A GENTLE TUTORIAL ON GRAPH NEURAL NETWORKS AND ITS APPLICATION TO PROGRAMMING LANGUAGE

Yizhou Sun
Department of Computer Science
University of California, Los Angeles
yzsun@cs.ucla.edu

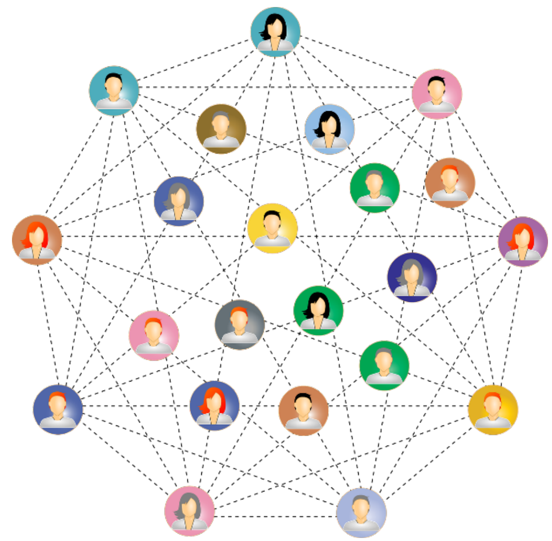
August 17, 2020

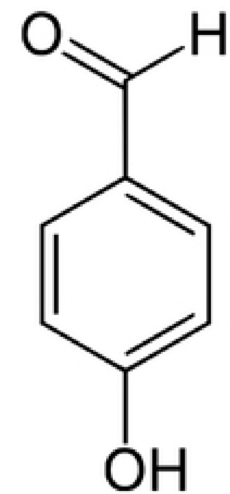
Outline

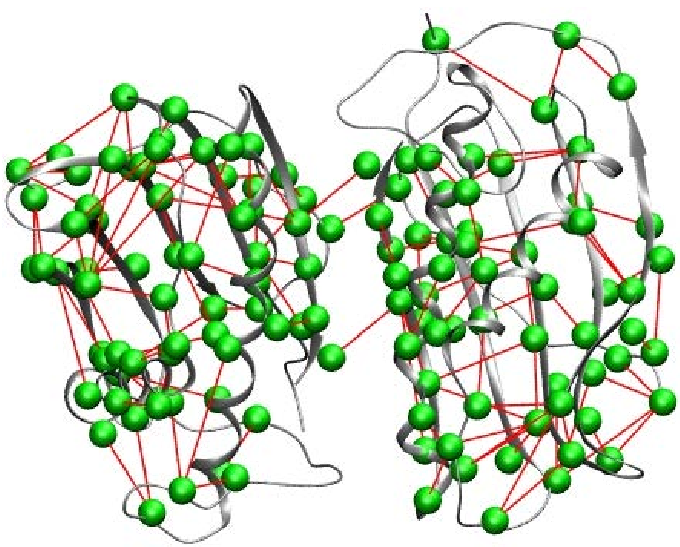
- Introduction
- Graph Neural Networks
- Downstream Tasks for Graphs
- Applications in PL
- Discussions

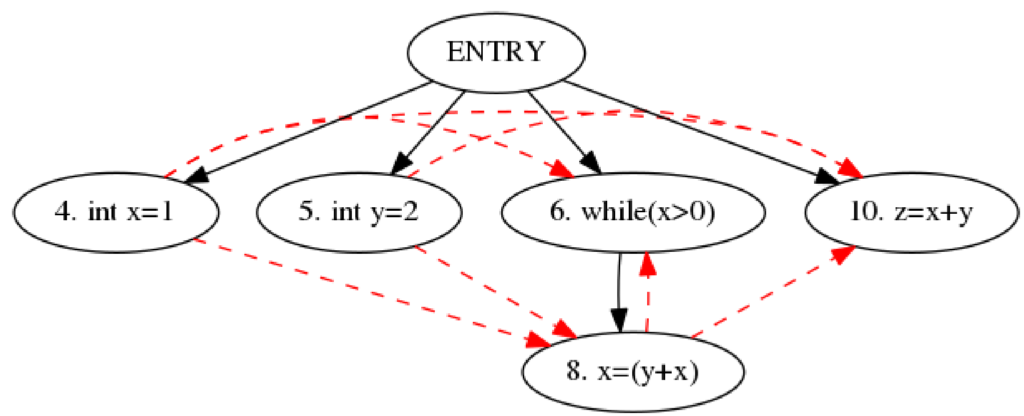
Graph Analysis

- Graphs are ubiquitous
  - Social networks
  - Proteins
  - Chemical compounds
  - Program dependence graph
  - …



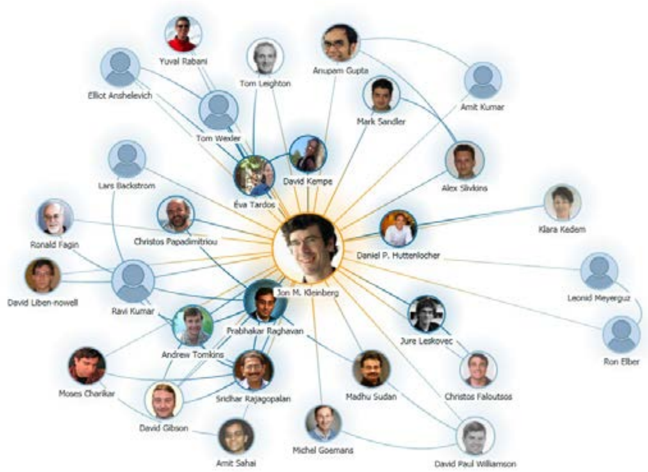






Representing Nodes and Graphs

- Important for many graph related tasks
- Discrete nature makes it very challenging
- Naïve solutions



