Introduction to Graph Neural Networks

Jiaxuan You, Stanford University

Adapted from Stanford CS 224W & CS 246

Many Types of Data are Graphs

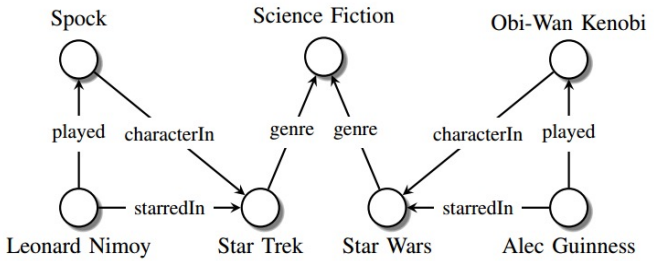


Image credit: Maximilian Nickel et al

Knowledge Graphs

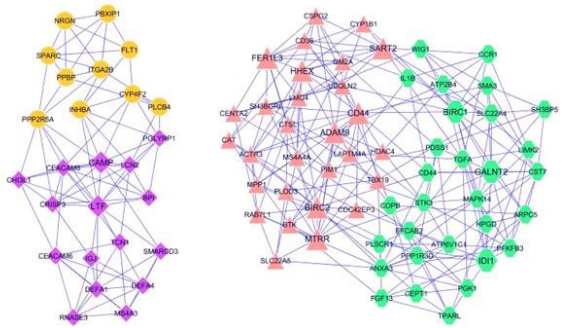


Image credit: ese.wustl.edu

Regulatory Networks

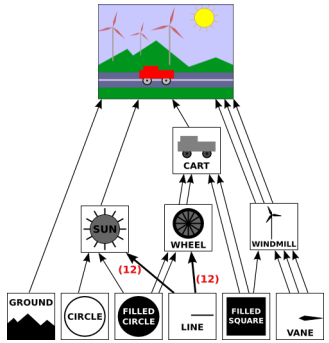


Image credit: math.hws.edu

Scene Graphs

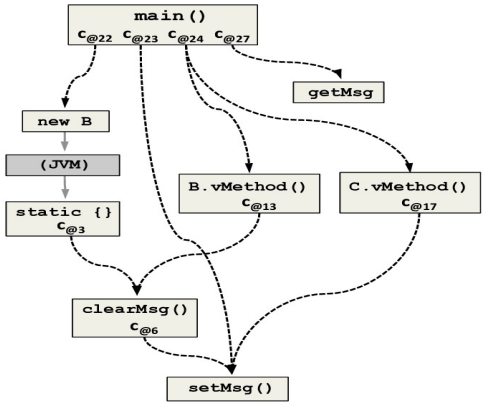


Image credit: ResearchGate

Code Graphs

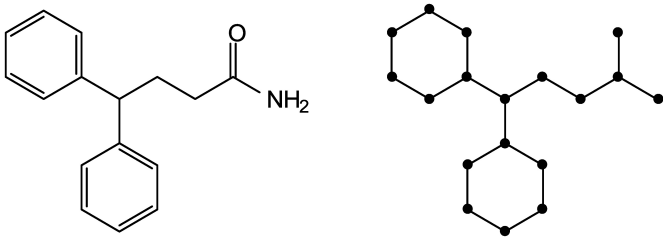


Image credit: MDPI

Molecules

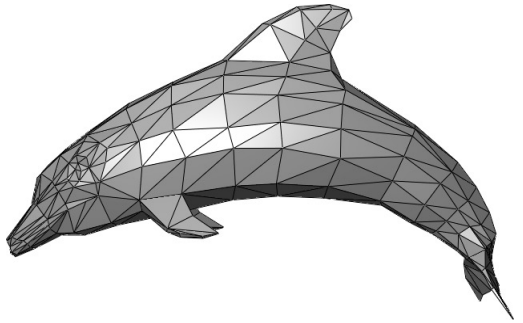
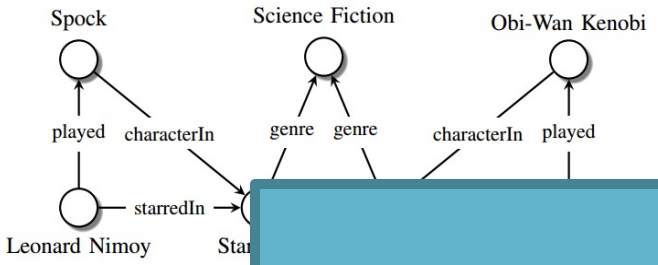
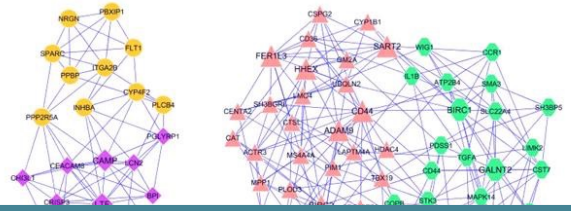


Image credit: Wikipedia

3D Shapes

Many Types of Data are Graphs





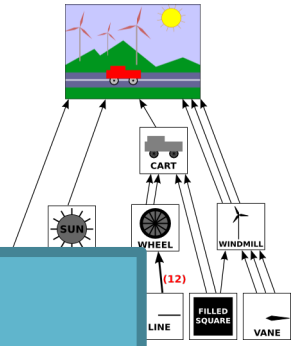


Image cred

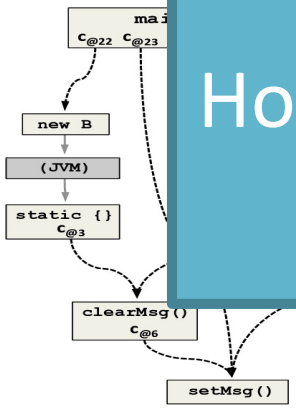
ath.hws.edu

Know

Graphs

Main question:

How do we perform ML over graphs?



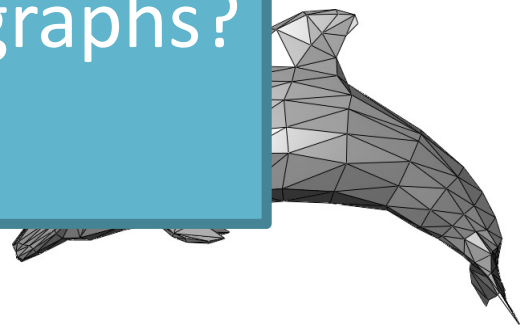


Image credit: ResearchGate

Image credit: MDPI

Image credit: Wikipedia

Code Graphs

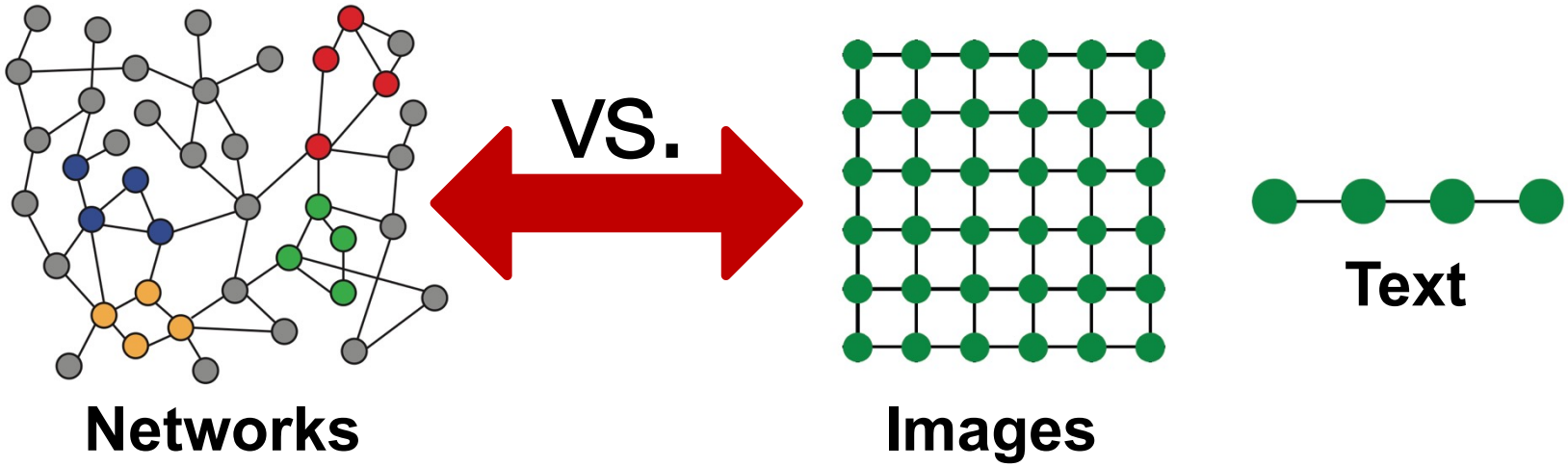
Molecules

3D Shapes

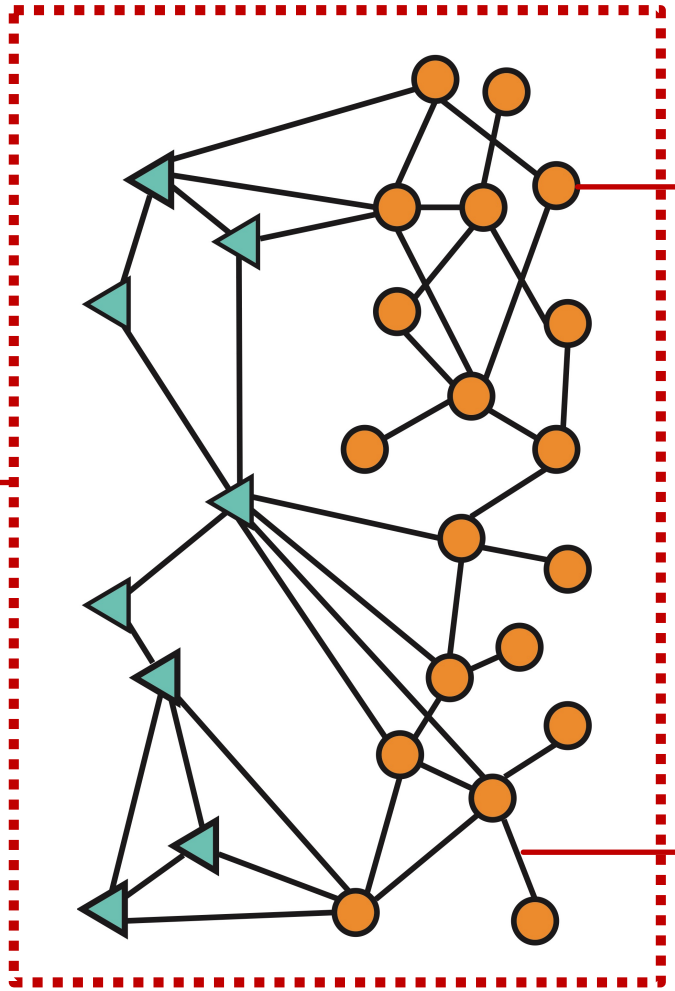
Why is it Hard?

Networks are complex.

- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)



Learning From Graphs



Node-level prediction

"What is the area of this research paper?"

Graph-level prediction

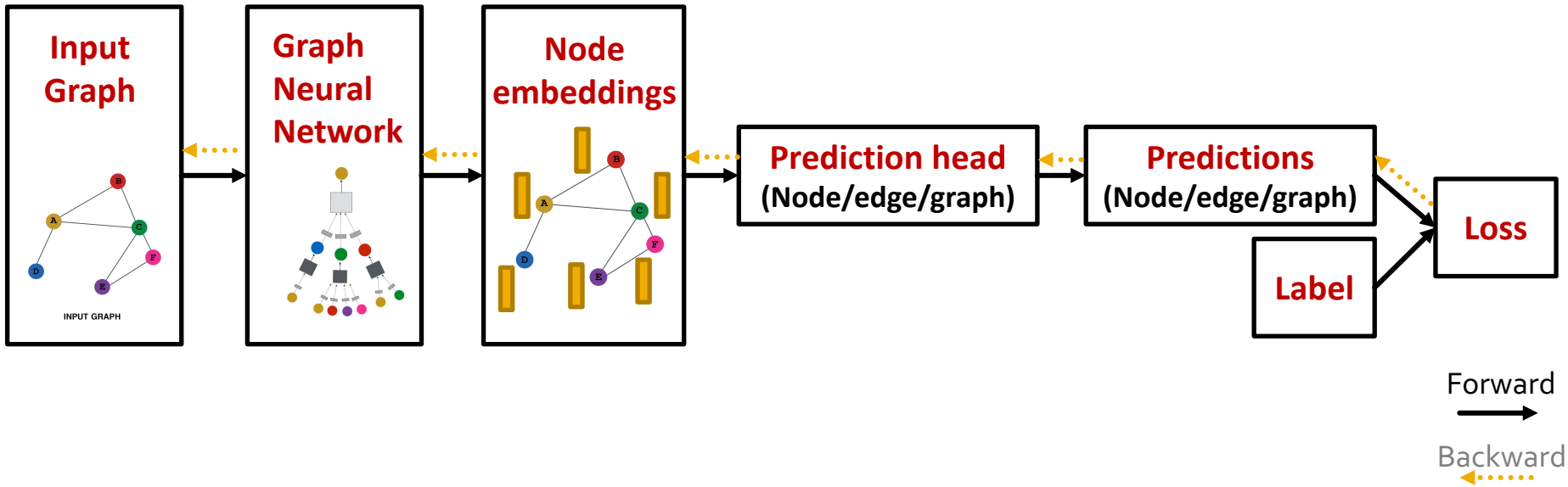
"Is this molecular graph toxic?"

Edge-level prediction

"Is this transaction fraudulent?"

