Stanford CS224W: A General Perspective on Graph Neural Networks

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Announcements

- Colab 1 due this Thursday
- Colab 2 out on Thursday (same day)
- Thank you for the suggestion for in-person & group office hours!

Group Office Hours for Homeworks #127

T Tyler Nichols
Yesterday in General

PIN STAR WATCH 113 VIEWS

6

Beloved CAs,

Might it be possible to have some group office hours for general homework questions throughout the week?

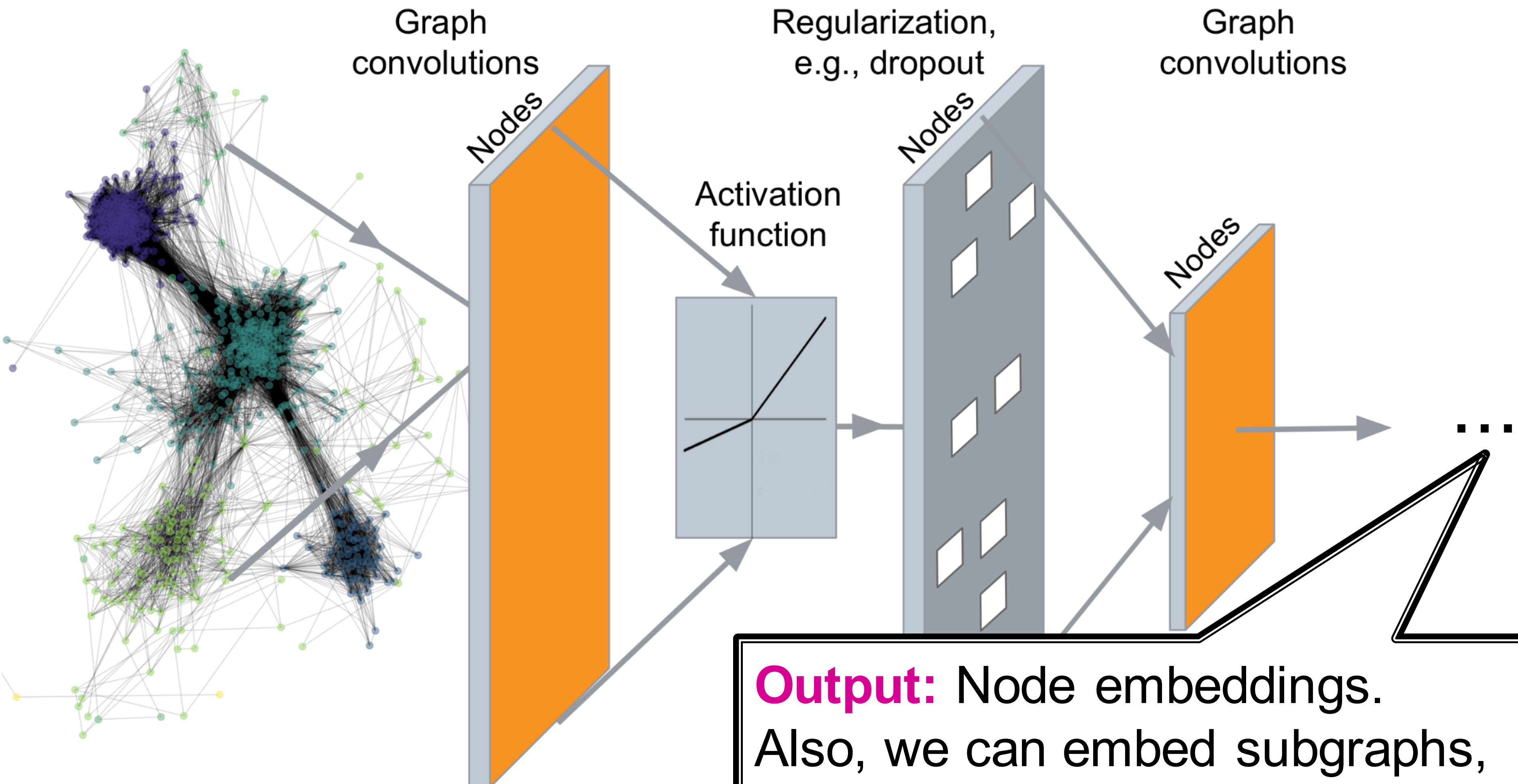
I understand that 1-1 meetings are important for students that need help with some work they've already done or with their code but I suspect there are a *lot* of students who would like to ask general questions (and listen to answers to questions from other students) about how to approach or start working through homework questions.

Answering these general questions with batches of students seems likely to increase the efficiency of office hours a lot. From past experience, students can also get answers to questions that will inevitably pop up for them when they reach the corresponding problems on the assignments which will, in turn, alleviate the pressure on future office hours, as well.

Comment Edit Delete Endorse ···

  - We will be hosting 1 group OH in person a week. CA Zhuoyi Huang will lead these.

