Yale

Graph Attention and Multi-hop Attention

CPSC483: Deep Learning on Graph-Structured Data

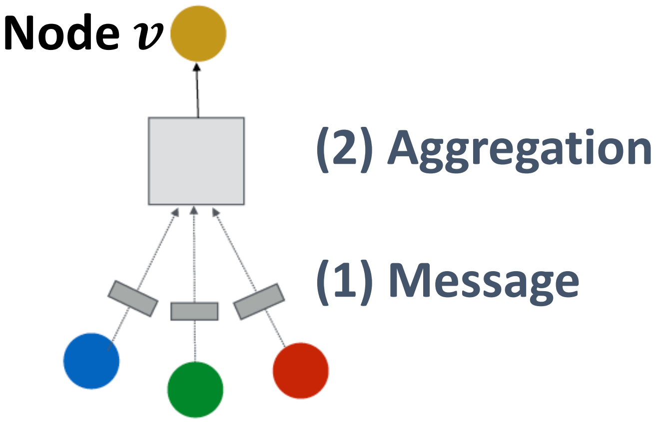
Rex Ying

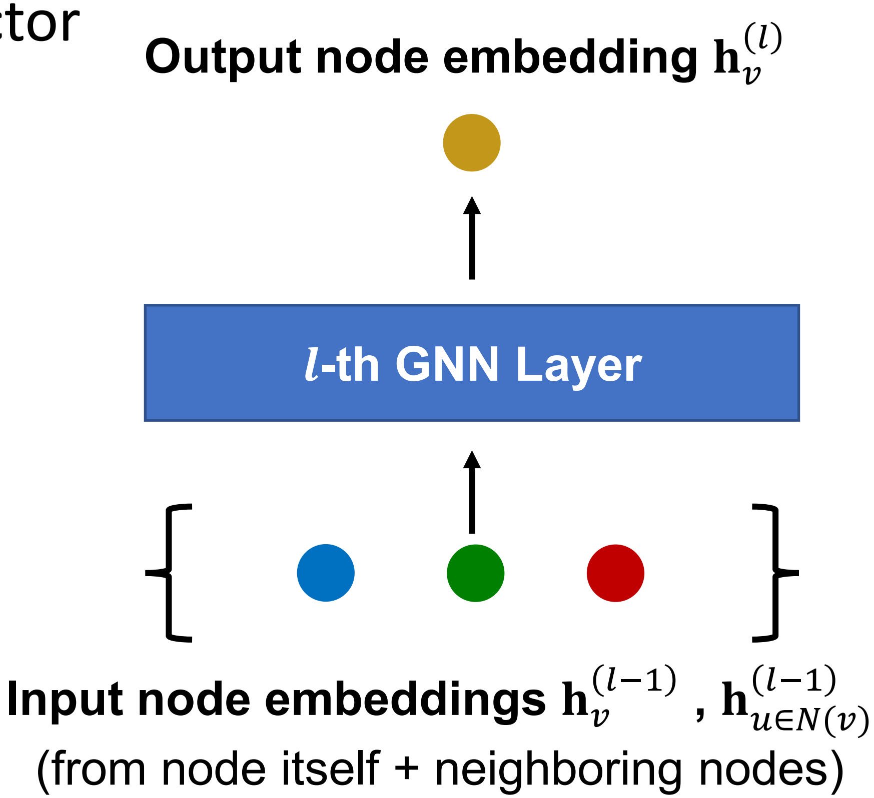
Readings

- Readings are updated on the website (syllabus page)
- **Lecture 6 readings**:
  - GraphSAINT
  - GNN AutoScale
- **Lecture 7 readings:**
  - Graph Attention Networks
  - Multi-hop Attention Graph Neural Networks

Recap: 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**





Recap: Message and Aggregation

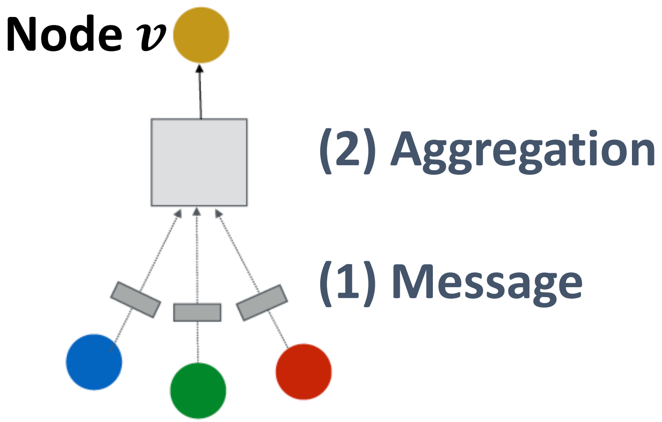
- **Putting things together:**
  - **(1) Message**: each node computes a message

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

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

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

  - **Nonlinearity (activation):** Adds expressiveness
    - Often written as $\sigma(\cdot)$: $\mathrm{ReLU}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, …
    - Can be added to **message or aggregation**



Recap: Classical GNN Layers: GraphSAGE

- **GraphSAGE**

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

- **How to write this as Message + Aggregation?**
  - **Message** is computed within the $\mathrm{AGG}(\cdot)$
  - **Two-stage aggregation**
    - **Stage 1:** Aggregate from node neighbors

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

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

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

Recap: GraphSAGE Neighbor Aggregation

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

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

Aggregation

Message computation

- **Pool:** Transform neighbor vectors and apply symmetric vector function $\text{Mean}(\cdot)$ or $\text{Max}(\cdot)$

$$\text{AGG} = \boxed{\text{Mean}}(\{\boxed{\text{MLP}}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

Aggregation Message computation

- **LSTM:** Apply LSTM to the reshuffled neighbors (not order invariant)

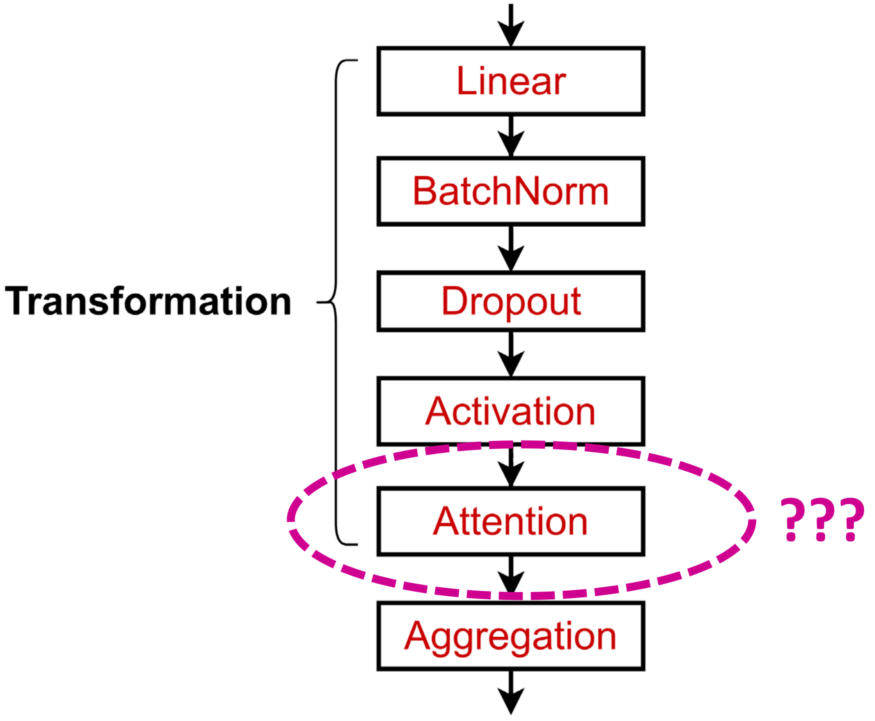
$$\text{AGG} = \boxed{\text{LSTM}}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$$

Aggregation

Recap: GNN Layer in Practice (1)

- **In practice, these classic GNN layers are a great starting point**
  - We can often get better performance by considering a general GNN layer design
  - Concretely, we can include modern deep learning modules that proved to be useful in many domains

An example GNN Layer



Machine Learning Tasks for Graph-structured Data

- **Graph Attention Network**
- **Introduction of Heterogeneous Graph**
- **Multi-hop Attention Graph Neural Network**

Machine Learning Tasks for Graph-structured Data

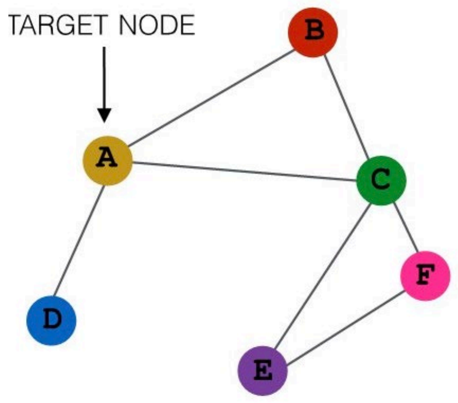
- **Graph Attention Network**
- Introduction of Heterogeneous Graph
- Multi-hop Attention Graph Neural Network

Setup

- **Assume we have a graph $\boldsymbol{G}$:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacent matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{d \times |V|}$ is a matrix of **node features**
    - v: a node in V; N_v: the set of neighbors of v
  - **Node features:**
    - Social networks: User profile, User image
    - Biological networks: Gene expression profiles, gene functional information
    - When there is no node feature in the graph dataset:
      - Indicator vectors (one-hot encoding of a node)
      - Vector of constant 1: [1, 1, …, 1]

Neighborhood Aggregation: Review

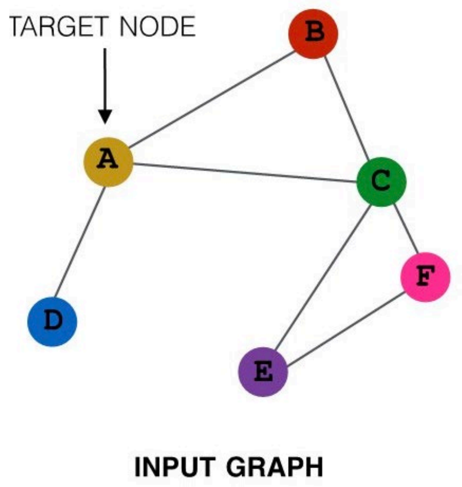
- How can a node aggregate information from their neighborhood?