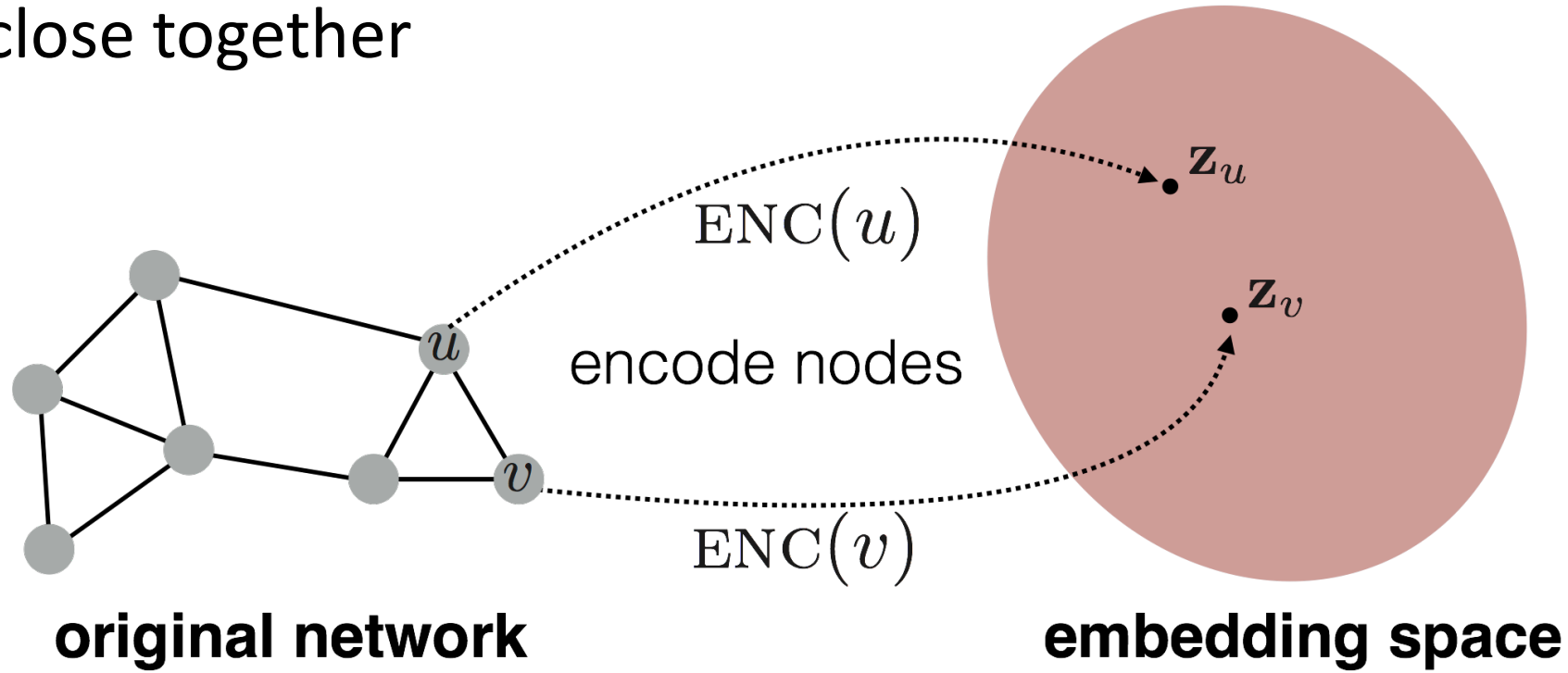
J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020

Deep Learning Pipeline for Graphs



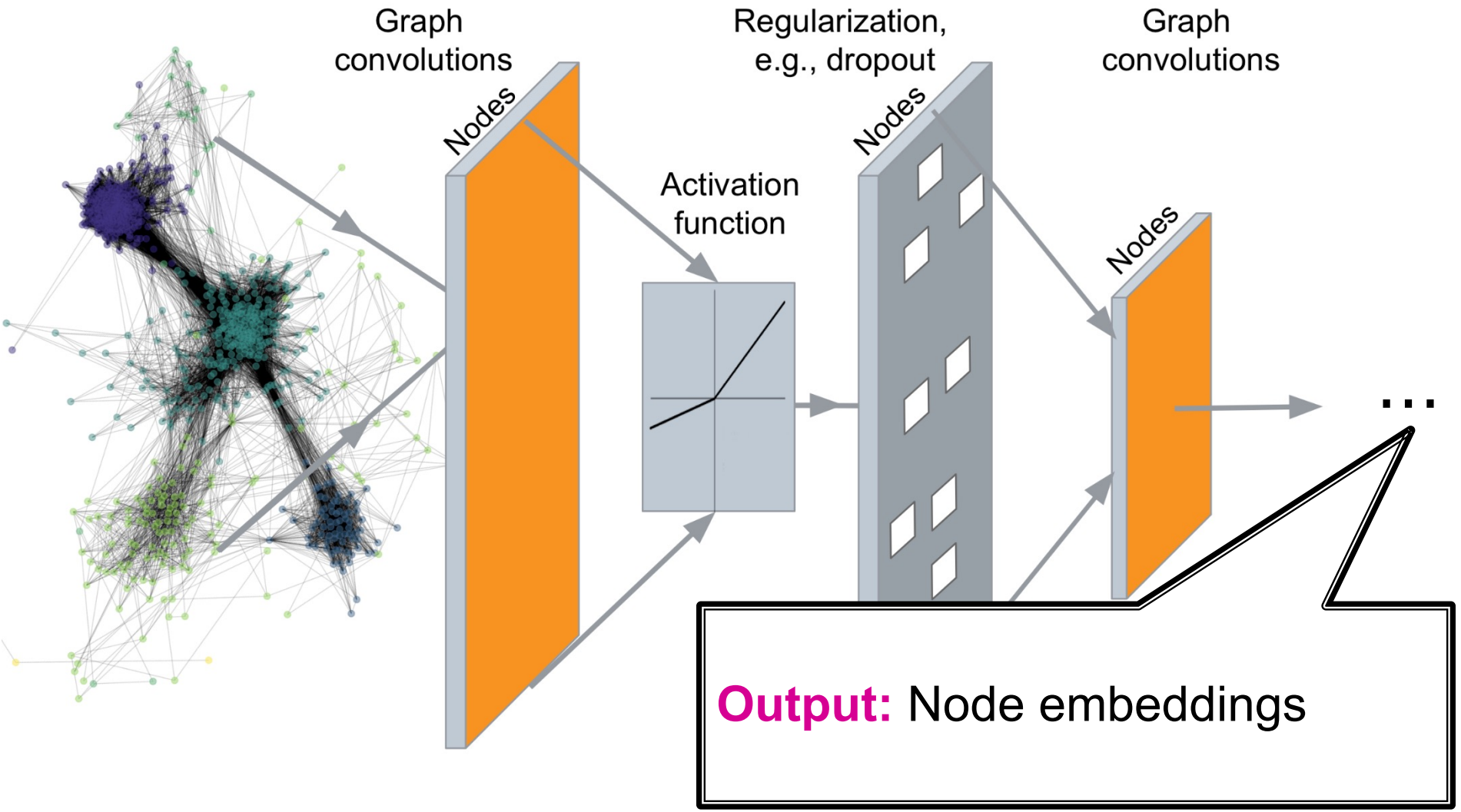
Key Concept: Node Embeddings

- **Intuition:** Map nodes to d-dimensional embeddings such that similar nodes in the graph are embedded close together



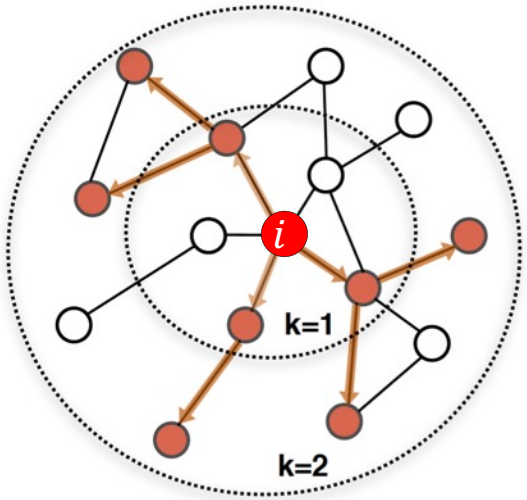
- **How to learn the encoder function $ENC(\cdot)$?**

Deep Graph Encoders

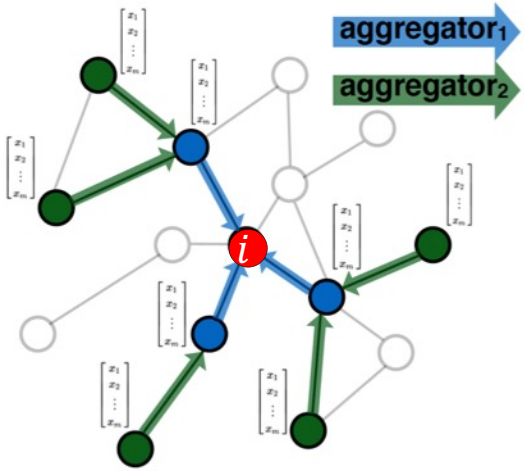


Recap: Graph Neural Networks

Idea: Node's neighborhood defines a computation graph



Determine node computation graph

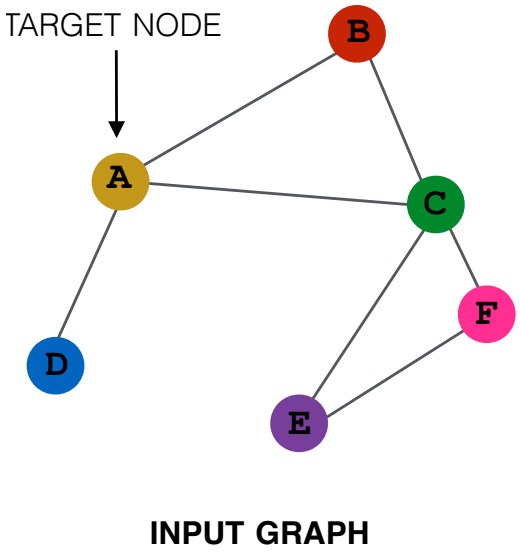


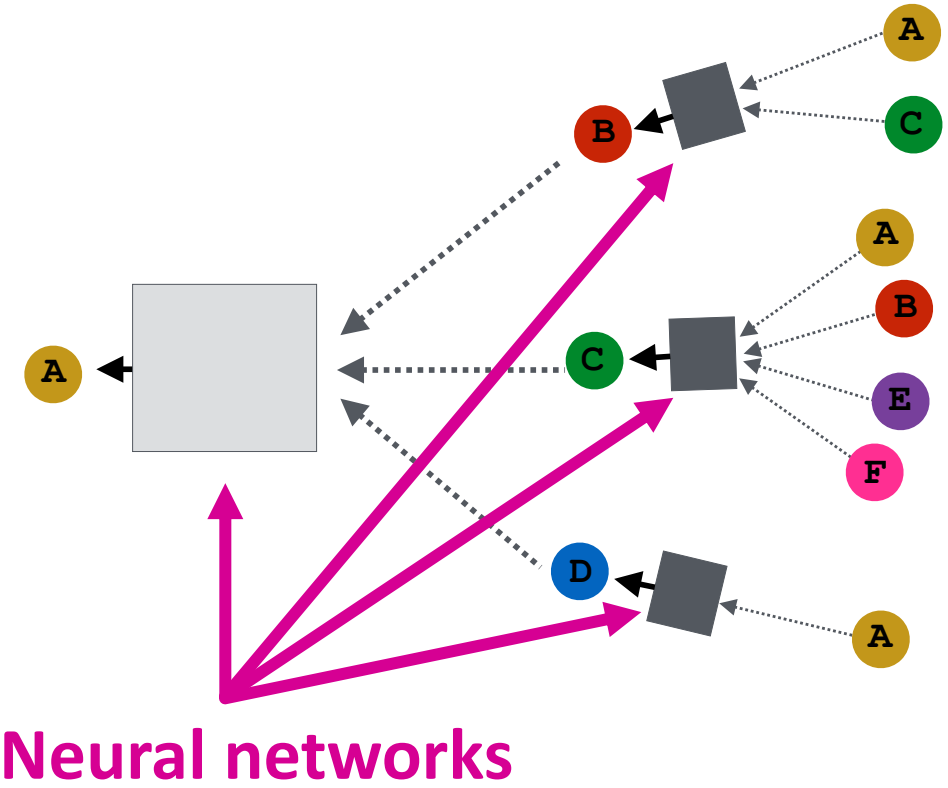
Propagate and transform information

Learn how to propagate information across the graph to compute node features

Recap: Aggregate from Neighbors

- **Intuition:** Nodes aggregate information from their neighbors using neural networks

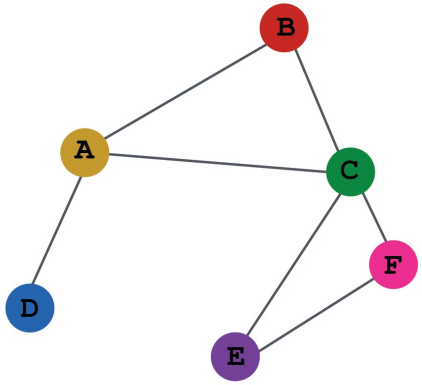




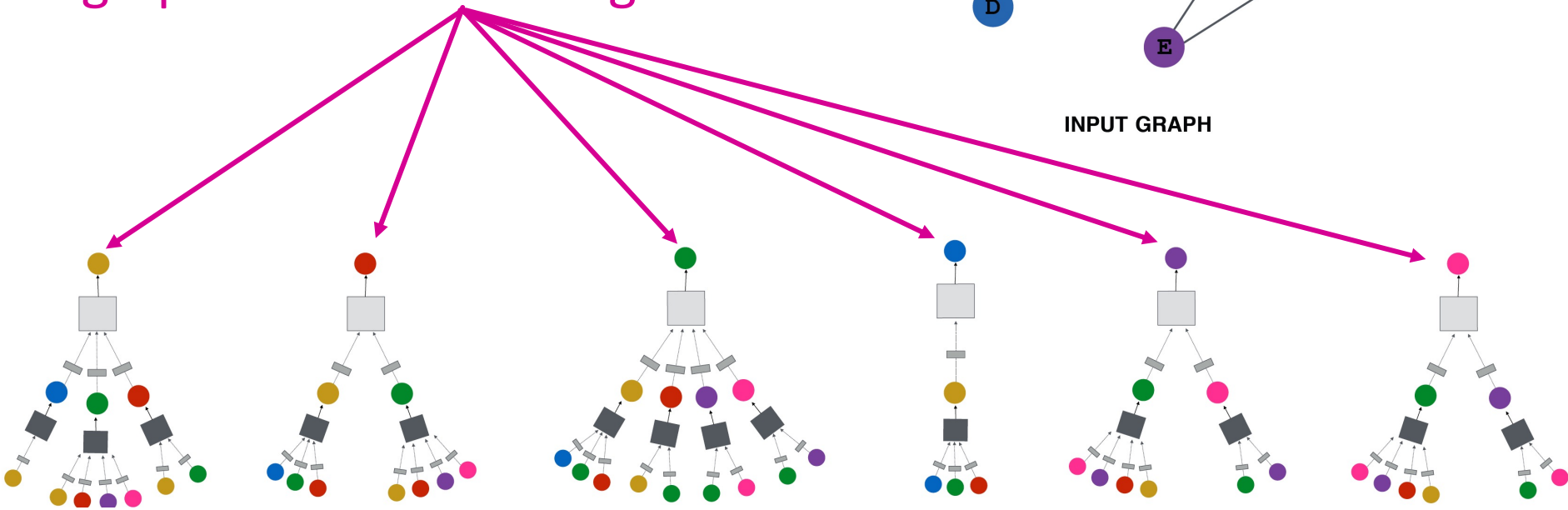
Recap: Aggregate Neighbors

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



INPUT GRAPH



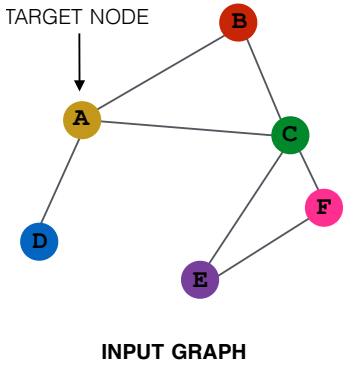
A General Perspective on Graph Neural Networks

