ANNOUNCEMENTS

• Project Proposal due today
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
CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu
Heterogeneous Graphs: Motivation

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
Heterogeneous Graphs: Motivation

A graph could have multiple types of nodes and edges! **2 types of nodes + 2 types of edges.**
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
A heterogeneous graph is defined as

\[ G = (V, E, \tau, \phi) \]

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

![Diagram of an E-Commerce Graph with nodes for User, Item, and Query, and edges for Search, Click, Guide, Purchase, Visit, Guide, Search]
Example: Academic Graph

- **Node types:** Author, Paper, Venue, Field, ...
- **Edge types:** Publish, Cite, ...
- **Benchmark dataset:** Microsoft Academic Graph
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?
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
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)
Stanford CS224W: Relational GCN (RGCN)
Recap: Classical GNN Layers: GCN

- (1) Graph Convolutional Networks (GCN)

\[
h_v^{(l)} = \sigma \left( W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|} \right)
\]

- How to write this as Message + Aggregation?

\[
h_v^{(l)} = \sigma \left( \sum_{u \in N(v)} W^{(l)} \frac{h_u^{(l-1)}}{|N(v)|} \right)
\]
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?
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?
What if the graph has multiple relation types?
What if the graph has multiple relation types?
Use different neural network weights for different relation types.
What if the graph has multiple relation types?
Use different neural network weights for different relation types!
Introduce a set of neural networks for each relation type!

- Weight for rel_1
- Weight for rel_N
- Weight for self-loop
Relational GCN (RGCN):

\[ h_v^{(l+1)} = \sigma \left( \sum \sum \frac{1}{c_{v,r}} W_r^{(l)} h_u^{(l)} + W_0^{(l)} h_v^{(l)} \right) \]

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
  \[ h_v^{(l+1)} = \sigma \left( \text{Sum} \left( \left\{ m_{u,r}^{(l)}, u \in N(v) \right\} \cup \left\{ m_v^{(l)} \right\} \right) \right) \]

Normalized by node degree of the relation \( c_{v,r} = |N_v^r| \)
Each relation has $L$ matrices: $\mathbf{W}_r^{(1)}, \mathbf{W}_r^{(2)} \ldots \mathbf{W}_r^{(L)}$

The size of each $\mathbf{W}_r^{(l)}$ is $d^{(l+1)} \times d^{(l)}$

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 $\mathbf{W}_r$

$$\mathbf{W}_r = \begin{bmatrix}
\end{bmatrix}$$

**Limitation**: only nearby neurons/dimensions can interact through $\mathbf{W}$

- If use $B$ low-dimensional matrices, then # param reduces from $d^{(l+1)} \times d^{(l)}$ to $B \times \frac{d^{(l+1)}}{B} \times \frac{d^{(l)}}{B}$
Key insight: **Share weights** across different relations!

Represent the matrix of each relation as a **linear combination** of basis transformations

\[ W_r = \sum_{b=1}^{B} a_{rb} \cdot V_b, \text{ where } V_b \text{ is shared across all relations} \]

- \( V_b \) 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
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): $\mathbf{h}^{(L)}_A \in \mathbb{R}^k$, each item in $\mathbf{h}^{(L)}_A$ represents the probability of that class
**Example: Link Prediction**

- **Link prediction split:**

  The original graph
  
  ![](image1)

  Split

  Split Graph with 4 categories of edges
  
  ![](image2)

  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.
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 \(h_A^{(L)} \in \mathbb{R}^d\)
- Relation-specific score function \(f_r: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}\)
  - One example \(f_{r_1}(h_E, h_A) = h_E^T W_{r_1} h_A, W_{r_1} \in \mathbb{R}^{d \times d}\)
RGCN for Link Prediction (2)

- **Training:**

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., \((E, r_3, B), (E, r_3, D)\)

---

**Input Graph**

- **Training supervision edges:** \((E, r_3, A)\)
- **Training message edges:** all the rest existing edges (solid lines)

---

E.g., \((E, r_3, C)\) is **NOT** a negative edge

---

Note the negative edges should **NOT** belong to training message edges or training supervision edges!
Training:

1. Use RGCN to score the training supervision edge \((E, r_3, A)\)
2. Create a **negative edge** by perturbing the supervision edge \((E, r_3, B)\)
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

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

\(\sigma \ldots \text{Sigmoid function}\)
Evaluation:

Validation time as an example, same at the test time

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., \((E, r_3, B)\)

**Input Graph**

- **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
Evaluation:

Validation time as an example, same at the test time

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., \((E, r_3, B)\)

Input Graph

1. Calculate the score of \((E, r_3, D)\)
2. Calculate the score of all the negative edges: \((E, r_3, v)\) where \((E, r_3, v)\) is NOT in the training message edges and 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 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 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**
**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 results:

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

**SOTA method: SeHGNN**

- **ComplEx (Next lecture) + Simplified GCN (Lecture 17)**
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
Stanford CS224W: Heterogeneous Graph Transformer

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu
Graph Attention Networks (GAT)

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

Attention weights

Not all node’s neighbors are equally important

- **Attention** is inspired by cognitive attention.
- The **attention** \( \alpha_{vu} \) 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?**
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)
HGT uses Scaled Dot-Product Attention (proposed in Transformer)

\[
\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V
\]

- Query: \(Q\), Key: \(K\), Value: \(V\)
  - \(Q, K, V\) have shape \((\text{batch\_size}, \text{dim})\)

How do we obtain \(Q, K, V\)?

