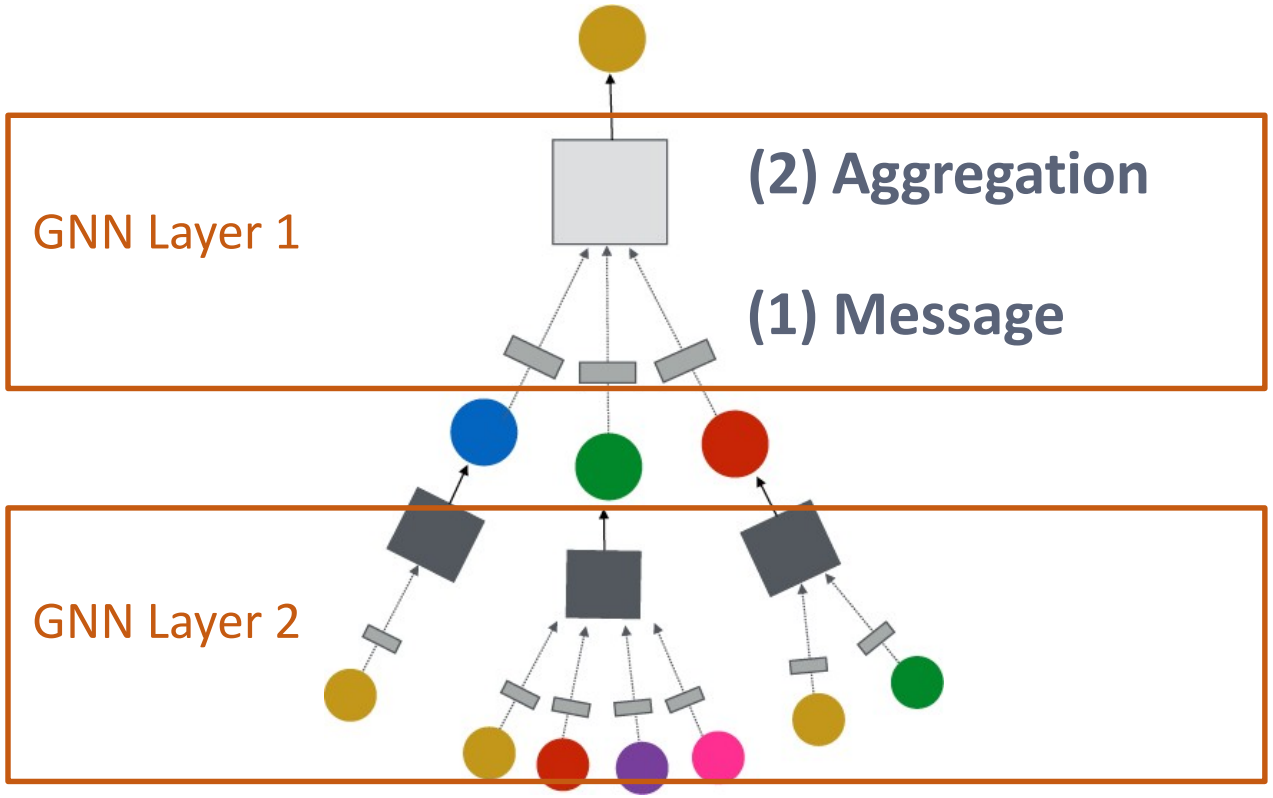
A General GNN Framework

GNN Layer = Message + Aggregation

- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**

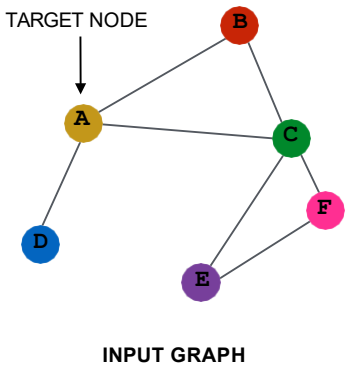
Connect GNN layers into a GNN





(3) Layer connection

A General GNN Framework

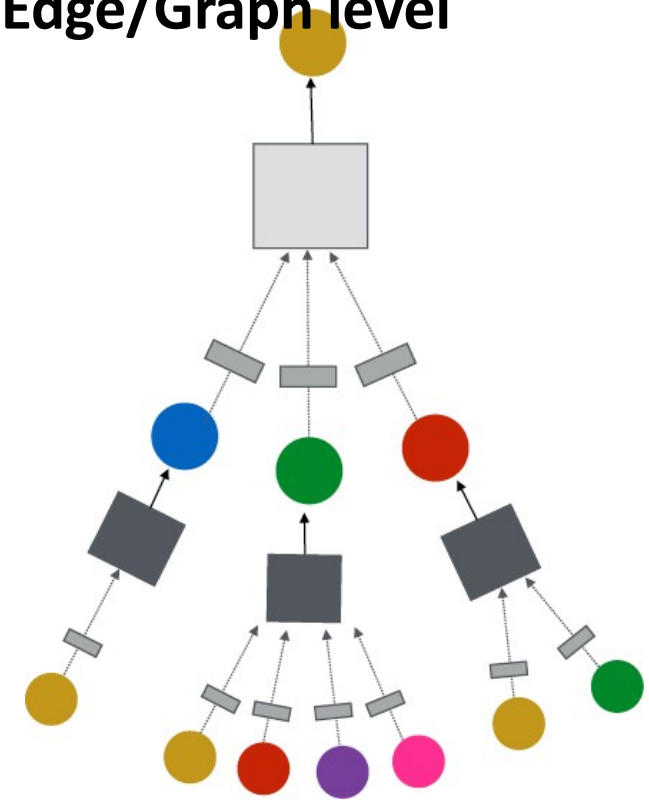


Graph manipulation

- **Graph feather augmentation**
- **Graph structure augmentation**

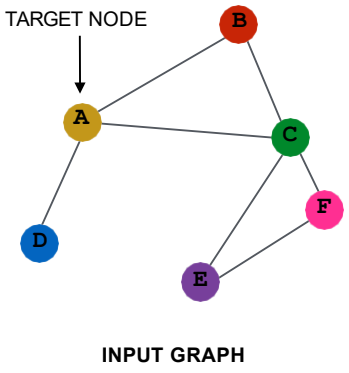
Learning Objective

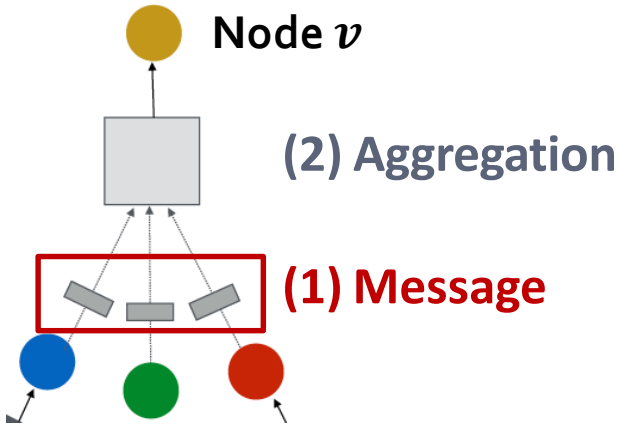
- **Supervise/unsupervised**
- **Node/Edge/Graph level**