Jiaxuan You, Stanford University

Adapted from Stanford CS 224W & CS 246

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020

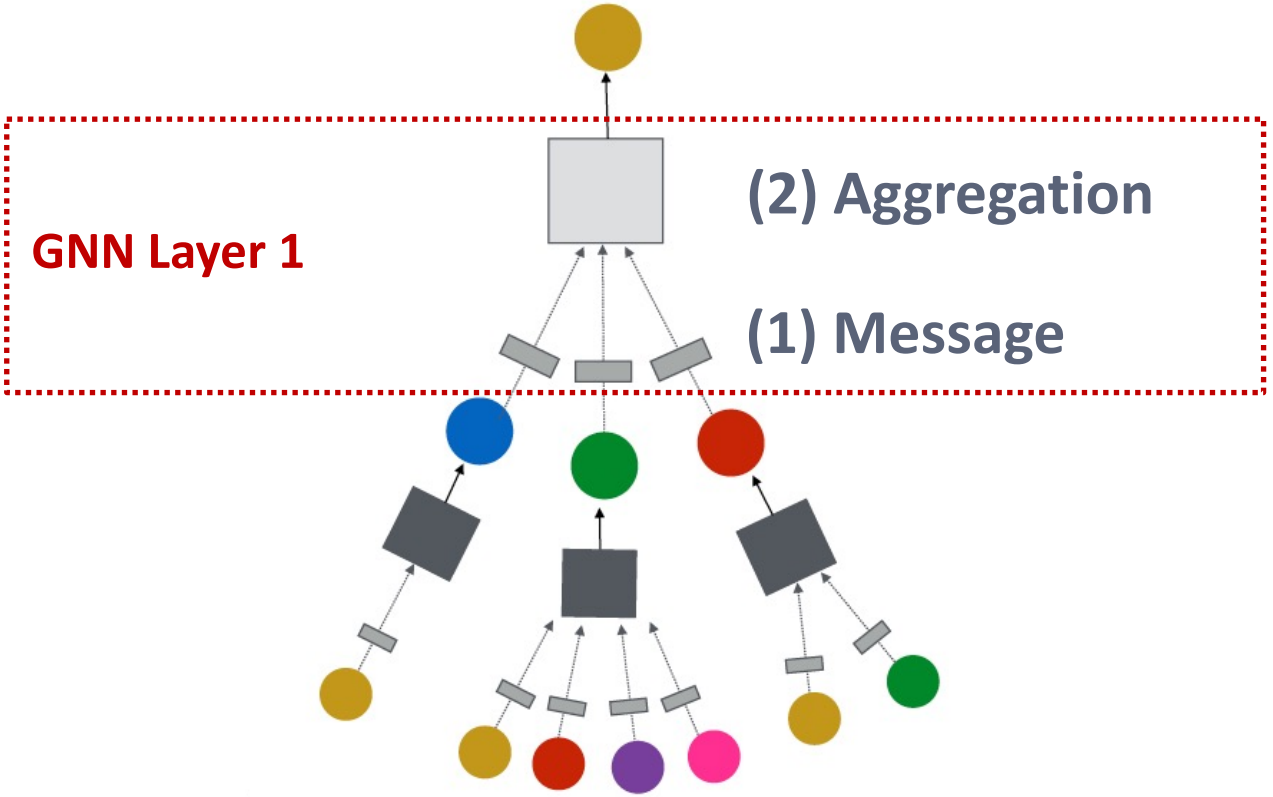
A General GNN Framework (1)



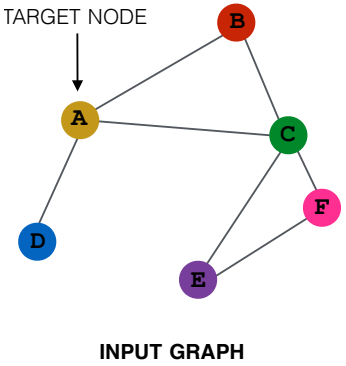
INPUT GRAPH

GNN Layer = Message + Aggregation

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



A General GNN Framework (2)

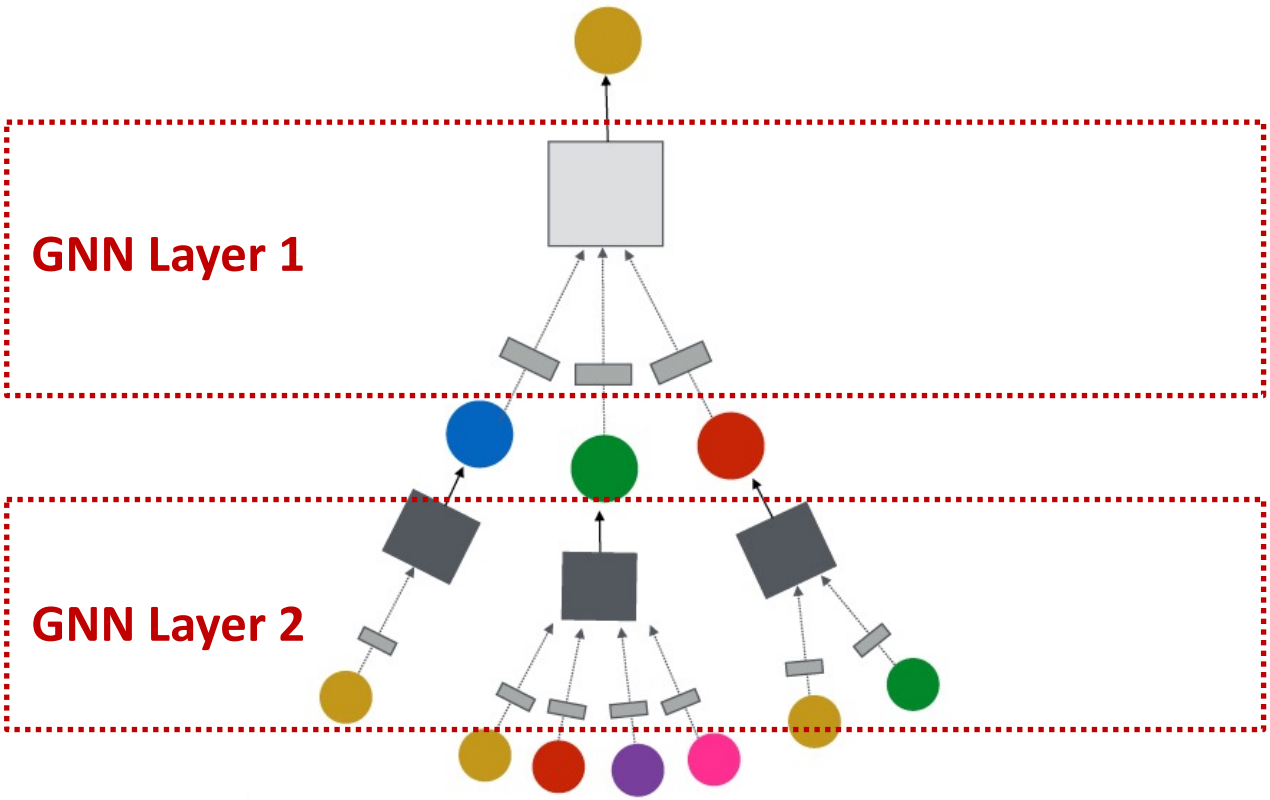


INPUT GRAPH

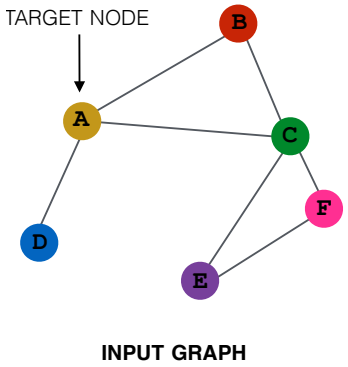
Connect GNN layers into a GNN

- Stack layers sequentially
- Ways of adding skip connections

(3) Layer connectivity



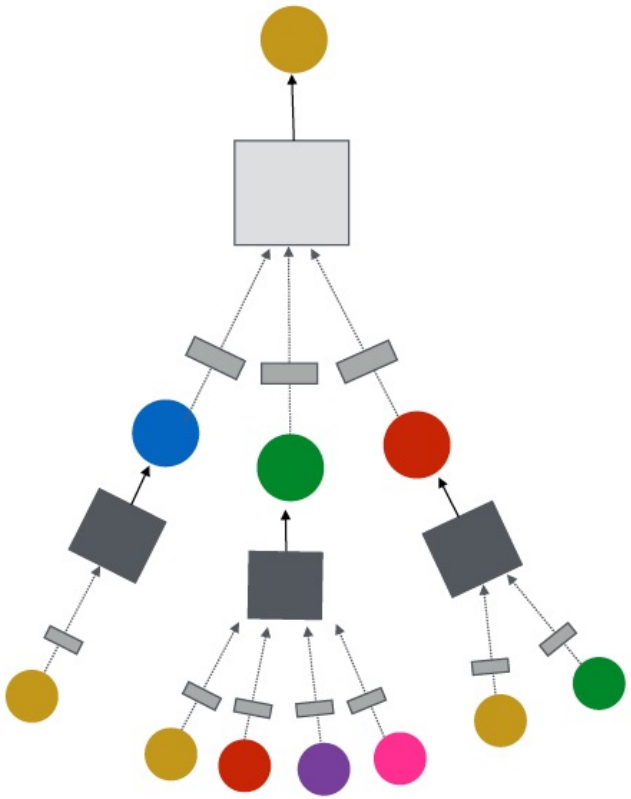
A General GNN Framework (3)



INPUT GRAPH

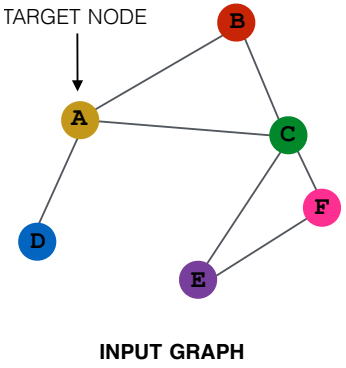
Idea: Raw input graph ≠ computational graph

- **Graph feature augmentation**
- **Graph structure augmentation**



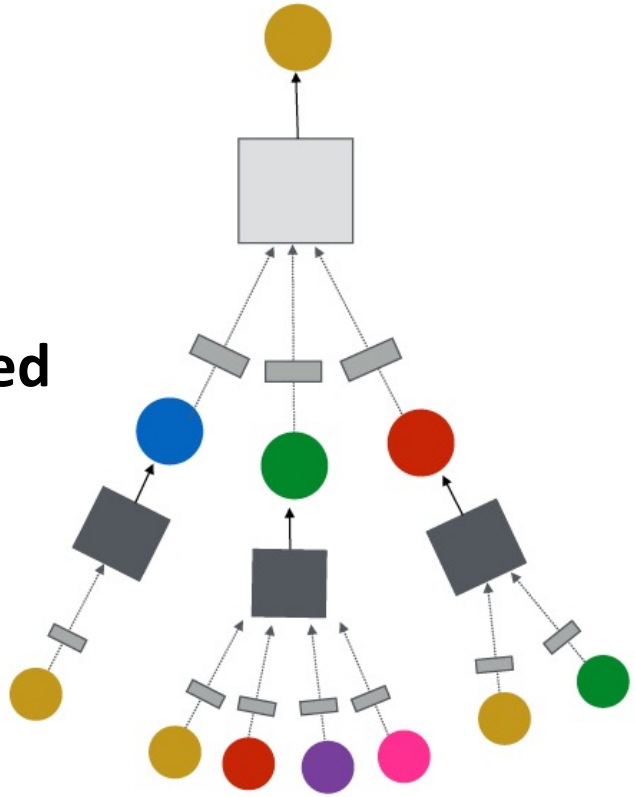
(4) Graph augmentation

A General GNN Framework (4)



INPUT GRAPH

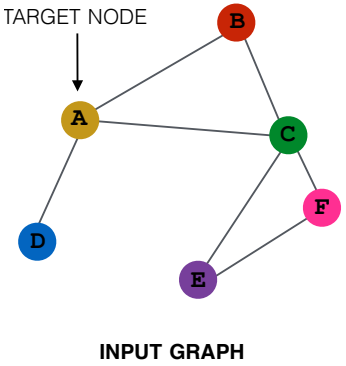
(5) Learning objective



How do we train a GNN

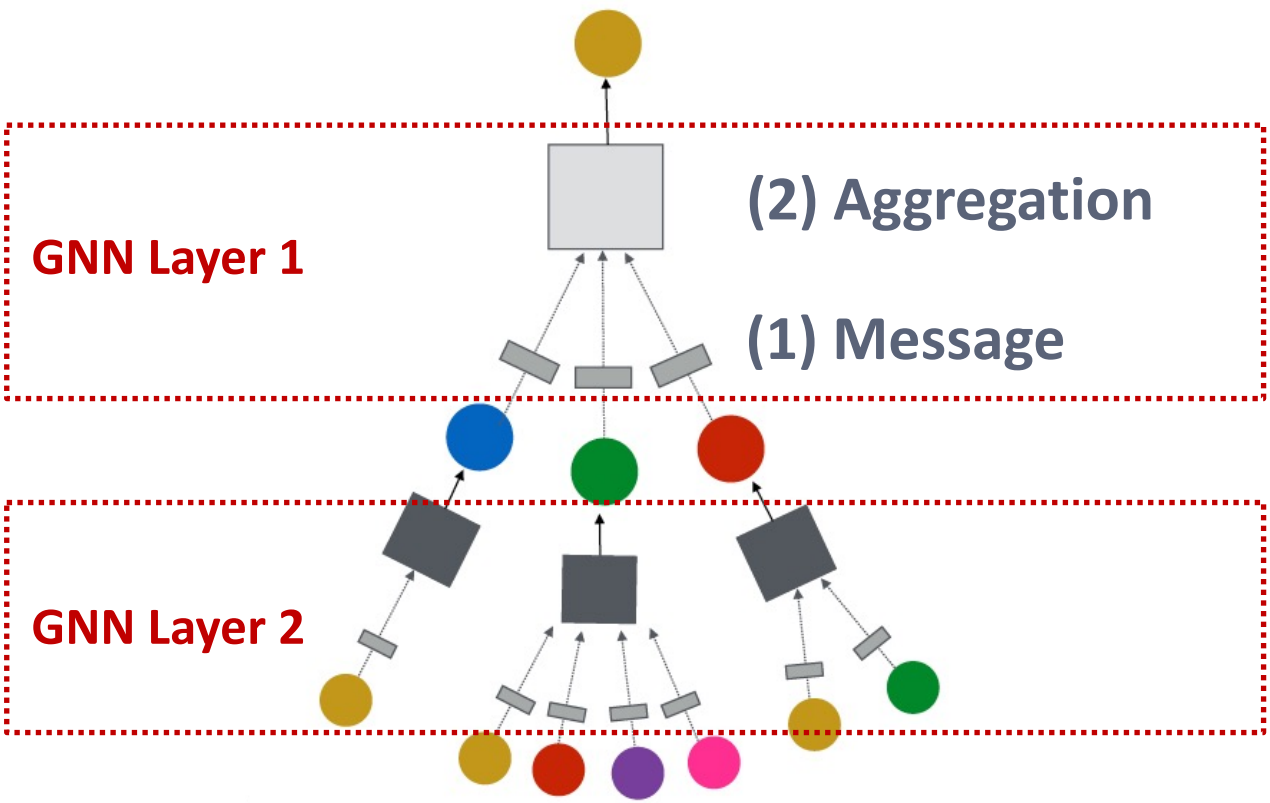
- **Supervised/Unsupervised objectives**
- **Node/Edge/Graph level objectives**

A General GNN Framework (5)