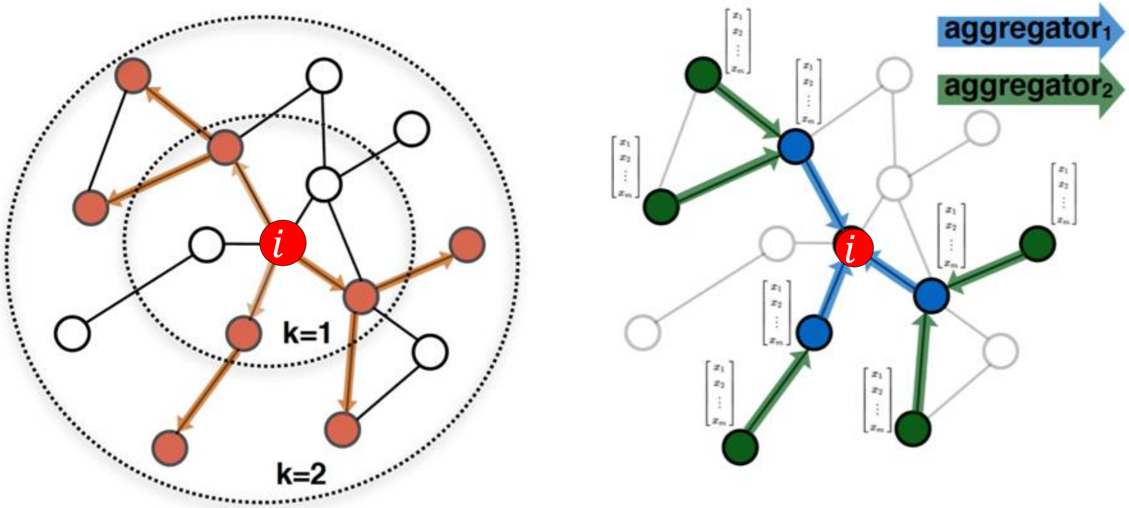
Recap: Deep Graph Encoders



Output: Node embeddings. Also, we can embed subgraphs, graphs

Recap: Graph Neural Networks

Idea: Node's neighborhood defines a computation graph



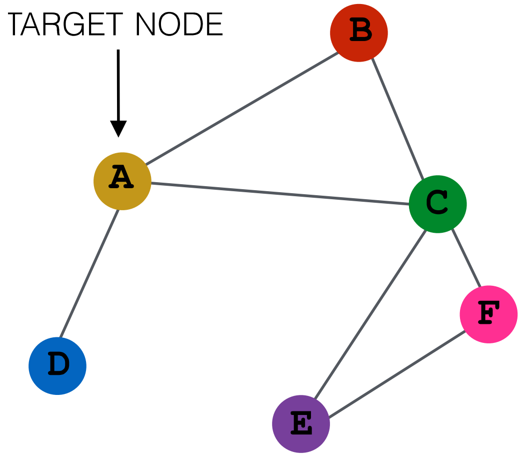
Determine node computation graph

Propagate and transform information

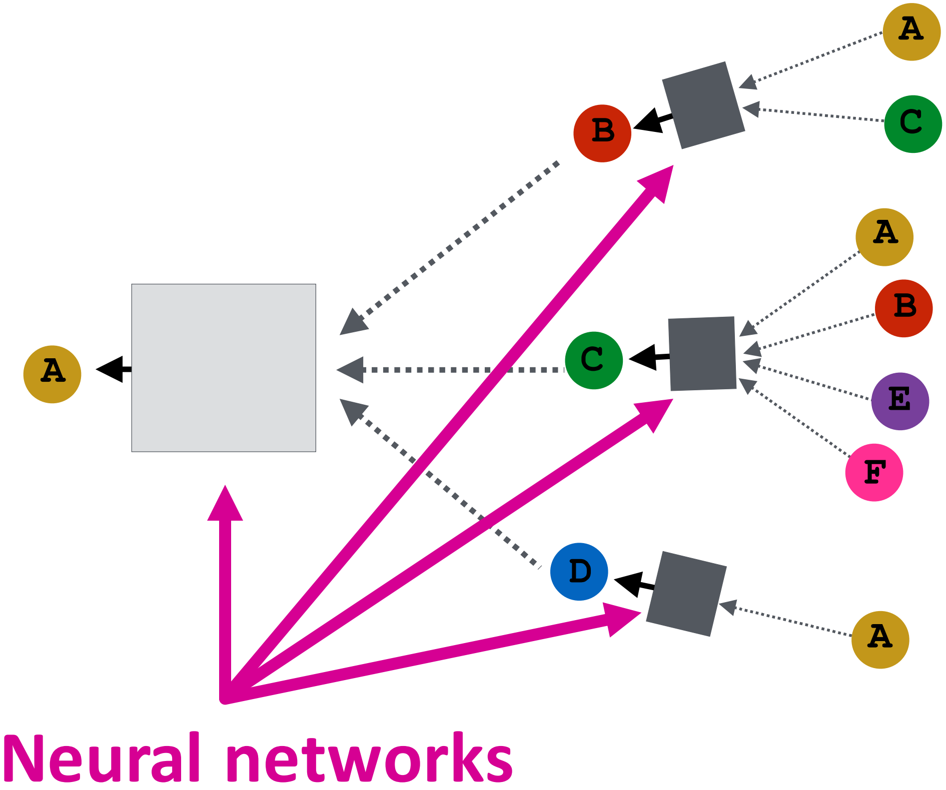
Learn how to propagate information across the graph to compute node features