Message Computation

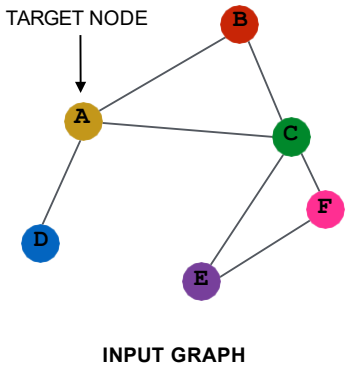
- **Message function**: $\mathrm{m}_u^l = MSG^l(h_u^{l-1})$
  - **Intuition:** Each node creates a message, which will be sent to other nodes later
  - **Example:** A Linear layer $\mathrm{m}_u^l = W^l h_u^{l-1}, \mathrm{m}_v^l = B^l h_v^{l-1}$
    - Multiply node features with weight matrix W^l, B^l

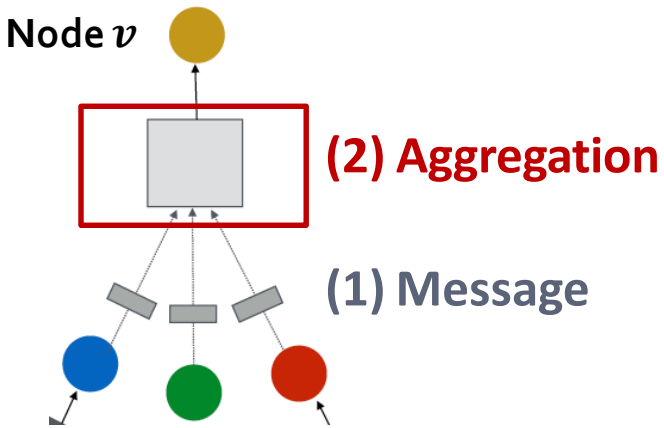




Message Aggregation

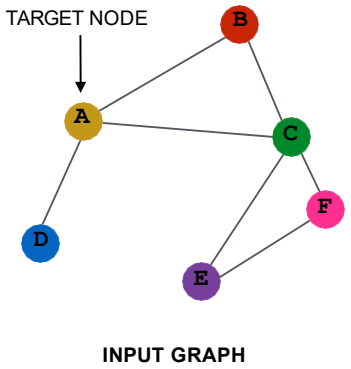
- Node v will aggregate the messages from its neighbors u:
  - $h_v^l = Agg^l(\{m_u^l,\ u \in N(v)\})$
  - **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator $h_v^l = Sum(\{m_u^l,\ u \in N(v)\})$
- Aggregate the message from node v itself, via concatenation or summation
  - $h_v^l = CONCAT(Agg^l(\{m_u^l,\ u \in N(v)\}), m_v^l)$
- Nonlinearity(activation): adds expressiveness to message or aggregation

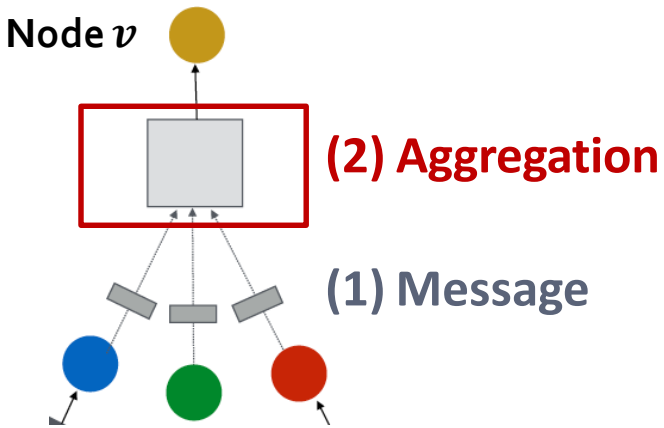




Message Aggregation

- Node v will aggregate the messages from its neighbors u:
  - $h_v^l = Agg^l(\{m_u^l,\ u \in N(v)\})$
- **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator
  - $h_v^l = Sum(\{m_u^l,\ u \in N(v)\})$





GNN Layers: GCN

- **(1) Graph Convolutional Networks (GCN)**

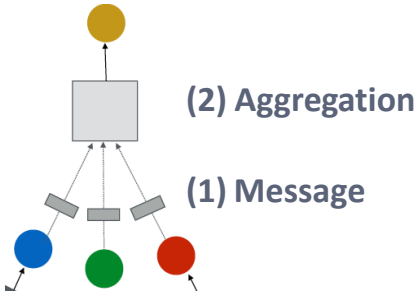
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- **How to write this as Message + Aggregation?**

Message

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

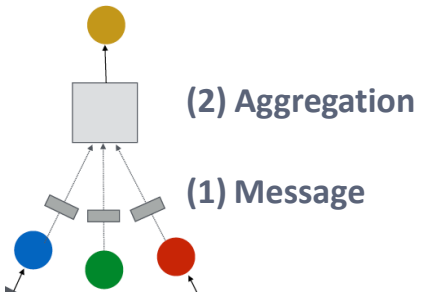
Aggregation



GNN Layers: GCN

- **(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: $m_u^l = \frac{1}{|N(v)|} W^l h_u^{l-1}$

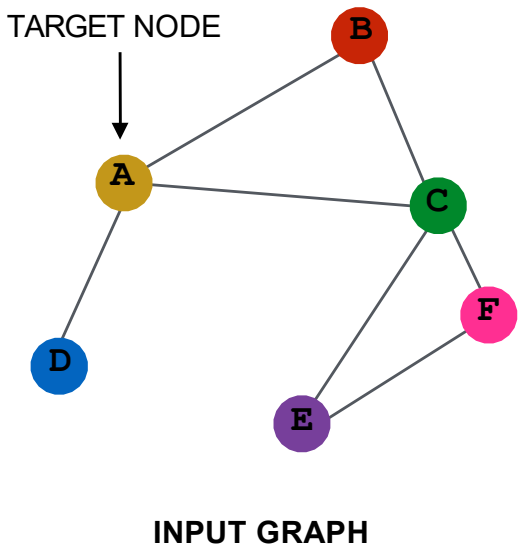
Normalized by node degree
(In the GCN paper they use a slightly different normalization)

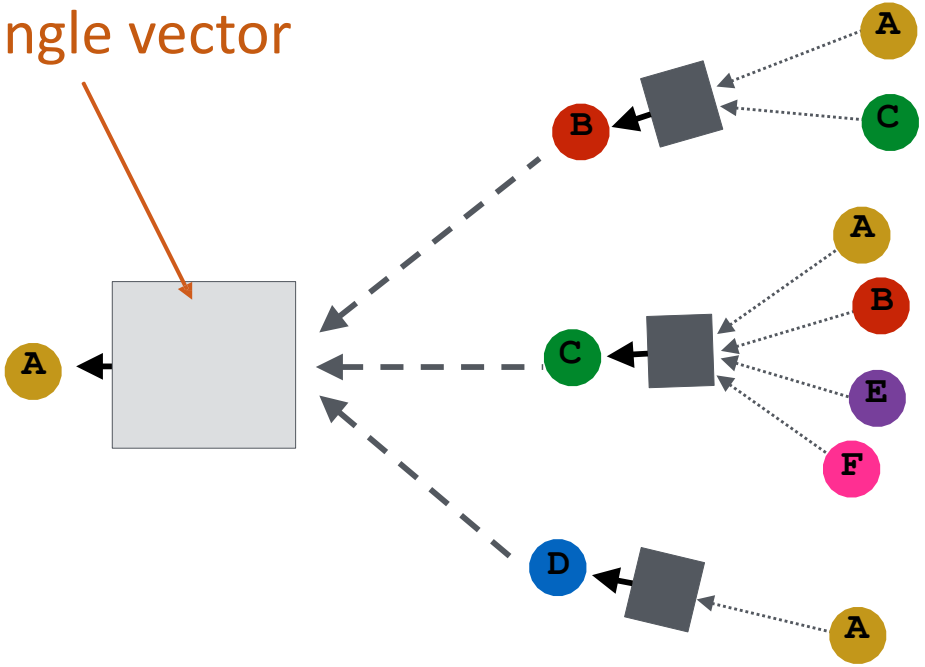
- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $h_v^l = \sigma(Sum(\{m_u^l,\ u \in N(v)\}))$

In GCN the input graph is assumed to have self-edges that are included in the summation.