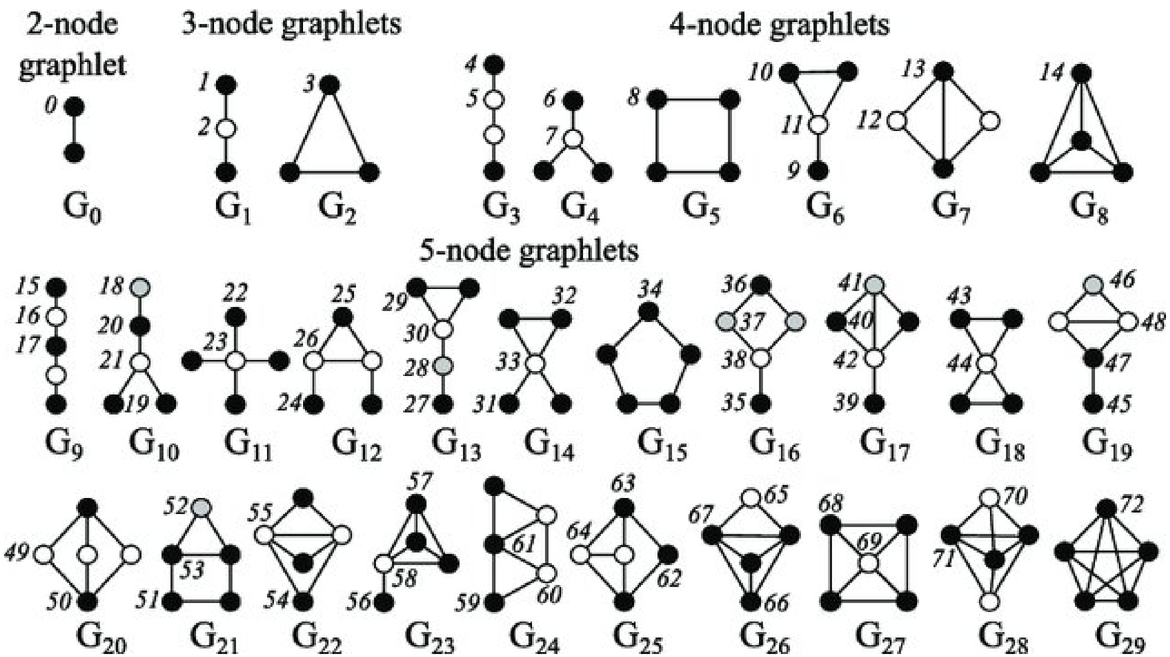
	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0

Limitations:

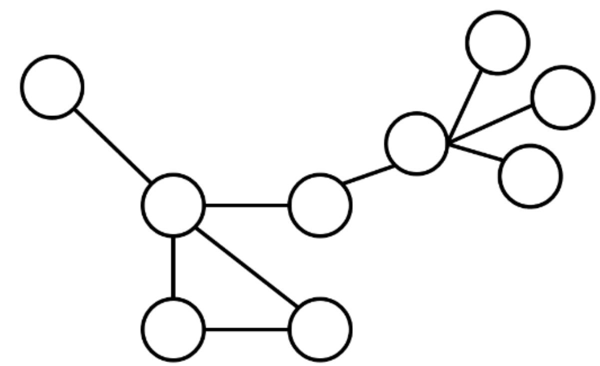
Extremely High-dimensional
No global structure information integrated
Permutation-variant

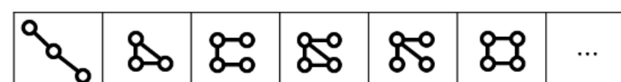
Even more challenging for graph representation

- Ex. Graphlet-based feature vector



Source: DOI: 10.1093/bioinformatics/btv130





12	1	4	1	6	0	…

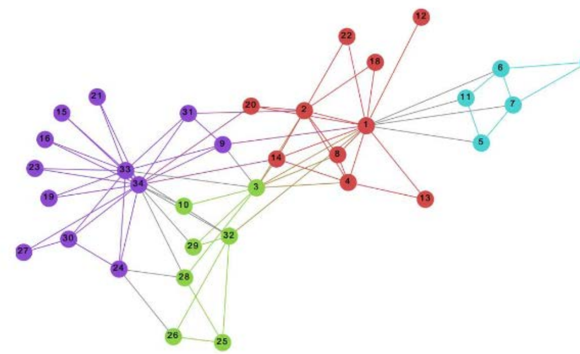
Requires subgraph isomorphism test: NP-hard

Source: https://haotang1995.github.io/projects/robust_graph_level_representation_learning_using_graph_based_structural_attentional_learning

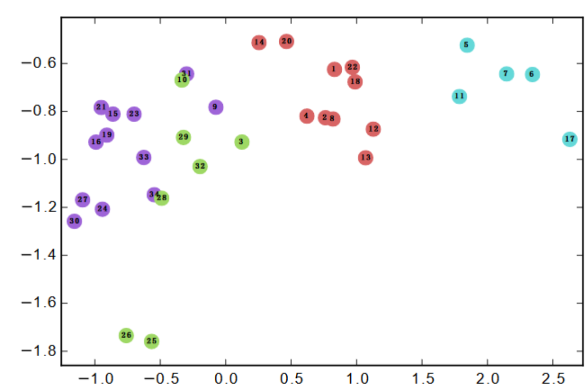
Automatic representation Learning

- Map each node/graph into a low dimensional vector
  - $\phi: V \rightarrow R^d$ or $\phi: \mathcal{G} \rightarrow R^d$
- Earlier methods
  - Shallow node embedding methods inspired by word2vec
    - DeepWalk [Perozzi, KDD'14]
    - LINE [Tang, WWW'15]
    - Node2Vec [Grover, KDD'16]

$\phi(v) = U^T x_v$, where U is the embedding matrix and x_v is the one-hot encoding vector



(a) Input: karate network



(b) Output: representations

Source: DeepWalk

Limitation of shallow embedding techniques

- Too many parameters
  - Each node is associated with an embedding vector, which are parameters
- Not inductive
  - Cannot handle new nodes
- Cannot handle node attributes

From shallow embedding to Graph Neural Networks

- The embedding function (encoder) is more complicated
  - Shallow embedding
    - $\phi(v) = U^T x_v$, where U is the embedding matrix and x_v is the one-hot encoding vector
  - Graph neural networks
    - $\phi(v)$ is a neural network depending on the graph structure