Recap: Aggregate from Neighbors

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



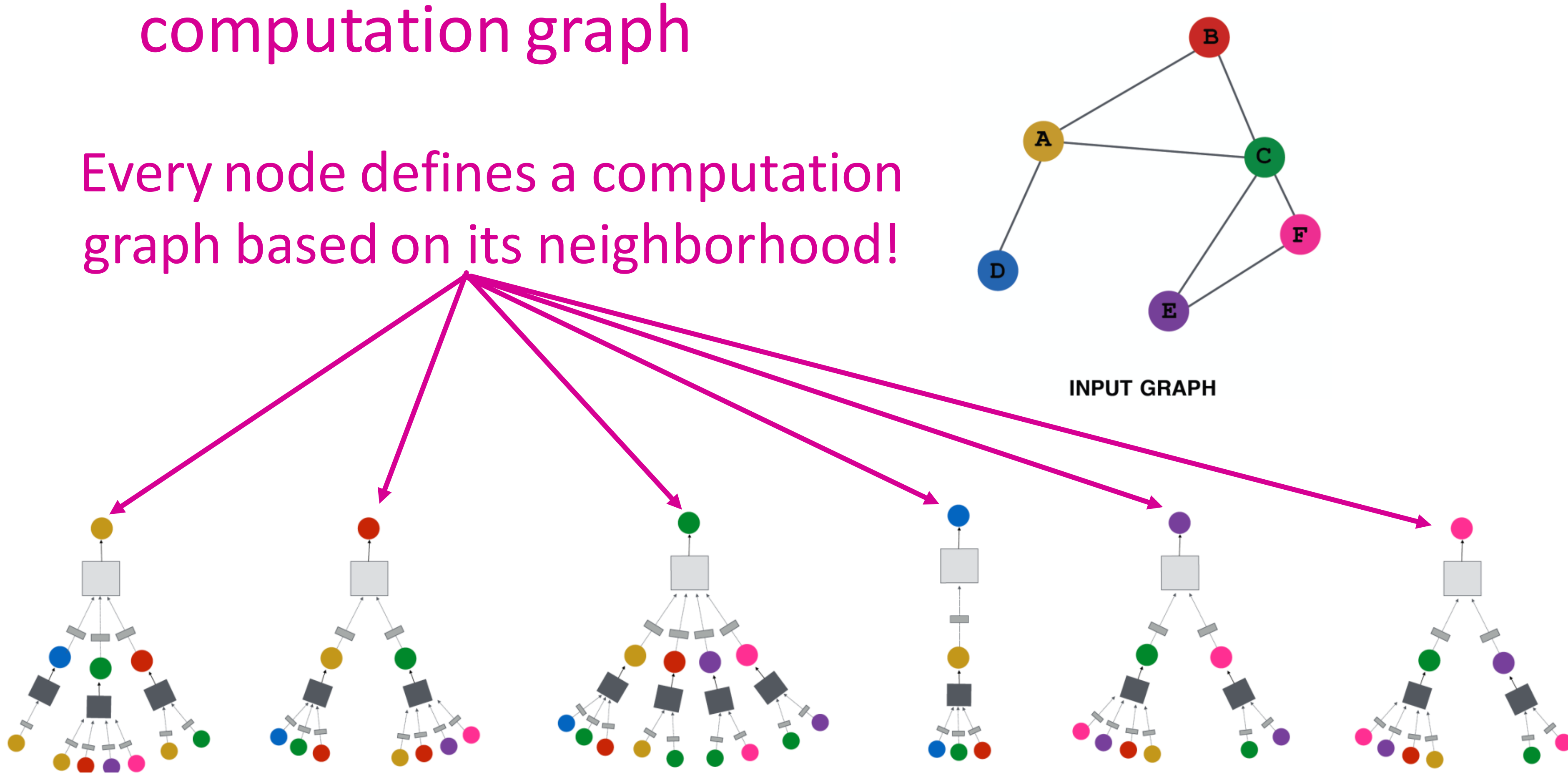
INPUT GRAPH



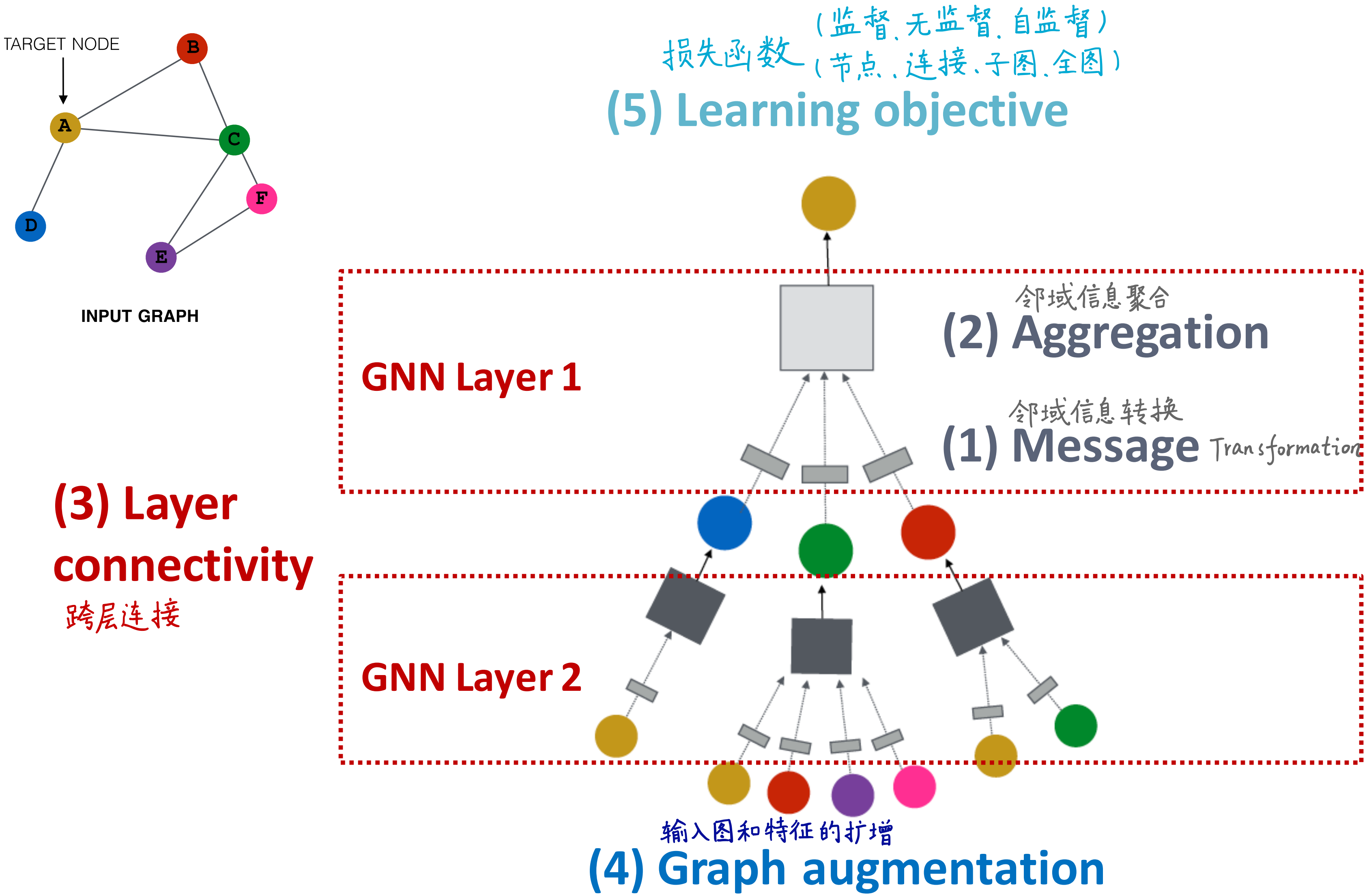
Recap: Aggregate Neighbors

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

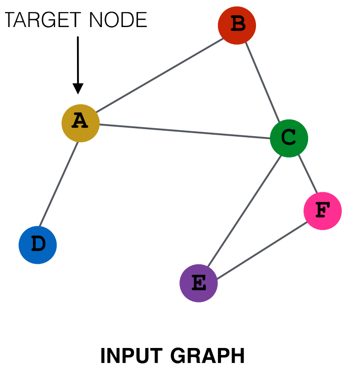
Every node defines a computation graph based on its neighborhood!