GNN Layers: GraphSAGE

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





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

How does this message passing architecture differ?

Neighborhood Aggregation

- **Simple neighborhood aggregation:**

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

- **GraphSAGE:**

Concatenate neighbor embedding and self embedding

$$\mathrm{h}_v^{(l+1)} = \sigma\left(\left[\mathrm{W}_l \cdot \boxed{\mathrm{AGG}\left(\left\{\mathrm{h}_u^{(l)}, \forall u \in N(v)\right\}\right)}, \mathrm{B}_l \mathrm{h}_v^{(l)}\right]\right)$$

Flexible aggregation function instead of mean

Neighborhood Aggregation: Variants

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

$$\mathrm{AGG} = \sum_{u \in N(v)} \frac{\mathrm{h}_u^{(l)}}{|N(v)|}$$

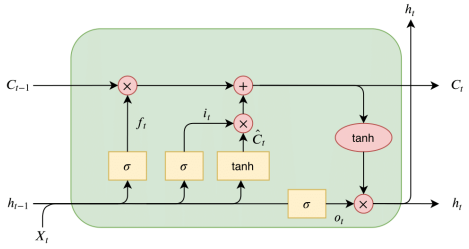
- **Pool:** Transform neighbor vectors and apply symmetric vector function

$$\mathrm{AGG} = \gamma\left(\left\{\mathrm{MLP}\left(\mathrm{h}_u^{(l)}\right), \forall u \in N(v)\right\}\right)$$

Element-wise mean/max

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

$$\mathrm{AGG} = \mathrm{LSTM}\left(\left[\mathrm{h}_u^{(l)}, \forall u \in \pi(N(v))\right]\right)$$



L2 Normalization

Optional: Apply L2 normalization to $\mathrm{h}_v^{(l+1)}$ embedding at every layer

- ℓ_2 **Normalization:**
  - $h_v^k \leftarrow \frac{h_v^k}{\|h_v^k\|_2} \forall v \in V$ where $\|u\|_2 = \sqrt{\sum_i u_i^2}$($\ell_2$−norm)
  - Without ℓ_2 normalization, the embedding vectors have different scales (ℓ_2-norm) for vectors
  - In some cases (not always), normalization of embedding results in performance improvement
  - After ℓ_2 normalization, all vectors will have the same ℓ_2 norm

GNN Layers: GAT

- **Graph Attention Networks**

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

Attention weights

- **In GCN / GraphSAGE**
  - $\alpha_{uv} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u's message to node v
  - α_{uv} is defined **explicitly** based on the structural properties of the graph (node degree)
  - All neighbors $u \in N(v)$ are equally important to node v

GNN Layers: GAT

- **Graph Attention Networks**

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

Attention weights

Not all node's neighbors are equally important

- **Attention** is inspired by cognitive attention.
- The **attention** $\boldsymbol{\alpha_{vu}}$ focuses on the important parts of the input data and fades out the rest.
  - **Idea:** the NN should devote more computing power on that small but important part of the data.
  - Which part of the data is more important depends on the context and is learned through training.

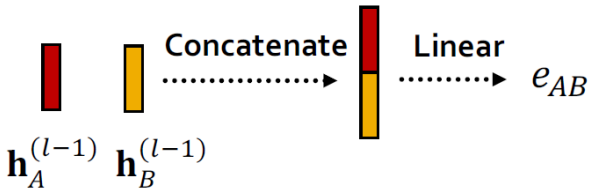
Attention Mechanism

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

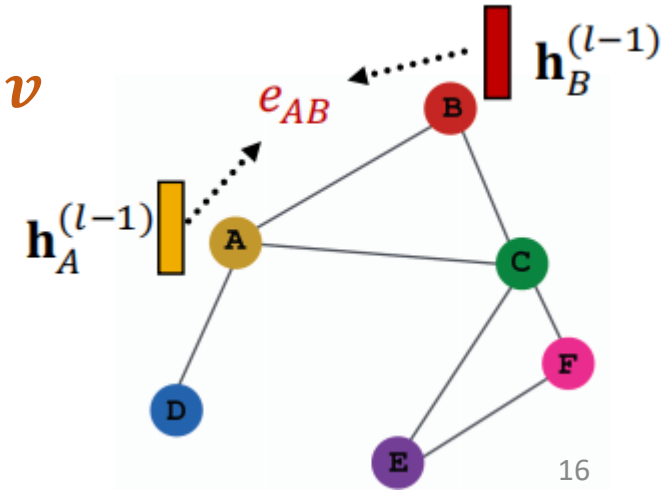
$$\boldsymbol{e_{vu} = a(W^l h_u^{l-1}, W^l h_v^{l-1})}$$

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

Example: use a simple single-layer neural network



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



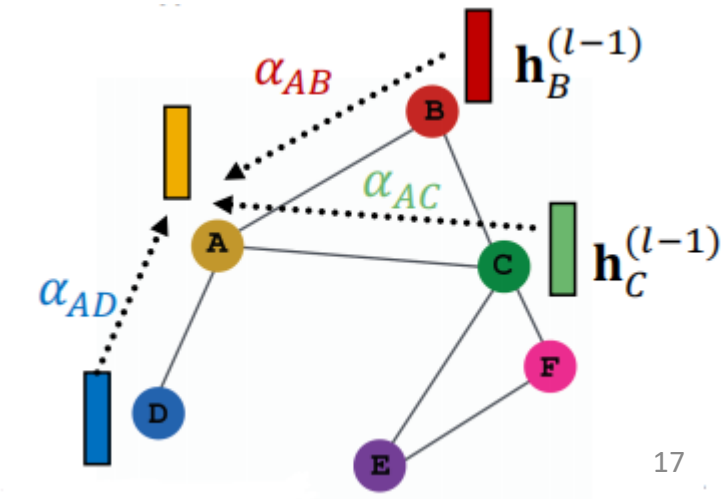
Attention Mechanism

- **Normalize** e_{vu} into the final attention weight α_{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** α_{vu} :

$$h_v^l = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} W^l h_u^{l-1}\right)$$