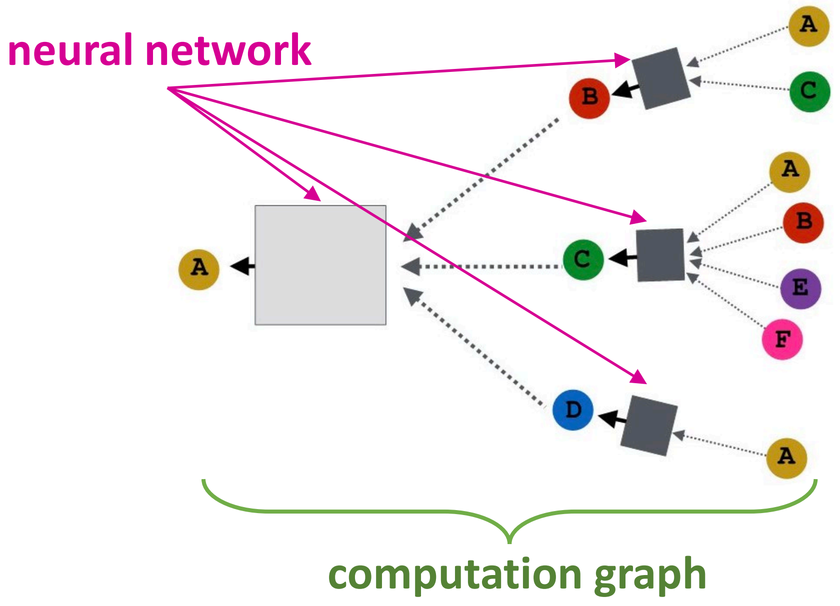


INPUT GRAPH

Neighborhood Aggregation: Review

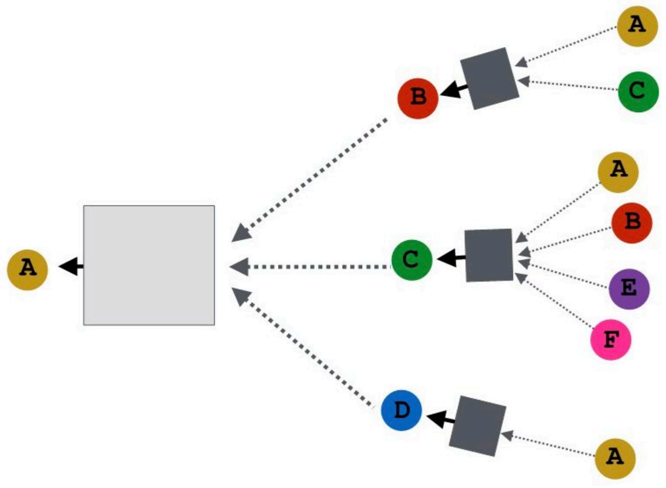
- How can a node aggregate information from their neighbors?
  - Firstly, build a **computation graph** based on its neighborhood
  - Then, average neighbor messages and apply a **neural network**





Neighborhood Aggregation: Review

- Message Aggregation details



Non-linearity

Embedding of u at layer l

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

Neighbors of v

Weighting factor of u's message to v

Learnable parameter

Importance of Neighbor

Weighted sum for each
u's message to v

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

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

Graph Attention Network (GAT)

Weighted sum for each u's message to v

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

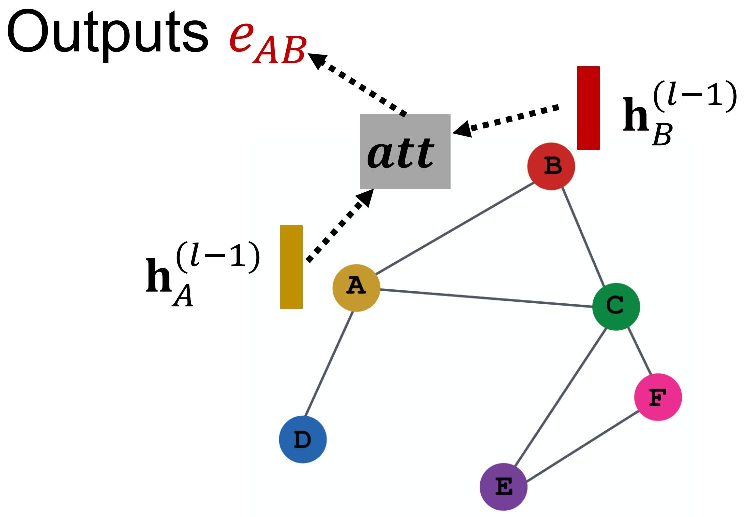
- **Can we do better than simple neighborhood aggregation?**
  - Let weighting factors α_{vu} to be learned!
- **Goal:** Specify **arbitrary importance** to different neighbors of each node in the graph
- **Idea:** Compute embedding $\boldsymbol{h}_v^{(l)}$ of each node in the graph following an **attention strategy**:
  - Nodes attend over nodes in their neighborhoods
  - Determine weights for different nodes in a neighborhood through optimization

Attention Mechanism (1)

- Let a be an **attention mechanism**
  - Attention coefficient e_{vu} is computed by att based on the messages of v, u:

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

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



Attention Mechanism (2)

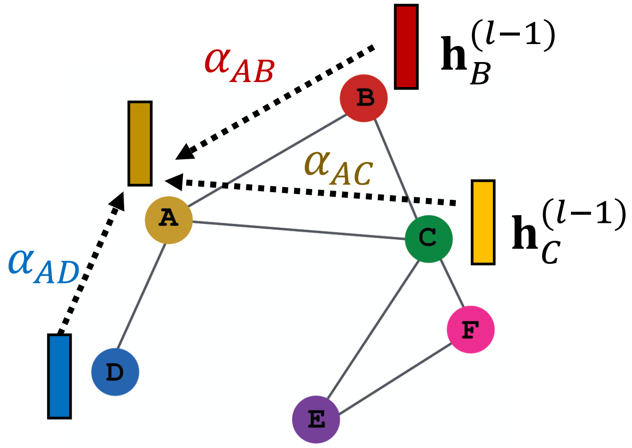
- **Normalize** e_{vu} into the **final attention weight** $\boldsymbol{\alpha_{vu}}$
  - Apply the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:

v's Attention to u: $$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N_v} \exp(e_{vk})}$$

Exponential function

- Aggregate the information based on α_{vu}:

$$\mathbf{h}_v^{(l)} = \sigma(\textstyle\sum_{u \in N_v} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$



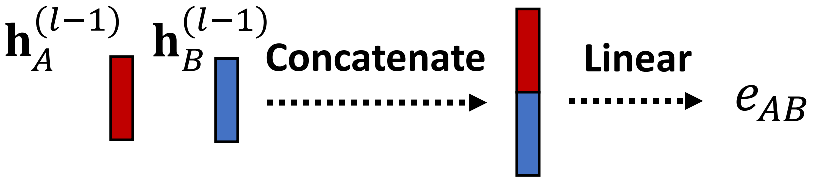
Weighted sum using α_{AB}, α_{AC}, α_{AD}:

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

Attention Mechanism (3)

- What is the form of **attention mechanism** att ?
  - The approach is agnostic to the choice of att
    - E.g., use a concatenate-based neural network

Recall edge-level prediction head in lecture 5

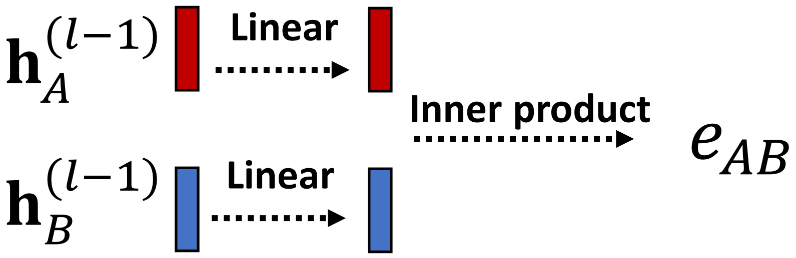


$$e_{AB} = att\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)$$
$$= \mathrm{Linear}\left(\mathrm{Concat}\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)\right)$$

    - att have trainable parameters (weights in the Linear layer)
      - Learn the parameters together with weight matrices (i.e., other parameter of the neural net $\mathbf{W}^{(l)}$) in an end-to-end fashion

Attention Mechanism (4)