Today: A General GNN Framework

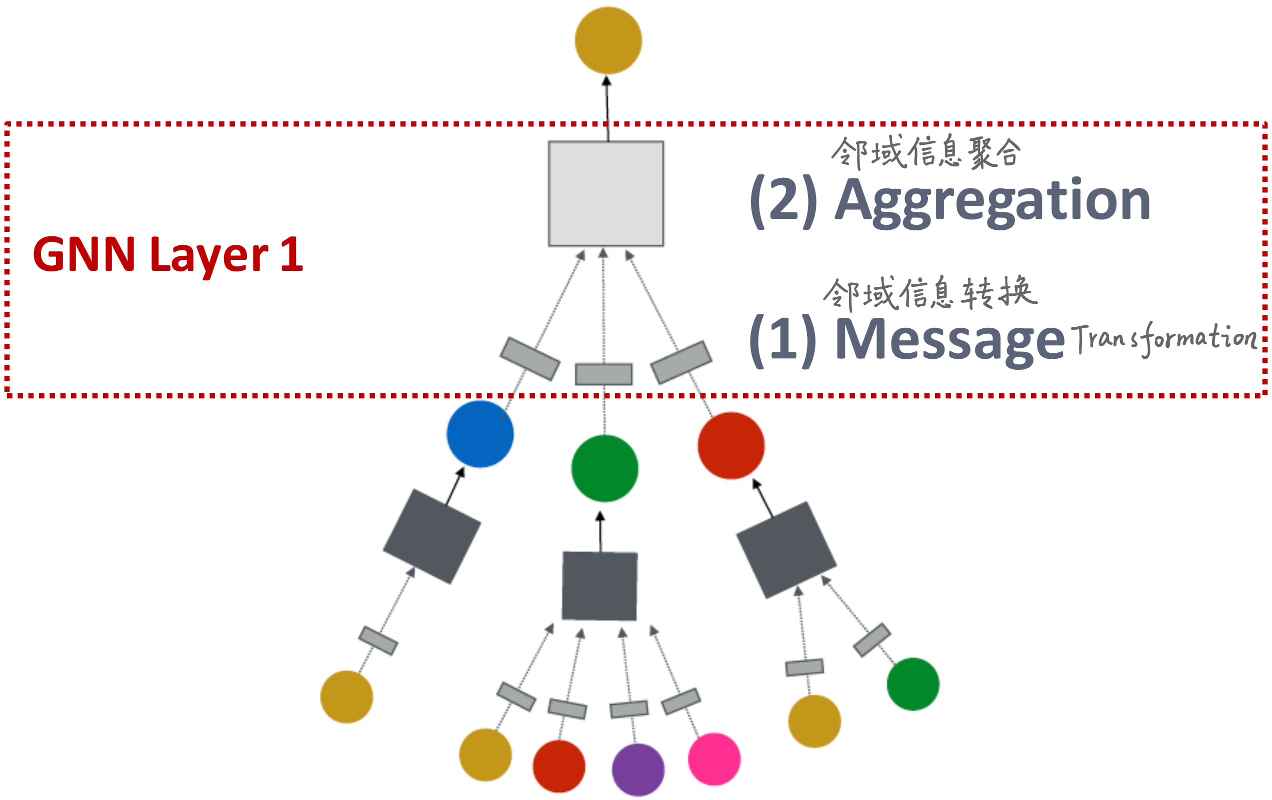


A General GNN Framework (1)

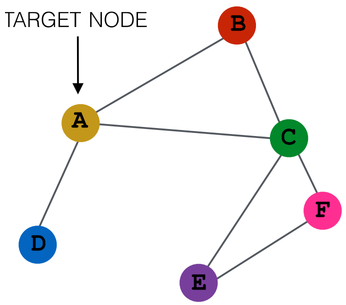


GNN Layer = Message + Aggregation

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



A General GNN Framework (2)

