Beyond Simple Graphs: Heterogeneous Graphs

Jiaxuan You

Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall

https://ulab-uiuc.github.io/CS598/

Logistics: Homework & Proposal Feedback

- Coding Assignment 2 Due
  - Please submit your code and written answers to Canvas.
  - The submission deadline is **Oct 13 (Sun) 11:59 PM, CT**.
- Proposal Feedback
  - Peer discussion on Slack will count towards 2% of your final grade. Please provide feedback to at least 2 groups. No restrictions on formats.
  - Prioritize providing feedback to proposals that have received relatively fewer comments. Ideally, we hope that each group's proposal receives at least 5 pieces of feedback.
  - Please provide your feedback by **Oct 13 (Sun) 11:59 PM, CT**.

Logistics: Submission Task

- Project submission Task Out
  - The submission task counts towards **15% (writing) + 15% (implementation) = 30%** of your final grade.
  - We are going to use the Open Review system to receive submissions. Detailed instructions will be updated on Canvas later. Expect the format as an ICLR conference submission (~9 pages).
  - The submission deadline is **Nov 17 (Sun) 11:59 PM, CT** (~5 weeks from now). Please plan the progress of your project reasonably.
    - We will begin peer-review right after the submission
    - The submission is not the final version of your project submission – **The final version** will be due on **Dec 8 (Sun) 11:59 PM, CT**

Recap: GNN Implementation

- GraphGym: Easy-to-use code platform for GNN
  - General guidelines for GNN design
  - Understandings of GNN tasks
  - Transferring best GNN designs across tasks
- PyG principles & highlights

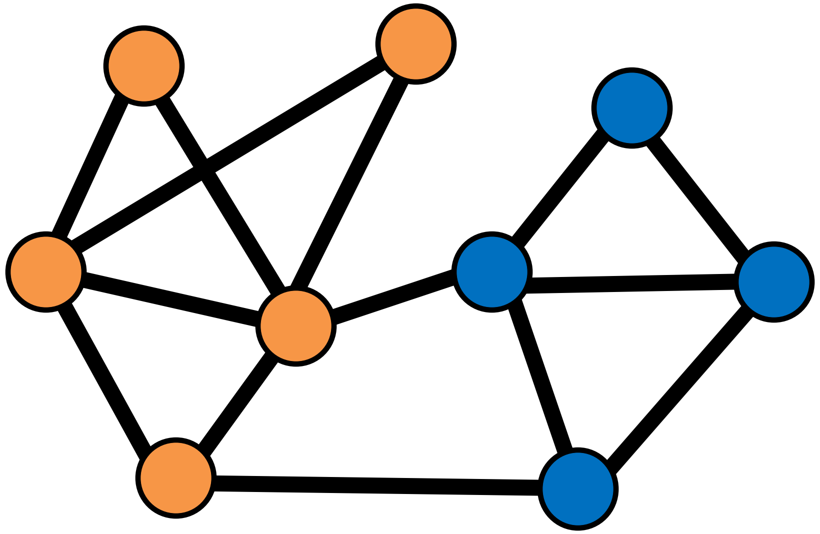
Today: Heterogeneous Graphs

- So far, we only handle graphs with one edge type
- How to handle graphs with multiple nodes or edge types (a.k.a **heterogeneous graphs**)?
- **Goal:** Learning with **heterogeneous graphs**
  - **Relational GCNs**
  - Heterogeneous Graph Transformer
  - **Design space for heterogeneous GNNs**

Beyond Simple Graphs: Heterogeneous Graphs

Heterogeneous Graphs

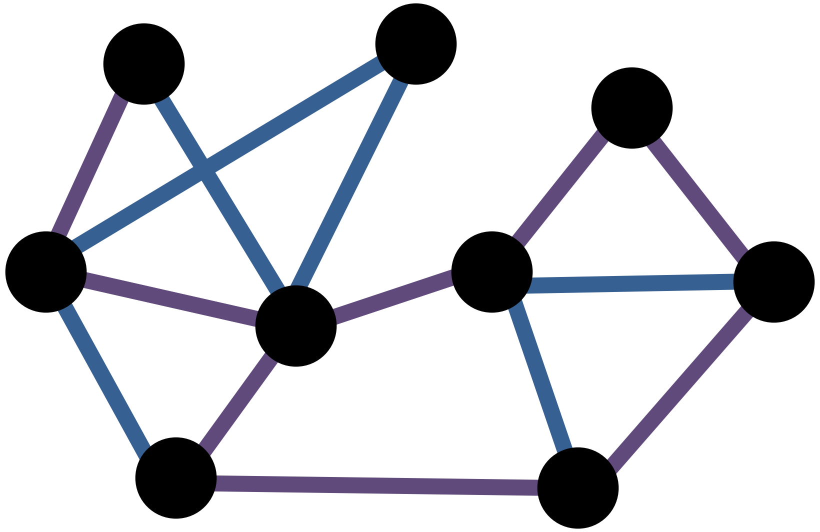
Heterogeneous Graphs: Motivation



2 types of nodes:

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

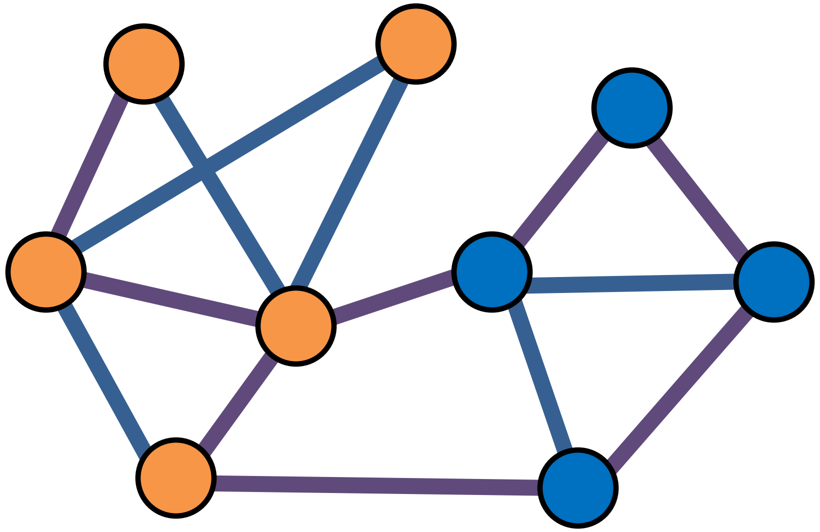
Heterogeneous Graphs: Motivation



2 types of edges:

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

Heterogeneous Graphs: Motivation



- A graph could have multiple types of nodes and edges! **2 types of nodes + 2 types of edges**.

Heterogeneous Graphs: Motivation

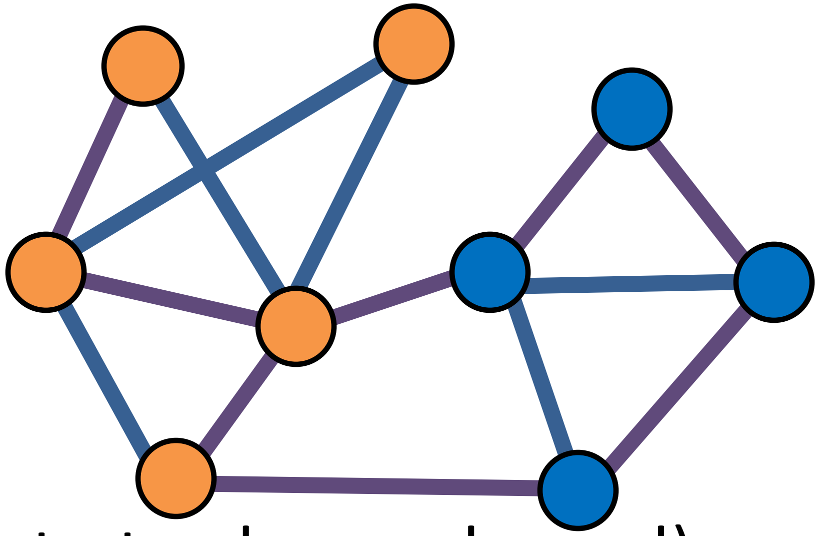
8 possible relation types!

(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

Heterogeneous Graphs

- **A heterogeneous graph is defined as**

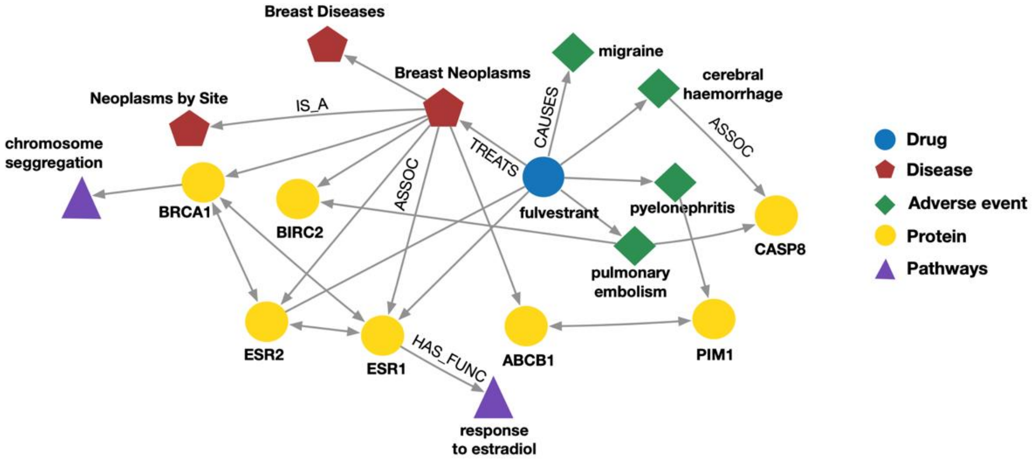
$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \tau, \phi)$$

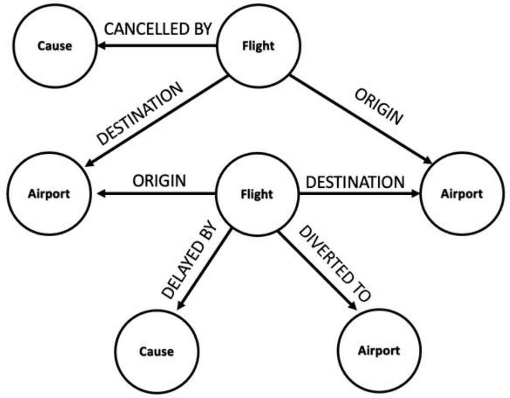
  - Nodes **with node types** $v \in V$
    - **Node type** for node v: $\tau(v)$
  - Edges **with edge types** $(u, v) \in E$
    - **Edge type** for edge (u, v): $\phi(u, v)$
    - **Relation type** for edge e is a tuple: $r(u, v) = (\tau(u), \phi(u, v), \tau(v))$