INPUT GRAPH

(5) Learning objective



(3) Layer connectivity

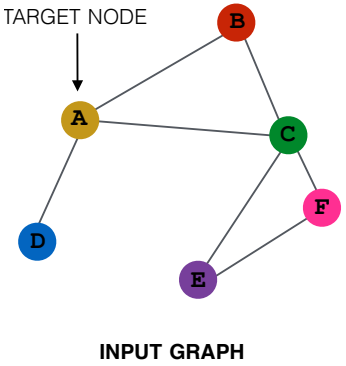
(4) Graph augmentation

A Single Layer of a GNN

Jiaxuan You, Stanford University

Adapted from Stanford CS 224W & CS 246

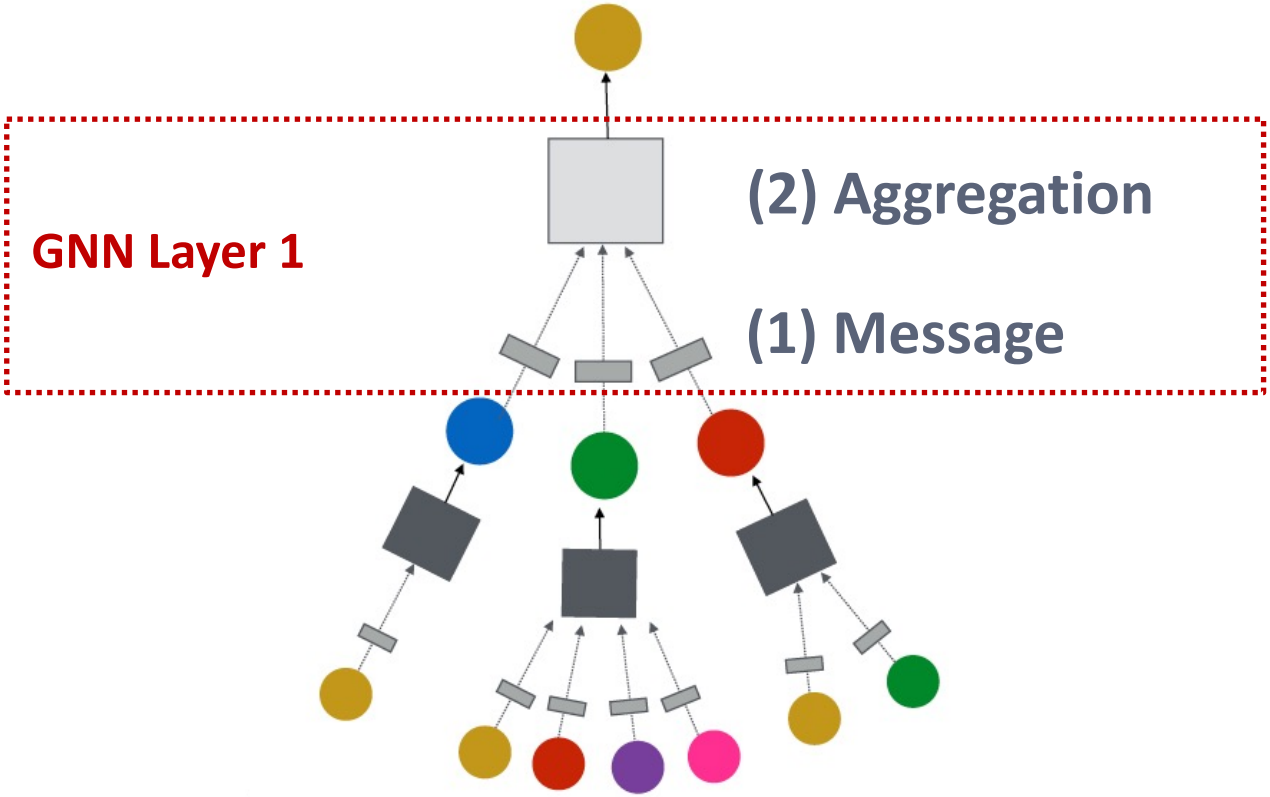
A GNN Layer



INPUT GRAPH

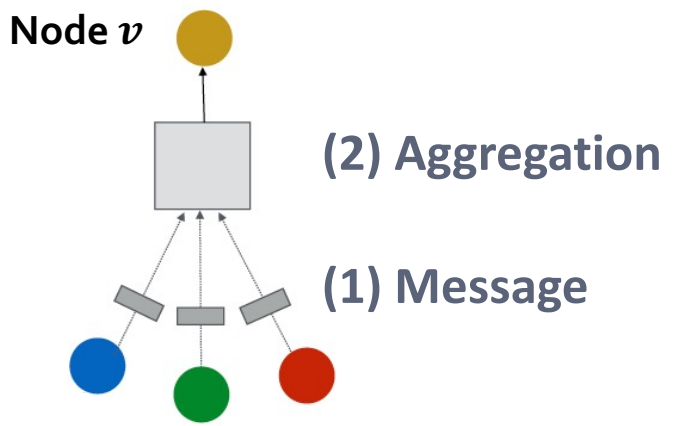
GNN Layer = Message + Aggregation

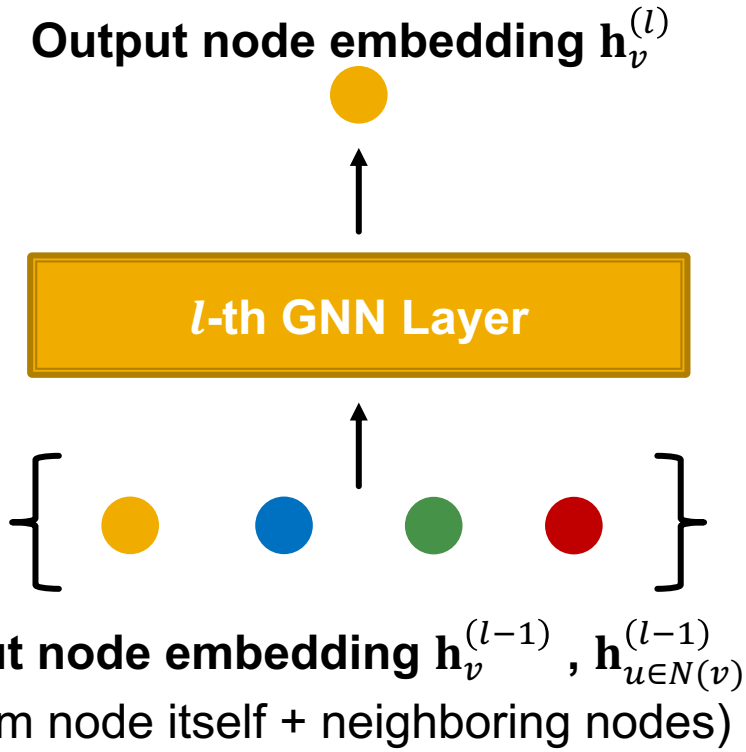
- Different instantiations under this perspective
- GCN, GraphSAGE, GAT, …



A Single GNN Layer

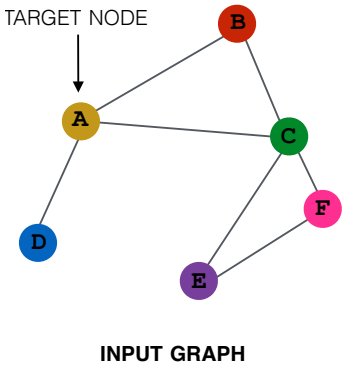
- **Idea of a GNN Layer:**
  - Compress a set of vectors into a single vector
  - **Two step process:**
    - **(1) Message**
    - **(2) Aggregation**



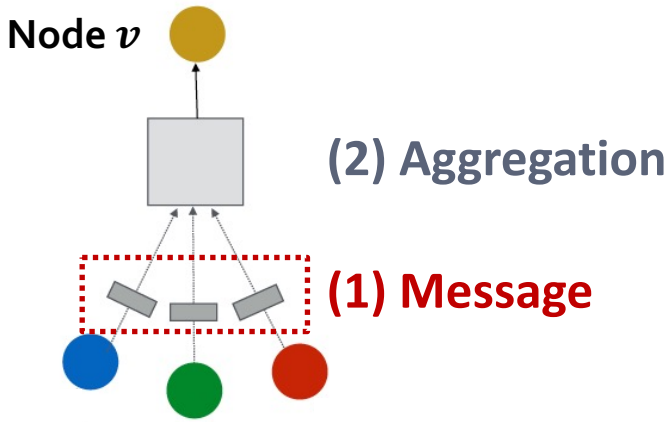


Message Computation

- **(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 $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$
      - Multiply node features with weight matrix $\mathbf{W}^{(l)}$



INPUT GRAPH

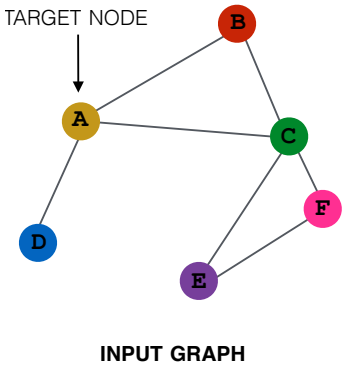


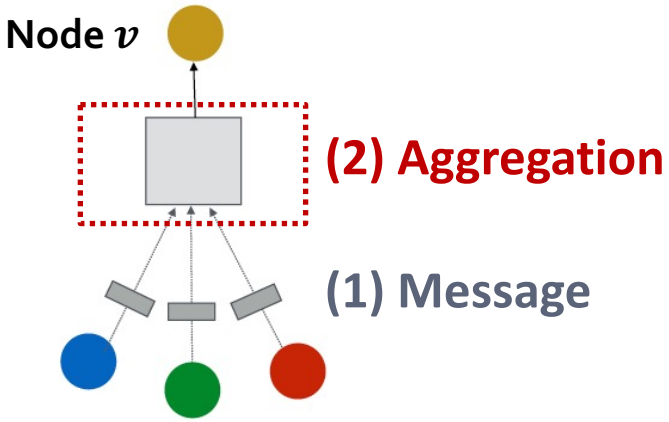
Message Aggregation

- **(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
    - $\mathbf{h}_v^{(l)} = \mathrm{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$





Message Aggregation: Issue

- **Issue:** Information from node v itself **could get lost**
  - Computation of $\mathbf{h}_v^{(l)}$ does not directly depend on $\mathbf{h}_v^{(l-1)}$
- **Solution:** Include $\mathbf{h}_v^{(l-1)}$ when computing $\mathbf{h}_v^{(l)}$
  - **(1) Message: compute message from node v itself**
    - Usually, a **different message computation** will be performed

$$\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$$

$$\mathbf{m}_v^{(l)} = \mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)}$$

  - **(2) Aggregation:** After aggregating from neighbors, we can **aggregate the message from node v itself**
    - Via **concatenation** or **summation**

Then aggregate from node itself

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

First aggregate from neighbors

A Single GNN Layer

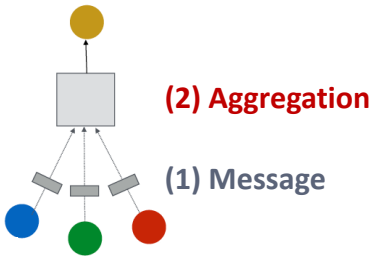
- **Putting things together:**
  - **(1) Message**: each node computes a message

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

  - **(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**



T. Kipf, M. Welling. Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

Classical GNN Layers: GCN (1)

- (1) Graph Convolutional Networks (GCN)

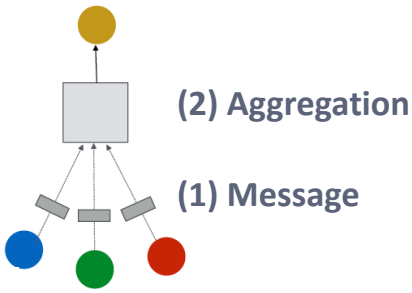
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- How to write this as Message + Aggregation?

Message

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

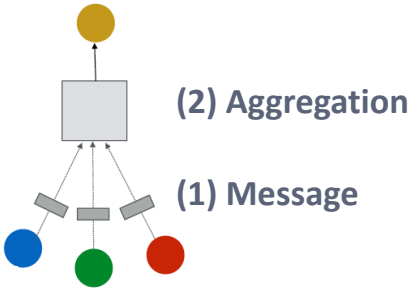
Aggregation



Classical GNN Layers: GCN (2)

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

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



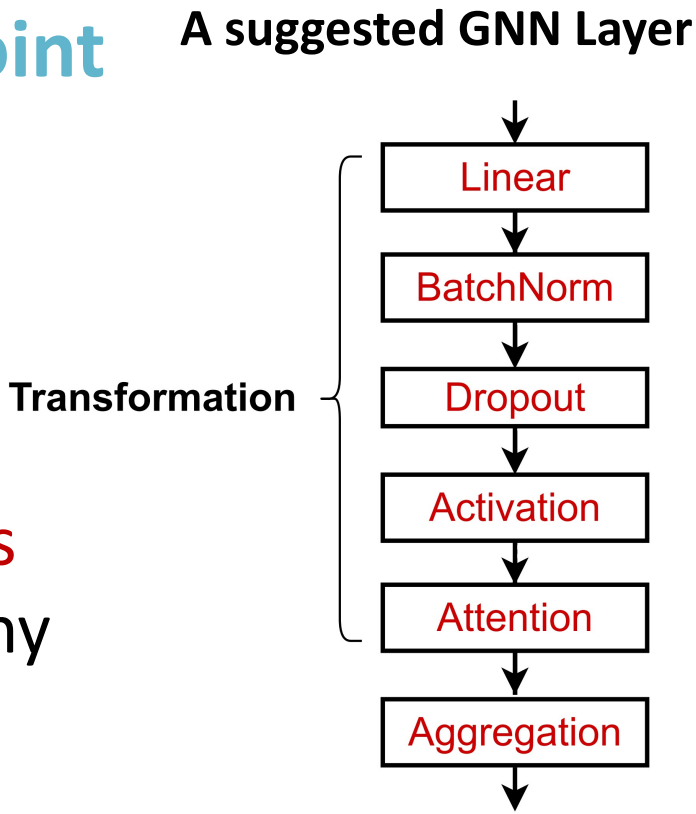
- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$

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

- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

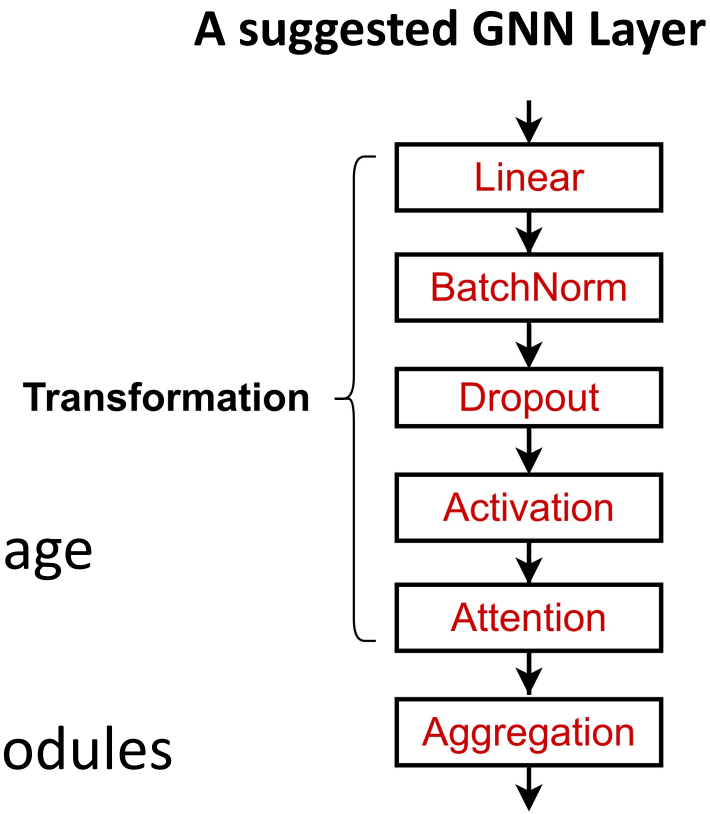
GNN Layer in Practice

- **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



GNN Layer in Practice

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



Batch Normalization

- **Goal**: Stabilize neural networks training
- **Idea**: Given a batch of inputs (node embeddings)
  - Re-center the node embeddings into zero mean
  - Re-scale the variance into unit variance

Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$
N node embeddings

Trainable Parameters:
$\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{D}$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D}$
Normalized node embeddings

Step 1:
Compute the mean and variance over N embeddings

$$\boldsymbol{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} \mathbf{X}_{i,j}$$

$$\boldsymbol{\sigma}_j^2 = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{X}_{i,j} - \boldsymbol{\mu}_j)^2$$

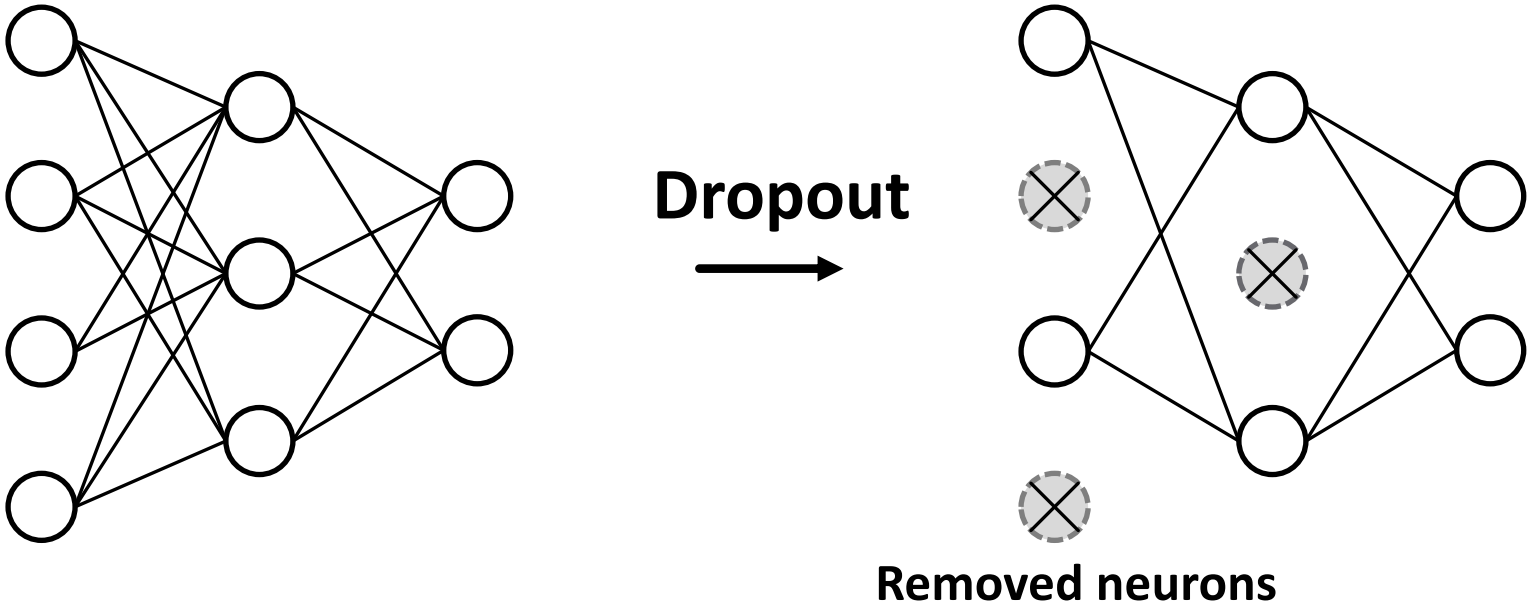
Step 2:
Normalize the feature using computed mean and variance

$$\widehat{\mathbf{X}}_{i,j} = \frac{\mathbf{X}_{i,j} - \boldsymbol{\mu}_j}{\sqrt{\boldsymbol{\sigma}_j^2 + \epsilon}}$$

$$\mathbf{Y}_{i,j} = \boldsymbol{\gamma}_j \widehat{\mathbf{X}}_{i,j} + \boldsymbol{\beta}_j$$

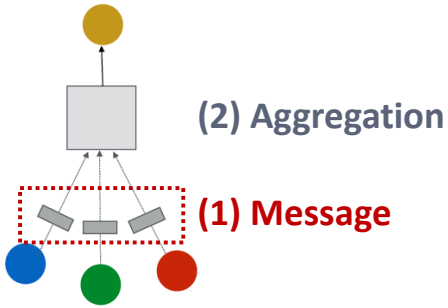
Dropout

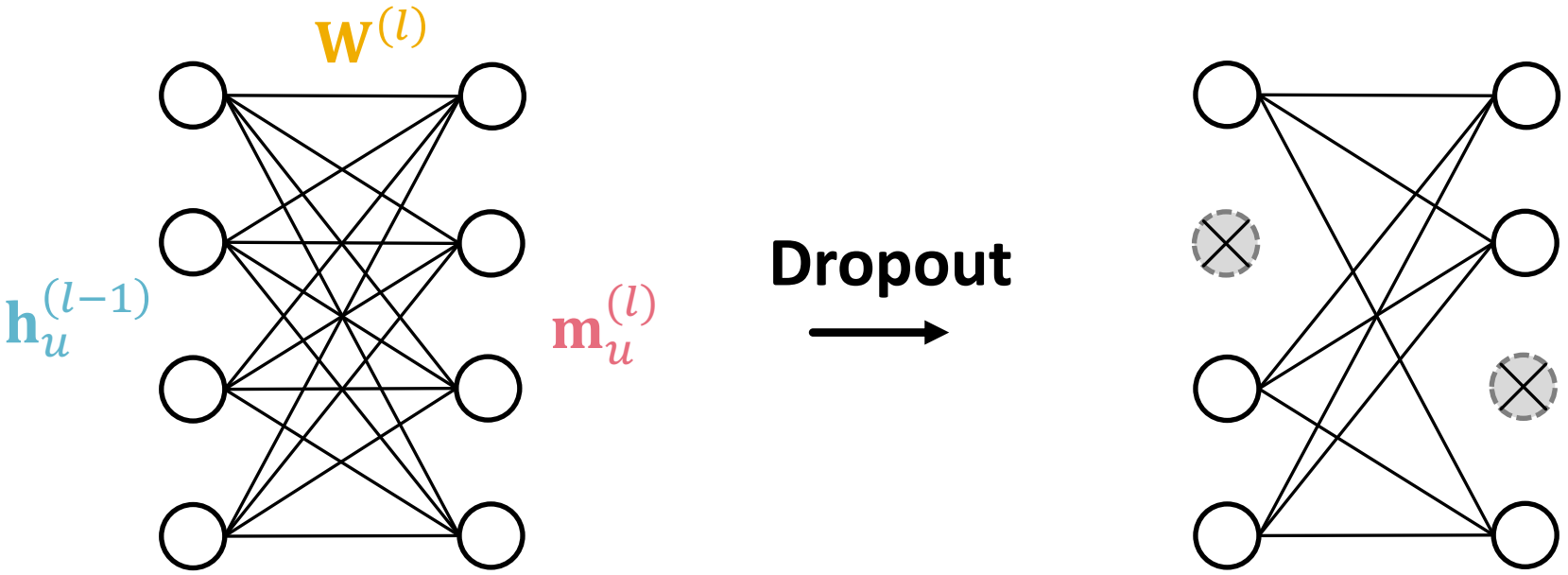
- **Goal**: Regularize a neural net to prevent overfitting.
- **Idea**:
  - **During training**: with some probability p, randomly set neurons to zero (turn off)
  - **During testing:** Use all the neurons for computation



Dropout for GNNs

- In GNN, Dropout is applied to **the linear layer in the message function**
  - **A simple message function with linear layer:** $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$





Visualization of a linear layer

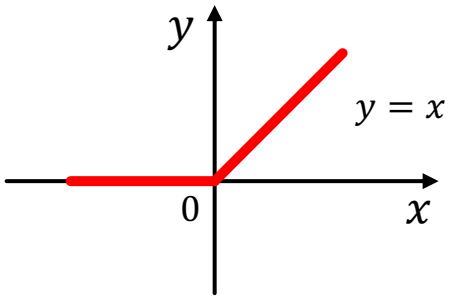
Activation (Non-linearity)

Apply activation to i-th dimension of embedding $\mathbf{x}$

- **Rectified linear unit (ReLU)**

  $$\mathrm{ReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0)$$

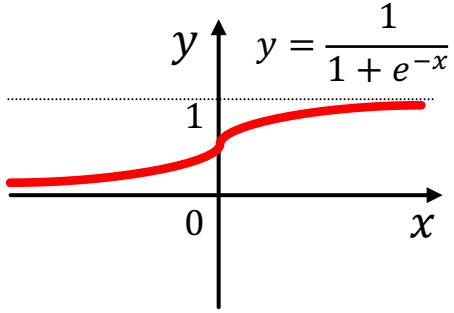
  - Most commonly used



- **Sigmoid**

  $$\sigma(\mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{x}_i}}$$

  - Used only when you want to restrict the range of your embeddings

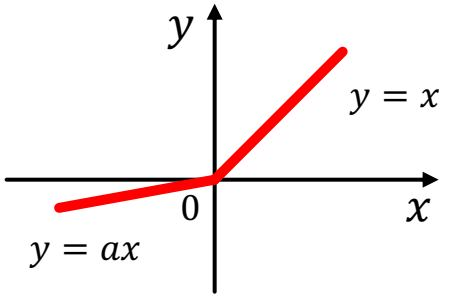


- **Parametric ReLU**

  $$\mathrm{PReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0) + a_i \min(\mathbf{x}_i, 0)$$

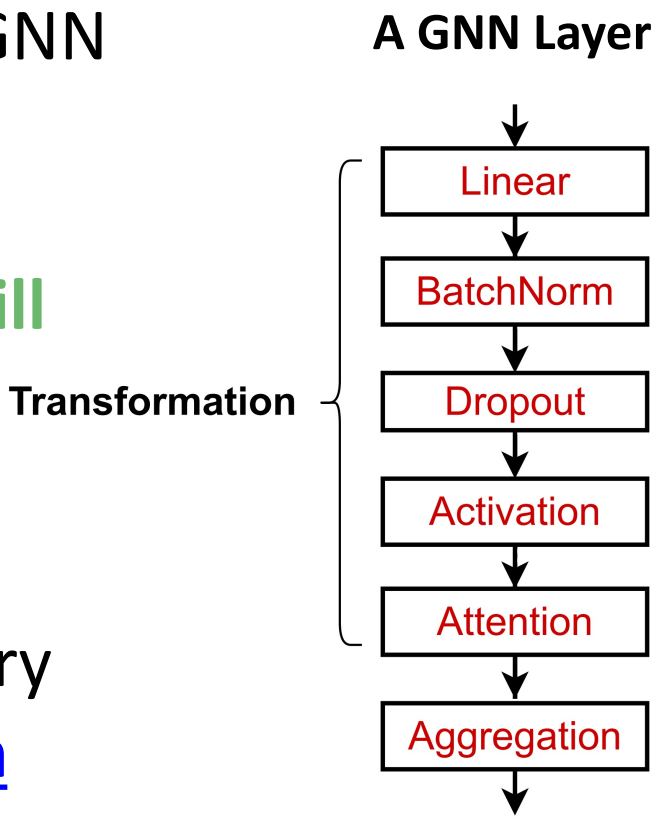
  a_i is a trainable parameter

  - Empirically performs better than ReLU



GNN Layer in Practice

- **Summary:** Modern deep learning modules can be included into a GNN layer for better performance
- **Designing novel GNN layers is still an active research frontier!**
- **Suggested resources:** You can explore diverse GNN designs or try out your own ideas in **GraphGym**

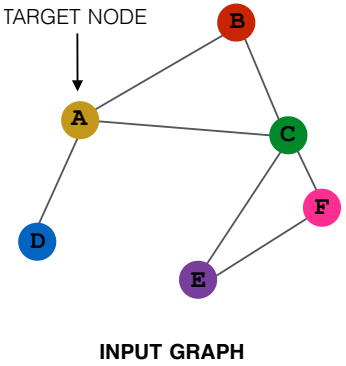


Stacking Layers of a GNN

Jiaxuan You, Stanford University

Adapted from Stanford CS 224W & CS 246

Stacking GNN Layers

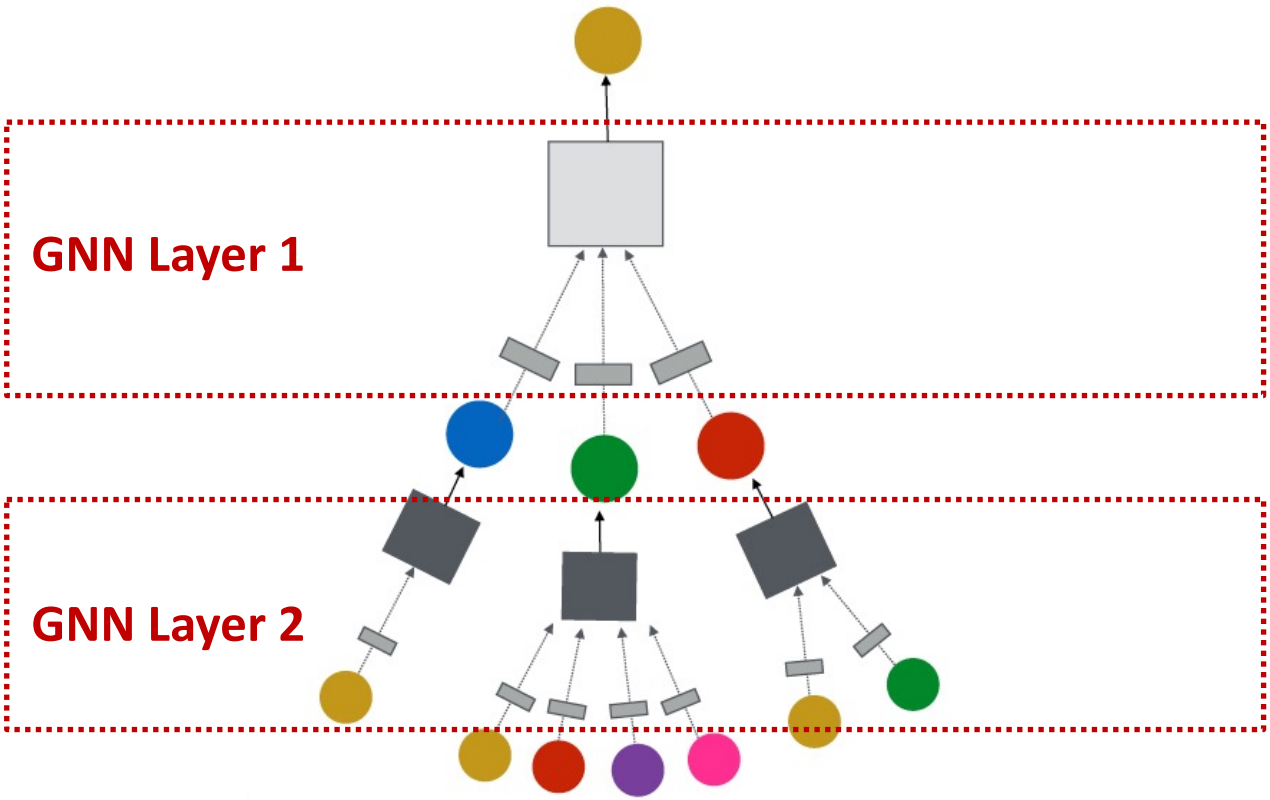


INPUT GRAPH

How to connect GNN layers into a GNN?