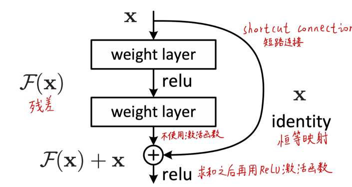


INPUT GRAPH

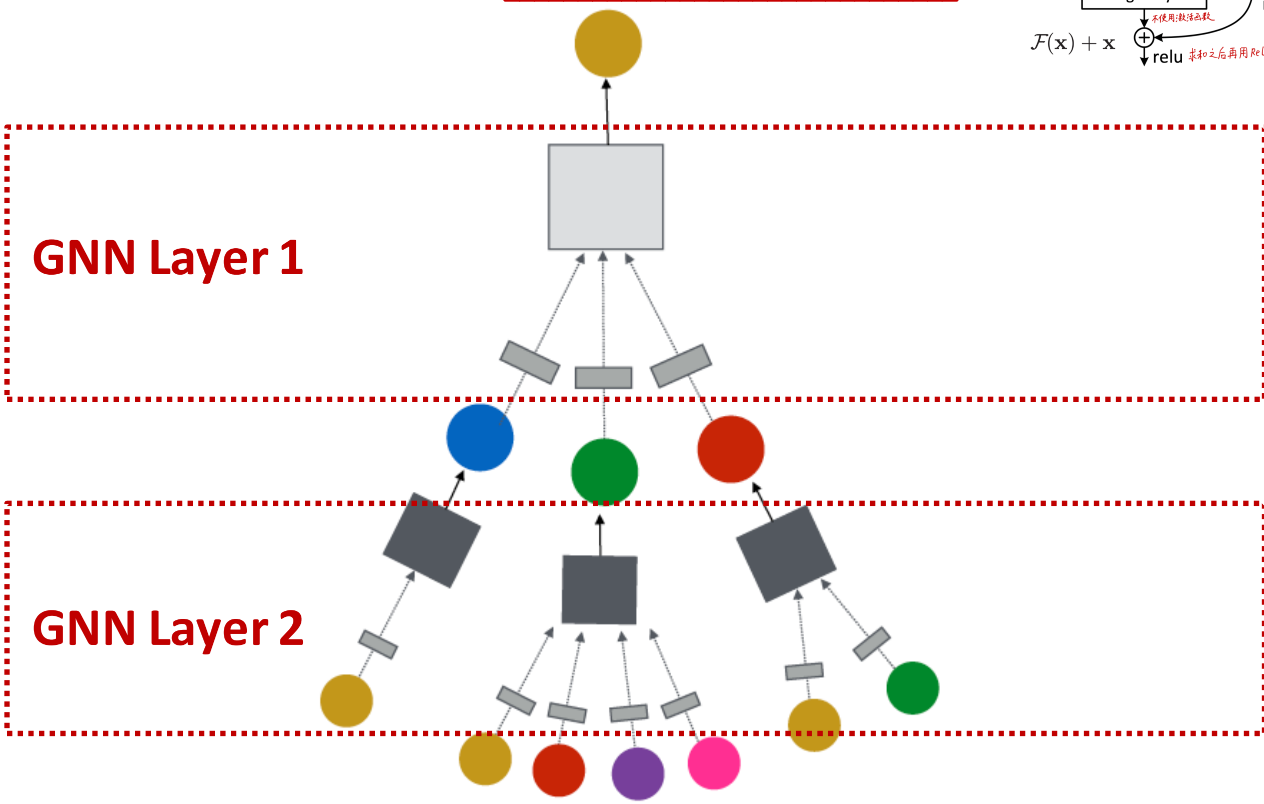
Connect GNN layers into a GNN

- Stack layers sequentially
- Ways of adding skip connections

见同济子豪兄ResNet论文精读

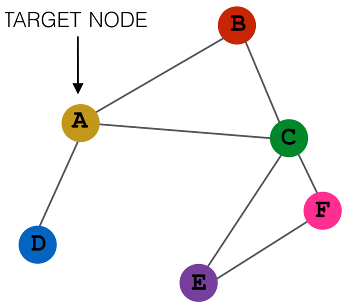


(3) Layer connectivity



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

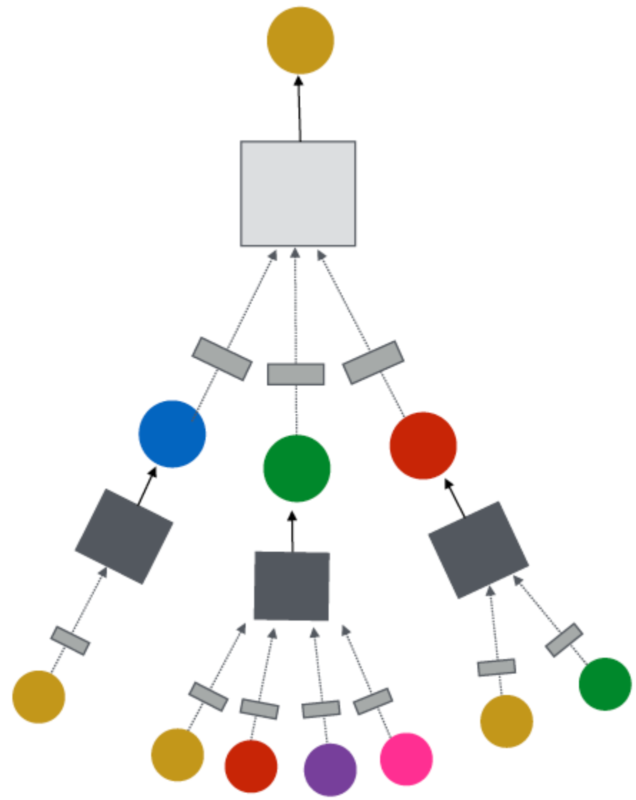
A General GNN Framework (3)



INPUT GRAPH

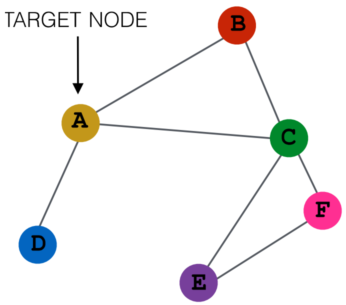
Idea: Raw input graph ≠ computational graph

- **Graph feature augmentation** 特征扩增
- **Graph structure augmentation** 结构扩增



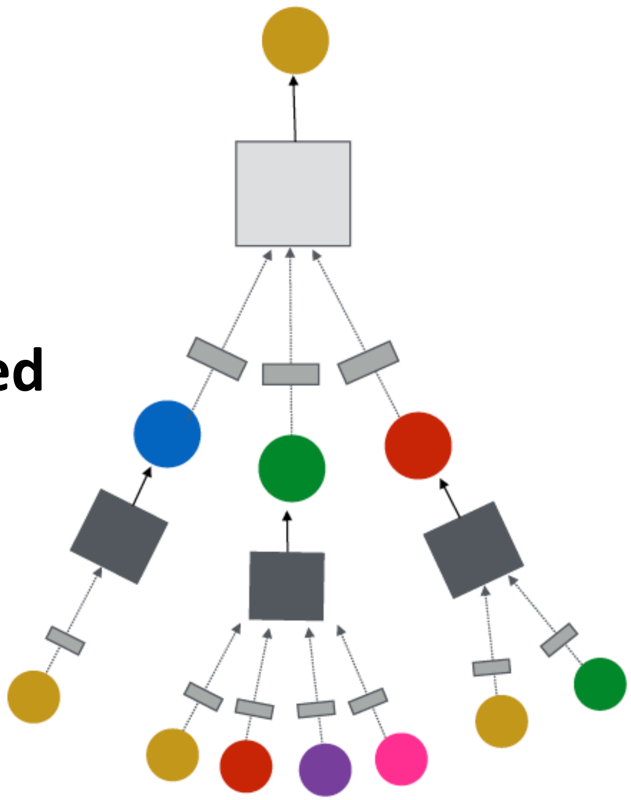
(4) Graph augmentation

A General GNN Framework (4)