An edge can be described as a pair of nodes

- There are other definitions for heterogeneous graphs as well – describe **graphs with node & edge types**

Many Graphs are Heterogeneous Graphs (1)





Biomedical Knowledge Graphs

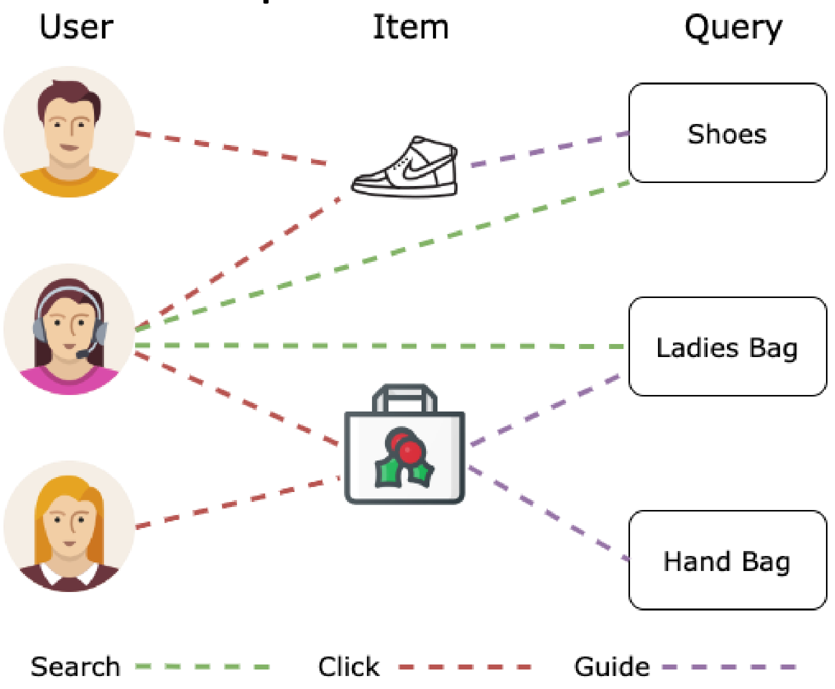
- Example node: Migraine
- Example relation: (fulvestrant, Treats, Breast Neoplasms)
- Example node type: Protein
- Example edge type: Causes

Event Graphs

- Example node: SFO
- Example relation: (UA689, Origin, LAX)
- Example node type: Flight
- Example edge type: Destination

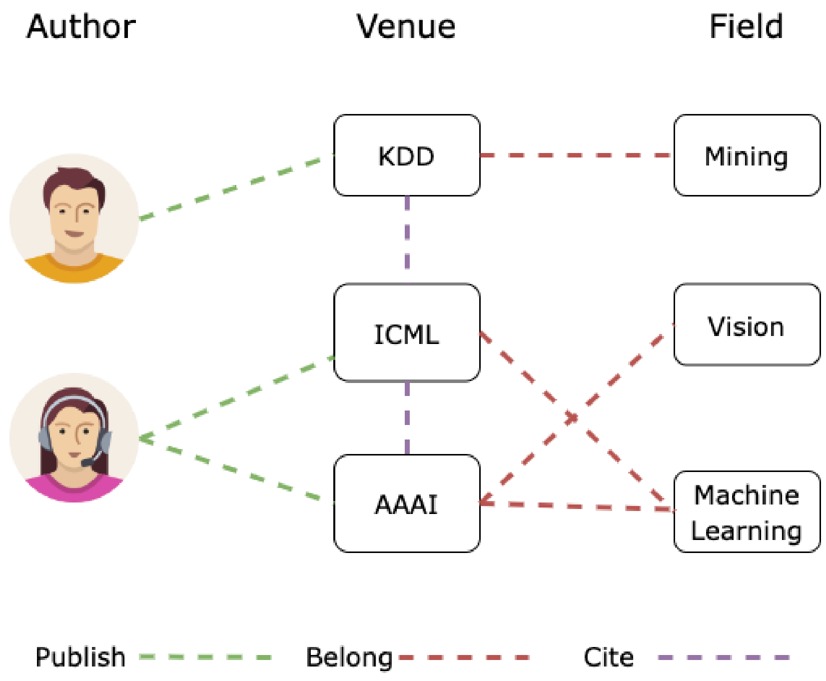
Many Graphs are Heterogeneous Graphs (2)

- 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!



Many Graphs are Heterogeneous Graphs (3)

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



Discussions: Type or Feature?

- **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?**

Discussions: Type or Feature?

- **When do we need a heterogeneous graph?**
  - **Case 1:** Different node/edge types **have different shapes/semantic of features**
    - An "author node" has 4-dim feature, a "paper node" has 5-dim feature
    - Both "author node" and "paper node" have 4-dim feature, but *with different semantics*
      - Author node features are: Institution, Gender, Age, Citation count
      - Paper node features are: Venue, Year, Length, Citation count
  - **Case 2**: We know different relation types represent **different types of interactions**
    - (English, translate, French) and (English, translate, Chinese) require different model weights/transformations

Discussions: 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)

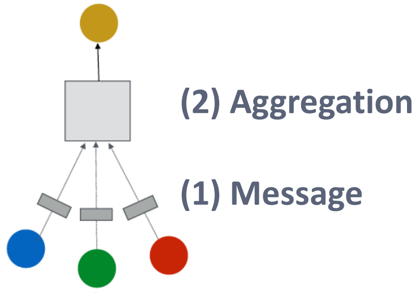
Recap: Classical GNN Layers: GCN

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

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

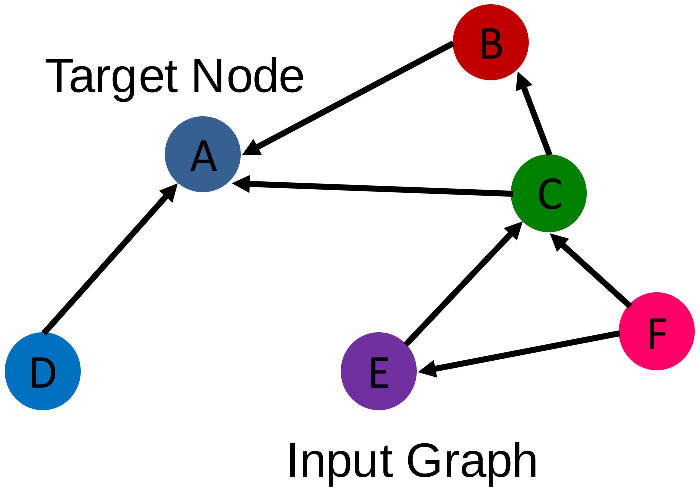
How to write this as Message + Aggregation?

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



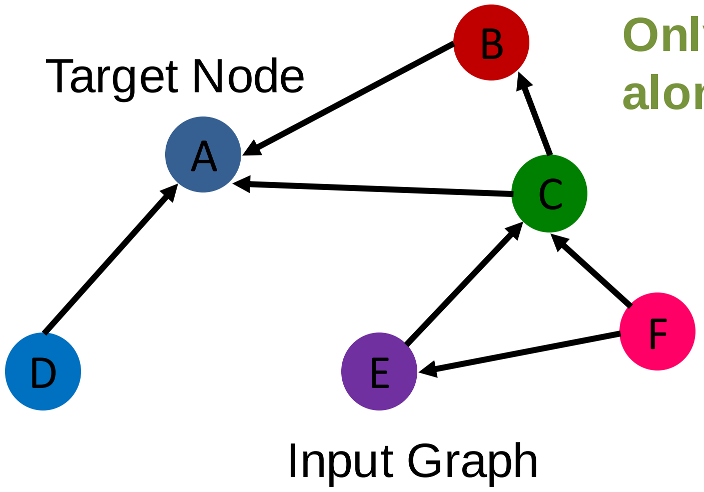
Relational GCN

- **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?

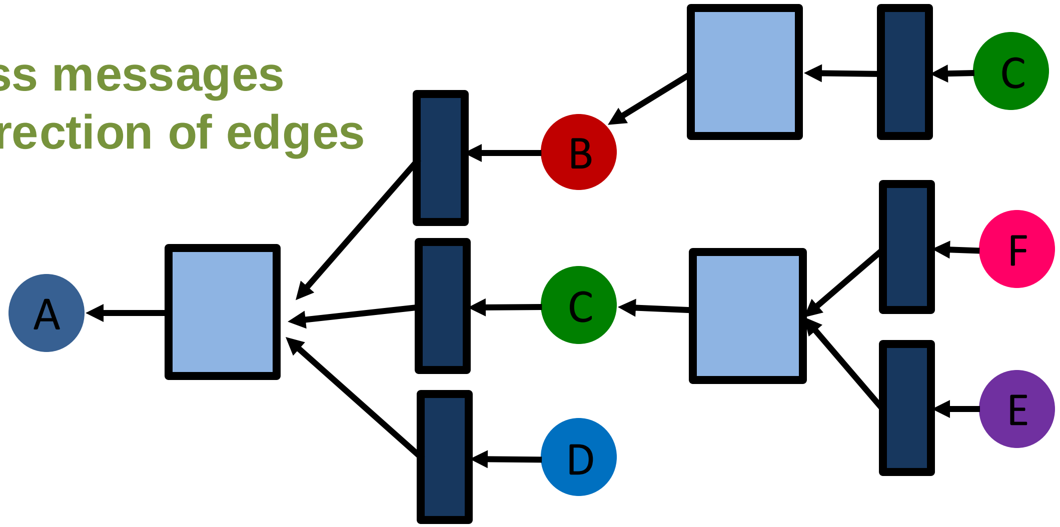


Relational GCN

- **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?