Outline

- Introduction
- Graph Neural Networks
- Downstream Tasks for Graphs
- Applications in PL
- Discussions

Notations

- An attributed graph $G = (V, E)$
  - V: vertex set
  - E: edge set
  - A: adjacency matrix
  - $X \in R^{d_0 \times |V|}$: feature matrix for all the nodes
  - $N(v)$: neighbors of node v
  - h_v^l: Representation vector of node v at Layer l
    - $Note\, h_v^0 = x_v$
  - $H^l \in R^{d_l \times |V|}$: representation matrix

The General Architecture of GNNs

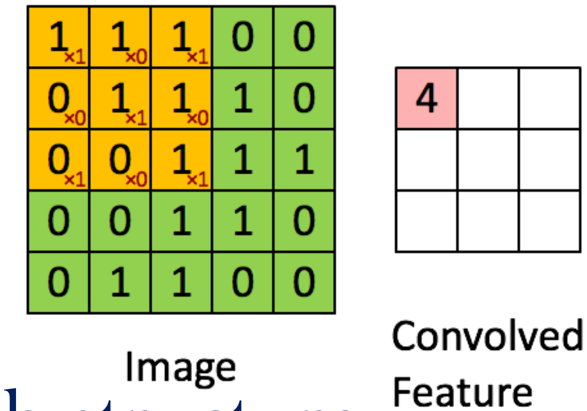
- For a node v at layer t

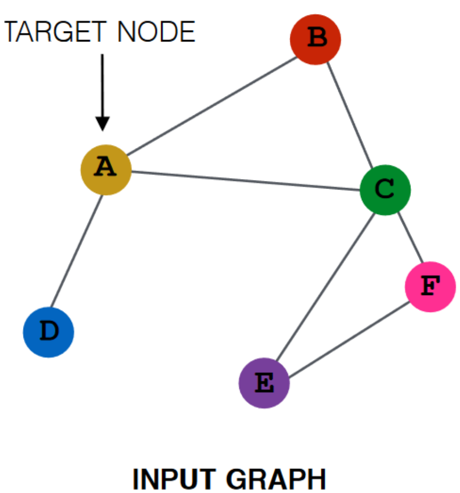
$$h_v^{(t)} = f\left(\underbrace{h_v^{(t-1)}}_{\text{representation vector from previous layer for node v}}, \left\{\underbrace{h_u^{(t-1)} | u \in \mathcal{N}(v)}_{\text{representation vectors from previous layer for node v's neighbors}}\right\}\right)$$

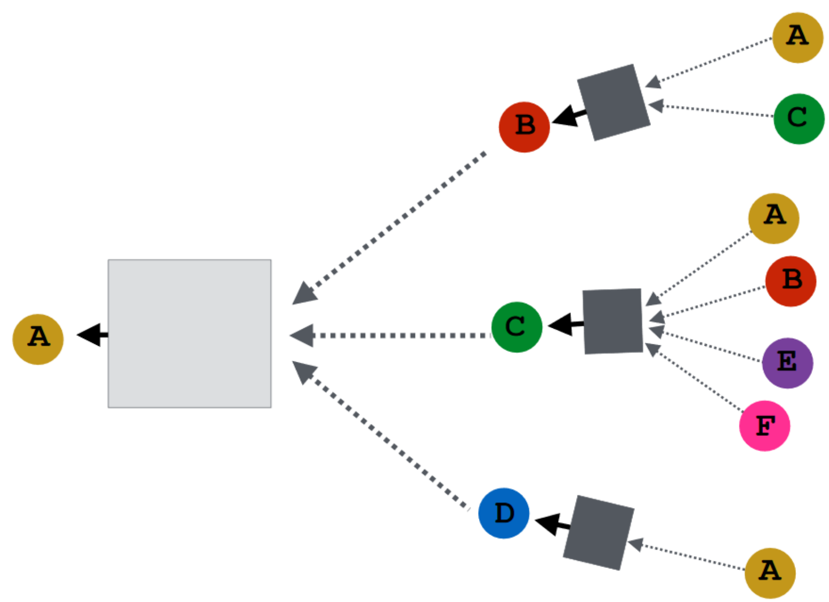
- A function of representations of neighbors and itself from previous layers
  - **Aggregation** of neighbors
  - **Transformation** to a different space
  - **Combination** of neighbors and the node itself

Compare with CNN

- Recall CNN
  - Regular graph
- GNN
  - Extend to irregular graph structure







Graph Convolutional Network (GCN)