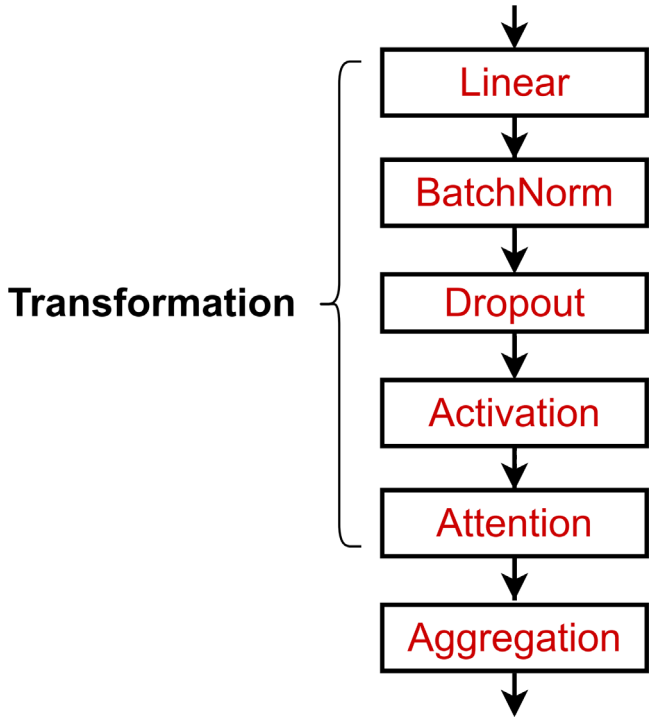
Benefits of Attention Mechanism

- **Key benefit:** Allows for (implicitly) specifying **different importance values (α_{vu}) to different neighbors**
- **Computationally efficient**:
  - Computation of attentional coefficients can be parallelized across 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
- **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

GNN Layer in Practice

- **Many modern deep learning modules can be incorporated into a GNN layer**
- **Batch Normalization:**
  - Stabilize neural network training
- **Dropout:**
  - Prevent overfitting
- **Attention/Gating:**
  - Control the importance of a message
- **More:**
  - Any other useful deep learning modules

A suggested GNN Layer

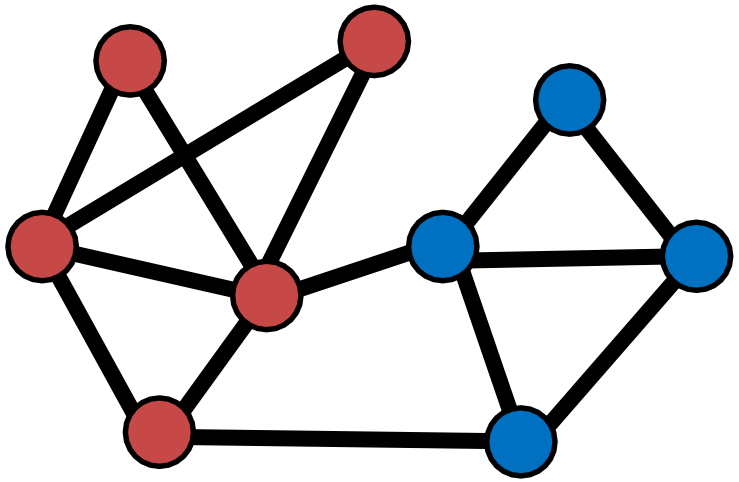


Graph Neural Networks on Heterogeneous Graphs

上海交通大学
约翰·霍普克罗夫特
计算机科学中心
John Hopcroft Center for Computer Science

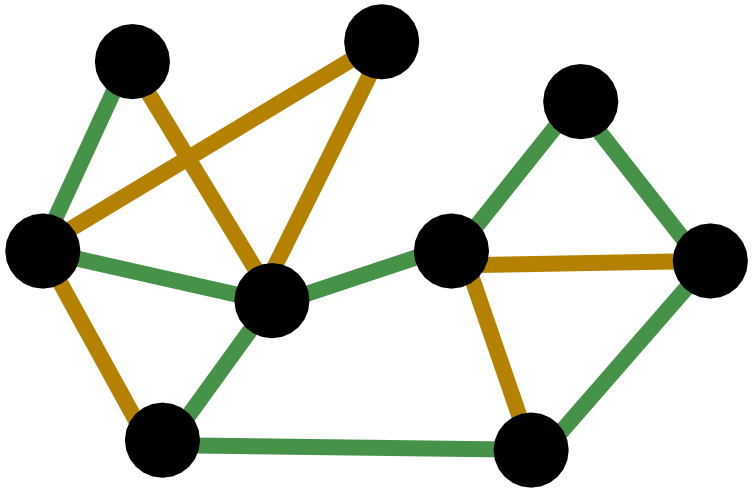
Heterogeneous Graphs

- A heterogeneous graph is a directed graph with nodes and edges of different types.



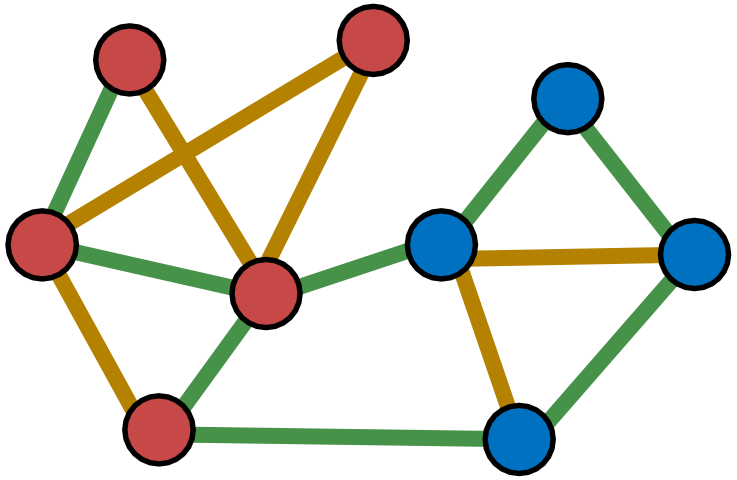
2 types of nodes:

- Node type A: Paper nodes
- Node type B: Author nodes



2 types of edges:

- Edge type A: Cite
- Edge type B: Like



A graph could have multiple types of nodes and edges!

2 types of nodes & edges

Heterogeneous Graphs

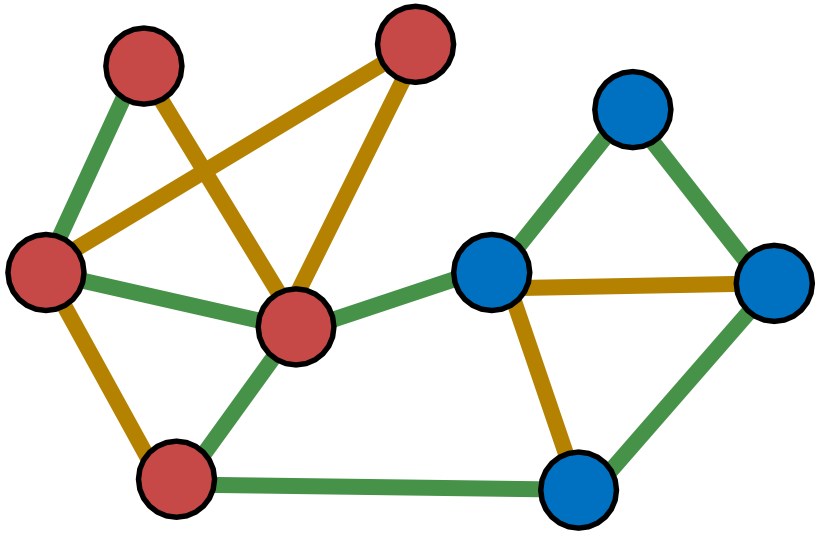
8 possible relation types!