- The approach is **agnostic** to the function we use to compute e
  - E.g., use inner product



$$e_{AB} = att\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)$$
$$= \mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)} \cdot \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}$$

Other functions to combine two vectors can be used as well.
For example, **bilinear form**

Multi-head Attention

- **Multi-head attention:** Stabilizes the learning process of attention mechanism
  - **Run through several attention heads with different parameters (vector computation)**:

$$\mathbf{h}_v^{(l)}[1] = \sigma(\sum_{u\in N(v)} \alpha_{vu}^1 \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)})$$
$$\mathbf{h}_v^{(l)}[2] = \sigma(\sum_{u\in N(v)} \alpha_{vu}^2 \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)})$$
$$\mathbf{h}_v^{(l)}[3] = \sigma(\sum_{u\in N(v)} \alpha_{vu}^3 \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)})$$

$\alpha_{vu}^1, \alpha_{vu}^2, \alpha_{vu}^3$ are Calculated by $\boldsymbol{a}^{(l)}[1], \boldsymbol{a}^{(l)}[2], \boldsymbol{a}^{(l)}[3]$ respectively

  - Outputs are aggregated:
    - By concatenation or summation
    - $\mathbf{h}_v^{(l)} = \mathrm{AGG}(\mathbf{h}_v^{(l)}[1], \mathbf{h}_v^{(l)}[2], \mathbf{h}_v^{(l)}[3])$

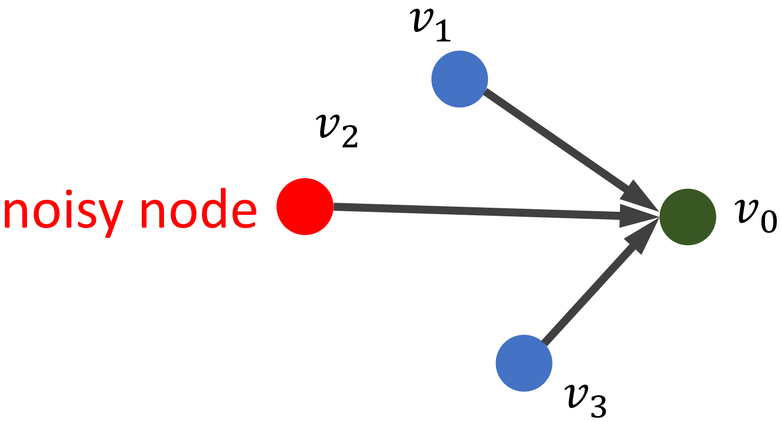
Graph Attention Network (GAT)

Learnable single-head or multi-head attention mechanism

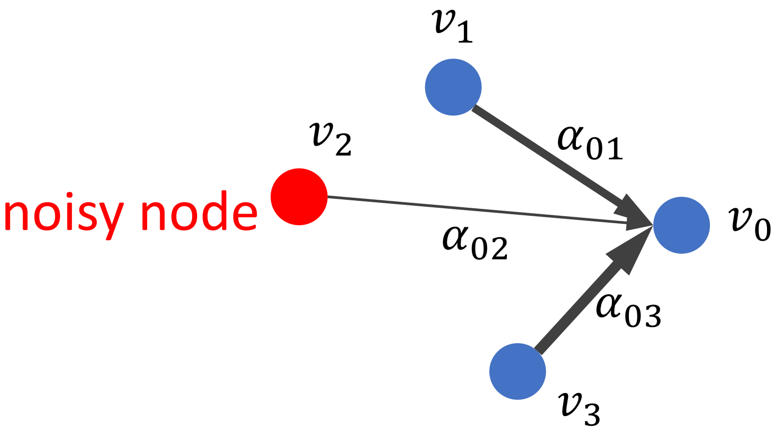
- A GAT layer (single head):
  - **Attention** computing: calculate the importance of neighbors
$$\alpha_{vu} = att\left(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)}\right)$$
  - **Message** computing: transform information of neighbor node to a message
$$\mathbf{m}_u^{(l)} = \alpha_{vu}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, u \in N_v$$
  - **Aggregate** message: aggregate messages from neighbor nodes
$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N_v} \mathbf{m}_u^{(l)}\right)$$

Benefit of Attention Mechanism (1)

- Allow GNNs to **adaptively** assign different weights to neighbors during training
  - In cases where the graphs contain many **noise**
    - Nodes with inaccurate feature
    - Edges that are incorrect



GCN: Aggregate the messages from neighbors with **same** weights

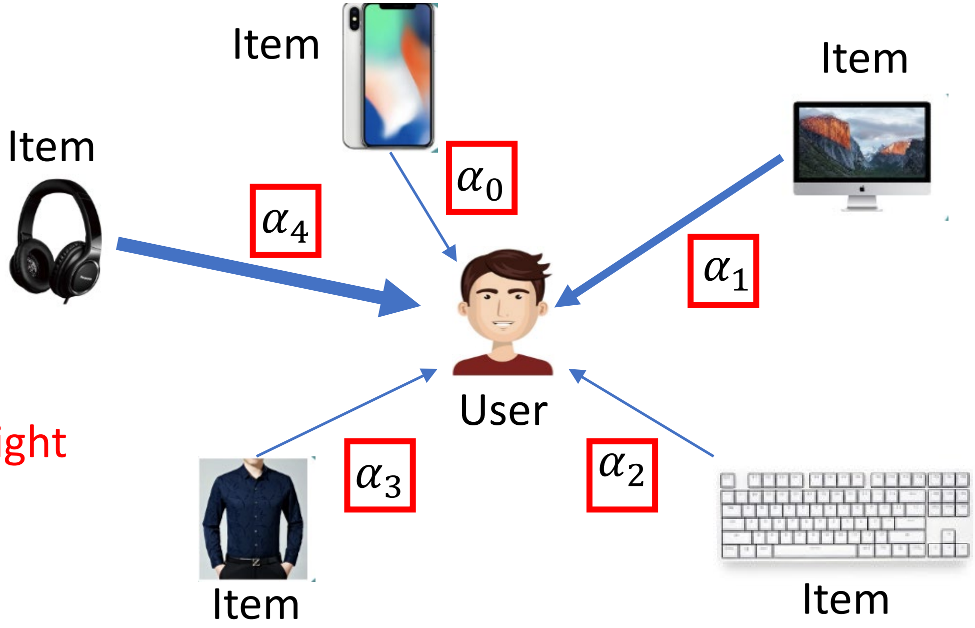


$\alpha_{01} > \alpha_{02}$
$\alpha_{03} > \alpha_{02}$

GAT: Aggregate the messages from neighbors with **adaptive** weights

Benefit of Attention Mechanism (2)

- Learnable weighting function can provide a good **interpretability**
  - Different edge weights indicates difference importance of the neighbor nodes
    - We can simply sum up the attention scores of all layers between two nodes
  - Take recommender system as an example



α_i: edge weight

- User aggregates the information from items with different weights
  - High attention weights indicate that user prefers these corresponding items

Machine Learning Tasks for Graph-structured Data

- Graph Attention Network
- **Introduction of Heterogeneous Graph**
- Multi-hop Attention Graph Neural Network

Heterogeneous Graph (1)

- What is heterogeneous graph (HG)?
  - A graph with multiple **node types** and **edge types**
- Example: E-Commerce graph
  - **Node types:** User, Item, Query, Location, …
  - **Edge types:** Purchase, Visit, Guide, Search, Click, …
  - Different node type's feature spaces can be different!

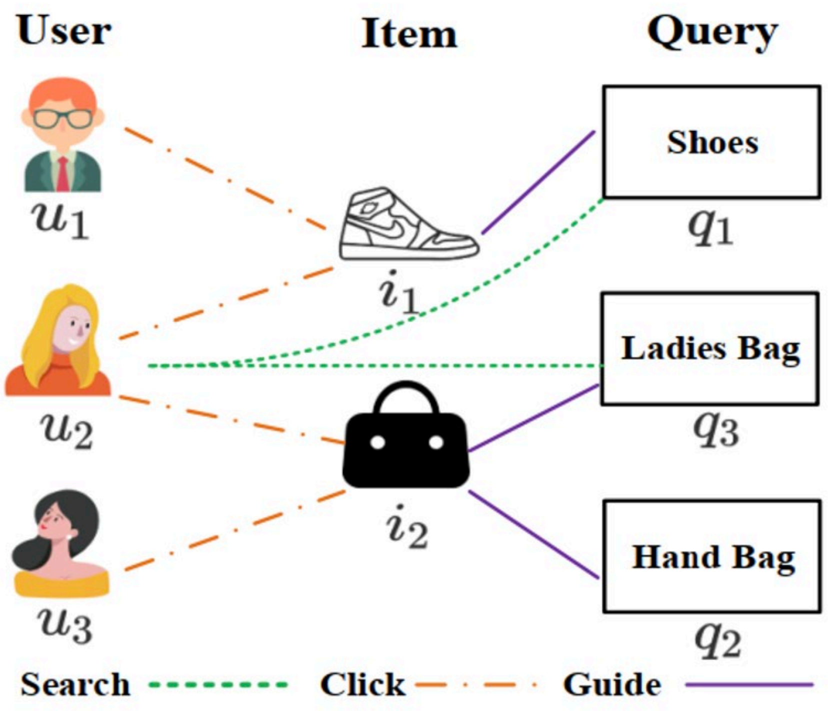
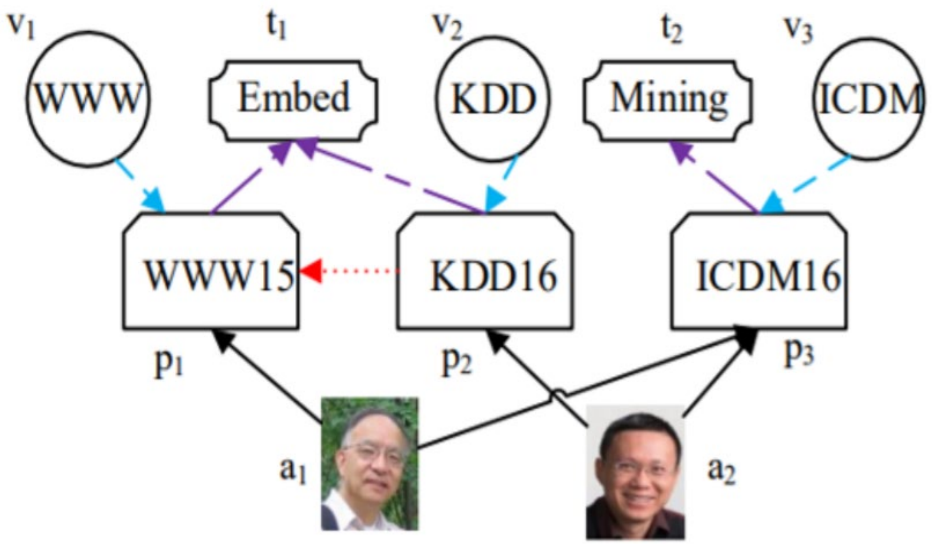