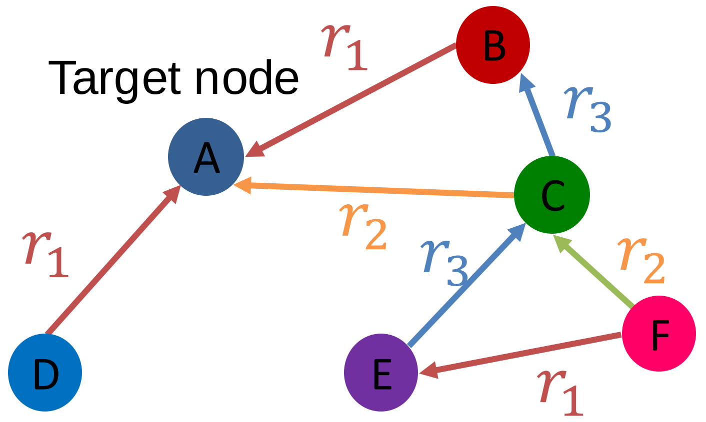


Only pass messages along direction of edges



Relational GCN (1)

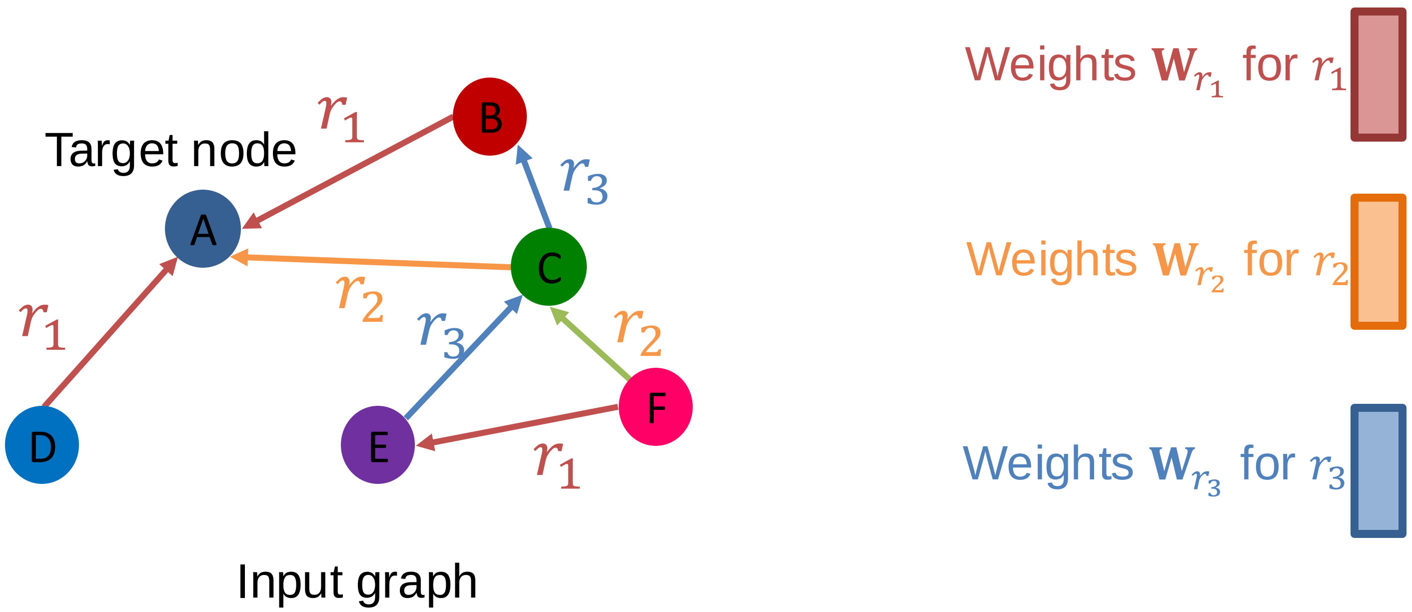
- **What if the graph has multiple relation types?**



Input graph

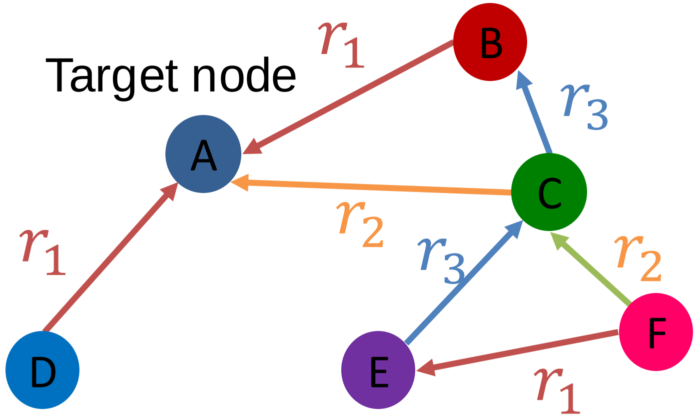
Relational GCN (2)

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

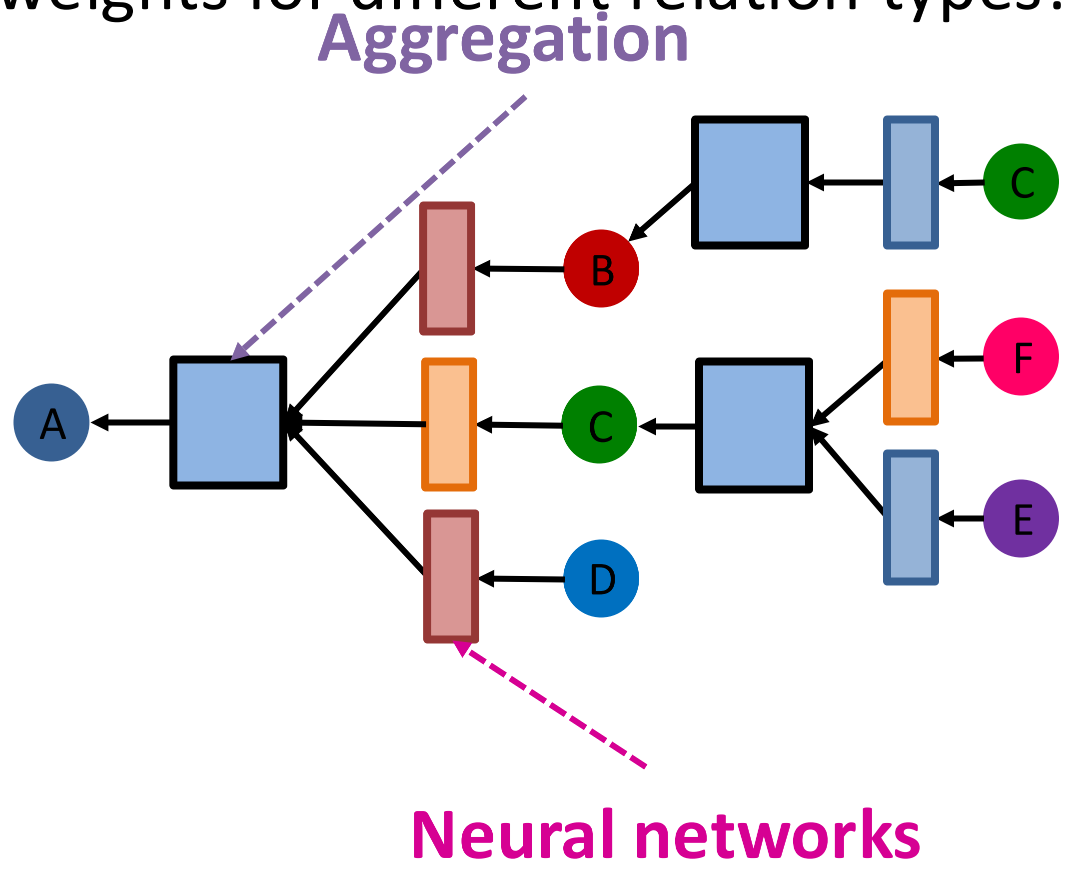


Relational GCN (3)

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



Input graph



Relational GCN (4)

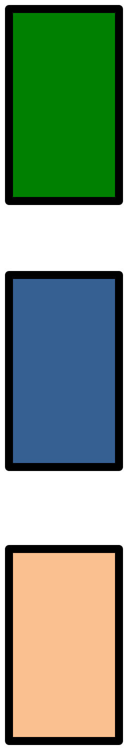
- Introduce a set of neural networks for each relation type!

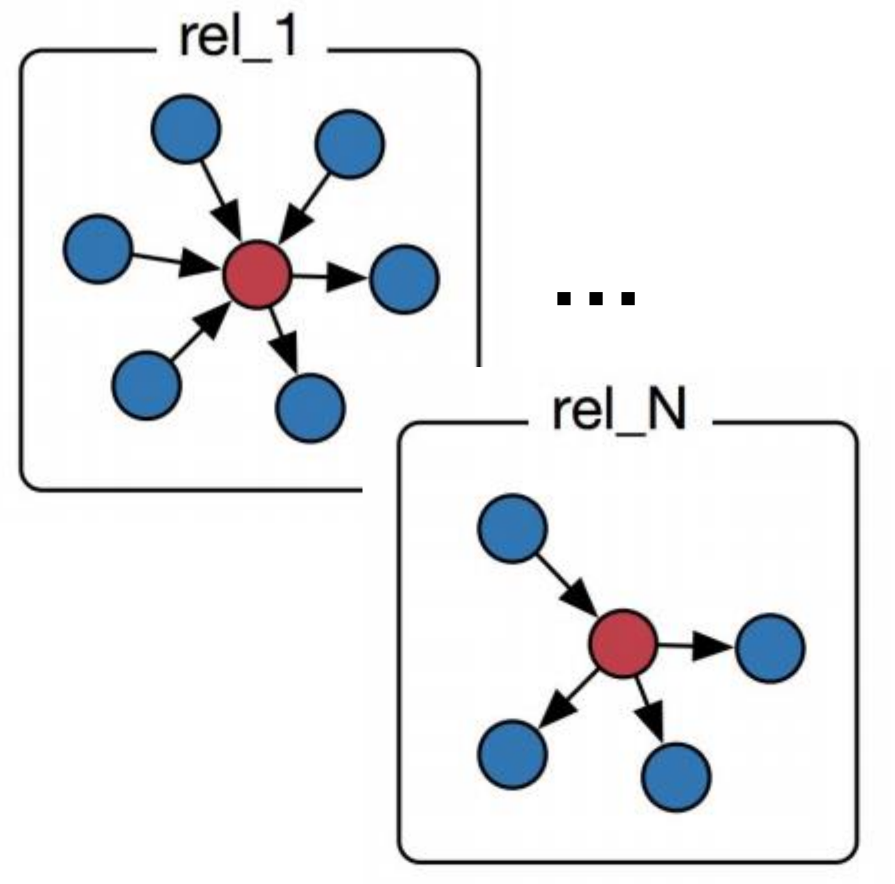
Weight for rel_1

...