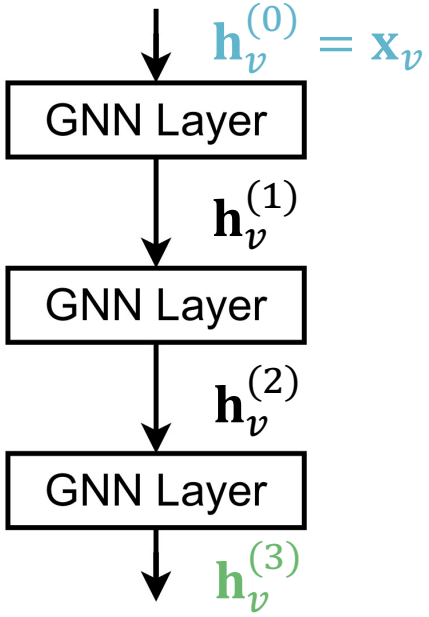
- **Stack layers sequentially**
- **Ways of adding skip connections**

(3) Layer connectivity



Stacking GNN Layers

- **How to construct a Graph Neural Network?**
  - **The standard way:** Stack GNN layers sequentially
  - **Input:** Initial raw node feature $\mathbf{x}_v$
  - **Output:** Node embeddings $\mathbf{h}_v^{(L)}$ after L GNN layers



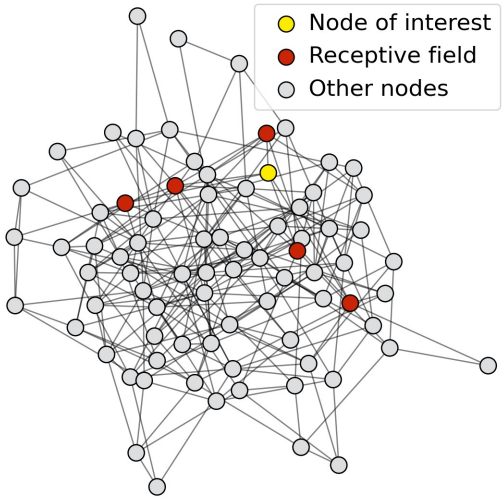
The Over-smoothing Problem

- **The Issue of stacking many GNN layers**
  - GNN suffers from **the over-smoothing problem**
- **The over-smoothing problem: all the node embeddings converge to the same value**
  - This is bad because we **want to use node embeddings to differentiate nodes**
- **Why does the over-smoothing problem happen?**

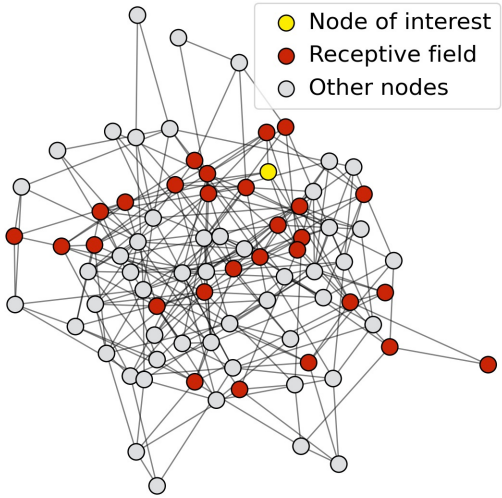
Receptive Field of a GNN

- **Receptive field:** the set of nodes that determine the embedding of a node of interest
  - **In a K-layer GNN, each node has a receptive field of K-hop neighborhood**

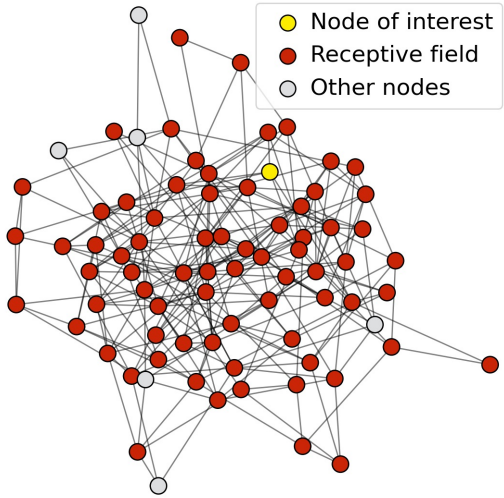
Receptive field for 1-layer GNN



Receptive field for 2-layer GNN



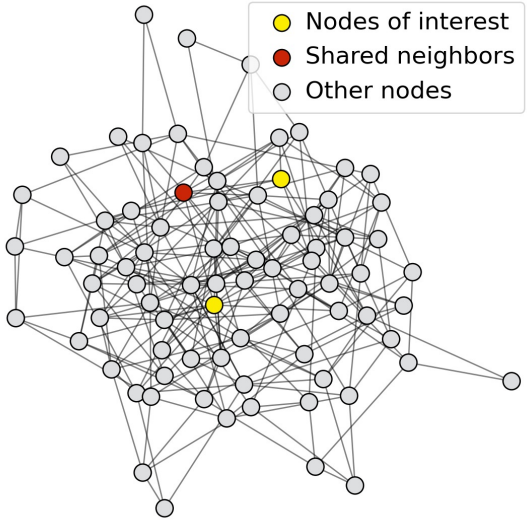
Receptive field for 3-layer GNN



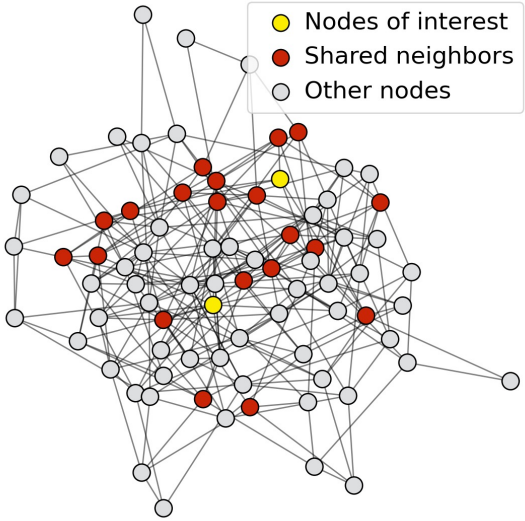
Receptive Field of a GNN

- **Receptive field overlap** for two nodes
  - **The shared neighbors quickly grows** when we increase the number of hops (num of GNN layers)

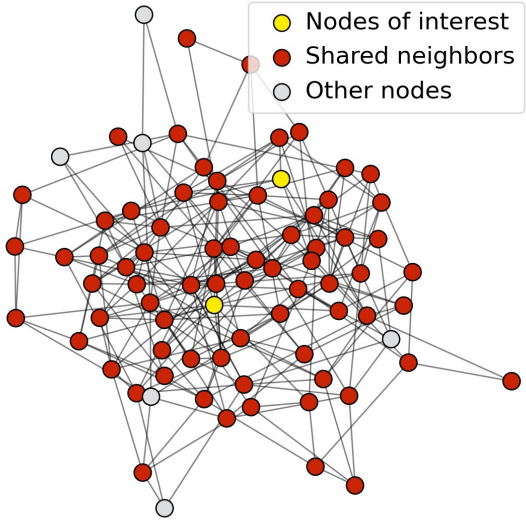
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



3-hop neighbor overlap
Almost all the nodes!



Receptive Field & Over-smoothing

- **We can explain over-smoothing via the notion of receptive field**
  - We knew **the embedding of a node is determined by its receptive field**
    - If two nodes **have highly-overlapped receptive fields, then their embeddings are highly similar**
  - **Stack many GNN layers → nodes will have highly-overlapped receptive fields → node embeddings will be highly similar → suffer from the over-smoothing problem**
- **Next:** how do we overcome over-smoothing problem?

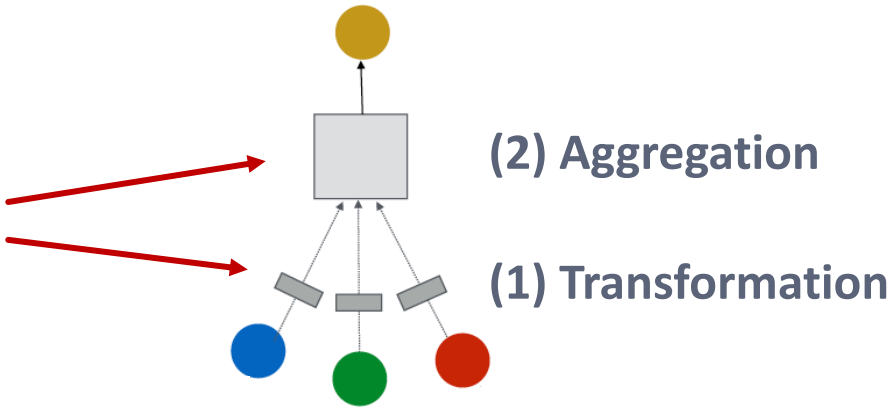
Design GNN Layer Connectivity

- **What do we learn from the over-smoothing problem?**
- **Lesson 1: Be cautious when adding GNN layers**
  - Unlike neural networks in other domains (CNN for image classification), **adding more GNN layers do not always help**
  - **Step 1: Analyze the necessary receptive field** to solve your problem. E.g., by computing the diameter of the graph
  - **Step 2:** Set number of GNN layers L to be a bit more than the receptive field we like. **Do not set L to be unnecessarily large**!
- **Question:** How to enhance the expressive power of a GNN, if the number of GNN layers is small?

Expressive Power for Shallow GNNs

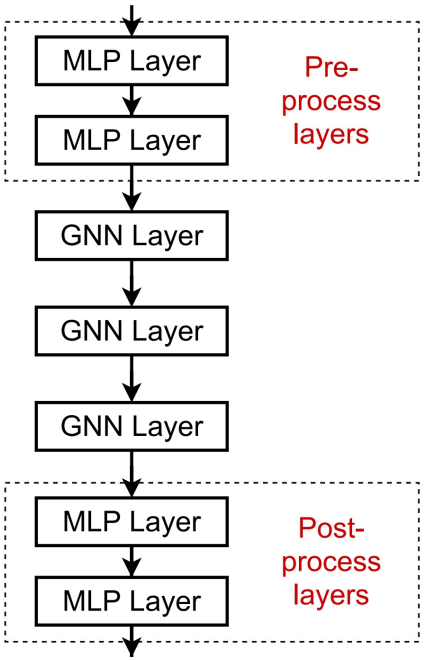
- **How to make a shallow GNN more expressive?**
- **Solution 1:** Increase the expressive power **within each GNN layer**
  - In our previous examples, each transformation or aggregation function only include one linear layer
  - We can **make aggregation / transformation become a deep neural network**!

If needed, each box could include a 3-layer MLP



Expressive Power for Shallow GNNs

- **How to make a shallow GNN more expressive?**
- **Solution 2:** Add layers that do not pass messages
  - A GNN does not necessarily only contain GNN layers
    - E.g., we can add **MLP layers** (applied to each node) before and after GNN layers, as **pre-process layers** and **post-process layers**



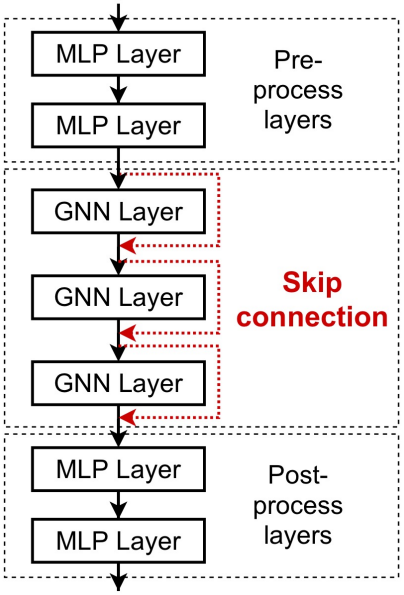
Pre-processing layers: Important when encoding node features is necessary.
E.g., when nodes represent images/text

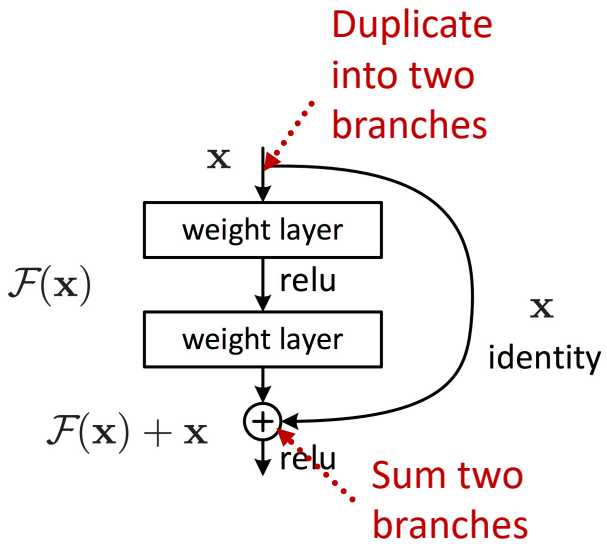
Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

In practice, adding these layers works great!

Design GNN Layer Connectivity

- **What if my problem still requires many GNN layers?**
- **Lesson 2: Add skip connections in GNNs**
  - **Observation from over-smoothing:** Node embeddings in earlier GNN layers can sometimes better differentiate nodes
  - **Solution:** We can increase the impact of earlier layers on the final node embeddings, **by adding shortcuts in GNN**





Idea of skip connections:

Before adding shortcuts:

$$\boldsymbol{F}(\mathbf{x})$$

After adding shortcuts:

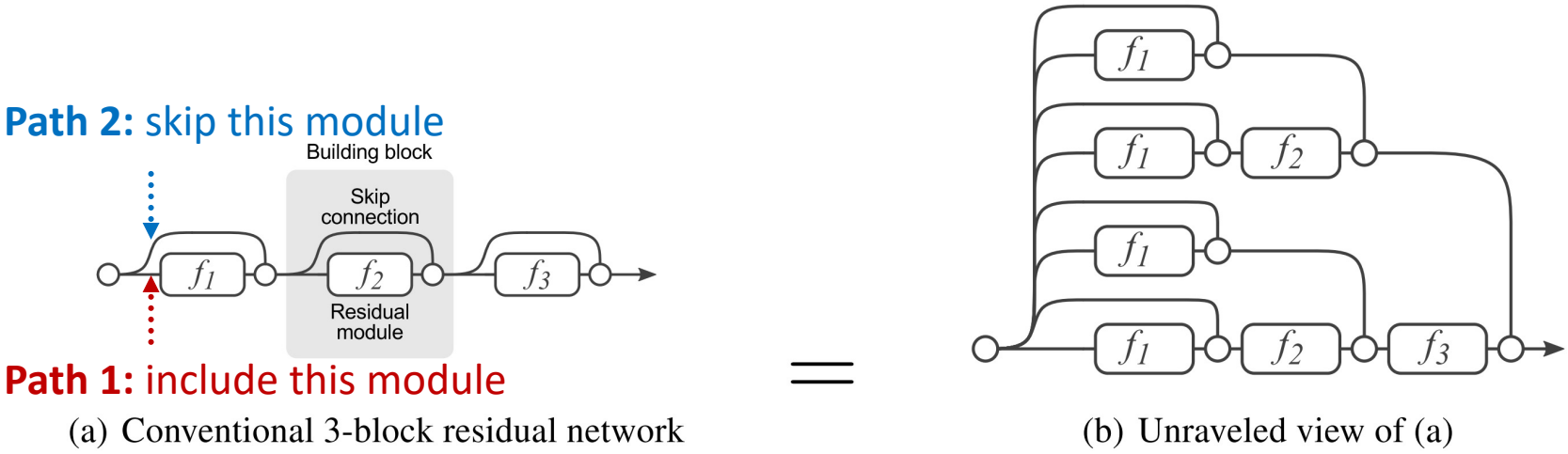
$$\boldsymbol{F}(\mathbf{x}) + \mathbf{x}$$