Image source

Heterogeneous Graph (2)

- Example: **Academic Graph**
  - Node type: Author, Paper, Venue, Field, …
  - Edge type: Publish, Citation, …

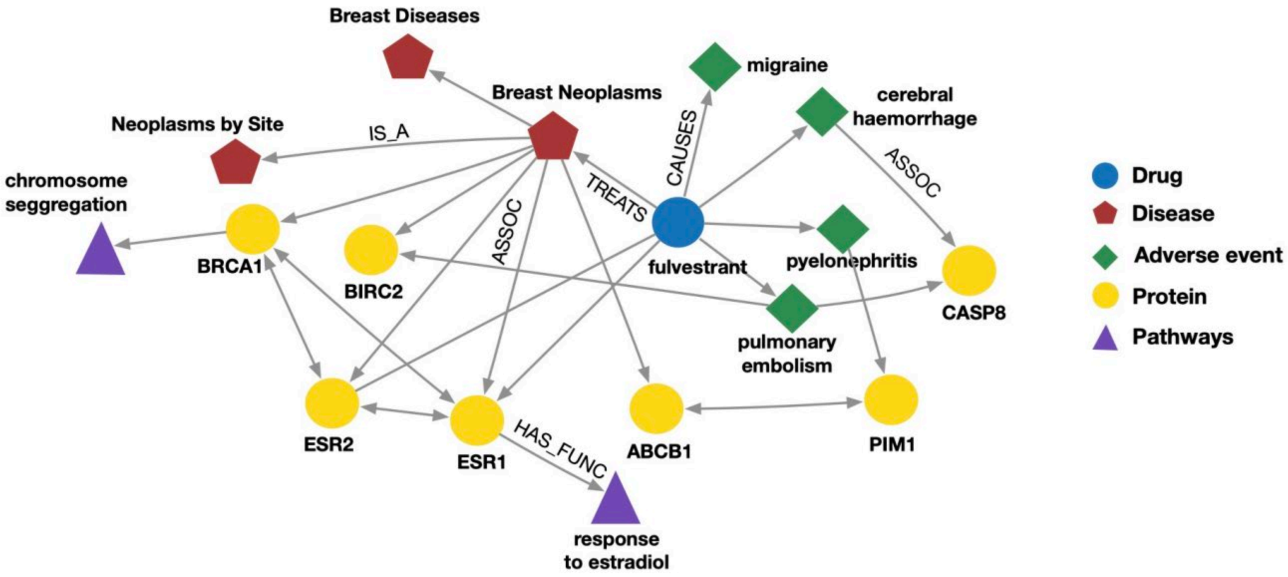


Node type: a: Author, t: Field, v: Venue
Edge type: p: Publish

Image source

Heterogeneous Graph (3)

- Example: **Biomedical Graph**
  - Node type: Drug, Disease, Protein, …
  - Edge type: Associate, Treat, Cause, …

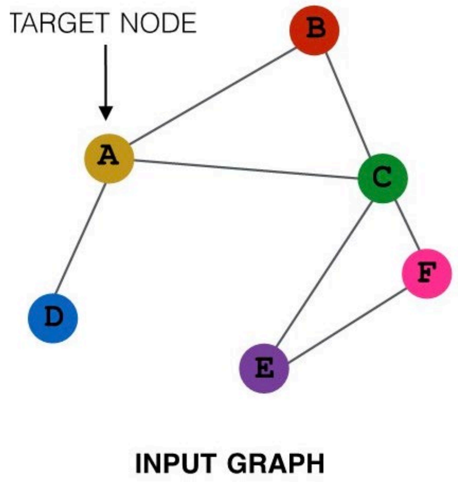


Heterogeneous Graph (3)

- **Heterogeneous Graph (HG)** $\boldsymbol{G(V, E, T, R)}$
  - V is the **vertex** set $\{v_i\}$
    - N_v: the set of neighbors of v
  - E is the **edge** sets with edge type $(v_i, r, v_j) \in E$
    - N_v^r: the set of neighbors with relation r of v
    - $|N_v^r|$: the set size of N_v^r
  - T is the **node type** set
  - R is the **edge(relation) type** set $r \in R$
    - $|R|$: the number of relations

Homogeneous GCN (1)

- How to learn the representation of node and edge in HG?
  - Previous GNNs (GCN, GraphSAGE, GAT) focus on **homogeneous** graphs
  - How to extend the GCN to **handle heterogeneous graphs?**
  - Recall the way GCN performs message passing on homogeneous graphs:
    - Message function
    - Aggregation of messages



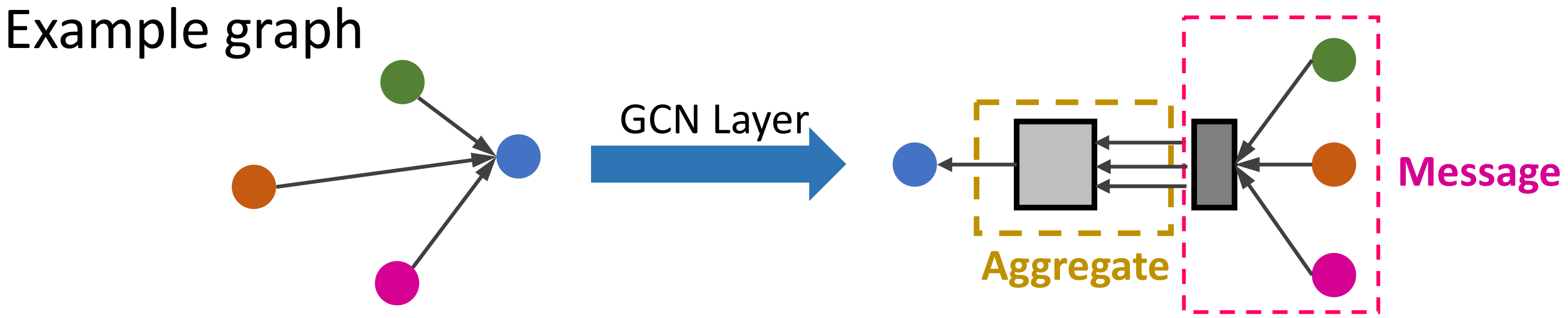
Homogeneous GCN (2)

- A GCN layer:
  - **Message** computing: transform information of neighbor node to a message

$$\mathbf{m}_u^{(l)} = \alpha_{vu}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, u \in N_v$$

  - **Aggregate** message: aggregate messages from neighbor nodes

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

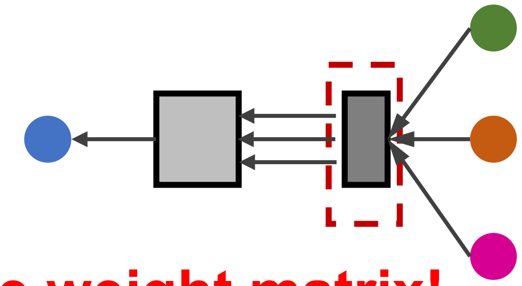


Homogeneous GCN (3)

- A GCN layer:
  - **Message** computing: transform information of neighbor node to a message

$$\mathbf{m}_u^{(l)} = \alpha_{vu}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, u \in N(v)$$