(Paper, Cite, Paper)

(Paper, Like, Paper)

(Paper, Cite, Author)

(Paper, Like, Author)



(Author, Cite, Author)

(Author, Like, Author)

(Author, Cite, Paper)

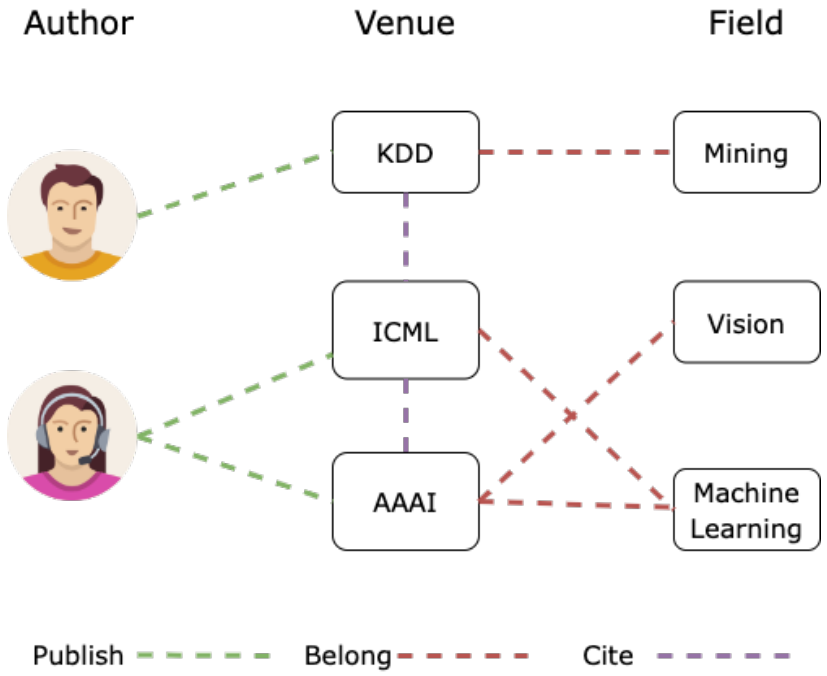
(Author, Like, Paper)

Relation types: (node_start, edge, node_end)

- We use **relation type to describe an edge** (as opposed to edge type)
- Relation type better captures the interaction between nodes and edges

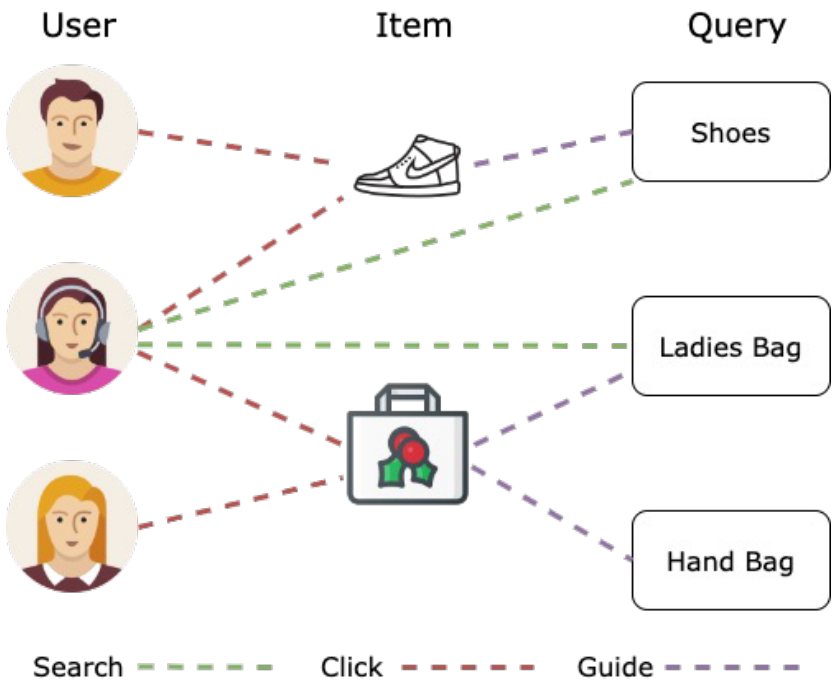
Examples of Heterogeneous Graphs

- Example: Academic Graph
  - **Node types:** Author, Paper, Venue, Field, ...
  - **Edge types:** Publish, Cite, ...
  - Benchmark dataset: **Microsoft Academic Graph**



Examples of Heterogeneous Graphs

- Example: E-Commerce Graph
  - **Node types:** User, Item, Query, Location, ...
  - **Edge types:** Purchase, Visit, Guide, Search, ...
  - Different node type's features spaces can be different!



Heterogeneous Graphs

- **A heterogeneous graph is defined as**

$$\boldsymbol{G}=(\boldsymbol{V}, \boldsymbol{E}, \tau, \phi)$$

  - Nodes **with node types** $v \in V$
  - **Node type** for node v: $\tau(v)$
- Edges **with edge types** $(u, v) \in E$
  - **Edge type** for edge (u, v): $\phi(u, v)$
    - **Relation type** for edge e is a tuple: $r(u, v)=(\tau(u), \phi(u, v), \tau(v))$

An edge can be described as a pair of nodes

How to model types?

- **Observation:** We can also treat types of nodes and edges as features
  - **Example:** Add a one-hot indicator for nodes and edges
    - Append feature [1, 0] to each "author node"; Append feature [0, 1] to each "paper node"
    - Similarly, we can assign edge features to edges with different types
  - Then, a heterogeneous graph reduces to a standard graph
- **When do we need a heterogeneous graph?**

Discussion: Type or Feature?

- **When do we need a heterogeneous graph?**
  - **Case 1:** Different node/edge types **have different shapes of features**
    - An “author node” has 4-dim feature, a “paper node” has 5-dim feature
  - **Case 2**: We know different relation types represent **different types of interactions**
    - (English, translate, French) and (English, translate, Chinese) require different models

Discussion: Heterogeneous?

- Ultimately, **heterogeneous graph** is a **more expressive graph representation**
  - Captures **different types of interactions between entities**
- But it also **comes with costs**
  - More expensive (computation, storage)
  - More complex implementation
- There are many ways to **convert a heterogeneous graph to a standard graph** (that is, a homogeneous graph)

Recap: Classical GCN

- **Graph Convolutional Networks (GCN)**

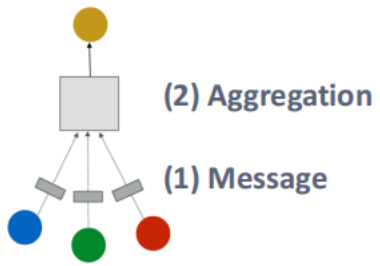
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- **How to write this as Message + Aggregation?**