Idea of Skip Connections

- **Why do skip connections work?**
  - **Intuition:** Skip connections create **a mixture of models**
  - N skip connections → 2^N possible paths
  - Each path could have up to N modules
  - We automatically get **a mixture of shallow GNNs and deep GNNs**

All the possible paths:

$2 * 2 * 2 = 2^3 = 8$



(a) Conventional 3-block residual network

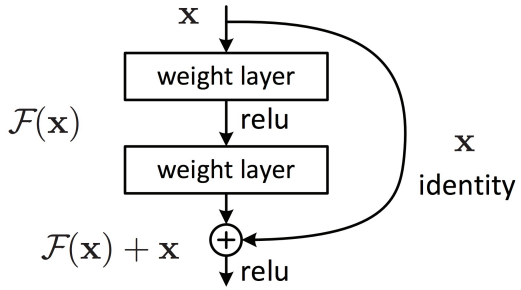
(b) Unraveled view of (a)

Veit et al. Residual Networks Behave Like Ensembles of Relatively Shallow Networks, ArXiv 2016

Example: GCN with Skip Connections

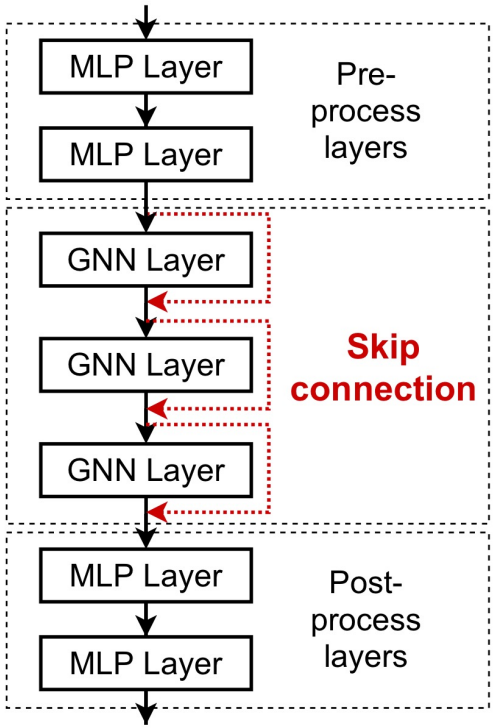
- **A standard GCN layer**
- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$

This is our $F(\mathbf{x})$



- **A GCN layer with skip connection**
- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} + \mathbf{h}_v^{(l-1)}\right)$

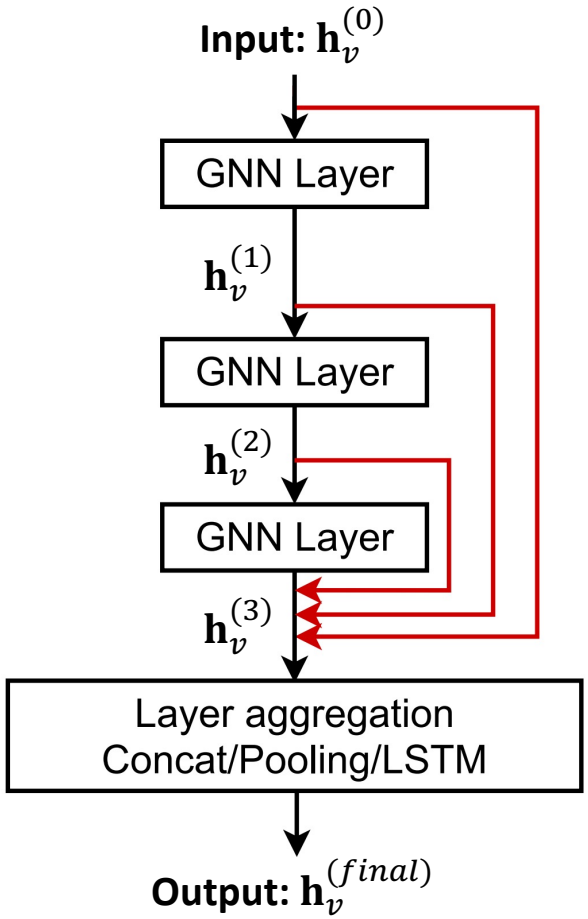
$F(\mathbf{x})$ + $\mathbf{x}$



Xu et al. Representation learning on graphs with jumping knowledge networks, ICML 2018

Other Options of Skip Connections

- **Other options:** Directly skip to the last layer
  - The final layer directly **aggregates from the all the node embeddings** in the previous layers

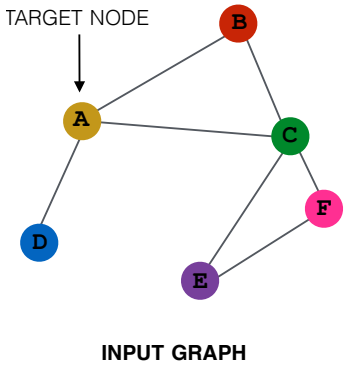


Graph Manipulation in GNNs

Jiaxuan You, Stanford University

Adapted from Stanford CS 224W & CS 246

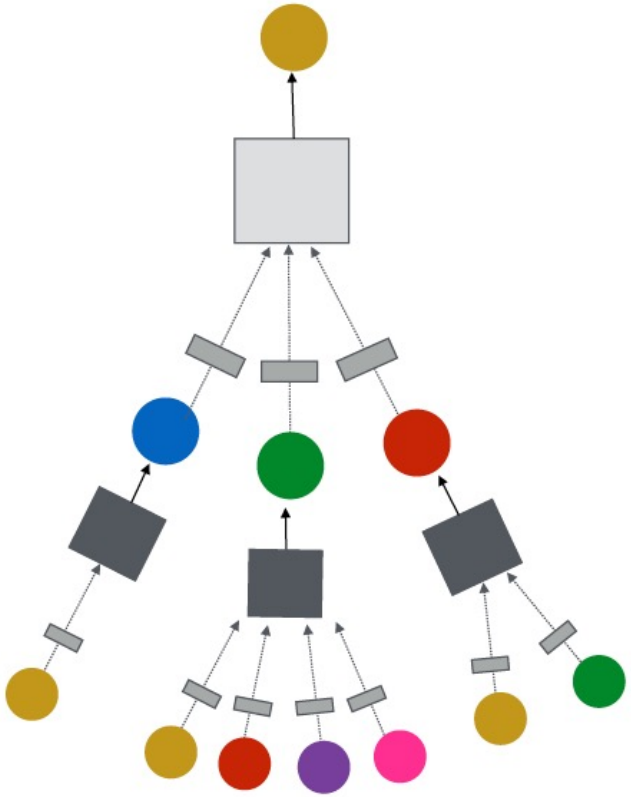
General GNN Framework



INPUT GRAPH

Idea: Raw input graph ≠ computational graph

- **Graph feature augmentation**
- **Graph structure manipulation**



(4) Graph manipulation

Why Manipulate Graphs

Our assumption so far has been

- **Raw input graph = computational graph**

Reasons for breaking this assumption

- **Feature level:**
  - The input graph **lacks features** → feature augmentation
- **Structure level:**
  - The graph is **too sparse** → inefficient message passing
  - The graph is **too dense** → message passing is too costly
  - The graph is **too large** → cannot fit the computational graph into a GPU
- It's just **unlikely that the input graph happens to be the optimal computation graph** for embeddings

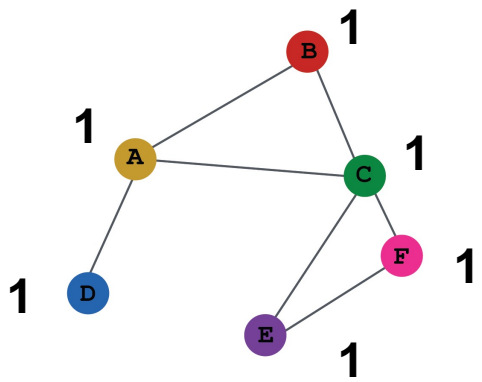
Graph Manipulation Approaches

- **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

Feature Augmentation on Graphs

Why do we need feature augmentation?

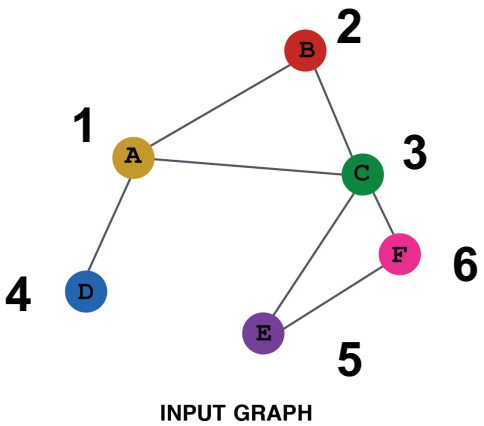
- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **a) Assign constant values to nodes**



Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **b) Assign unique IDs to nodes**
  - These IDs are converted into **one-hot vectors**



One-hot vector for node with ID=5

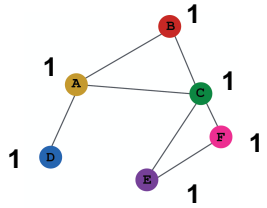
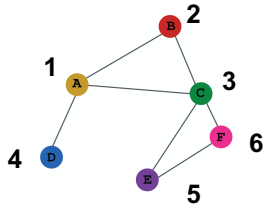
ID = 5

[0, 0, 0, 0, 1, 0]

Total number of IDs = 6

Feature Augmentation on Graphs

- Feature augmentation: **constant** vs. **one-hot**

	Constant node feature	**One-hot node feature**
		
Expressive power	**Medium**. All the nodes are identical, but GNN can still learn from the graph structure	**High**. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	**High**. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	**Low**. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	**Low**. Only 1 dimensional feature	**High**. $O(\|V\|)$ dimensional feature, cannot apply to large graphs
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

Feature Augmentation on Graphs

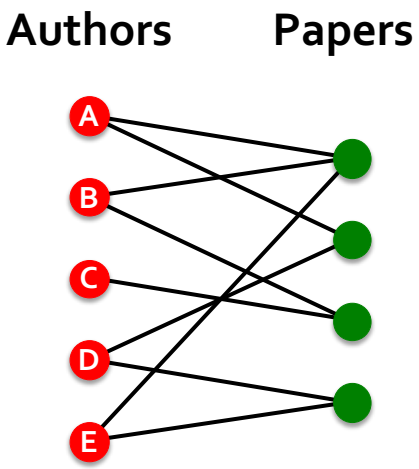
Why do we need feature augmentation?

- **(2) Certain features can help GNN learning**
- Other commonly used augmented features:
  - **Node degree**
  - **PageRank**
  - **Clustering coefficient**
  - **…**
- Any **useful graph statistics can be used!**

Add Virtual Nodes / Edges

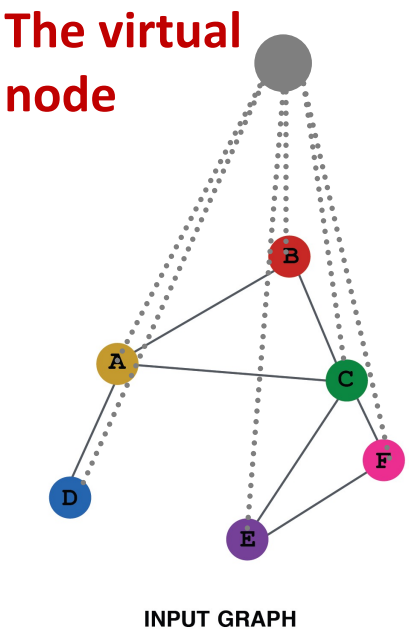
- **Motivation:** Augment sparse graphs
- **(1) Add virtual edges**
  - **Common approach:** Connect 2-hop neighbors via virtual edges
  - **Intuition:** Instead of using adj. matrix A for GNN computation, use $A + A^2$

- **Use cases:** Bipartite graphs
  - Author-to-papers (they authored)
  - 2-hop virtual edges make an author-author collaboration graph



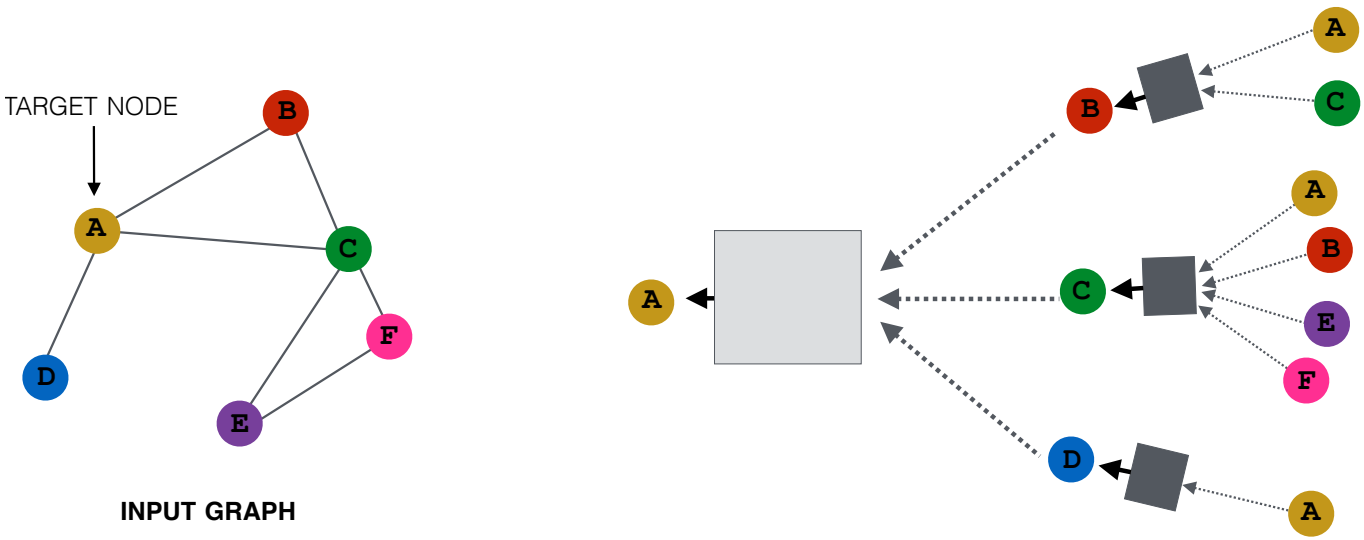
Add Virtual Nodes / Edges

- **Motivation:** Augment sparse graphs
- **(2) Add virtual nodes**
  - The virtual node will connect to all the nodes in the graph
    - Suppose in a sparse graph, two nodes have shortest path distance of 10
    - After adding the virtual node, **all the nodes will have a distance of 2**
      - **Node A – Virtual node – Node B**
  - **Benefits:** Greatly **improves message passing in sparse graphs**