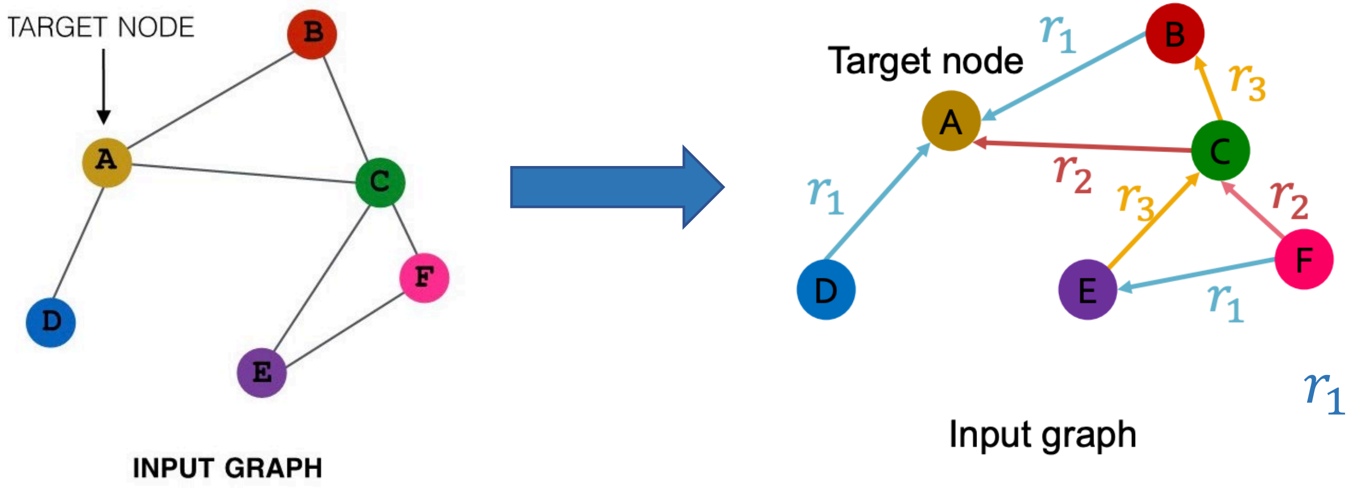
  - In **homogeneous** graph, we use **same** weight matrix to perform message computing



Same weight matrix!

Relational GCN (1)

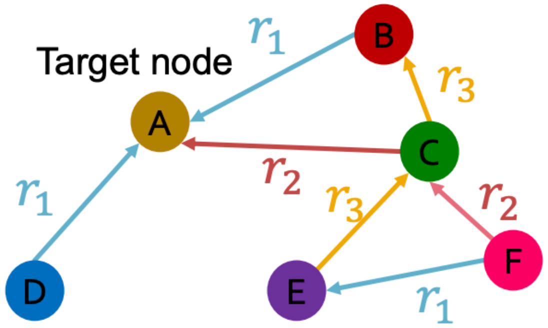
- How about the graph with multiple relational types?



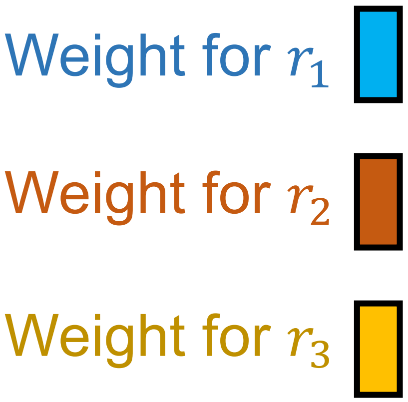
r_1, r_2, r_3: **edge types**

Relational GCN (2)

- How about the graph with multiple relational types?
  - Extend GCN to **Relational GCN** (RGCN)!
  - Use different weight matrix of **message process** for different relation types

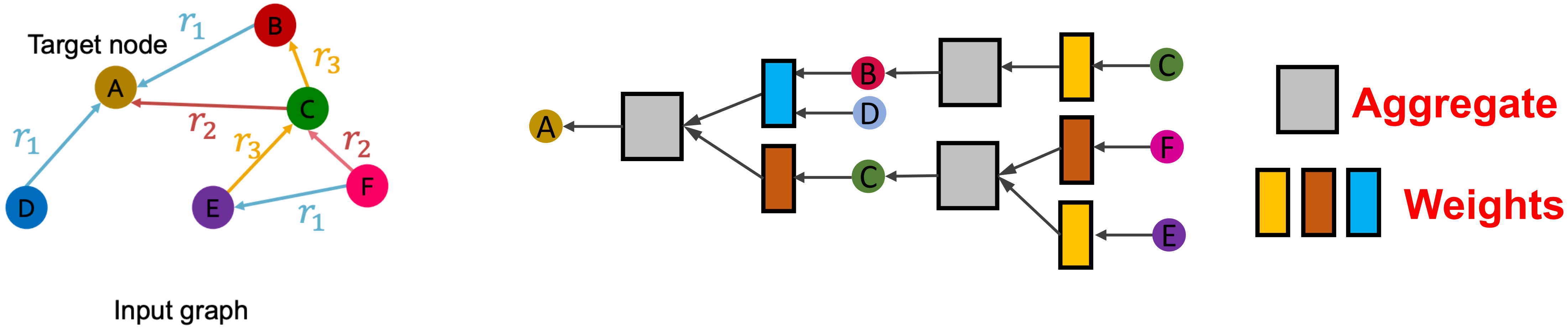


Input graph



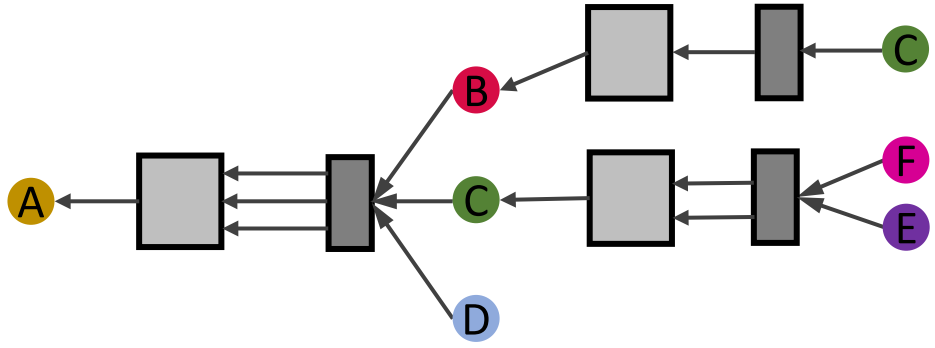
Relational GCN (3)

- How about the graph with multiple relational types?
  - Extend GCN to **Relational GCN** (RGCN)!
  - Use different weight matrix of **message process** for different relation types

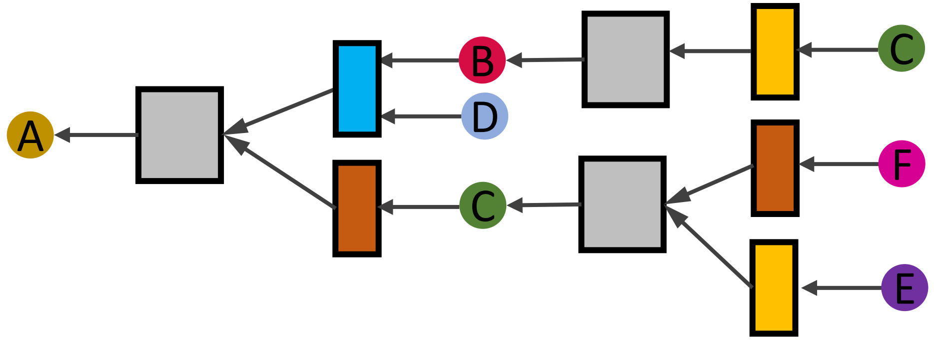


Relational GCN (4)

- Comparison between GCN and Relational GCN



All the nodes in a layer share same weight



Node with different type uses different weight

Relational GCN (5)

- A Relational GCN layer:
  - **Message** computing: transform information of neighbor node of relation r to a message

Normalized by node degree of the relation

$$\mathbf{m}_{u,r}^{(l)} = \frac{1}{|N_v^r|} \mathbf{W}_r^{(l)} \mathbf{h}_u^{(l-1)}, u \in N_v^r$$

  - **Aggregate** message: aggregate messages from neighbor nodes

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{r \in R} \sum_{u \in N_v^r} \mathbf{m}_{u,r}^{(l)}\right)$$

For every relation type

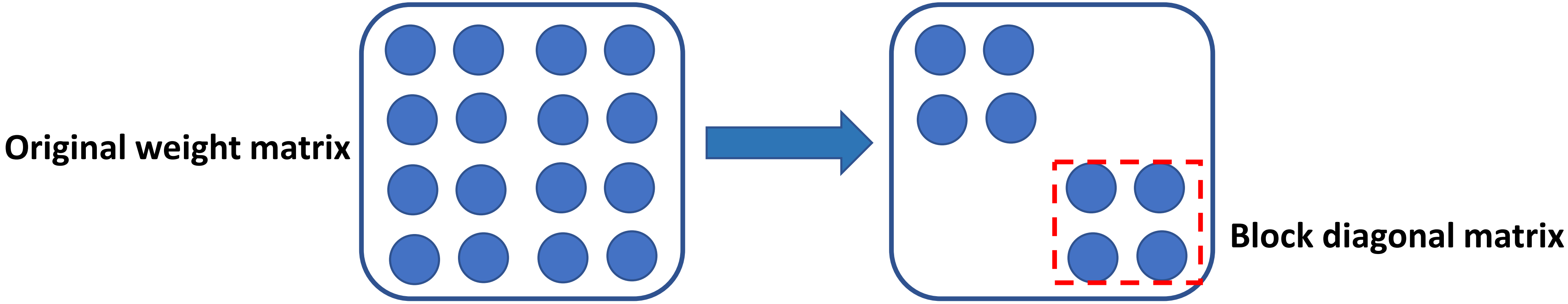
Scalability of RGCN (1)

- **Parameters of RGCN**
  - Suppose we have L layers, $|R|$ relations, and the hidden size d at every layer is identical
  - For each layer, model has $|R|$ weights $\mathbf{W}_{\boldsymbol{r}}^{(l)} \in \mathbb{R}^{d \times d}$. So parameters of every layer can be $|R| \times d^2$
  - Considering L layers, the parameters of RGCN can be $L \times |R| \times d^2$!
  - How to improve the scalability of RGCN?