Message

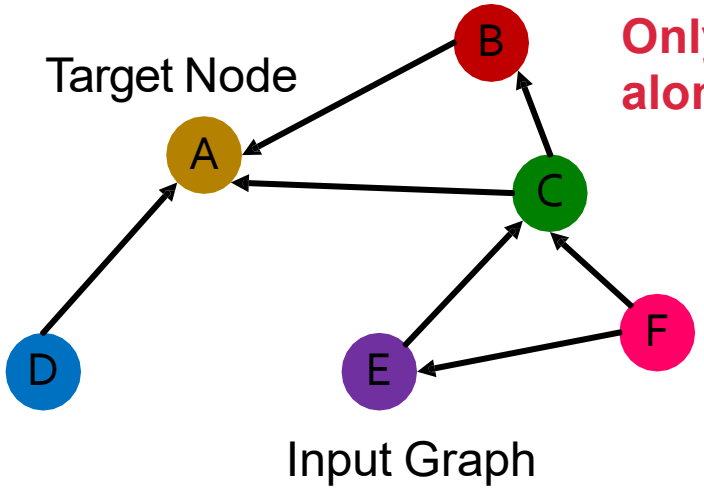
$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

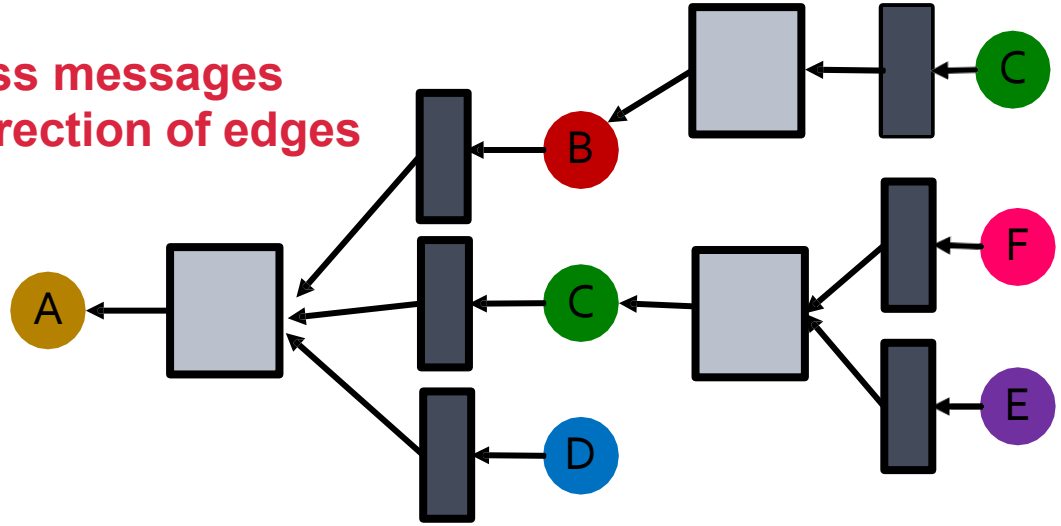
Aggregation



Relational GCN

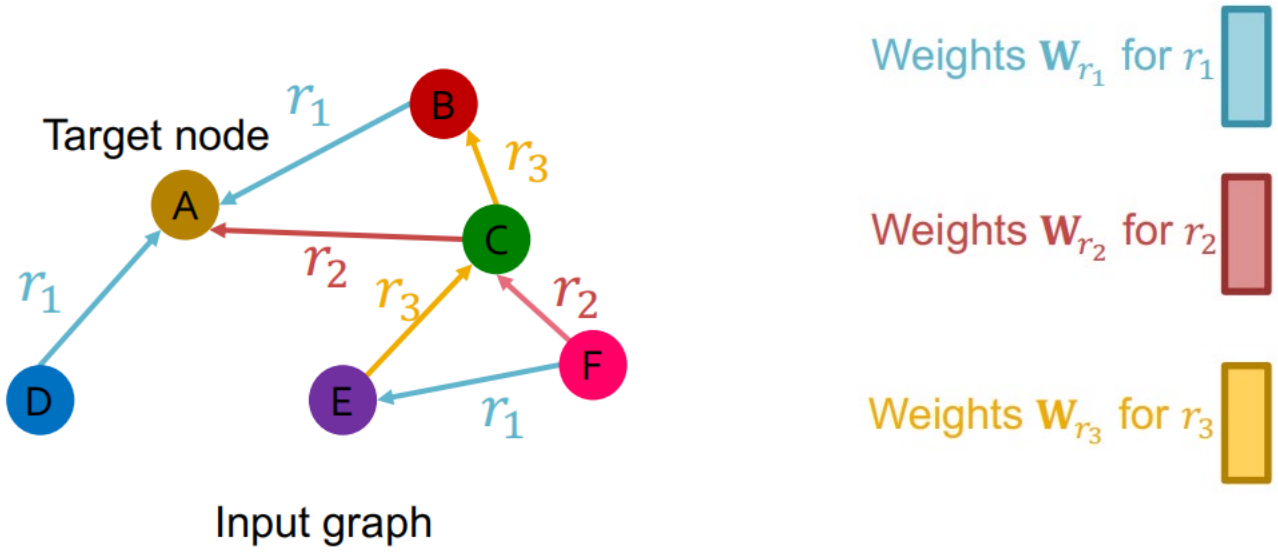
- **We will extend GCN to handle heterogeneous graphs with multiple edge/relation types**
- We start with a directed graph with **one** relation
  - How do we run GCN and update the representation of the **target node A** on this graph?