INPUT GRAPH

(5) Learning objective

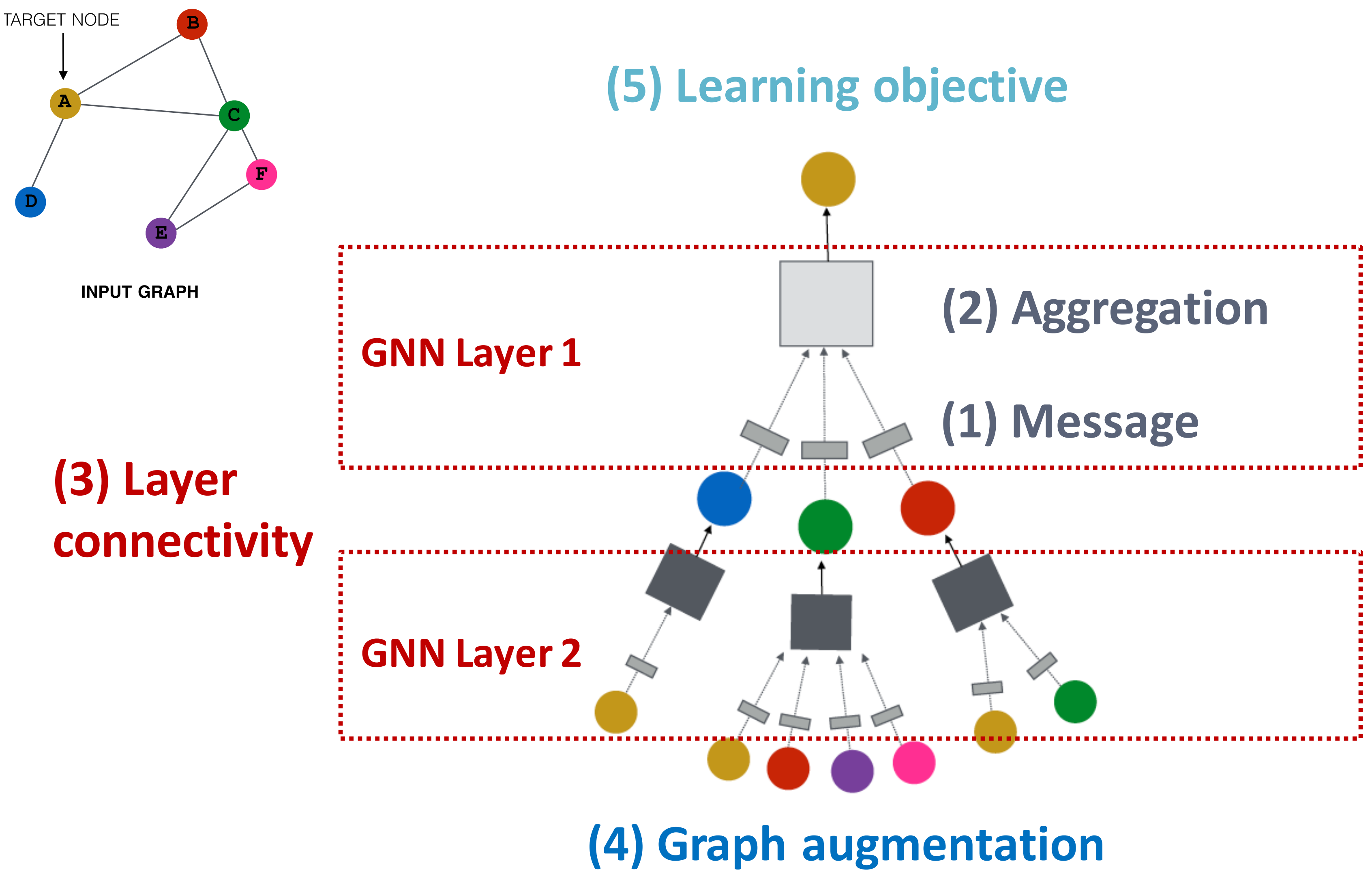


How do we train a GNN

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

(We will discuss all of these later in class)

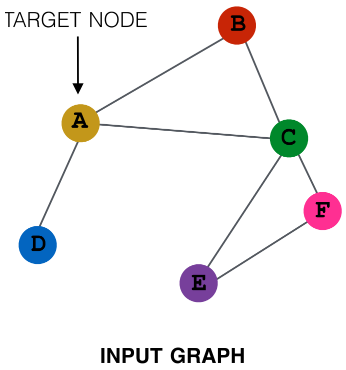
GNN Framework: Summary



Stanford CS224W: A Single Layer of a GNN

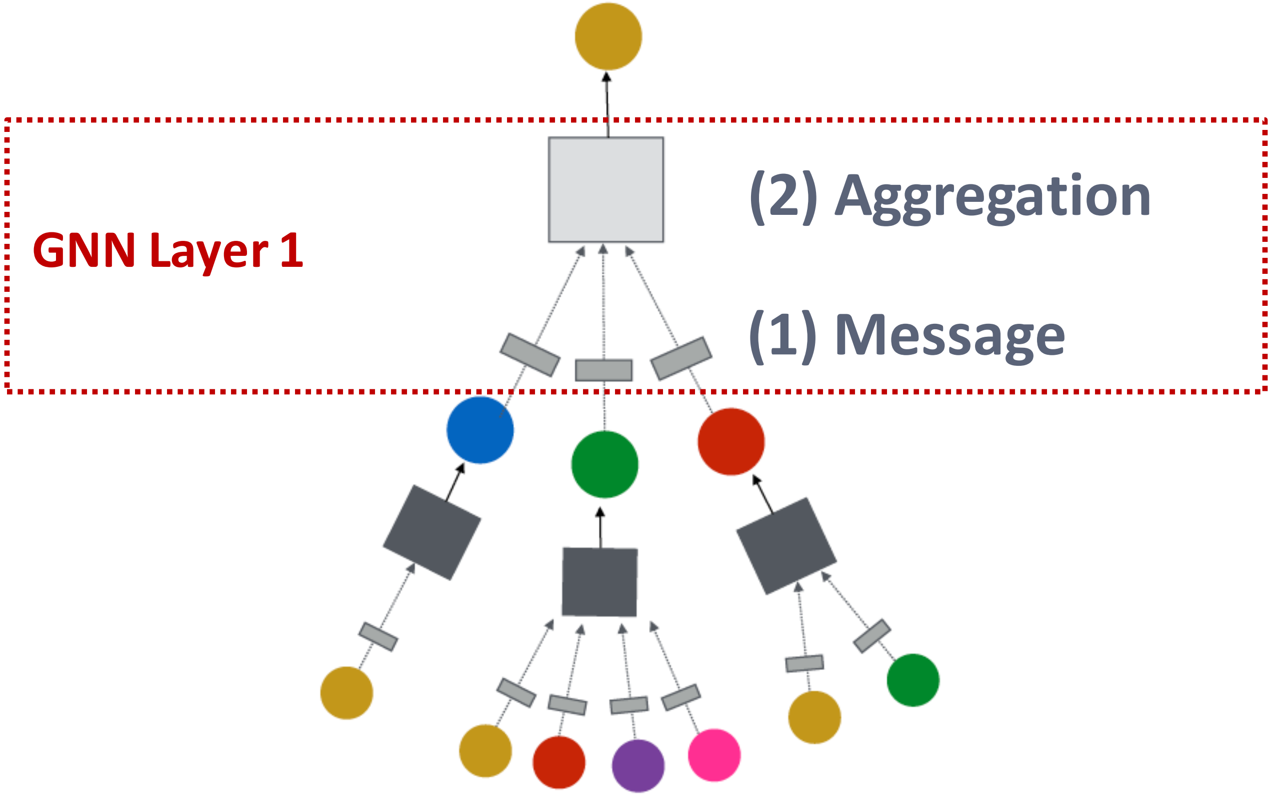
CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