Scalability of RGCN (2)

- Block Diagonal Matrix
  - Use **sparser** weight matrices
  - Parameter of a weight matrix can be reduced from d^2 to $\frac{d^2}{B}$

B is the number of block diagonal matrix



Scalability of RGCN (3)

- Basis Learning
  - Share weights across different relations
  - Use B **shared** basis matrices $\mathbf{M}_b^{(l)}$ and **relation-specific** learnable weight $a_{rb}^{(l)}$ to represent a relation matrix:

$$\mathbf{W}_{\boldsymbol{r}}^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)} \cdot \mathbf{M}_b^{(l)}$$

  - Parameter of all weight matrices in a layer can be reduced from $|R|d^2$ to $|R|B + Bd^2$

***B* scalars** $\{\boldsymbol{a}_{\boldsymbol{r0}}^{(\boldsymbol{l})}, \dots, \boldsymbol{a}_{\boldsymbol{rB}}^{(\boldsymbol{l})}\}$ **Basis matrix**

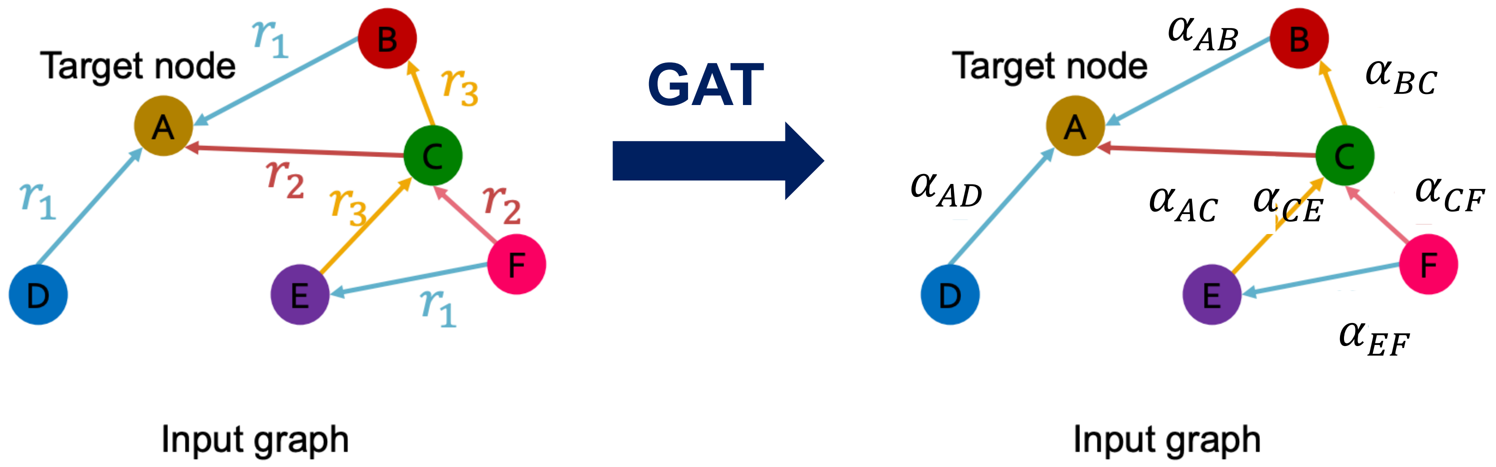
Machine Learning Tasks for Graph-structured Data

- Graph Attention Network
- Introduction of Heterogeneous Graph
- **Multi-hop Attention Graph Neural Network**

  Multi-hop attention graph neural network. *IJCAI 2020*

Homogeneous GAT (1)

- How to extend Graph Attention Network (GAT) to heterogenous graphs?



α: edge weight produced by GAT

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Homogeneous GAT (2)

Learnable attention mechanism

- A GAT layer:
  - **Attention** computing: calculate the importance of neighbors

$$\alpha_{vu} = a\left(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)}\right)$$

  - **Message** computing: transform information of neighbor node to a message

$$\mathbf{m}_u^{(l)} = \alpha_{vu}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, u \in N_v$$

  - **Aggregate** message: aggregate messages from neighbor nodes

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N_v} \mathbf{m}_u^{(l)}\right)$$

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Homogeneous GAT (3)

Learnable weights for source node and target node

- Attention mechanism a: **Linear layer**
  - Compute **attention coefficient** e_{vu} based on v, u:

$$e_{vu} = \text{Linear}\left(\text{Concat}\left(\mathbf{W}_t^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}_s^{(l)}\mathbf{h}_v^{(l-1)}\right)\right)$$

  - **Normalize** e_{vu} by the softmax function

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

N_v **: neighborhood nodes of** v

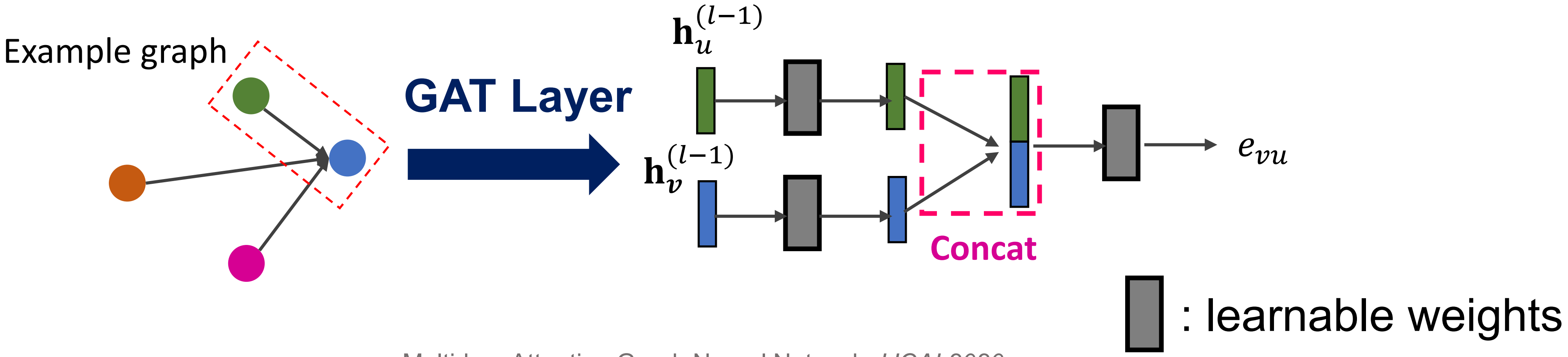
Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Homogeneous GAT (4)

- Attention mechanism a:
  - Compute **attention coefficient** e_{vu} based on v, u:

$$e_{vu} = \text{Linear}\left(\text{Concat}\left(\mathbf{W}_t^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}_s^{(l)}\mathbf{h}_v^{(l-1)}\right)\right)$$

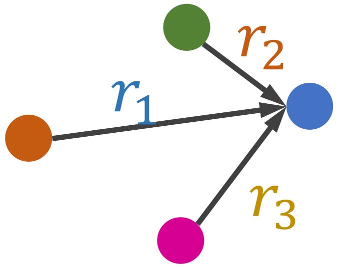
  - Assign learnable weights on node embeddings:



Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Heterogeneous GAT (1)

- How to compute attention coefficient with edge type?
  - Similar as RGCN, we can use an additional weight to model the relation type!



Weights for r_1

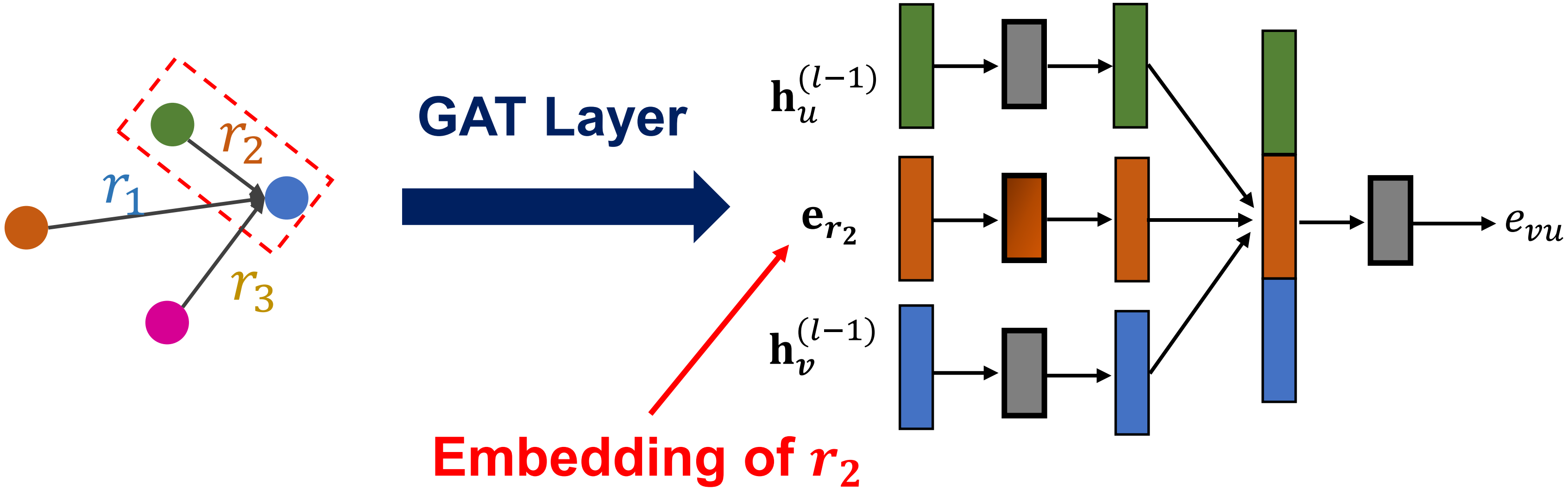
Weights for r_2

Weights for r_3

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Heterogeneous GAT (2)