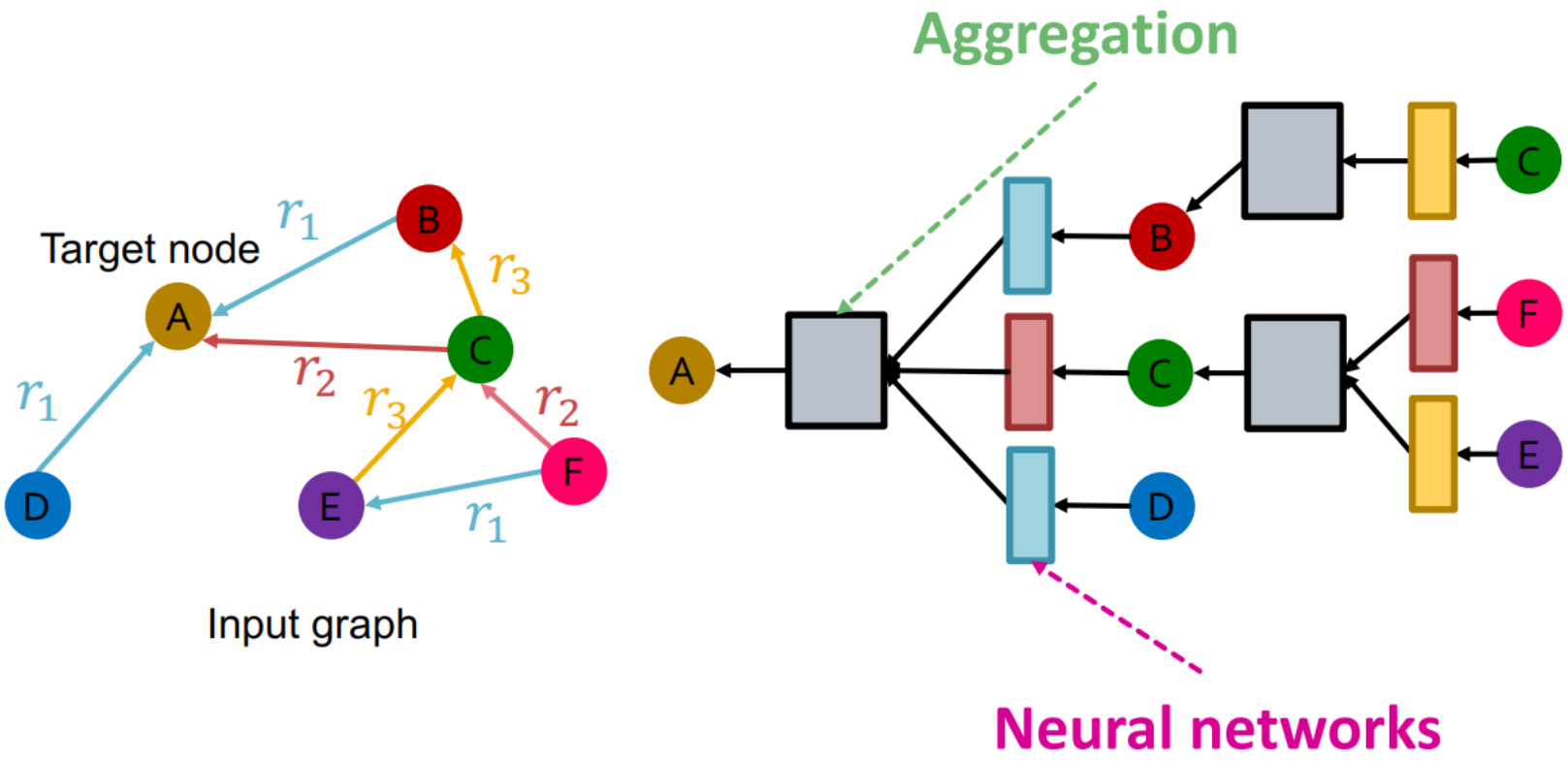
Relational GCN

- **What if the graph has multiple relation types?**
- Use different neural network weights for different relation types.

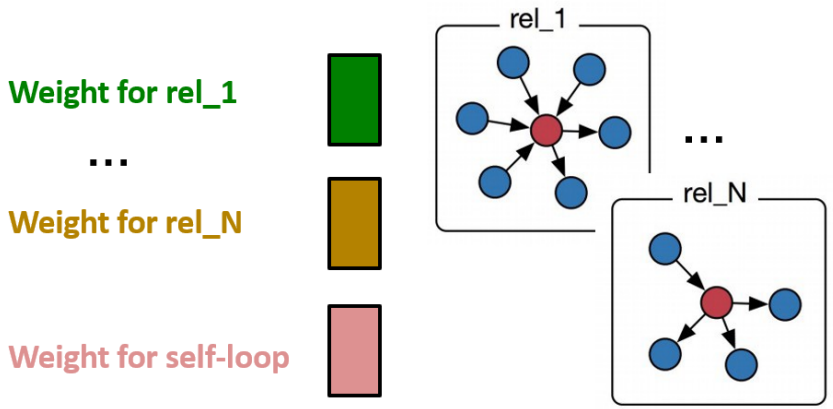


Relational GCN

- **What if the graph has multiple relation types?**
- Use different neural network weights for different relation types.



Relational GCN: Definition



- **Relational GCN (RGCN):**

$$\mathrm{h}_v^{l+1} = \sigma\Big(\sum_{r \in R} \sum_{u \in N_v^r} \frac{1}{c_{v,r}} W_r^l h_u^l + W_0^l h_v^l\Big)$$

- **How to write this as Message + Aggregation?**
- **Message:**
  - Each neighbor of a given relation:

$$m_{u,r}^l = \frac{1}{c_{v,r}} W_r^l h_u^l$$

  - Self-loop:

$$m_v^l = W_0^l h_v^l$$

- **Aggregation:**
  - Sum over messages from neighbors and self-loop, then apply activation
  - $\mathrm{h}_v^{l+1} = \sigma\big(Sum\big(\{m_{u,r}^l, u \in N_v^r\} \cup \{m_v^l\}\big)\big)$

Relational GCN: Scalability

- Each relation has L matrices: $W_r^1, W_r^2, \cdots, W_r^L$
- The size of each W_r^l is $d^{l+1} \times d^l$

  d^l is the hidden dimension in layer l
- **Rapid growth of the number of parameters w.r.t number of relations!**
- **Two methods to regularize the weights**

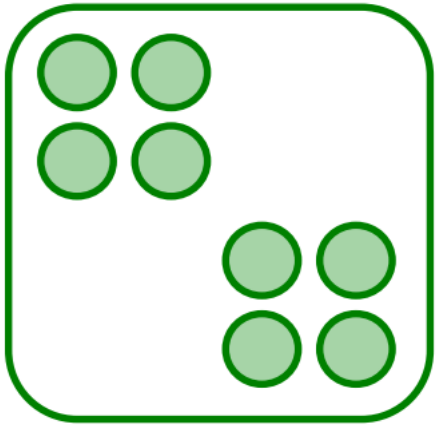
  (1) Use block diagonal matrices

  (2) Basis/Dictionary learning

Block Diagonal Matrices

- **Key insight**: make the weights **sparse**!
- Use **block diagonal matrices** for **W**

$$\mathbf{W}_r =$$



Limitation: only nearby neurons/dimensions can interact through W

- If use B low-dimensional matrices, then # param reduce from $d^{l+1} \times d^l$ to $B \times \frac{d^{l+1}}{B} \times \frac{d^l}{B}$

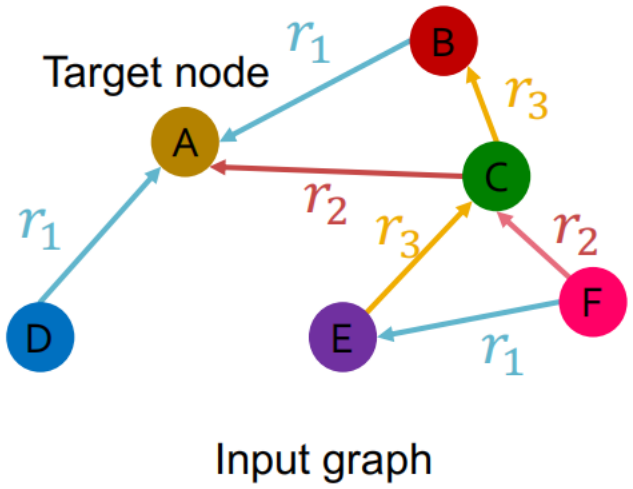
Basis Learning

- **Key insight**: **Share weights** across different relations!
- Represent the matrix of each relation as a linear combination of basis transformations

  $W^r = \sum_{b=1}^{B} a_{rb} V_b$, where V_b is shared across all relations
  - V_b are the basis matrices
  - a_{rb} is the importance weight of matrix V_b
- Now each relation only needs to learn $\{a_{rb}\}_{b=1}^{B}$, which is B scalars

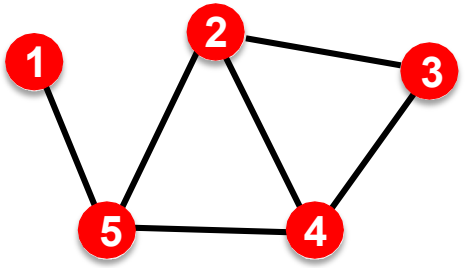
Example: Node Classification

- RGCN uses the representation of the final layer:
  - Take the final layer as the probability of the node belonging to a class



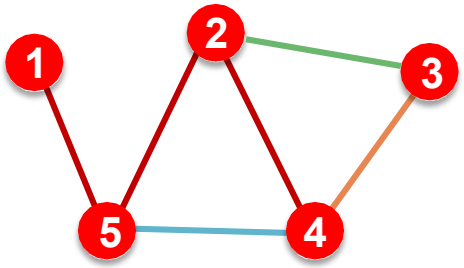
Example: Link Prediction

- **Link prediction split:**



The original graph

Split



Split Graph with 4 categories of edges

Every edge also has a relation type, this is independent of the 4 categories.