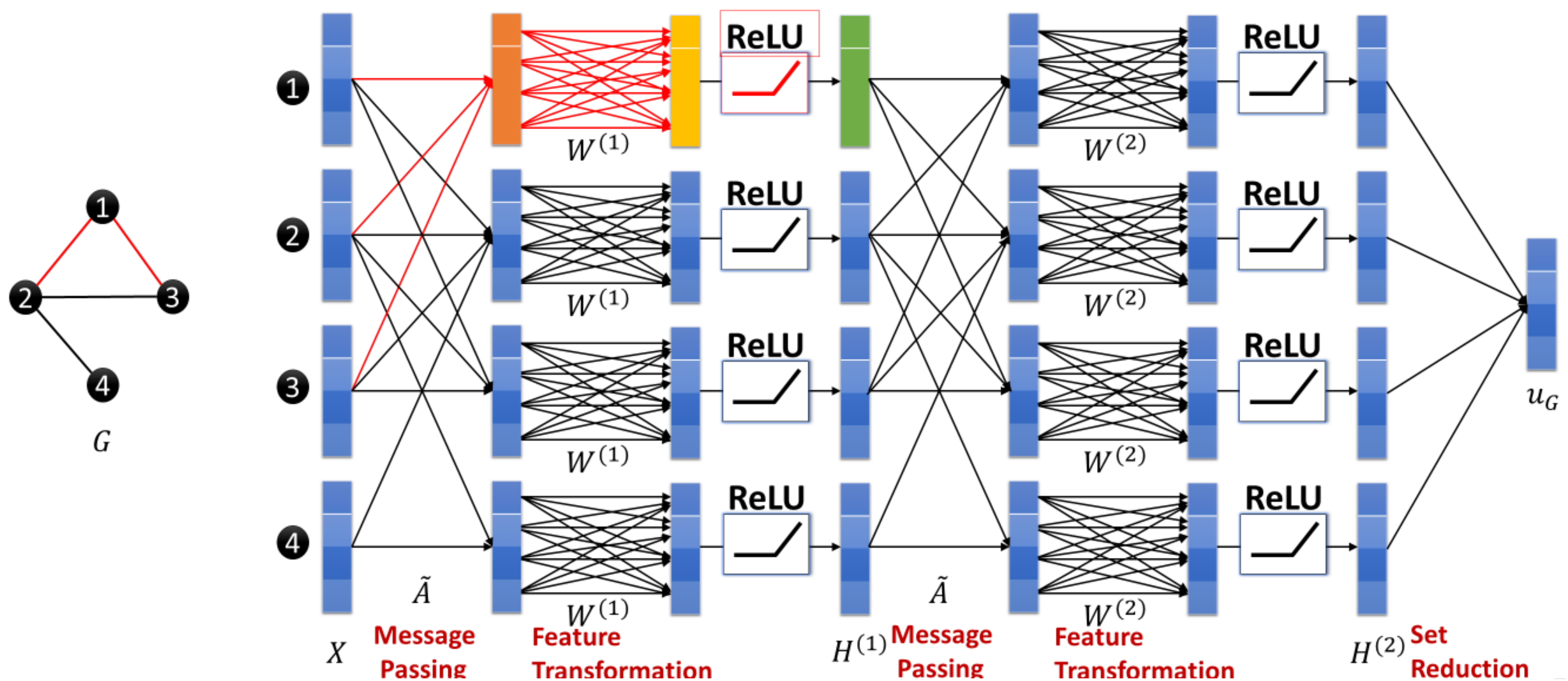
- Kipf and Welling, ICLR'17
  - $f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$, $\hat{A} = A + I$
  - f: graph filter
- From a node v's perspective

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

W_k: *weight matrix at Layer k, shared across different nodes*

A toy example of 2-layer GCN on a 4-node graph

- Computation graph



GraphSAGE

- Inductive Representation Learning on Large Graphs
  William L. Hamilton*, Rex Ying*, Jure Leskovec, NeurIPS'17

$$\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}^{k-1}_u, \forall u \in \mathcal{N}(v)\})$$

$$\mathbf{h}^k_v \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}^{k-1}_v, \mathbf{h}^k_{\mathcal{N}(v)})\right)$$

A more general form

$$\mathbf{h}^k_v = \sigma\left(\left[\mathbf{W}_k \cdot \boxed{\text{AGG}\left(\{\mathbf{h}^{k-1}_u, \forall u \in N(v)\}\right)}, \mathbf{B}_k \mathbf{h}^{k-1}_v\right]\right)$$

More about AGG

- Mean

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

- LSTM

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

  - $\pi(\cdot)$: *a random permutation*

- Pool

$$\text{AGG} = \gamma\left\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\right\}$$

  - $\gamma(\cdot)$: Element-wise mean/max pooling of neighbor set

Message-Passing Neural Network

- Gilmer et al., 2017. Neural Message Passing for Quantum Chemistry. *ICML.*
- *A general framework that subsumes most GNNs*
  - Can also include edge information
- Two steps
  - Get messages from neighbors at step k

$$\mathbf{m}_v^k = \sum_{u \in N(v)} M(\mathbf{h}_u^{k-1}, \mathbf{h}_v^{k-1}, \mathbf{e}_{u,v})$$ e.g., Sum or MLP

  - Update the node latent represent based on the msg

$$\mathbf{h}_v^k = U(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$ e.g., LSTM, GRU

$A\ special\ case$: $GGNN$, **Li et al., Gated graph sequence neural networks, ICLR 2015**

Graph Attention Network (GAN)

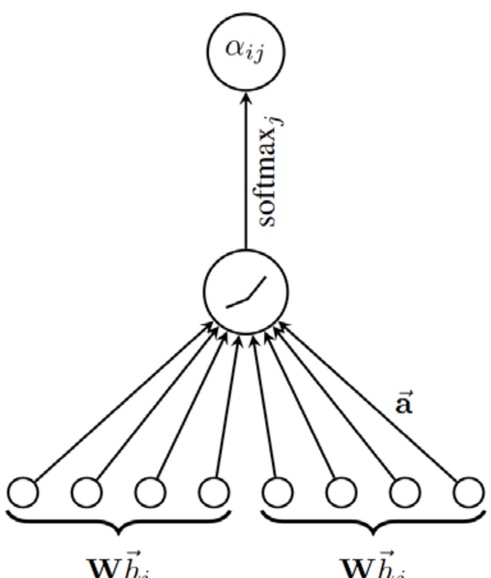
- How to decide the importance of neighbors?
  - GCN: a predefined weight
  - Others: no differentiation
- GAN: decide the weights using learnable attention
  - Velickovic et al., 2018. Graph Attention Networks. *ICLR.*

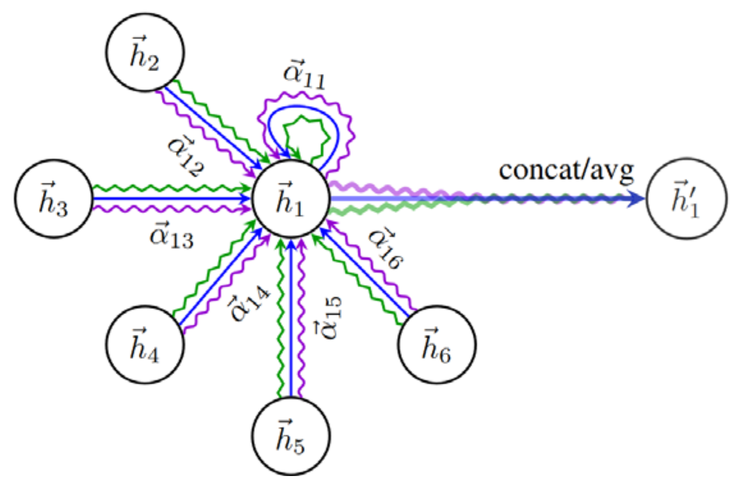
$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \boxed{\alpha_{ij}} \mathbf{W}\vec{h}_j\right)$$