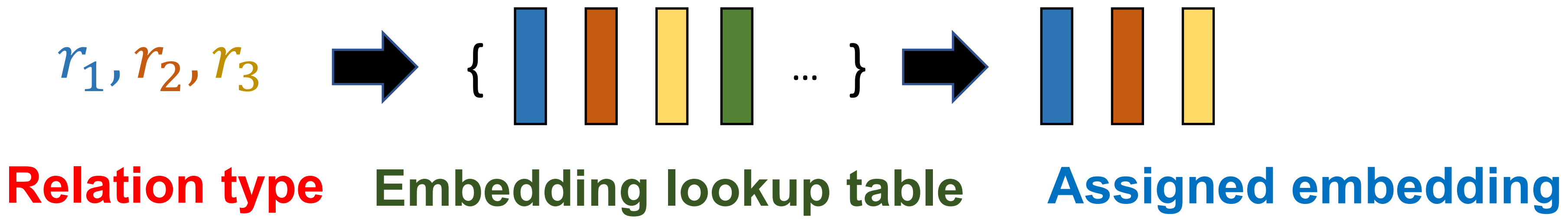
- How to compute attention coefficient with edge type?
  - Similar as RGCN, we can use an additional weight to model the relation type!
  - Attention mechanism with relation:



Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Heterogeneous GAT (3)

- How to encode the relation type into a vector?
  - Simplest way: assign a learnable vector to every relation type



Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Heterogeneous GAT (4)

- Attention mechanism att for heterogeneous graph:
  - Compute **attention coefficient** e_{vu} based on v, u, r_{vu}:

$$e_{vu} = \mathrm{Linear}\left(\mathrm{Concat}\left(\mathbf{W}_t^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}_s^{(l)}\mathbf{h}_v^{(l-1)}, \mathbf{W}_{r_{vu}}^{(l)}\mathbf{e}_{r_{vu}}\right)\right)$$

$\boldsymbol{r_{vu}}$: relation type between $\boldsymbol{v}$ and $\boldsymbol{u}$

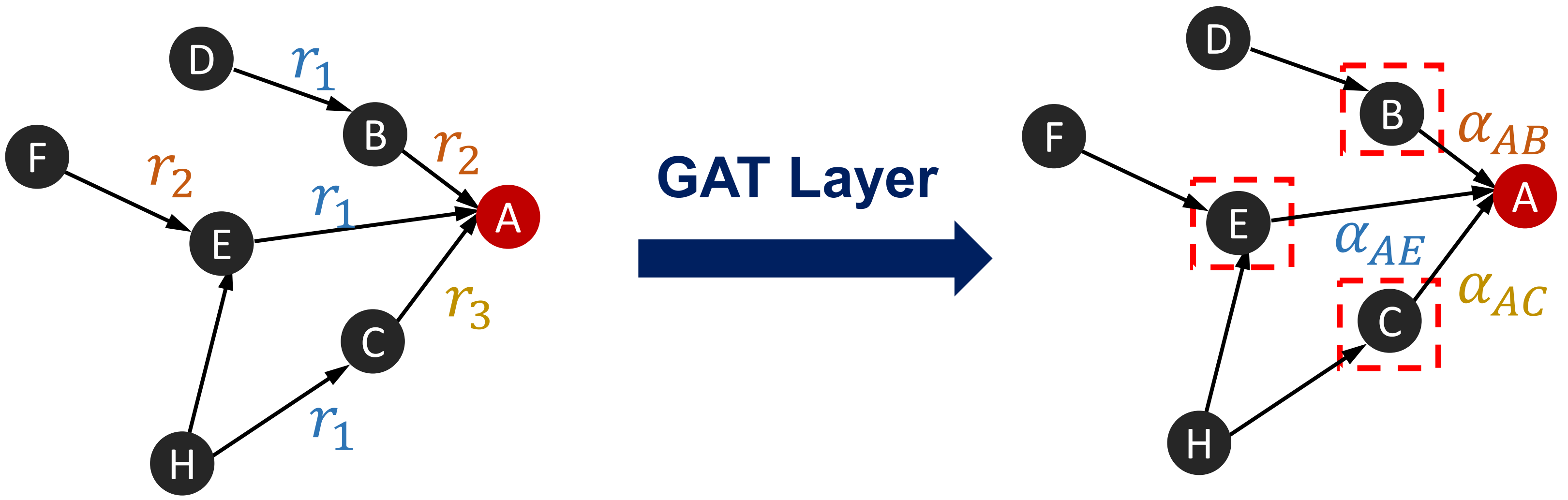
  - **Normalize** e_{vu} by softmax function

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

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

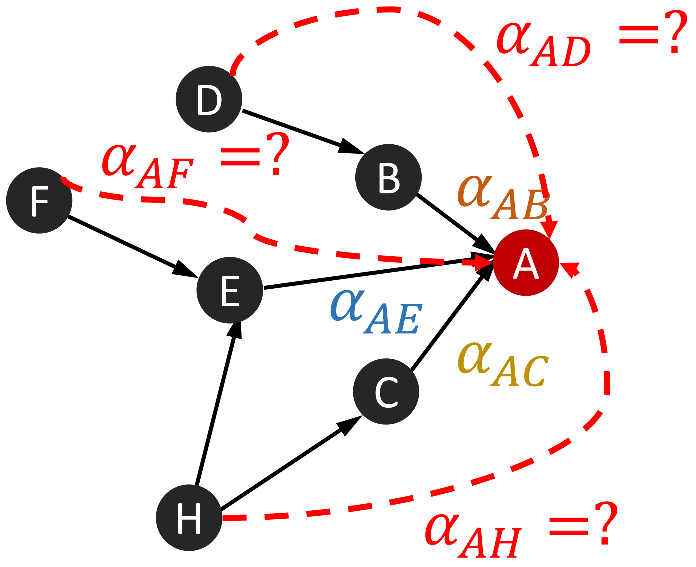
Limitation of Single Hop Attention (1)

- A single GAT layer can only explore the relationship between a node and its **one-hop** neighbors
  - Target node only attends to its immediate neighbors



Limitation of Single Hop Attention (2)

- A single hop attention falls short in exploring **broader graph structure** and **multi-hop** neighbors
  - Stacking multiple GAT layers causes **over-smoothing** and **over-fitting**

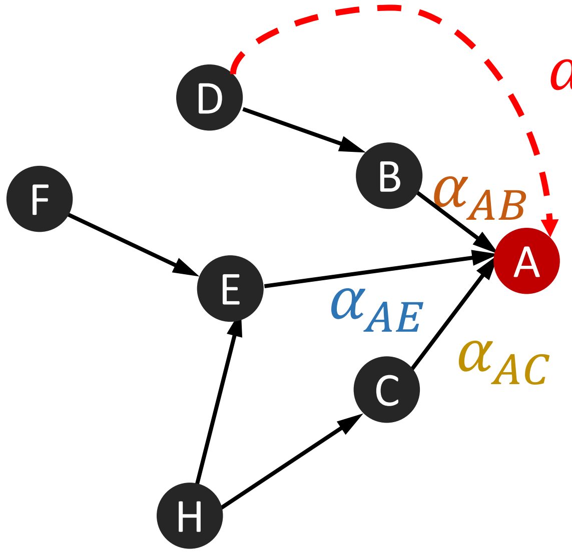


Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Benefit of Multi-Hop Attention (1)

- Benefit of using multi-hop neighbors in a GAT layer
  - Exploit **important** nodes that are **not** directed connected
  - **Less number of message-passing layers** is needed to propagate information

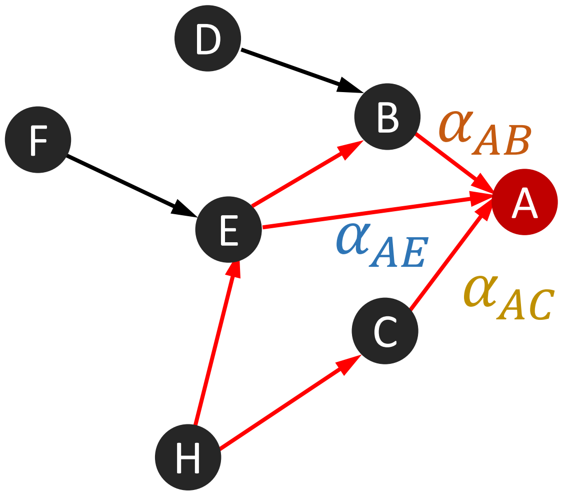
$\alpha_{AD} \gg \alpha_{AB}, \alpha_{AE}, \alpha_{AC}$



Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Benefit of Multi-Hop Attention (2)

- Benefit of using multi-hop neighbors in a GAT layer
  - Attention score **not only** depends on node representation
  - Compute the attention score over **all the possible paths** connecting two nodes



All possible paths between A and E

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

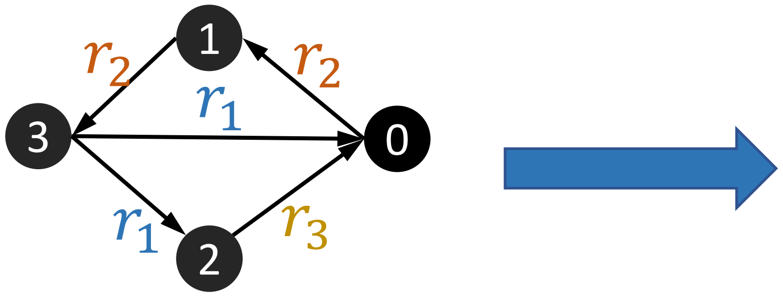
Multi-Hop Attention (1)

- How to incorporate multi-hop neighbors in a GAT layer?
  - We can first calculate the attention of one-hop neighbors

$$\alpha_{vu} = a(\mathbf{h}_u, \mathbf{h}_v, \mathbf{e}_{r_{vu}})$$

Relation embedding

  - Attention scores can be organized as an adjacent matrix A:



$$A = \begin{bmatrix} 0 & 0 & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_{23} \\ 0 & \alpha_{31} & 0 & 0 \end{bmatrix}$$

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Multi-Hop Attention (2)

- Each node can access its l-hop neighbors by $A^l = \overbrace{AAA}^{l}\cdots$
  - For example, A_{ij}^2 sums up number of **all the paths** of length 1 between **each of v_i's neighbors and v_j**

v_0's neighbors: v_2, v_3 **$v_0 \rightarrow v_3 \rightarrow v_1$**

$$A^2 = \begin{bmatrix} 0 & 0 & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_{23} \\ 0 & \alpha_{31} & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_{23} \\ 0 & \alpha_{31} & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \alpha_{03}\alpha_{31} & 0 & \alpha_{02}\alpha_{23} \\ 0 & 0 & \alpha_{10}\alpha_{02} & \alpha_{10}\alpha_{13} \\ 0 & \alpha_{23}\alpha_{31} & 0 & 0 \\ \alpha_{31}\alpha_{10} & 0 & 0 & 0 \end{bmatrix}$$

v_3's neighbors: v_1

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Multi-Hop Attention (3)

- How to incorporate multi-hop neighbors in a GAT layer?
  - Attention **diffusion**!

$$\mathcal{A} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k A^k, 0 < \alpha < 1$$

  - Increasing the receptive field of the attention
    - Attention between two nodes not only depends on node representation, but also the **paths** between them:

$$\mathcal{A}_{ij} = \alpha A_{ij}^0 + \alpha(1-\alpha)A_{ij}^1 + \alpha(1-\alpha)^2 A_{ij}^2 + \alpha(1-\alpha)^3 A_{ij}^3 + \cdots$$

1-hop path attention between v_i and v_j

2-hop path attention between v_i and v_j

3-hop path attention between v_i and v_j

Multi-Hop Attention (4)

- Why do we need $\alpha(1-\alpha)^k$?
  - It can control the weight of attention score of different hop
  - Nodes further away should be weighted **less** in message aggregation!
  - For example, $\alpha = 0.5$:

$$\mathcal{A}_{ij} = 0.5A_{ij}^0 + 0.25A_{ij}^1 + 0.125A_{ij}^2 + 0.0625A_{ij}^3 + \cdots$$

Weight decays gradually

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Multi-Hop Attention GNN

- Multi-hop attention GNN:
  - **One-hop** attention computing:

Attention score between u, v

$$A_{vu}^{(l-1)} = att(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}, \mathbf{e}_{r_{vu}})$$

  - Building **multi-hop** attention diffusion matrix:

$$\mathcal{A} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k {A^{(l-1)}}^k, 0 < \alpha < 1$$

  - **Aggregate** message: aggregate messages based on multi-hop attention

$$\mathbf{h}_v^{(l)} = \sum_{u \in N_v} \mathcal{A}_{vu} \mathbf{h}_u^{(l-1)}$$

    - Note: N_v here is defined as the set of **multi-hop neighbors** (instead of immediate neighbors). It can be the set of all nodes for larger k

Limitation of Attention Diffusion

- But computing the attention diffusion is **costly**
  - $\mathcal{A}^l$ will be **denser** with the growing of l
  - Using $\mathcal{A}$ will lead to computational complexity and memory requirement of $\boldsymbol{O}(\boldsymbol{n}^2)$
  - How to compute it efficiently?

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Approximate Computation for Attention Diffusion (1)

- Let's first rewrite the aggregation step in matrix form:

$$\mathbf{H}^{(l)} = \mathcal{A}\mathbf{H}^{(l-1)} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k A^k \mathbf{H}^{(l-1)}$$

  - Expand the formula:

$$\mathbf{H}^{(l)} = \underbrace{\alpha\mathbf{H}^{(l-1)}}_{k=0} + \underbrace{\alpha(1-\alpha)A\mathbf{H}^{(l-1)}}_{k=1} + \underbrace{\alpha(1-\alpha)^2 A^2\mathbf{H}^{(l-1)}}_{k=2} + \cdots$$

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Approximate Computation for Attention Diffusion (2)

- When $k = 1$:

$$\mathbf{Z}^{(1)} = \overbrace{\alpha \mathbf{H}^{(l-1)}}^{k=0} + \overbrace{\alpha(1-\alpha)A^{(l-1)}\mathbf{H}^{(l-1)}}^{k=1}$$

- When $k = 2$:

$$\begin{aligned}\mathbf{Z}^{(2)} &= \overbrace{\alpha \mathbf{H}^{(l-1)}}^{k=0} + \overbrace{\alpha(1-\alpha)A\mathbf{H}^{(l-1)}}^{k=1} + \overbrace{\alpha(1-\alpha)^2 A^2 \mathbf{H}^{(l-1)}}^{k=2} \\ &= \alpha \mathbf{H}^{(l-1)} + (1-\alpha)A\left(\alpha \mathbf{H}^{(l-1)} + \alpha(1-\alpha)A\mathbf{H}^{(l-1)}\right) \\ &= \alpha \mathbf{H}^{(l-1)} + (1-\alpha)A\mathbf{Z}^{(1)}\end{aligned}$$

- We find a pattern!

For simplicity, we rewrite $A^{(l-1)}$ as A here

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Approximate Computation for Attention Diffusion (3)

- When $k = 3$:

$$\begin{aligned}\mathbf{Z}^{(3)} &= \underbrace{\alpha\mathbf{H}^{(l-1)}}_{k=0} + \underbrace{\alpha(1-\alpha)A\mathbf{H}^{(l-1)}}_{k=1} + \underbrace{\alpha(1-\alpha)^2A^2\mathbf{H}^{(l-1)}}_{k=2} + \underbrace{\alpha(1-\alpha)^3A^3\mathbf{H}^{(l-1)}}_{k=3} \\ &= \alpha\mathbf{H}^{(l-1)} + (1-\alpha)A\left(\alpha\mathbf{H}^{(l-1)} + \alpha(1-\alpha)A\mathbf{H}^{(l-1)} + \alpha(1-\alpha)^2A^2\mathbf{H}^{(l-1)}\right) \\ &= \alpha\mathbf{H}^{(l-1)} + (1-\alpha)A\mathbf{Z}^{(2)}\end{aligned}$$

- So we can conclude:

$$\mathbf{Z}^{(k)} = \alpha\mathbf{Z}^{(0)} + (1-\alpha)A\mathbf{Z}^{(k-1)}, \mathbf{Z}^{(0)} = \mathbf{H}^{(l-1)}$$
$$\mathbf{H}^{(l)} = \mathbf{Z}^{(\infty)}$$

  - An **approximated** iterative computation to the original attention diffusion!

For simplicity, we rewrite $A^{(l-1)}$ as A here

Approximate Computation for Attention Diffusion (4)

- An approximated multi-hop attention GNN:
  - **One-hop** attention computation:

$$A_{vu}^{(l-1)} = a(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}, \mathbf{e}_{r_{vu}})$$

  - **Aggregate** message: iteratively perform the following computation

$$\mathbf{Z}^{(0)} = \mathbf{H}^{(l-1)}$$
$$\mathbf{Z}^{(i+1)} = \alpha \mathbf{Z}^{(0)} + (1-\alpha) A^{(l-1)} \mathbf{Z}^{(i)}, i = 0, \dots, I-1$$
$$\mathbf{H}^{(l)} = \mathbf{Z}^{(I)}$$

Multi-hop Attention Graph Neural Network. *IJCAI 2020*

Summary of the Lecture

- **Recap: Graph attention mechanism**
  - **Graph attention network on homogeneous graph**:
    - Compute the importance score of neighbor by learnable weights
    - Multi-head attention
  - **Heterogeneous graph**:
    - Use cases
    - Relational GCN
  - **Multi-hop attention network:**
    - Single-hop graph attention network on heterogeneous graph
    - Multi-hop graph attention network on heterogeneous graph through **diffusion**