Weight for rel_N

Weight for self-loop





Relational GCN: Definition

- **Relational GCN (RGCN):**

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\sum_{r \in R} \sum_{u \in N_v^r} \frac{1}{c_{v,r}} \mathbf{W}_r^{(l)} \mathbf{h}_u^{(l)} + \mathbf{W}_{\tau(v)}^{(l)} \mathbf{h}_v^{(l)}\right)$$

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

$$\mathbf{m}_{u,r}^{(l)} = \frac{1}{c_{v,r}} \mathbf{W}_r^{(l)} \mathbf{h}_u^{(l)}$$

  - Self-loop, based on the node type $\tau(v)$ for the node of interest:

$$\mathbf{m}_v^{(l)} = \mathbf{W}_{\tau(v)}^{(l)} \mathbf{h}_v^{(l)}$$

- **Aggregation:**
  - Sum over messages from neighbors and self-loop, then apply activation
  - $\mathbf{h}_v^{(l+1)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_{u,r}^{(l)}, u \in N(v)\right\} \cup \left\{\mathbf{m}_v^{(l)}\right\}\right)\right)$

Normalized by node degree of the relation

$c_{v,r} = |N_v^r|$

RGCN: Scalability

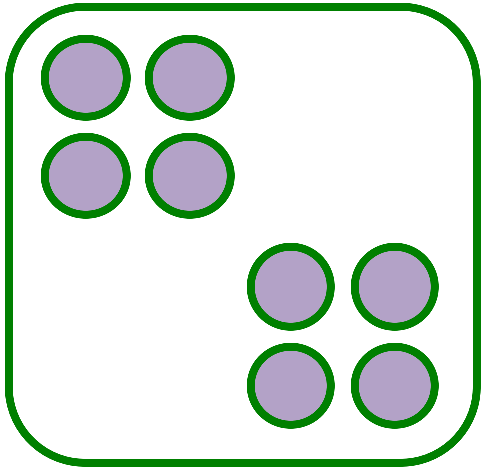
- Each relation has L matrices: $\mathbf{W}_r^{(1)}, \mathbf{W}_r^{(2)} \cdots \mathbf{W}_r^{(L)}$
- The size of each $\mathbf{W}_r^{(l)}$ is $d^{(l+1)} \times d^{(l)}$

$d^{(l)}$ is the hidden dimension in layer l

- **Rapid growth of the number of parameters w.r.t number of relations!**
  - **Overfitting becomes an issue**
- **Two methods to regularize the weights $\mathbf{W}_r^{(l)}$**
  - **(1)** Use block diagonal matrices
  - **(2)** Basis/Dictionary learning

(1) Block Diagonal Matrices

- **Key insight**: make the weights **sparse**!
- Use **block diagonal matrices** for $\mathbf{W}_r$

$\mathbf{W}_r =$ 

Limitation: only nearby neurons/dimensions can interact through W

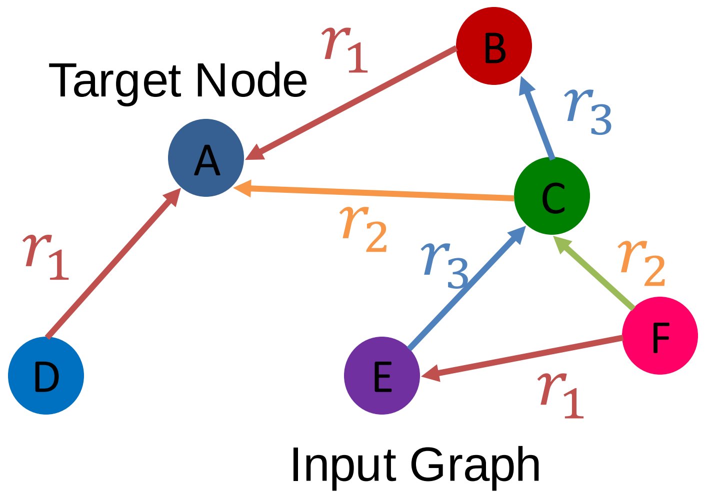
- If use B low-dimensional matrices, then # param reduces from $d^{(l+1)} \times d^{(l)}$ to $B \times \frac{d^{(l+1)}}{B} \times \frac{d^{(l)}}{B}$

(2) Basis Learning

- **Key insight**: **Share weights** across different relations!
- Represent the matrix of each relation as a linear combination of basis transformations
  $\mathbf{W}_r = \sum_{b=1}^{B} a_{rb} \cdot \mathbf{V}_b$, where $\mathbf{V}_b$ is shared across all relations
  - $\mathbf{V}_b$ are the basis matrices
  - a_{rb} is the importance weight of matrix $\mathbf{V}_b$
- Now each relation only needs to learn $\{a_{rb}\}_{b=1}^{B}$, which is B scalars

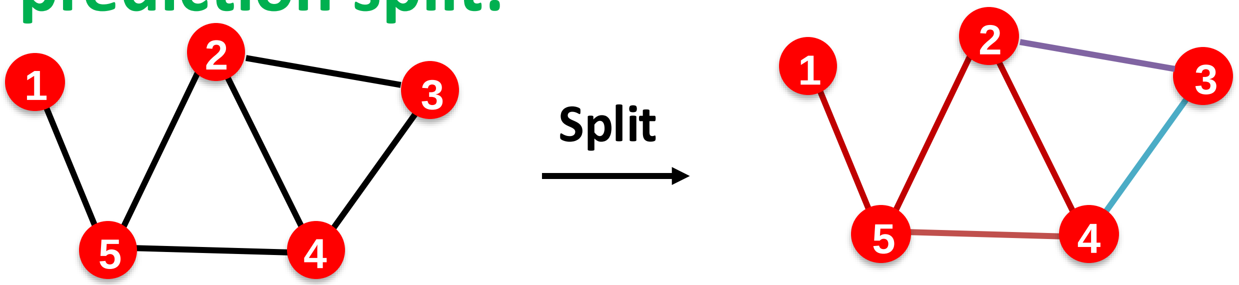
Example: Entity/Node Classification

- **Goal**: **Predict the label of a given node**
- **RGCN** uses the representation of the final layer:
  - If we predict the class of **node A** from **k classes**
  - Take the **final layer (prediction head)**: $\mathbf{h}_A^{(L)} \in \mathbb{R}^k$, each item in $\mathbf{h}_A^{(L)}$ represents **the probability of that class**



Example: Link Prediction

- **Link prediction split:**



The original graph

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

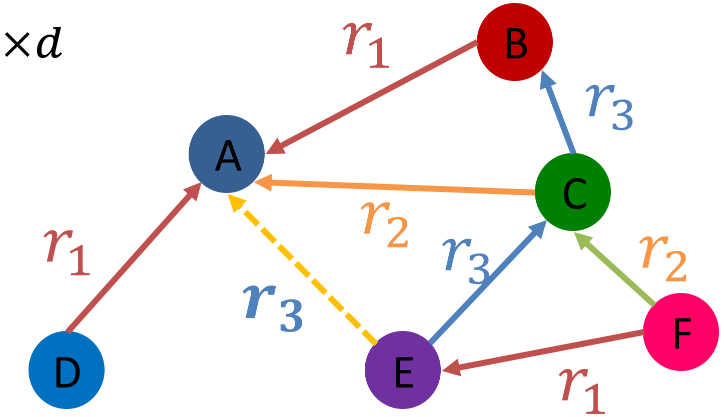
⋮

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 (1)

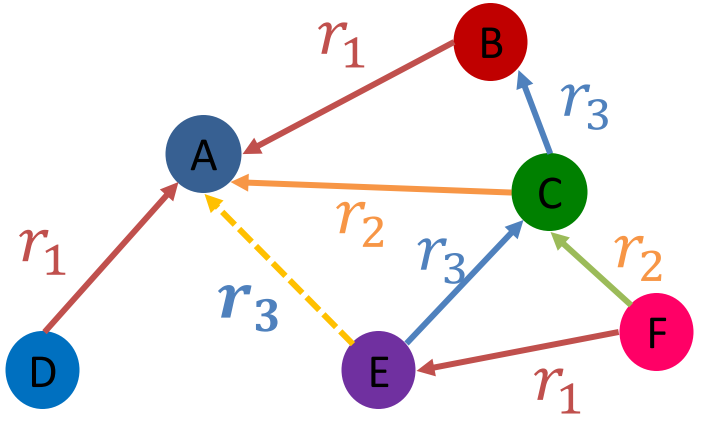
- **Assume (E, r_3, A) is training supervision edge, all the other edges are training message edges**
- **Use RGCN to score (E, r_3, A)!**
  - Take the final layer of E and A: $\mathbf{h}_E^{(L)}$ and $\mathbf{h}_A^{(L)} \in \mathbb{R}^d$
  - Relation-specific score function $f_r: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
    - One example $f_{r_1}(\mathbf{h}_E, \mathbf{h}_A) = \mathbf{h}_E^T \mathbf{W}_{r_1} \mathbf{h}_A, \mathbf{W}_{r_1} \in \mathbb{R}^{d \times d}$



Input Graph

RGCN for Link Prediction (2)

- **Training:**



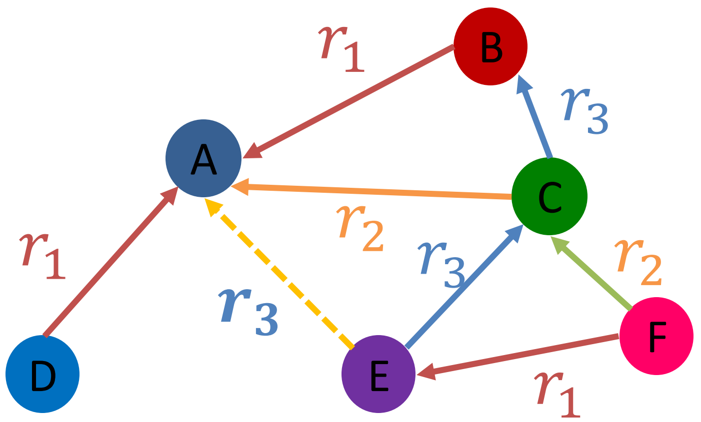
training supervision edges: (E, r_3, A)
training message edges: **all the rest existing edges (solid lines)**
(1) Use **training message edges** to predict **training supervision edges**