The attention mechanism

- Potentially many possible designs

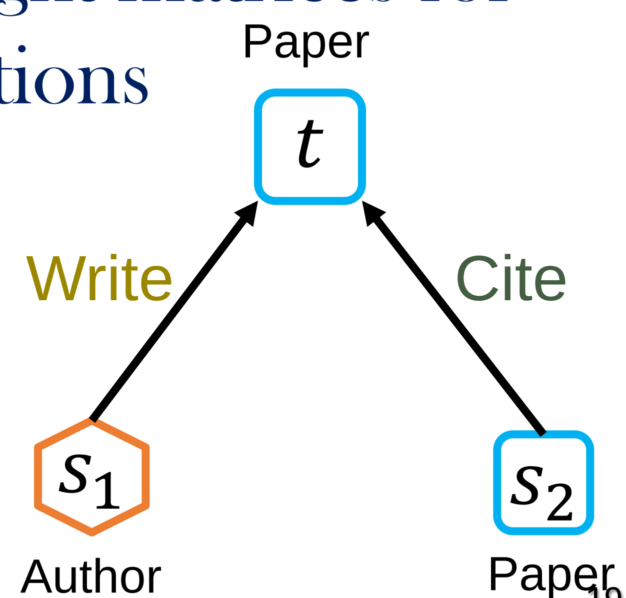
$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$





Heterogeneous Graph Transformer (HGT)

- How to handle heterogeneous types of nodes and relations?
  - Introduce different weight matrices for different types of nodes and relations
  - Introduce different attention weight matrices for different types of nodes and relations

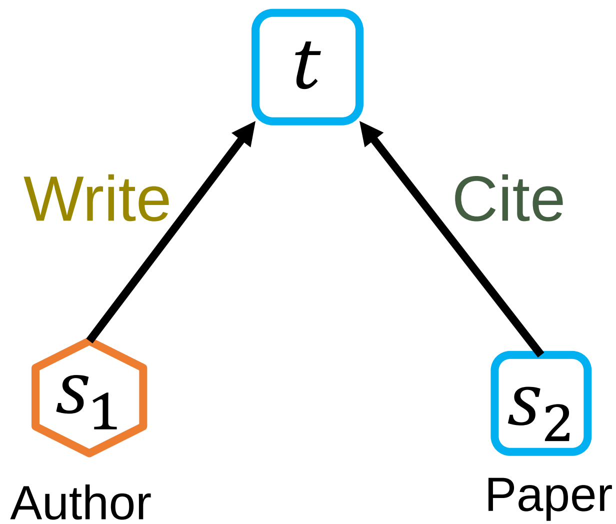


Hu et al., Heterogeneous Graph Transformer, WWW'20

Meta-Relation-based Parametrization

- Introduce node- and edge- dependent parameterization
  - Leverage meta relation <*source node type*, *edge type*, *target node type*> to parameterize attention and message passing weight.

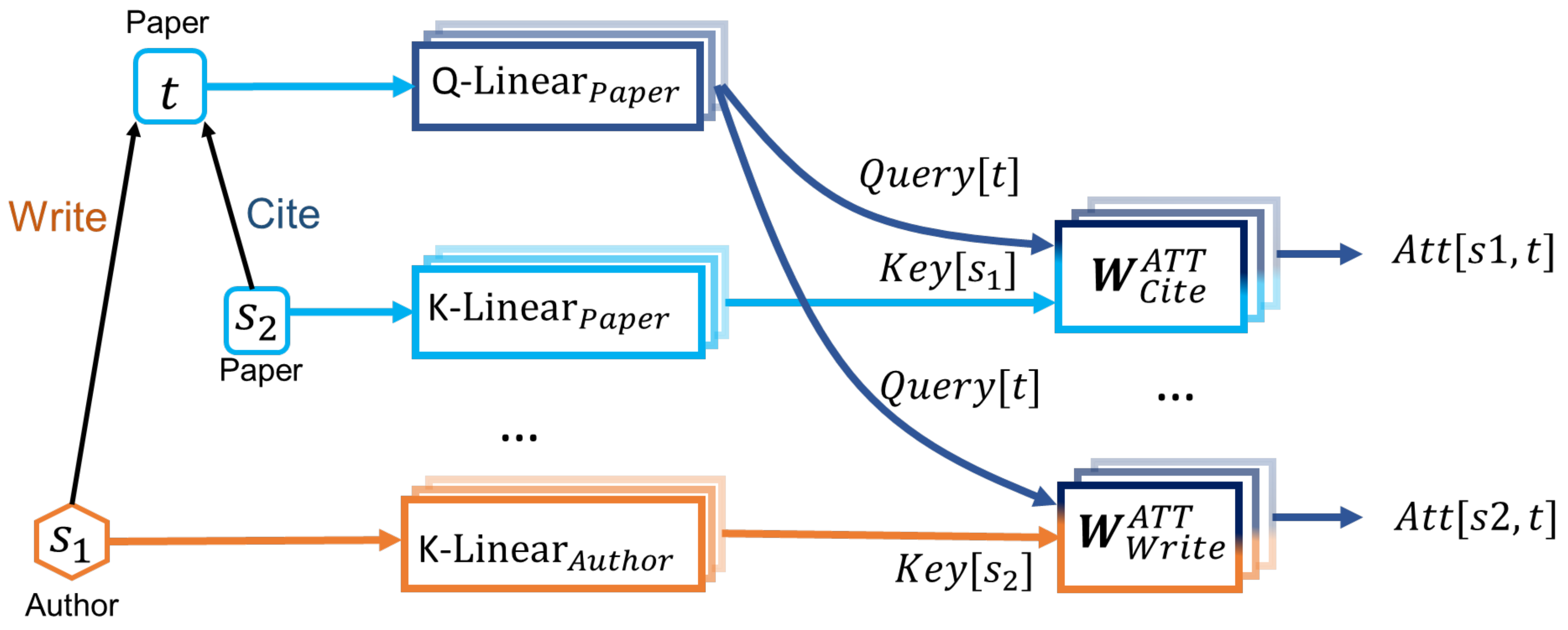
$$W_{\langle \text{Author}, \text{Write}, \text{Paper} \rangle} = W_{\text{Author}} W_{\text{Write}} W_{\text{Paper}}$$



$$W_{\langle \text{Paper}, \text{Cite}, \text{Paper} \rangle} = W_{\text{Paper}} W_{\text{Cite}} W_{\text{Paper}}$$