Node Neighborhood Sampling

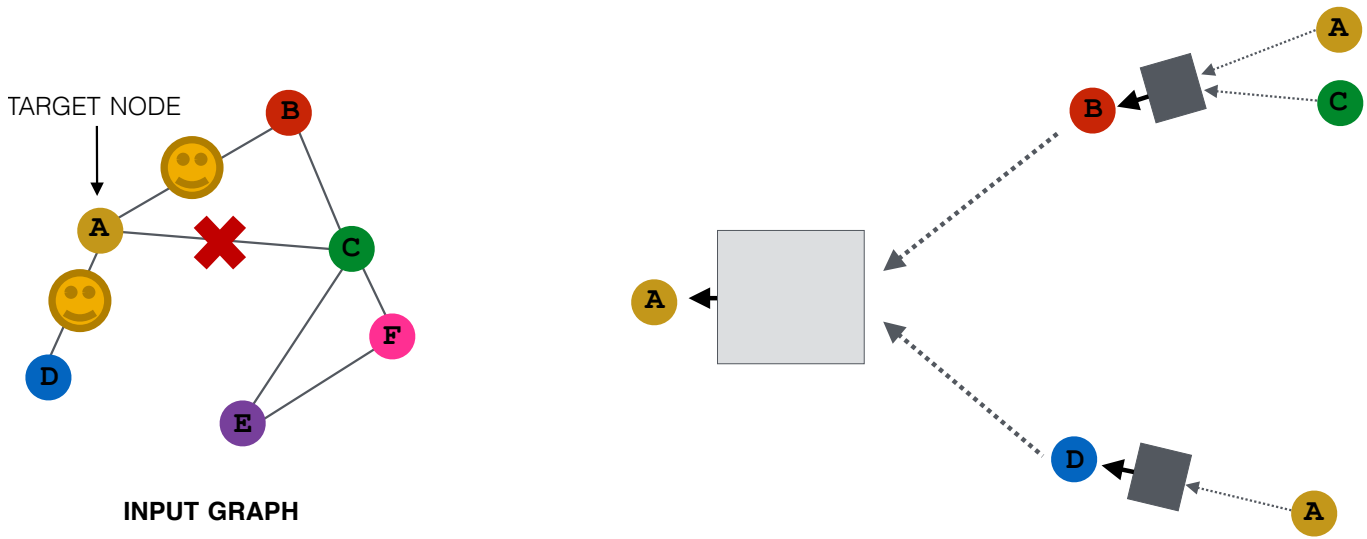
- **Previously:**
  - All the nodes are used for message passing



- **New idea:** (Randomly) sample a node's neighborhood for message passing

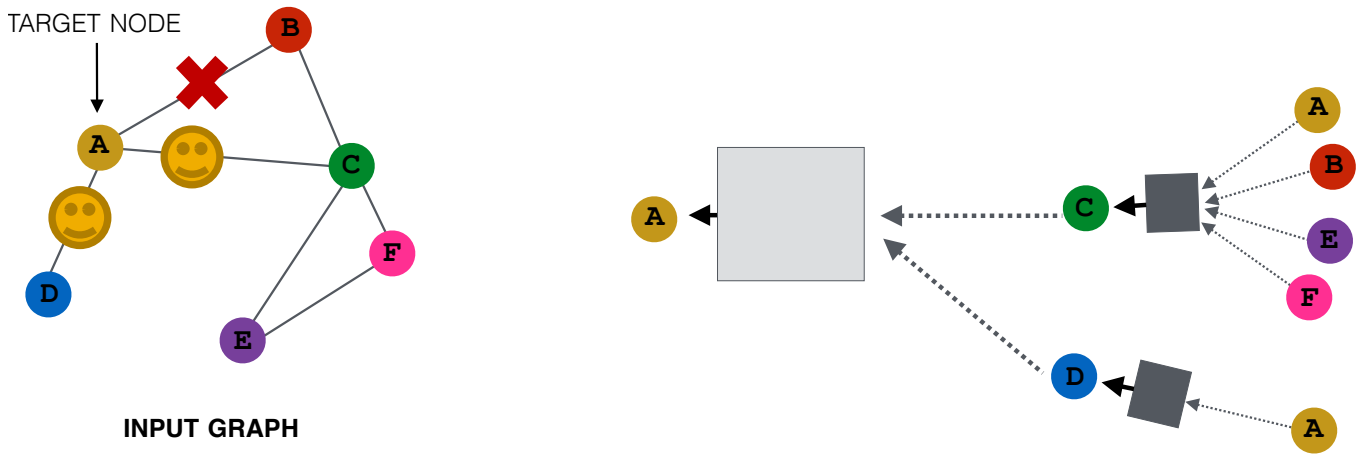
Neighborhood Sampling Example

- **For example, we can randomly choose 2 neighbors to pass messages**
  - Only nodes B and D will pass message to A



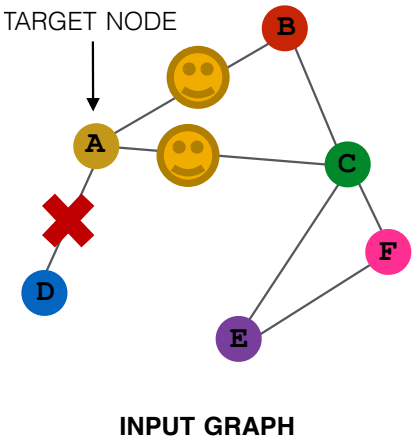
Neighborhood Sampling Example

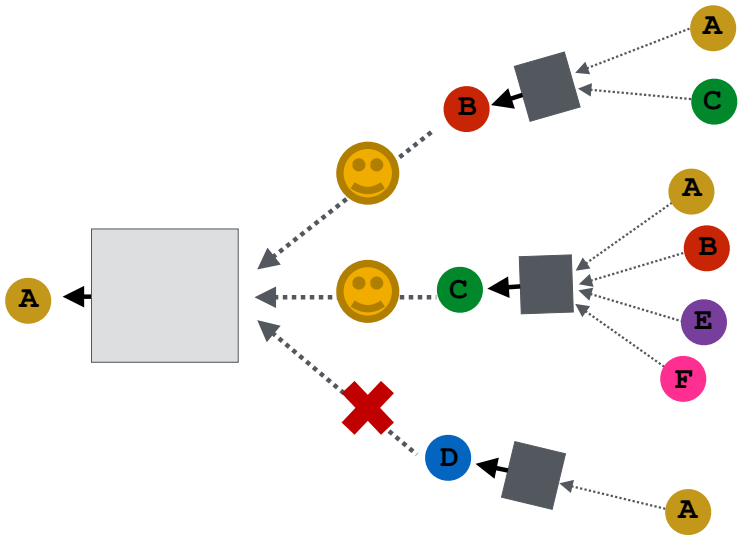
- **Next time when we compute the embeddings, we can sample different neighbors**
  - Only nodes C and D will pass message to A



Neighborhood Sampling Example

- In expectation, we can get embeddings similar to the case where all the neighbors are used
  - **Benefits:** greatly reduce computational cost
  - And in practice it works great!



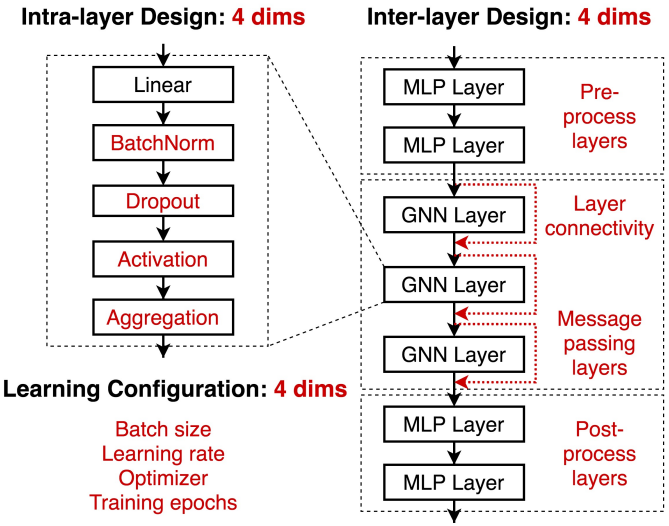


Summary of the Talk

- **Recap: A general perspective for GNNs**
  - **GNN Layer**:
    - Transformation + Aggregation
    - Classic GNN layers: GCN, GraphSAGE, GAT
  - **Layer connectivity**:
    - Deciding number of layers
    - Skip connections
  - **Graph Manipulation:**
    - Feature augmentation
    - Structure manipulation
- **Resources:** PyTorch Geometric + GraphGym

GraphGym: Code Platform for GNN Design

- **Highly modularized** pipeline for GNN research:
  - Data loading, splitting
  - GNN implementation
  - Tasks: node/edge/graph
  - Evaluation: accuracy, ROC AUC, ..
  - ...



Prediction head for different tasks

```
head_dict = {
        'node': GNNNodeHead,
        'edge': GNNEdgeHead,
        'link_pred': GNNEdgeHead,
        'graph': GNNGraphHead
}
```

GNN layers

```
layer_dict = {
    'linear': Linear,
    'mlp': MLP,
    'gcnconv': GCNConv,
    'sageconv': SAGEConv,
    'gatconv': GATConv,
    'splineconv': SplineConv,
    'ginconv': GINConv,
    'generalconv': GeneralConv,
    'generaledgeconv': GeneralE
    'generalsampleedgeconv': Ge
}
```

Layer connectivity

```
stage_dict = {
    'stack': GNNStackStage,
    'skipsum': GNNSkipStage,
    'skipconcat': GNNSkipStage,
}
```

GraphGym: Reproducible experiment management

design_v2.yaml

```yaml
out_dir: results
dataset:
  format: PyG
  name: Cora
  task: node
  task_type: classification
  transductive: True
  split: [0.8, 0.2]
  augment_feature: []
  augment_feature_dims: [10]
  augment_feature_repr: position
  augment_label: ''
  augment_label_dims: 5
  transform: none
train:
  batch_size: 32
  eval_period: 20
  ckpt_period: 100
model:
  type: gnn
  loss_fun: cross_entropy
  edge_decoding: dot
  graph_pooling: add
gnn:
  layers_pre_mp: 1
  layers_mp: 2
  layers_post_mp: 1
  dim_inner: 256
  layer_type: generalconv
  stage_type: stack
  batchnorm: True
  act: prelu
  dropout: 0.0
  agg: add
  normalize_adj: False
optim:
  optimizer: adam
  base_lr: 0.01
  max_epoch: 400
```

Dataset

Training

Model

Optimizer

- One experiment: fully described by a configuration file
- Running an experiment is as simple as

```
python main.py --cfg design_v2.yaml --repeat 3
```

GraphGym: Scalable experiment management

- **A Grid of experimental settings**

```
# dataset: TU, task: graph
dataset.format format ['PyG']
dataset.name dataset ['TU_BZR','TU_COX2','TU_DD','TU_IMDB']
dataset.task task ['graph']
dataset.transductive trans [False]
dataset.augment_feature feature [[]]
dataset.augment_label label ['']
gnn.layers_pre_mp l_pre [1,2]
gnn.layers_mp l_mp [2,4,6,8]
gnn.layers_post_mp l_post [2,3]
gnn.stage_type stage ['skipsum','skipconcat']
gnn.agg agg ['add','mean','max']
```

Run different datasets

Run different models

- **Launching thousands of GNNs in parallel**

```
# generate configs
python configs_gen.py --config design_v2ogb.yaml  --grid round2ogb.txt --out_dir configs
# run batch of configs
# Args: config_dir, num of repeats, max jobs running, sleep time
bash run_batch.sh configs/design_v2ogb_grid_round2ogb 3 8 1
```

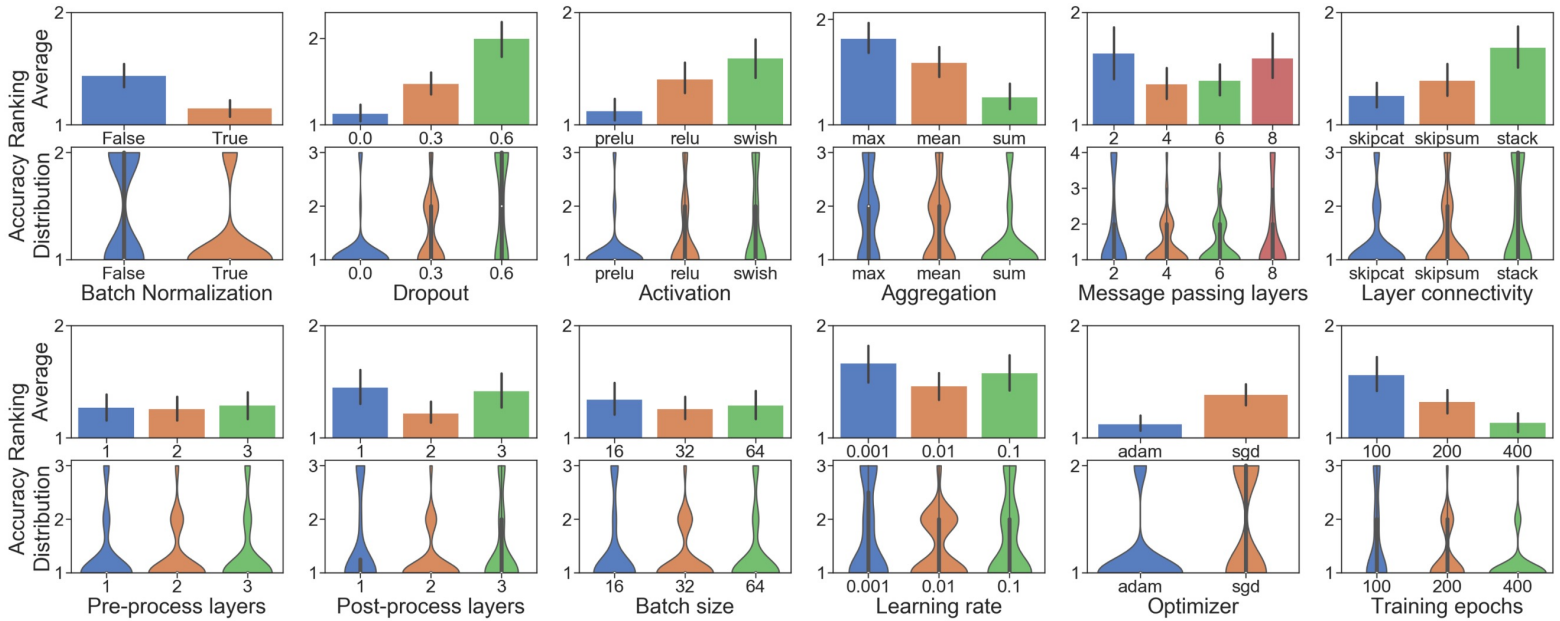
Repeat each experiment for 3 random seeds

Run 8 experiments concurently

GraphGym: Scalable experiment management

- **Automatically generate experiment reports and figures**

l_pre	l_mp	l_post	stage	agg	epoch	loss	loss_std	params	time_iter	time_iter_st	accuracy	accuracy_std
1	2	2	skipconcat	add	399	0.5678	0.0248	217256	0.1098	0.0075	0.886	0.0017
1	2	2	skipconcat	max	399	0.3754	0.0236	217256	0.0896	0.0026	0.9164	0.0017
1	2	2	skipconcat	mean	399	0.4885	0.0122	217256	0.0859	0.0046	0.9083	0.0011
1	2	2	skipsum	add	399	0.5624	0.022	295119	0.1121	0.0155	0.8853	0.0039
1	2	2	skipsum	max	399	0.3966	0.0054	295119	0.1049	0.003	0.9151	0.0025
1	2	2	skipsum	mean	399	0.4701	0.0118	295119	0.1027	0.0038	0.909	0.0028
1	2	3	skipconcat	add	399	0.5944	0.0231	199611	0.1138	0.0376	0.8844	0.0082



Stanford Graph Learning Workshop

Stanford ENGINEERING | Stanford Computer Forum

Stanford | Data Science

- https://snap.stanford.edu/graphlearning-workshop/
  - **Sept 16, 8am-5pm Pacific Time**
  - **Speakers: leaders from academia + industry**
  - **Will be live-streamed, free registration!**

- New graph learning platform: **Kumo**
  - Pytorch Geometric + GraphGym + more!