In a heterogeneous graph, the homogeneous graphs formed by every single relation also have the 4 splits.

Training message edges for r_1
Training supervision edges for r_1
Validation edges for r_1
Test edges for r_1

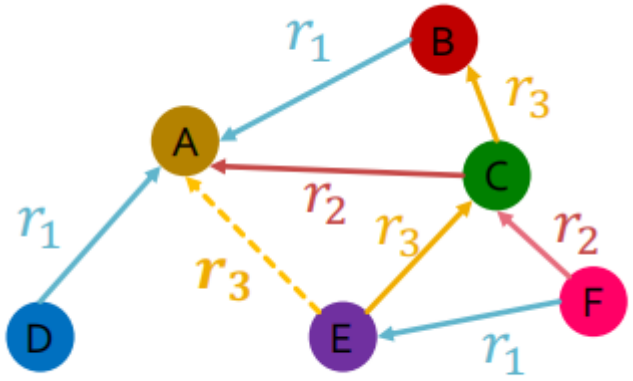
⋮

Training message edges for r_n
Training supervision edges for r_n
Validation edges for r_n
Test edges for r_n

Training message edges
Training supervision edges
Validation edges
Test edges

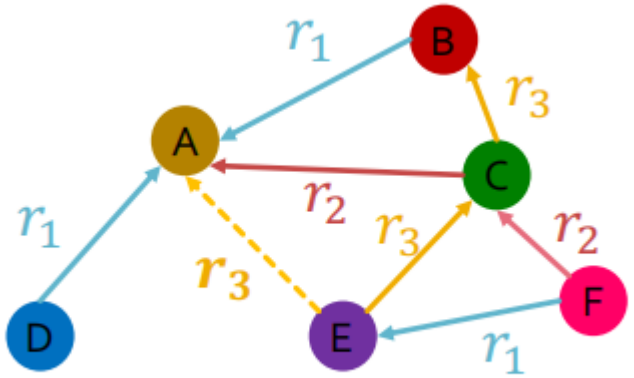
RGCN for Link Prediction

- **Assume (E, r_3, A) is training supervision edge, all the other edges are training message edges**
- **Use RGCN to score (E, r_3, A)**
  - Take the final layer of E and A: h_E^l and $h_A^l \in R^d$
  - Relation-specific score function $f_r: R^d \times R^d \rightarrow R$
    - One example $f_{r_1}(h_E, h_A) = h_E^T W_{r_1} h_A, W_{r_1} \in R^d \times R^d$



Input Graph

RGCN for Link Prediction



Input Graph

1. Use RGCN to score the **training supervision edge** (E, r_3, A)
2. Create a **negative edge** by perturbing the **supervision edge** (E, r_3, B)
   - Corrupt the **tail** of (E, r_3, A)
     - **e.g.,** (E, r_3, B), (E, r_3, D)

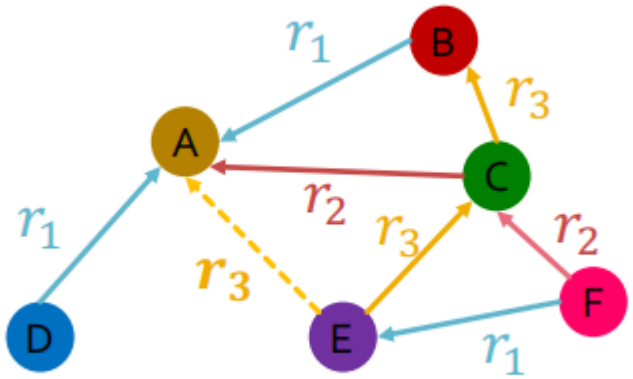
training supervision edges: (E, r_3, A)
training message edges: all the rest existing edges (solid lines)

(1) Use **training message edges** to predict **training supervision edges**

Note the negative edges should NOT belong to training message edges or training supervision edges!
e.g., (E, r_3, C) **is NOT a negative edge**

RGCN for Link Prediction

- **Training:**



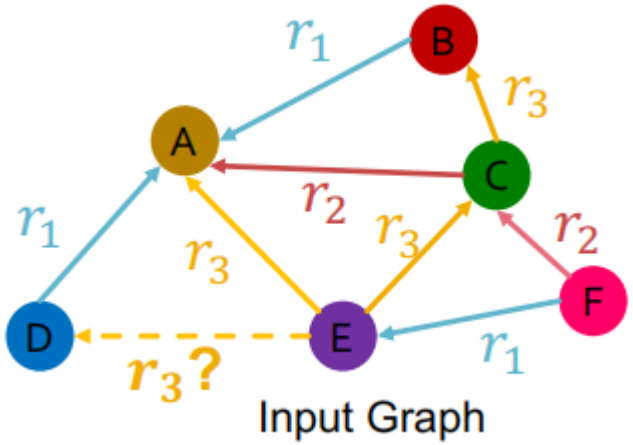
Input Graph

1. Use RGCN to score the **training supervision edge** $(\boldsymbol{E}, \boldsymbol{r_3}, \boldsymbol{A})$
2. Create a **negative edge** by perturbing the **supervision edge** $(\boldsymbol{E}, \boldsymbol{r_3}, \boldsymbol{B})$
3. Use GNN model to score **negative edge**
4. Optimize a standard cross entropy loss
   1. **Maximize** the score of **training supervision edge**
   2. **Minimize** the score of **negative edge**

$$l = -log\sigma\left(f_{r_3}(h_E, h_A)\right) - \log(1 - \sigma(f_{r_3}(h_E, h_B)))$$

RGCN for Link Prediction

- **Evaluation:**
  - Validation and test



- Evaluate how the model can predict the validation edges with the relation types
- predict validation edge (E, r_3, D)
- Intuition: the score of (E, r_3, D) should be higher than all (E, r_3, v) where (E, r_3, v) is **NOT** in the **training message edges** and **training supervision edges**, e.g., (E, r_3, B)

1. Calculate the score of (E, r_3, D)
2. Calculate the score of all the negative edges: $\{(E, r_3, v) | v \in \{B, F\}\}$, since (E, r_3, A), (E, r_3, C) belong to **training message edges & training supervision edges**
3. Obtain the ranking RK of (E, r_3, D)
4. Calculate metrics
   1. **Hits@k**: $\mathbf{1}\,[RK \leq k]$. Higher is better
   2. **Reciprocal Rank**: $\frac{1}{RK}$ Higher is better

Summary

- **In this lecture, we introduced**
  - GCN: the most basic GNN model
  - GraphSAGE: more flexible aggregation
  - GAT: use attention mechanism to evaluate importance of neigborhood
  - Heterogeneous Graphs: when nodes and edges have different types
  - RGCN: GNN for Heterogeneous Graphs