- Apply Linear layer to the input
  - \(Q = Q\_Linear(X)\)
  - \(K = K\_Linear(X)\)
  - \(V = V\_Linear(X)\)
Recall: Applying GAT to a homogeneous graph

- $H^{(l)}$ is the $l$-th layer representation:

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

How do we take relation type (node\_s, edge, node\_e) into attention computation?
**Innovation:** Decompose heterogeneous attention to Node- and edge-type dependent attention mechanism

- **3 node weight matrices, 2 edge weight matrices**
- **Without decomposition:** $3 \times 2 \times 3 = 18$ relation types $\rightarrow 18$ weight matrices (suppose all relation types exist)
Heterogeneous Mutual Attention:

\[
\text{ATT-head}^i(s, e, t) = \left(K^i(s) \, W^{ATT}_{\phi(e)} \, Q^i(t)^T\right)
\]

\[
K^i(s) = \text{K-Linear}^i_{\tau(s)}(H^{(l-1)}[s])
\]

\[
Q^i(t) = \text{Q-Linear}^i_{\tau(t)}(H^{(l-1)}[t])
\]

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 \(K_{\text{Linear}}^T(s)\) & \(Q_{\text{Linear}}^T(t)\), which further return Key and Query vectors \(K(s)\) & \(Q(t)\)
- Edge type \(R(e)\) directly parameterizes \(W_{R(e)}\)
More Details on HGT

- A full HGT layer

\[ \tilde{H}^{(l)}[t] = \bigoplus_{s \in N(t)} \left( \text{Attention}_{HGT}(s, e, t) \cdot \text{Message}_{HGT}(s, e, t) \right) \]

We have just computed

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

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

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

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*

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

- 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
Stanford CS224W: Design Space of Heterogeneous GNNs
How do we extend the general GNN design space to heterogeneous graphs?

(1) Message
(2) Aggregation
(3) Layer connectivity
(4) Graph augmentation
(5) Learning objective
(1) Message computation

Message function:

\[ m_u^{(l)} = MSG^{(l)}(h_u^{(l-1)}) \]

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

- **Example**: A Linear layer \( m_u^{(l)} = W^{(l)} h_u^{(l-1)} \)
(1) Heterogeneous message computation

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

- \( m^{(l)}_u = MSG^{(l)}_r \left( h^{(l-1)}_u \right) \), \( r = (u, e, v) \) 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 \( m^{(l)}_u = W^{(l)}_r h^{(l-1)}_u \)
(2) Aggregation

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

$$h_v^{(l)} = \text{AGG}^{(l)} \left( \left\{ m_u^{(l)}, u \in N(v) \right\} \right)$$

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

$$h_v^{(l)} = \text{Sum}(\{m_u^{(l)}, u \in N(v)\})$$
(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

\[
\mathbf{h}_v^{(l)} = \text{AGG}_{all}^{(l)} \left( \text{AGG}_r^{(l)} \left( \left\{ \mathbf{m}_u^{(l)} , u \in N_r(v) \right\} \right) \right)
\]

- Given all the messages sent to a node
- Within each message type, aggregate the messages that belongs to the edge type with \(\text{AGG}_r^{(l)}\)
- Aggregate across the edge types with \(\text{AGG}_{all}^{(l)}\)
- **Example:**

\[
\mathbf{h}_v^{(l)} = \text{Concat} \left( \text{Sum} \left( \left\{ \mathbf{m}_u^{(l)} , u \in N_r(v) \right\} \right) \right)
\]
(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 pre/post-process layers:

- MLP layers *with respect to each node type*
  - Since the output of GNN are *node embeddings*
  - \( h^{(l)}_v = \text{MLP}_{T(v)}(h^{(l)}_v) \)
  - \( T(v) \) 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 $\rightarrow$ feature augmentation

- **Graph Structure manipulation**
  - The graph is too sparse $\rightarrow$ Add virtual nodes / edges
  - The graph is too dense $\rightarrow$ Sample neighbors when doing message passing
  - The graph is too large $\rightarrow$ 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:
\[ \hat{y}_v = \text{Head}_{\text{node}}(h^{(L)}_v) = W^{(H)} h^{(L)}_v \]

Edge-level prediction:
\[ \hat{y}_{uv} = \text{Head}_{\text{edge}}(h^{(L)}_u, h^{(L)}_v) = \text{Linear}(\text{Concat}(h^{(L)}_u, h^{(L)}_v)) \]

Graph-level prediction:
\[ \hat{y}_G = \text{Head}_{\text{graph}}(\{h^{(L)}_v \in \mathbb{R}^d, \forall v \in G\}) \]
Heterogeneous Prediction Heads

**Node-level prediction:**

\[ \hat{y}_v = \text{Head}_{\text{node}}(T(v), h_v^{(L)}) = W_{T(v)}^{(H)} h_v^{(L)} \]

**Edge-level prediction:**

\[ \hat{y}_{uv} = \text{Head}_{\text{edge}}(r(h_u^{(L)}, h_v^{(L)})) = \text{Linear}_r(\text{Concat}(h_u^{(L)}, h_v^{(L)})) \]

**Graph-level prediction:**

\[ \hat{y}_G = \text{AGG}((\text{Head}_{\text{graph}}(i, \{h_v^{(L)} \in \mathbb{R}^d, \forall T(v) = i \})))) \]
Heterogeneous GNNs extend GNNs by separately modeling node/relation types + additional AGG

(1) Message

(2) Aggregation

(3) Layer connectivity

(4) Graph augmentation

(5) Learning objective
Heterogeneous graphs: graphs with multiple nodes or edge types
- Key concept: relation type \((\text{node}_s, \text{edge}, \text{node}_e)\)
- 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