Meta-Relation-based Attention

- Attention learning is also parameterized based on node type and link type



Outline

- Introduction
- Graph Neural Networks
- Downstream Tasks for Graphs
- Applications in PL
- Discussions

Typical Graph Functions

- Node level
  - Similarity search
  - Link prediction
  - Classification
  - Community detection
  - Ranking

- Graph level
  - Similarity search
  - Frequent pattern mining
  - Graph isomorphism test
  - Graph matching
  - Classification
  - Clustering
  - Graph generation

1. Semi-supervised Node Classification

- Decoder using $z_v = h_v^L$
  - Feed into another fully connected layer
  - $\hat{y}_v = \sigma(\theta^T z_v)$
- Loss function
  - Cross entropy loss
  - In a binary classification case
    - $l_v = y_v log\hat{y}_v + (1 - y_v)\log(1 - \hat{y}_v)$

Applications of Node Classification

- Social network
  - An account is bot or not
- Citation network
  - A paper's research field
- A program-derived graph
  - The type of a variable

2. Link Prediction

- Decoder using $z_v = h_v^L$
  - Given a node pair (u, v)
  - Determine its probability $p_{uv} = z_u^T R z_v$
  - R could be different for different relation type
- Loss function
  - Cross entropy loss
    - $l_{uv} = y_{uv} log p_{uv} + (1 - y_{uv})\log(1 - p_{uv})$

Link Prediction Applications

- Social network
  - Friend recommendation
- Citation network
  - Citation recommendation
- Medical network
  - Drug and target binding or not
- A program-derived graph
  - Code autocomplete

3. Graph Classification

- Decoder using $h_G = g(\{z_v\}_{v \in V})$
  - $g(\cdot)$: $a\ read\ out\ function$, e.g., sum
  - Feed h_G into another fully connected layer
  - $\hat{y}_G = \sigma(\theta^T h_G)$
- Loss function
  - Cross entropy loss
  - In a binary classification case
    - $l_G = y_G log\hat{y}_G + (1 - y_G)\log(1 - \hat{y}_G)$

Graph Classification Applications

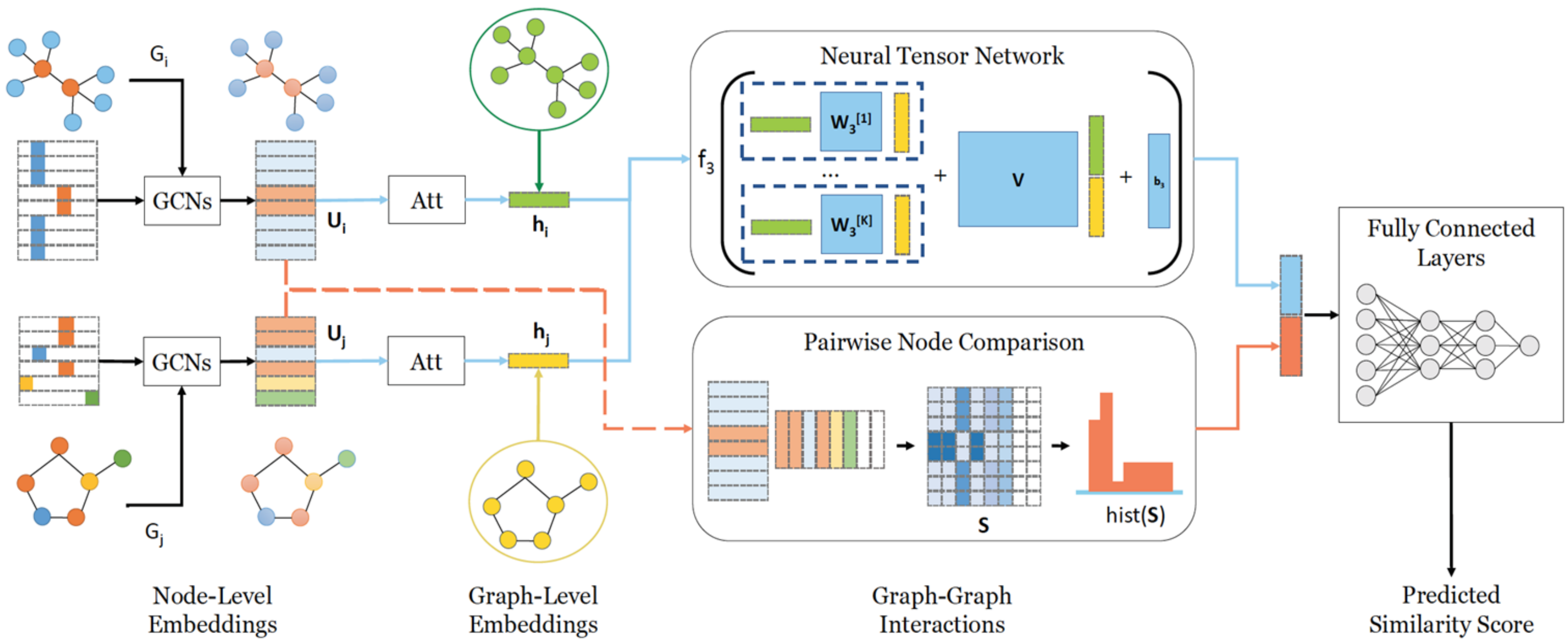
- Chemical compounds
  - Toxic or not
- Proteins
  - Has certain function or not
- Program-derived graphs
  - Contains bugs or not