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)

- **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 (3)

- **Training:**



Input Graph

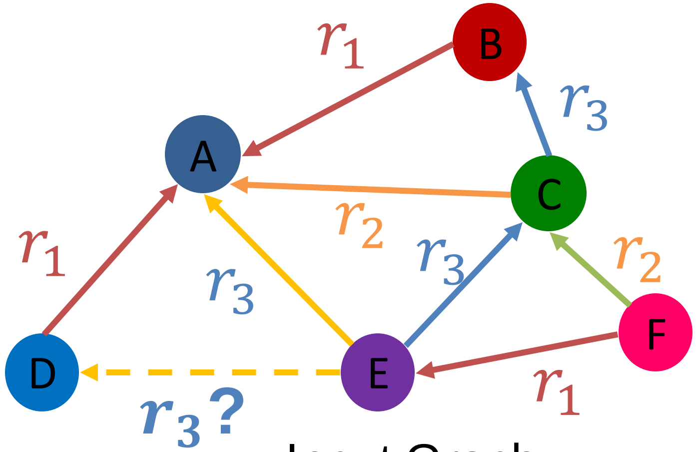
$$\ell = -\log \sigma\left(f_{r_3}(h_E, h_A)\right) - \log(1 - \sigma\left(f_{r_3}(h_E, h_B)\right))$$

σ … Sigmoid function

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 (as discussed in Lecture 6)**
   1. **Maximize the score of training supervision edge**
   2. **Minimize the score of negative edge**

RGCN for Link Prediction (4)

- **Evaluation:**
  - **Validation time as an example, same at the test time**



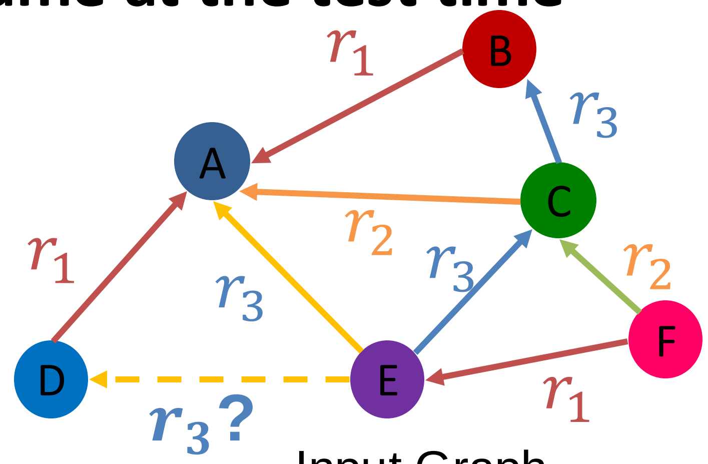
Input Graph

- **Evaluate how the model can predict the validation edges with the relation types.**
- **Let's 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)**

validation edges: (E, r_3, D)
training message edges & training supervision edges: **all existing edges (solid lines)**
(2) At validation time:
Use **training message edges & training supervision edges** to predict **validation edges**

RGCN for Link Prediction (4)

- **Evaluation:**
  - **Validation time as an example, same at the test time**



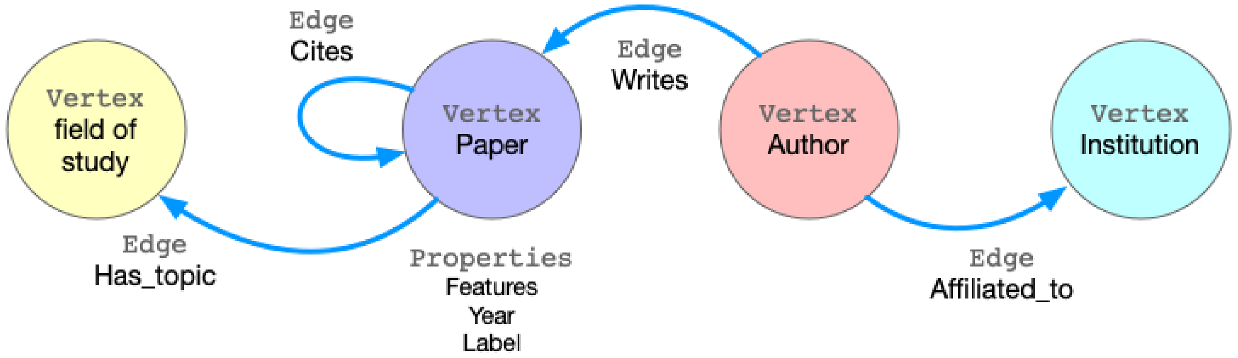
Input Graph

- **Evaluate how the model can predict the validation edges with the relation types.**
- **Let's 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: $1\,[RK \leq k]$. Higher is better; (2) Reciprocal Rank: $\frac{1}{RK}$. Higher is better**

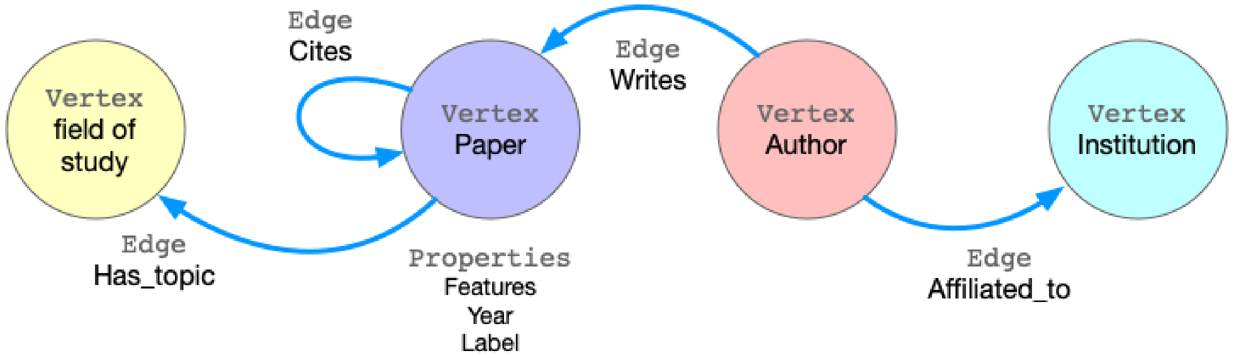
Benchmark for Heterogeneous Graphs (1)

- Benchmark dataset
  - **ogbn-mag** from Microsoft Academic Graph (MAG)
- Four (4) **types of entities**
  - Papers: 736k nodes
  - Authors: 1.1m nodes
  - Institutions: 9k nodes
  - Fields of study: 60k nodes



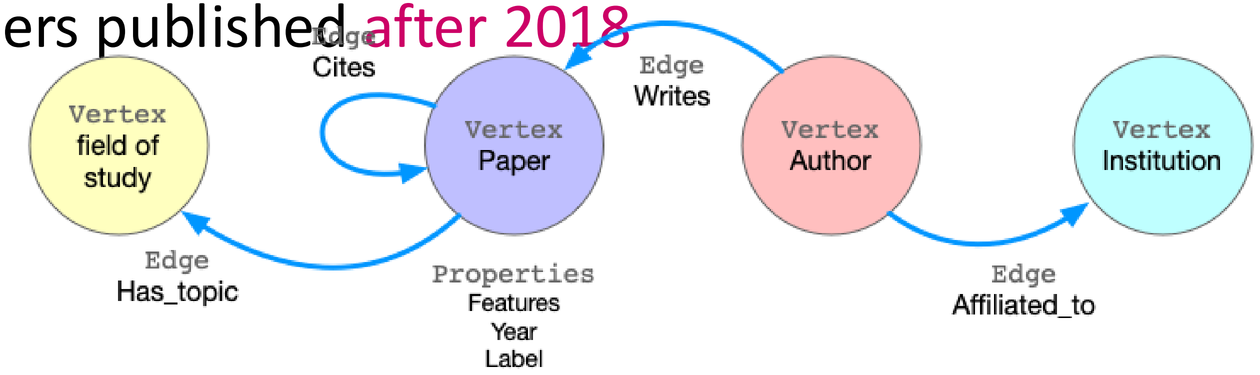
Benchmark for Heterogeneous Graphs (2)

- Benchmark dataset
  - **ogbn-mag** from Microsoft Academic Graph (MAG)
- Four (4) **directed relations**
  - An author is "affiliated with" an institution
  - An author "writes" a paper
  - A paper "cites" a paper
  - A paper "has a topic of" a field of study



Benchmark for Heterogeneous Graphs (3)

- **Prediction task**
  - Each paper has a 128-dimensional **word2vec** feature vector
  - Given the content, references, authors, and author affiliations from ogbn-mag, predict the **venue of each paper**
  - **349-class** classification problem due to 349 venues considered
- **Time-based dataset splitting**
  - **Training set**: papers published before 2018
  - **Test set**: papers published after 2018

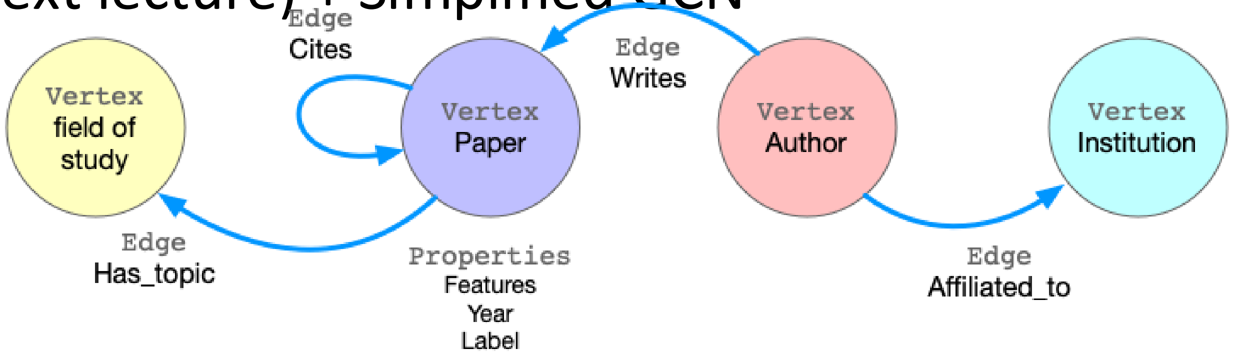


Benchmark for Heterogeneous Graphs (4)