A GNN Layer



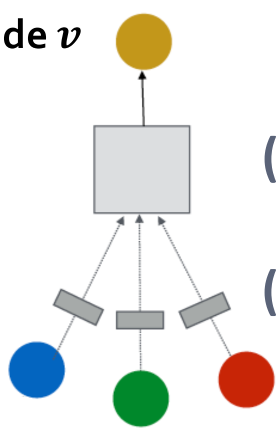
GNN Layer = Message + Aggregation

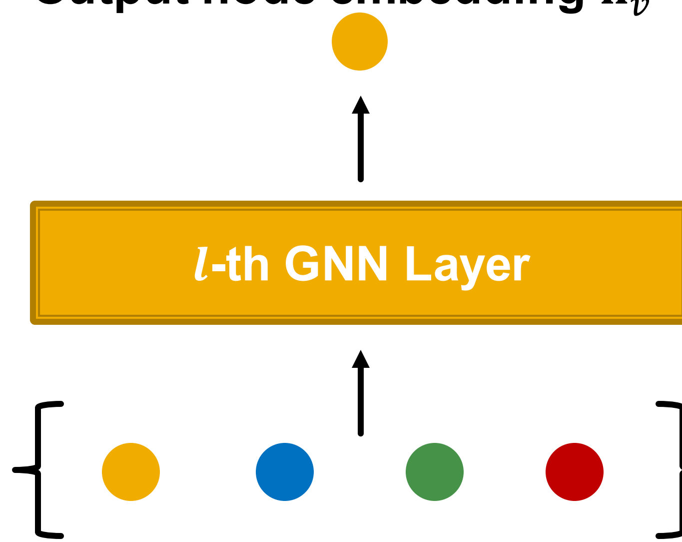
- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**



A Single GNN Layer

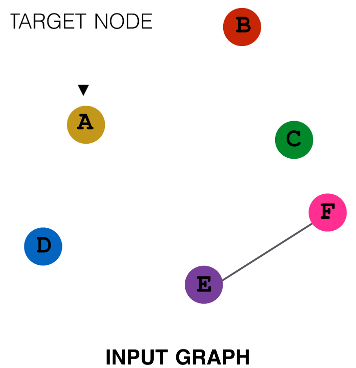
- **Idea of a GNN Layer:**
  - Compress a set of vectors into a single vector
  - **Two-step process:**
    - **(1) Message**
    - **(2) Aggregation**

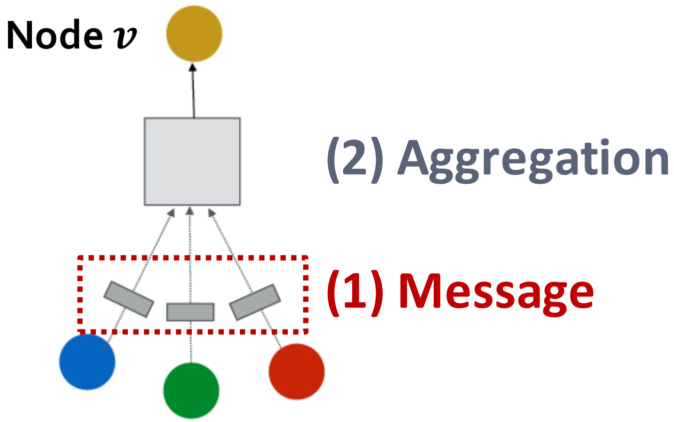




Message Computation

- **(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 $\mathbf{W}^{(l)}$



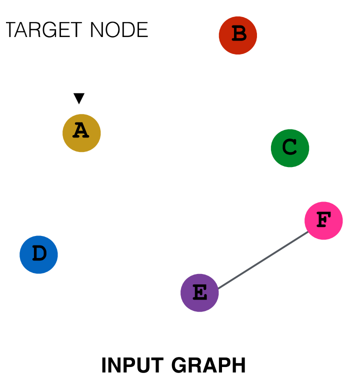


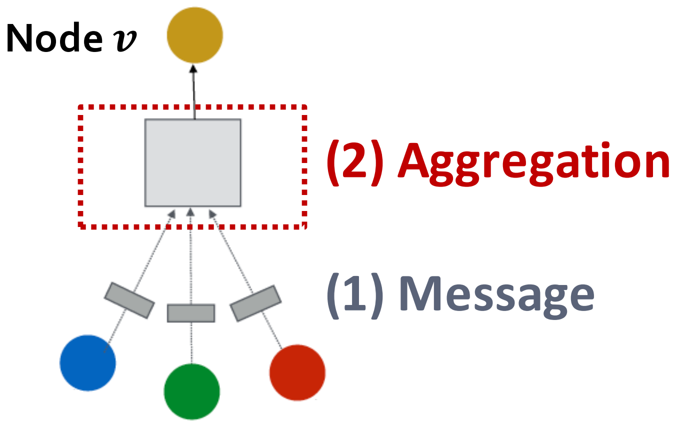
Message Aggregation

- **(2) Aggregation** 聚合
  - **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

上一层自己的Embedding信息　　　人不能永远活在别人的评价里

- **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

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

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

Then aggregate from node itself

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

First aggregate from neighbors

A Single GNN Layer

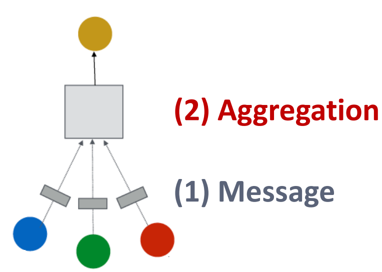
- **Putting things together:**
  - **(1) Message**: each node computes a message

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

  - **(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 非线性表示能力
    - Often written as $\sigma(\cdot)$: $\mathrm{ReLU}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, ...
    - Can be added to **message or aggregation**



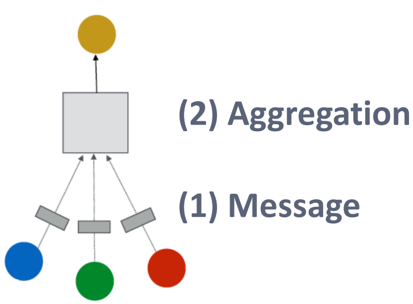
Classical GNN Layers: GCN (1)

- **(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?**

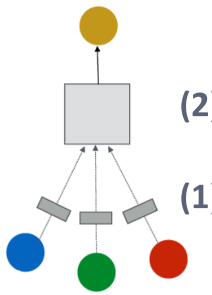
$$\mathbf{h}_v^{(l)} = \sigma\left(\underbrace{\sum_{u \in N(v)}}_{\text{Aggregation}} \overbrace{\mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}^{\text{Message}}\right)$$



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)
Normalized Adjacency Matrix
$D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

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

In GCN graph is assumed to have self-edges that are included in the summation.