4. Graph Similarity Computation

- Decoder using $h_G = g(\{z_v\}_{v \in V})$
  - Given a graph pair (G_1, G_2)
  - Determine its score $s_{G_1 G_2} = h_{G_1}^T R h_{G_2}$
- Loss function
  - E.g., Square loss
    - $l_{G_1 G_2} = (y_{G_1 G_2} - s_{G_1 G_2})$^2

A Concrete solution by SimGNN [Bai et al., AAAI 2019]

- Goal: learn a GNN ϕ: $\mathcal{G} \times \mathcal{G} \rightarrow R^{+}$ to approximate Graph Edit Distance between two graphs



1. **Attention**-based graph-level embedding
2. **Histogram** features from pairwise node similarities

Graph Similarity Computation Applications

- Drug database
  - Drug similarity search
- Program database
  - Code recommendation
    - Search ninja code for novice code
    - Search java code for COBOL code

CNN BUSINESS

Wanted urgently: People who know a half century-old computer language so states can process unemployment claims

By Alicia Lee, CNN

Updated 4:00 PM ET, Wed April 8, 2020

Outline

- Introduction
- Graph Neural Networks
- Downstream Tasks for Graphs
- Applications in PL
- Discussions

Deep Learning in PL

- Programs as sequences

```
public class FooBar {
    int BAZ_CONST = 42;
}
```

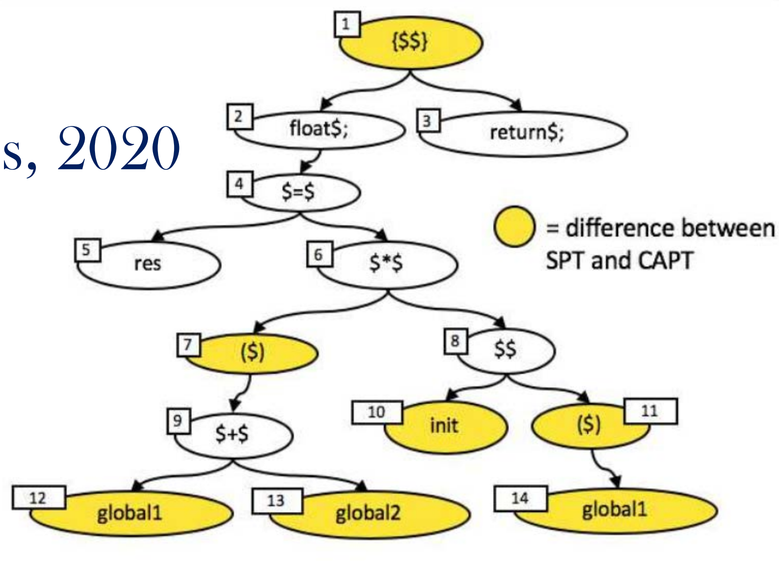
Lexer++

“public”, “class”, “CLASS0”, “{”, “int”, “VAR0”, “=”, “42”, “;”, “}”

  - Follow NLP techniques

```
float func()
{
    float res = (global1 + global2) * init(global1);
    return res;
}
```

- Programs as trees
  - Ye et al., Context-Aware ParseTrees, 2020



Simplified Parse Tree (SPT)

Programs as graphs

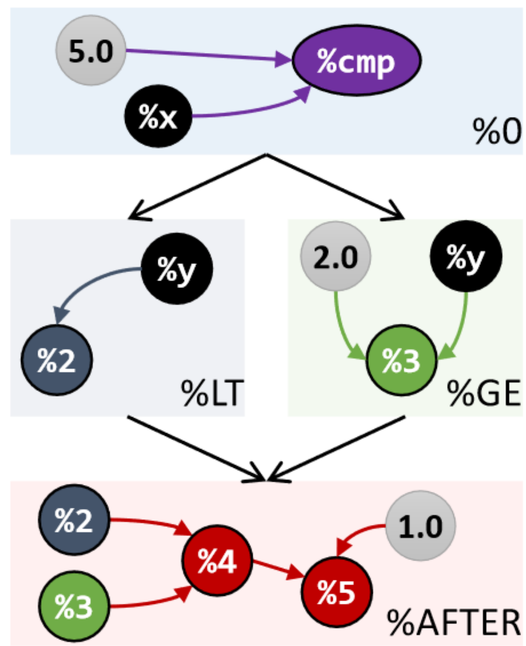
- **Ben-Nun et al.,** Neural Code Comprehension: A Learnable Representation of Code Semantics, NeurIPS 2018

```
double thres = 5.0;
if (x < thres)
    x = y * y;
else
    x = 2.0 * y;
x += 1.0;
```