- **Benchmark results:**

	Rank	Method	Ext. data	Test Accuracy	Validation Accuracy	Contact	References	#Params	Hardware	Date
SOTA	1	**SeHGNN (ComplEx embs)**	No	0.5719 ± 0.0012	0.5917 ± 0.0009	Xiaocheng Yang (ICT-GIMLab)	Paper, Code	8,371,231	NVIDIA Tesla T4 (15 GB)	Jul 7, 2022
R-GCN	21	NeighborSampling (R-GCN aggr)	No	0.4678 ± 0.0067	0.4761 ± 0.0068	Matthias Fey – OGB team	Paper, Code	154,366,772	GeForce RTX 2080 (11GB GPU)	Jun 26, 2020

- **SOTA method: SeHGNN**
  - ComplEx (Next lecture) + Simplified GCN



Summary of RGCN

- **Relational GCN**, a graph neural network for **heterogeneous graphs**
- Can perform entity classification as well as link prediction tasks.
- Ideas can easily be extended into RGNN (RGraphSAGE, RGAT, etc.)
- **Benchmark**: ogbn-mag from Microsoft Academic Graph, to predict paper venues

Beyond Simple Graphs: Heterogeneous Graphs

Heterogeneous Graph Transformer

Recap: Graph Attention Networks

- **Graph Attention Networks (GAT)**

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

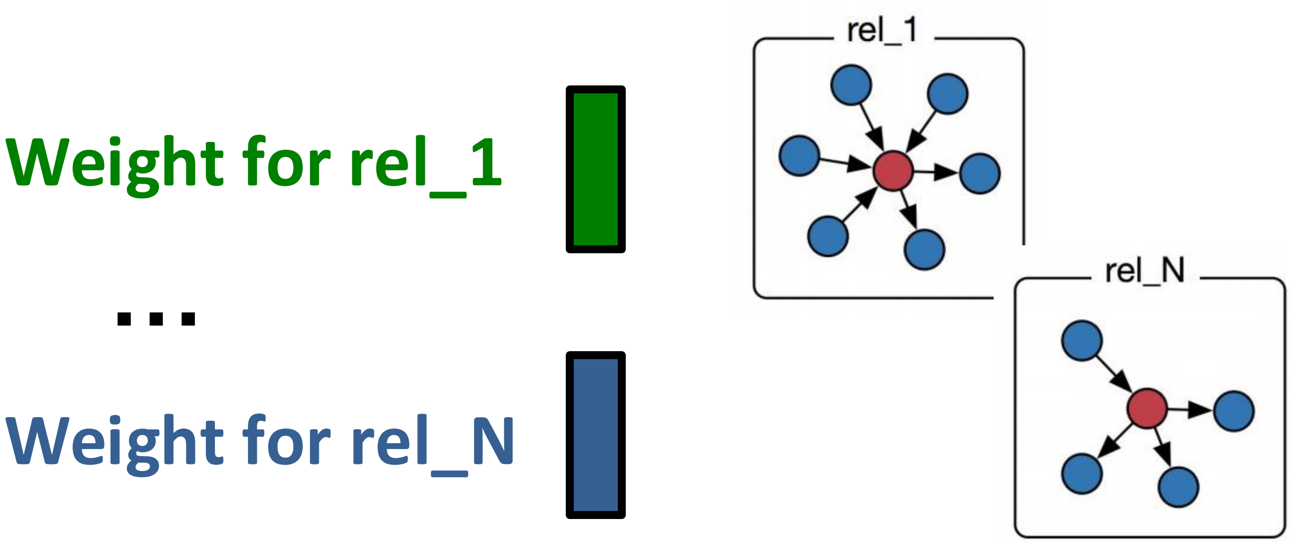
Attention weights

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.
- **Can we adapt GAT for heterogeneous graphs?**

Heterogeneous Graph Transformer

- **Motivation:** GAT is unable to represent different node & different edge types
- Introduce a set of neural networks for each relation type is **too expensive** for attention
  - **Recall:** relation describes (node_s, edge, node_e)



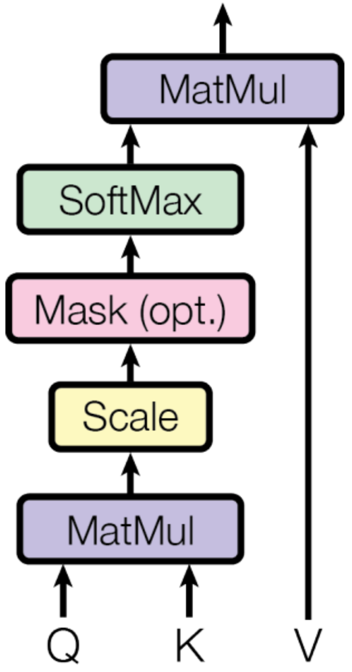
Too expensive!

Basics: Attention in Transformer

- HGT uses Scaled Dot-Product Attention (proposed in Transformer)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention



- Query: Q, Key: K, Value: V
  - Q, K, V have shape (batch_size, dim)

How do we obtain Q, K, V?

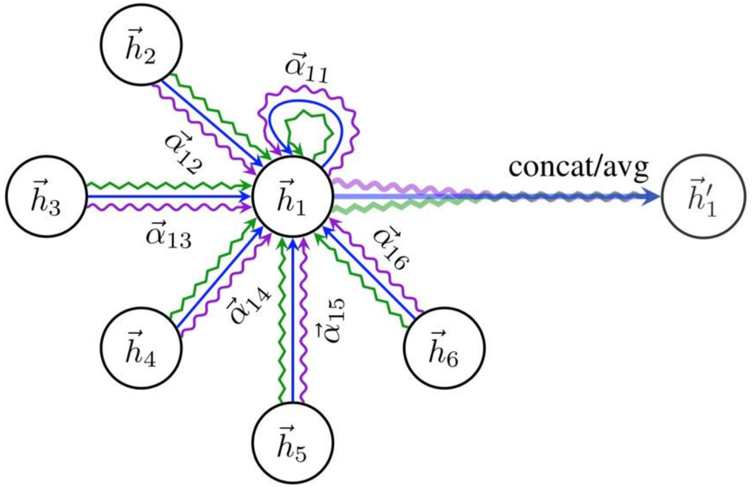
- Apply Linear layer to the input
  - $Q = Q\_Linear(X)$
  - $K = K\_Linear(X)$
  - $V = V\_Linear(X)$

Heterogeneous Mutual Attention (1)

- **Recall:** Applying GAT to a homogeneous graph
  - $H^{(l)}$ is the l-th layer representation:

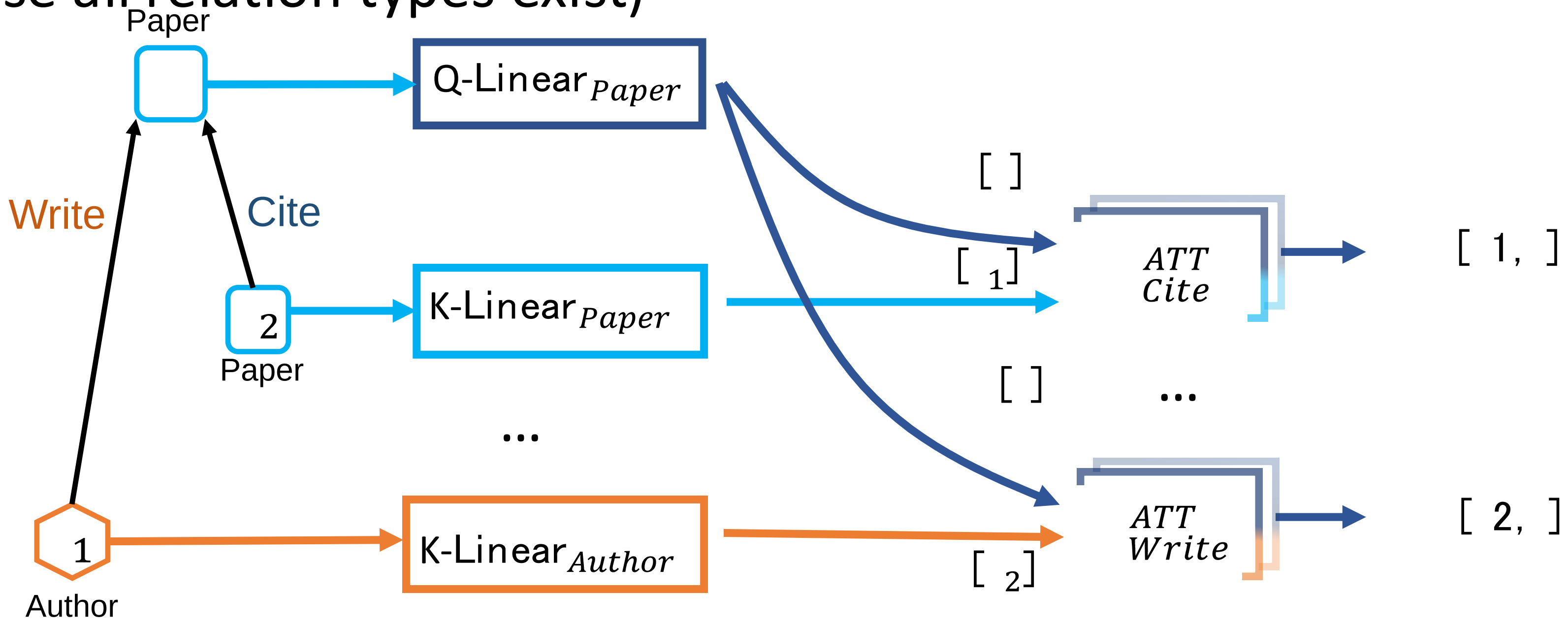
$$H^l[t] \leftarrow \underset{\forall s \in N(t), \forall e \in E(s,t)}{\text{Aggregate}} \left(\boxed{\text{Attention}(s,t)} \cdot \text{Message}(s) \right)$$

How do we take relation type (node_s, edge, node_e) into attention computation?