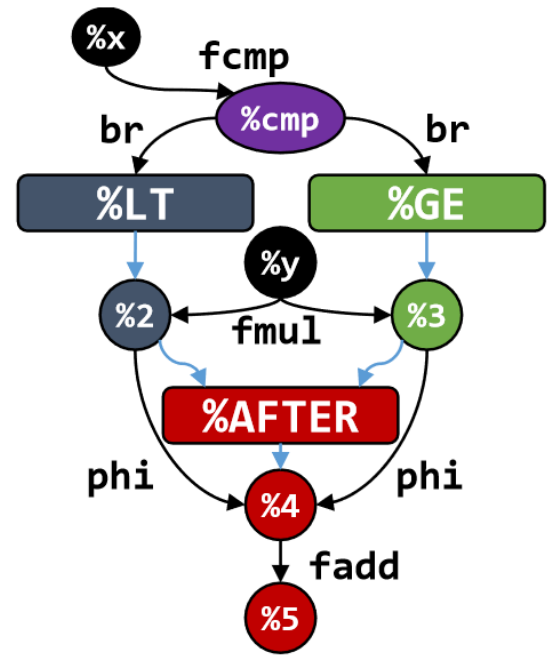
(a) Source code

```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```

(b) LLVM IR



(c) Dataflow basic blocks

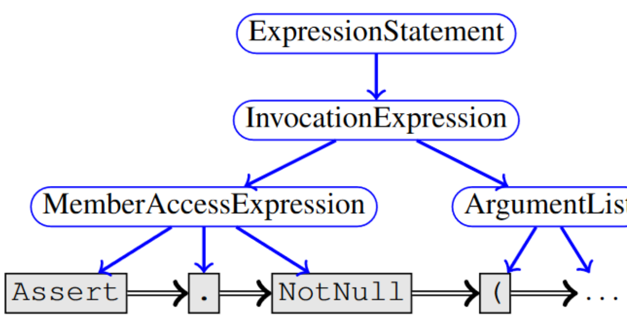


(d) Contextual Flow Graph

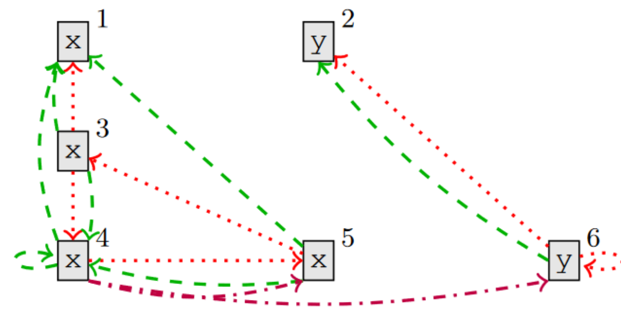
- Then shallow embedding: inst2vec

GNNs in PL

- Allamanis et al., Learning to represent programs with graphs, ICLR 2018
  - Tasks: (1) predict the name of a variable; (2) predict the right variable for a given location
  - Methodology: Gated GNN



(a) Simplified syntax graph for line 2 of Fig. 1, where blue rounded boxes are syntax nodes, black rectangular boxes syntax tokens, blue edges Child edges and double black edges NextToken edges.

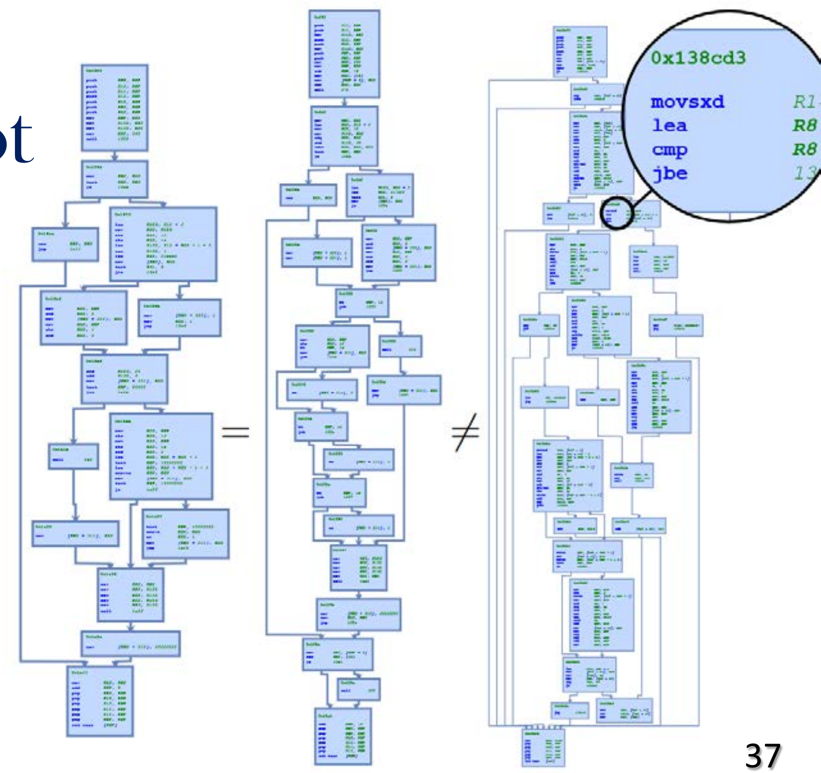


(b) Data flow edges for ($\boxed{\texttt{x}}^1$, $\boxed{\texttt{y}}^2$) = `Foo();` `while` ($\boxed{\texttt{x}}^3$ > 0) $\boxed{\texttt{x}}^4$ = $\boxed{\texttt{x}}^5$ + $\boxed{\texttt{y}}^6$ (indices added for clarity), with red dotted LastUse edges, green dashed LastWrite edges and dashdotted purple ComputedFrom edges.

GNNs in PL (Cont.)

- Li et al., Graph Matching Networks for Learning the Similarity of Graph Structured Objects, ICML 2019
  - Tasks: decide whether two functions are the same or not
  - Methodology: extend message passing across different graphs

Control flow graph for functions



Outline

- Introduction
- Graph Neural Networks
- Downstream Tasks for Graphs
- Applications in PL
- Discussions

Open Questions

- How to represent programs into graphs?
  - Iyer, Sun, Wang, Gottschlich, Software Language Comprehension using a Program-Derived Semantic Graph, arXiv:2004.00768
    - Capture program semantics at many levels of granularity
    - A hierarchical graph

Open Questions

- Why GNNs work?
  - Is the nonlinear transformation necessary?
  - Chen et al., Are Powerful Graph Neural Nets Necessary? A Dissection on Graph Classification, arXiv:1905.04579
  - A concatenate feature vector from graph propagation, followed by a MLP works equally well, and much faster!

$$X^G = \gamma(G, X) = \left[\boldsymbol{d}, X, \tilde{A}^1 X, \tilde{A}^2 X, \cdots, \tilde{A}^K X\right],$$

Q & A

- Thanks to my collaborators:
  - Yunsheng Bai, Wei Wang, Derek Xu, Hao Ding, Ting Chen, Ziniu Hu, etc...

- Thanks to my funding agencies and industry support:
  - NSF, DARPA, PPDAI, Yahoo!, Nvidia, Snapchat, Amazon, Okawa Foundation