Heterogeneous Mutual Attention (2)

- **Innovation:** Decompose heterogeneous attention to Node- and edge-type dependent attention mechanism
  - **3 node weight matrices, 2 edge weight matrices**
  - **Without decomposition:** 3*2*3=18 relation types -> 18 weight matrices (suppose all relation types exist)



Heterogeneous Mutual Attention (3)

- **Heterogeneous Mutual Attention:**

$$ATT\text{-}head^i(s,e,t) = \left(K^i(s) W^{ATT}_{\phi(e)} Q^i(t)^T\right)$$
$$K^i(s) = \text{K-Linear}^i_{\tau(s)}\left(H^{(l-1)}[s]\right)$$
$$Q^i(s) = \text{Q-Linear}^i_{\tau(t)}\left(H^{(l-1)}[t]\right)$$

- Each relation $(T(s), R(e), T(t))$ has a distinct set of **projection weights**
  - $T(s)$: type of node s, $R(e)$: type of edge e
  - $T(s)$ & $T(t)$ parameterize $K\_Linear_{T(s)}$ & $Q\_Linear_{T(t)}$, which further return Key and Query vectors $K(s)$ & $Q(t)$
  - Edge type $R(e)$ directly parameterizes $W_{R(e)}$
  - **Without decomposition:** 3*2*3=18 relation types -> 18 weight matrices
  - **With decomposition:** 3+2+3=8 weight matrices

More Details on HGT

- **A full HGT layer**

$$\widetilde{H}^{(l)}[t] = \bigoplus_{\forall s \in N(t)} \left(\boxed{\mathbf{Attention}_{HGT}(s,e,t)} \cdot \mathbf{Message}_{HGT}(s,e,t) \right)$$

We have just computed

- Similarly, HGT **decomposes weights** with node & edge types in the **message computation**

$$\mathbf{Message}_{HGT}(s,e,t) = \big\Vert_{i \in [1,h]} \; MSG\text{-}head^{i}(s,e,t)$$

$$MSG\text{-}head^{i}(s,e,t) = \boxed{\text{M-Linear}^{i}_{\tau(s)}} \left(H^{(l-1)}[s] \right) \boxed{W^{MSG}_{\phi(e)}}$$

Weights for each node type

Weights for each edge type

HGT vs R-GCN: Performance

- **Benchmark**: ogbn-mag from Microsoft Academic Graph, to predict paper venues

Rank	Method	Ext. data	Test Accuracy	Validation Accuracy	Contact	References	#Params	Hardware	Date
18	**HGT (LADIES Sample)**	No	0.4927 ± 0.0061	0.4989 ± 0.0047	Ziniu Hu	Paper, Code	21,173,389	Tesla K80 (12GB GPU)	Jan 26, 2021
21	NeighborSampling (R-GCN aggr)	No	0.4678 ± 0.0067	0.4761 ± 0.0068	Matthias Fey – OGB team	Paper, Code	154,366,772	GeForce RTX 2080 (11GB GPU)	Jun 26, 2020

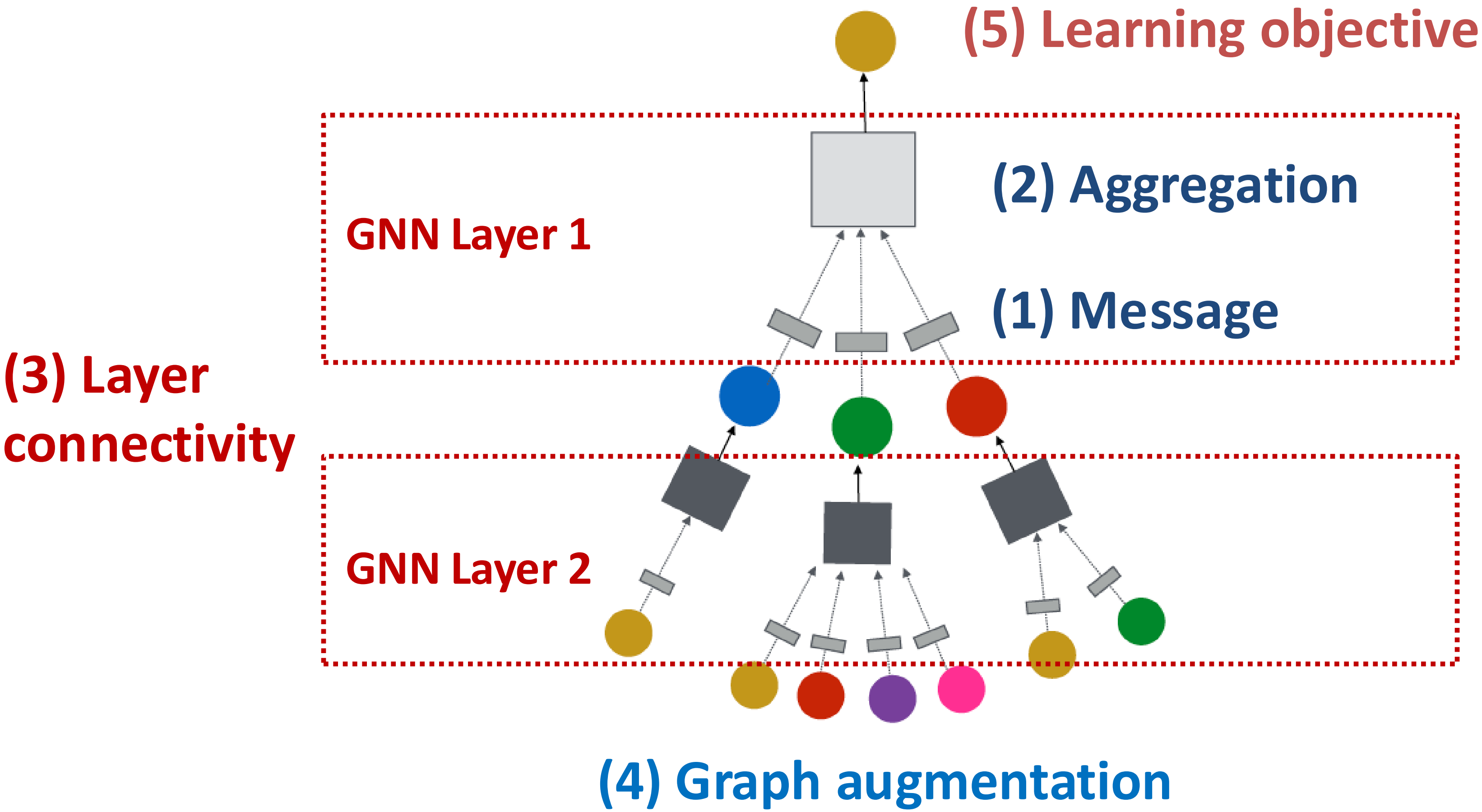
- HGT uses **much fewer parameters**, even though the attention computation is expensive, while **performs better than R-GCN**
  - Thanks to the weight decomposition over node & edge types

Beyond Simple Graphs: Heterogeneous Graphs

Design Space of Heterogeneous GNNs

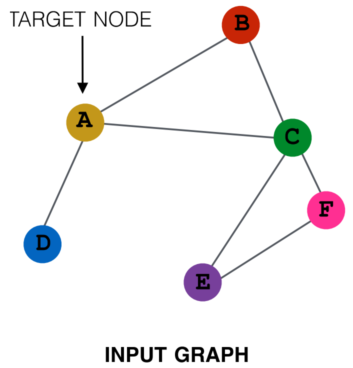
Recap: GNN Framework

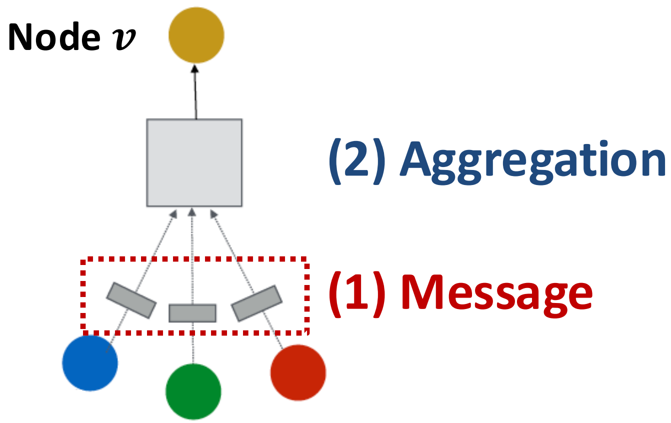
- **How do we extend the general GNN design space to heterogeneous graphs?**



Recap: Message Computation

- **(1) Message computation**
  - **Message function**: $\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
    - **Intuition:** Each node will create a message, which will be sent to other nodes later
    - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$



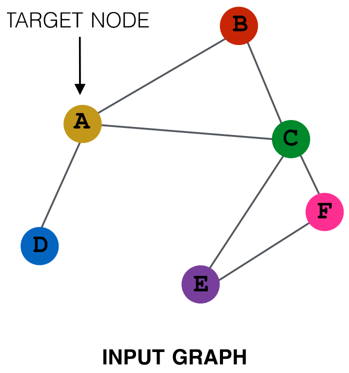


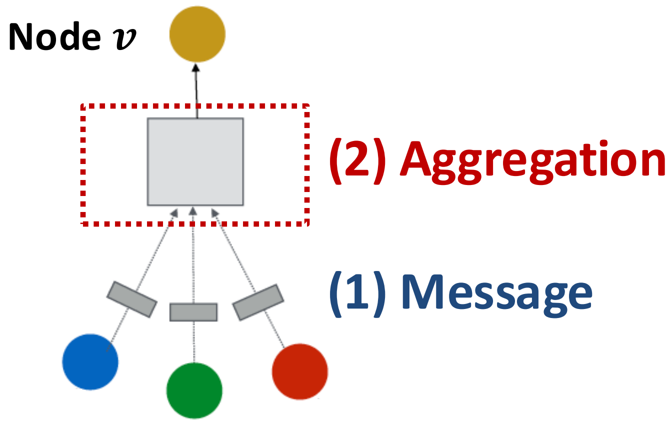
Heterogeneous Message

- **(1) Heterogeneous message computation**
  - **Message function**: $\mathbf{m}_u^{(l)} = \mathrm{MSG}_r^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
    - **Observation:** A node could receive multiple types of messages. **Num of message type = Num of relation type**
    - **Idea:** Create a different message function for each relation type
      - $\mathbf{m}_u^{(l)} = \mathrm{MSG}_r^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), r = (u, e, v)$ is the relation type between node u that sends the message, edge type e , and node v that receive the message
    - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}_r^{(l)}\mathbf{h}_u^{(l-1)}$

Recap: Message Aggregation

- **(2) Aggregation**
  - **Intuition:** Each node will aggregate the messages from node v's neighbors
$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)$$
  - **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator
$$\mathbf{h}_v^{(l)} = \mathrm{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)$$



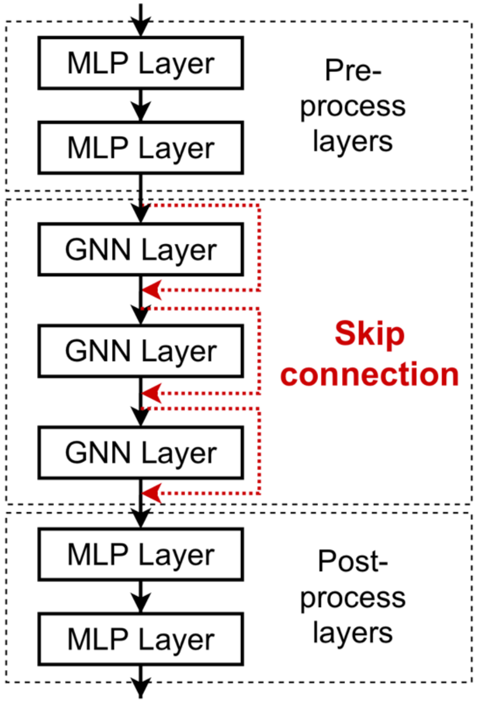


Heterogeneous Aggregation

- **(2) Heterogeneous Aggregation**
  - **Observation:** Each node could receive multiple types of messages from its neighbors, and multiple neighbors may belong to each message type.
  - **Idea:** We can define a 2-stage message passing
    - $\mathbf{h}_v^{(l)} = \mathrm{AGG}_{all}^{(l)}\left(\mathrm{AGG}_r^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N_r(v)\right\}\right)\right)$
    - Given all the messages sent to a node
    - Within each message type, aggregate the messages that belongs to the edge type with $\mathrm{AGG}_r^{(l)}$
    - Aggregate across the edge types with $\mathrm{AGG}_{all}^{(l)}$
  - **Example:** $\mathbf{h}_v^{(l)} = \mathrm{Concat}\left(\mathrm{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N_r(v)\right\}\right)\right)$

Recap: Layer connectivity

- **(3) Layer connectivity**
  - Add skip connections, pre/post-process layers



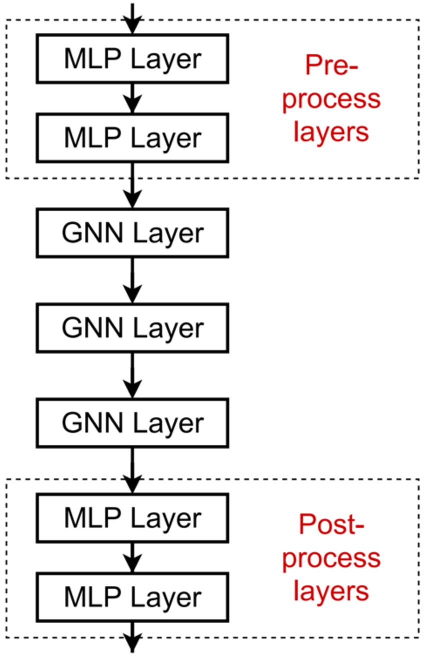
Pre-processing layers: Important when encoding node features is necessary.
E.g., when nodes represent images/text

Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

In practice, adding these layers works great!

Heterogeneous GNN Layers

- **Heterogeneous** pre/post-process layers:
  - MLP layers **with respect to each node type**
    - Since the output of GNN are **node embeddings**
  - $\mathbf{h}_v^{(l)} = \mathrm{MLP}_{T(v)}(\mathbf{h}_v^{(l)})$
    - $T(v)$ is the type of node v
- Other successful GNN designs are also encouraged for heterogeneous GNNs: skip connections, batch/layer normalization, …



Recap: Graph Manipulation

- **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

Heterogeneous Graph Manipulation

- **Graph Feature manipulation**
  - **2 Common options:** compute graph statistics (e.g., node degree) within each relation type, or across the full graph (ignoring the relation types)
- **Graph Structure manipulation**
  - Neighbor and subgraph sampling are also common for heterogeneous graphs.
  - **2 Common options:** sampling within each relation type (ensure neighbors from each type are covered), or sample across the full graph

Recap: GNN Prediction Heads

Node-level prediction:

- $\widehat{\boldsymbol{y}}_v = \mathrm{Head}_{\mathrm{node}}(\mathbf{h}_v^{(L)}) = \mathbf{W}^{(H)}\mathbf{h}_v^{(L)}$

Edge-level prediction:

- $\widehat{\boldsymbol{y}}_{\boldsymbol{uv}} = \mathrm{Head}_{\mathrm{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}) = \mathrm{Linear}(\mathrm{Concat}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}))$

Graph-level prediction:

- $\widehat{\boldsymbol{y}}_G = \mathrm{Head}_{\mathrm{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$

Heterogeneous Prediction Heads

Node-level prediction:

- $\widehat{\boldsymbol{y}}_{\boldsymbol{v}} = \mathrm{Head}_{\mathrm{node},\, T(v)}(\mathbf{h}_v^{(L)}) = \mathbf{W}_{T(v)}^{(H)} \mathbf{h}_v^{(L)}$

Edge-level prediction:

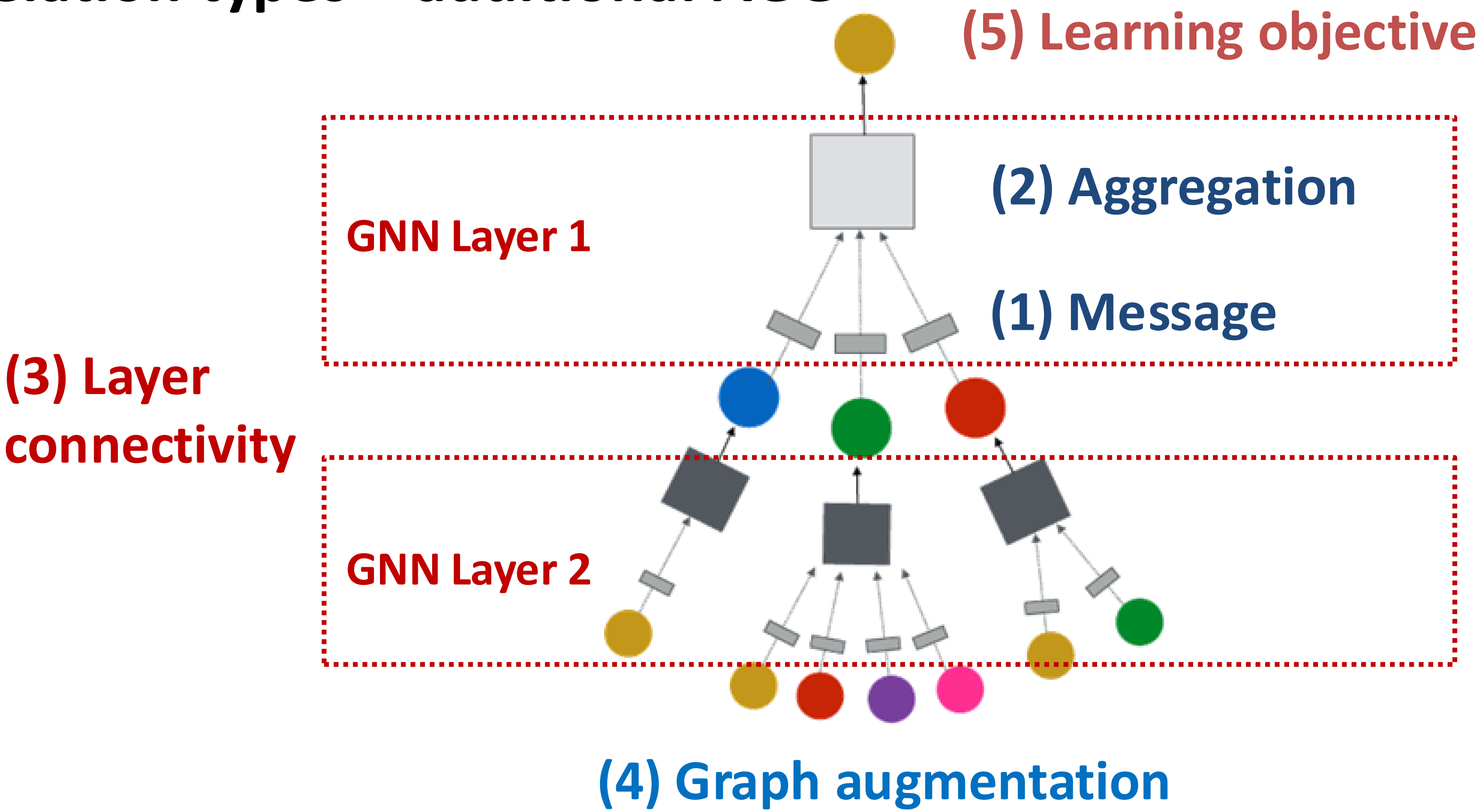
- $\widehat{\boldsymbol{y}}_{\boldsymbol{uv}} = \mathrm{Head}_{\mathrm{edge},\, r}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}) = \mathrm{Linear}_r(\mathrm{Concat}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}))$

Graph-level prediction:

- $\widehat{\boldsymbol{y}}_G = \mathrm{AGG}(\mathrm{Head}_{\mathrm{graph},\, i}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall T(v) = i\}))$

Summary: Heterogeneous GNN

- **Heterogeneous GNNs extend GNNs by separately modeling node/relation types + additional AGG**



Summary of the Lecture

- **Heterogeneous graphs:** graphs with multiple nodes or edge types
  - **Key concept: relation type** (node_s, edge, node_e)
  - Be aware that we don't always need heterogeneous graphs
- Learning with **heterogeneous graphs**
  - **Key idea: separately model each relation type**
  - **Relational GCNs**
  - Heterogeneous Graph Transformer
  - **Design space for heterogeneous GNNs**