Classical GNN Layers: GraphSAGE

- **(2) GraphSAGE**

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

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

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

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

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

GraphSAGE Neighbor Aggregation

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

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

Aggregation   Message computation

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

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

Aggregation   Message computation

- **LSTM:** Apply LSTM to reshuffled of neighbors 长短时记忆神经网络（打乱节点顺序输入）

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

Aggregation

GraphSAGE: L2 Normalization

- ℓ_2 **Normalization:**
  - **Optional:** Apply ℓ_2 normalization to $\mathbf{h}_v^{(l)}$ at every layer
  - $\mathbf{h}_v^{(l)} \leftarrow \frac{\mathbf{h}_v^{(l)}}{\left\|\mathbf{h}_v^{(l)}\right\|_2} \; \forall v \in V$ where $\|u\|_2 = \sqrt{\sum_i u_i^2}$ (ℓ_2-norm)
  - Without ℓ_2 normalization, the embedding vectors have different scales (ℓ_2-norm) for vectors
  - In some cases (not always), normalization of embedding results in performance improvement
  - After ℓ_2 normalization, all vectors will have the same ℓ_2-norm

类似对比两个国家的人均GDP

Classical GNN Layers: GAT (1)

- **(3) Graph Attention Networks**

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

Attention weights

- **In GCN / GraphSAGE**
  - $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u's message to node v
  - $\Longrightarrow$ α_{vu} is defined **explicitly** based on the structural properties of the graph (node degree)
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v 不同邻居带来的信息权重相同（连接的权重相同）

Classical GNN Layers: GAT (2)

- **(3) Graph Attention Networks**

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

Attention weights

Not all node's neighbors are equally important

- **Attention** is inspired by cognitive attention. 认知科学中的"注意力"
- 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.
  - Which part of the data is more important depends on the context and is learned through training.

Graph Attention Networks

Can we do better than simple neighborhood aggregation?

Can we let weighting factors α_{vu} to be learned? 通过学习得到注意力权重

- **Goal:** Specify **arbitrary importance** to different neighbors of each node in the graph
- **Idea:** Compute embedding $\boldsymbol{h}_v^{(l)}$ of each node in the graph following an **attention strategy**:
  - Nodes attend over their neighborhoods' message
  - Implicitly specifying different weights to different nodes in a neighborhood

Attention Mechanism (1)

副产品

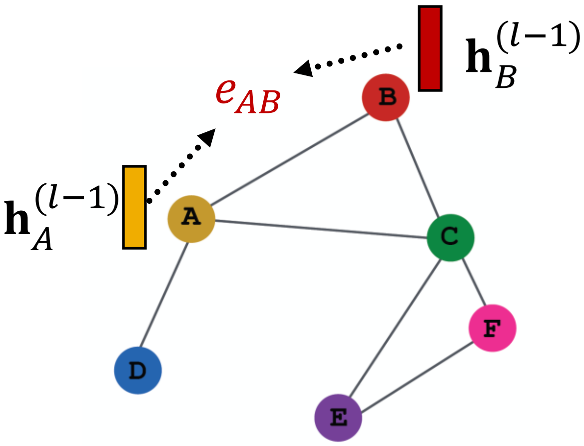
- Let α_{vu} be computed as a byproduct of an **attention mechanism $\boldsymbol{a}$:**
  - (1) Let a compute **attention coefficients $\boldsymbol{e_{vu}}$** across pairs of nodes u, v based on their messages:

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

a：自注意力函数

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

注意力分数



$$e_{AB} = a(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)})$$

Attention Mechanism (2)

注意力权重

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

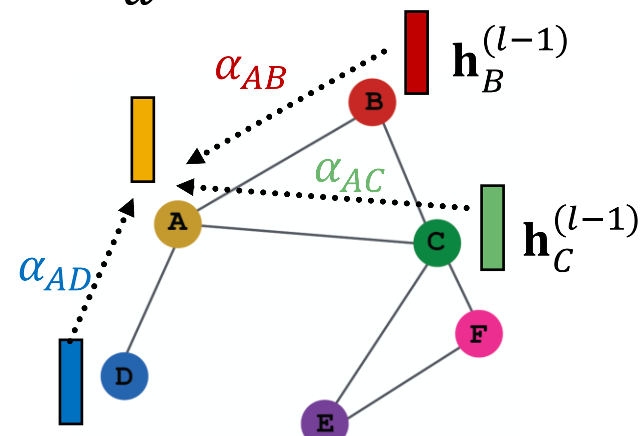
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

- **Weighted sum** based on the **final attention weight** α_{vu} 加权求和

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

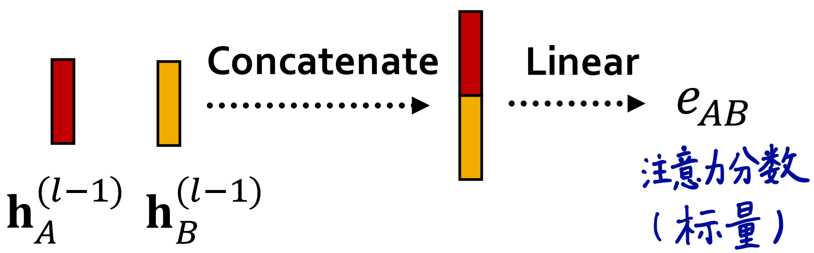
Weighted sum using $\alpha_{AB}, \alpha_{AC}, \alpha_{AD}$**:**

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



Attention Mechanism (3)

- **What is the form of attention mechanism a?**
  - The approach is agnostic to the choice of a
    - E.g., use a simple single-layer neural network 逻辑回归去掉sigmoid
      - a have trainable parameters (weights in the Linear layer)



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

- Parameters of a are trained jointly:
  - Learn the parameters together with weight matrices (i.e., other parameter of the neural net $\mathbf{W}^{(l)}$) in an end-to-end fashion 端到端

Attention Mechanism (4)

多头注意力机制　分别训练不同的α函数　每个α函数对应一套α权重

- **Multi-head attention:** Stabilizes the learning process of attention mechanism 避免偏见 陷入局部最优
  - **Create multiple attention scores** (each replica with a different set of parameters):

$$\mathbf{h}_v^{(l)}[1] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^{1} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$ 《狼性文化》

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

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

  - **Outputs are aggregated:** 《三体》
    - By concatenation or summation
    - $\mathbf{h}_v^{(l)} = \mathrm{AGG}(\mathbf{h}_v^{(l)}[1], \mathbf{h}_v^{(l)}[2], \mathbf{h}_v^{(l)}[3])$

Benefits of Attention Mechanism

- **Key benefit:** Allows for (implicitly) specifying **different importance values (α_{vu}) to different neighbors**
  隐式 指定
- **Computationally efficient**: 计算高效
  - Computation of attentional coefficients can be parallelized across all edges of the graph
  - Aggregation may be parallelized across all nodes
- **Storage efficient**: 存储高效
  - Sparse matrix operations do not require more than $O(V + E)$ entries to be stored α函数参数量固定,与节点和连接数无关
  - **Fixed** number of parameters, irrespective of graph size
- **Localized**:
  - Only **attends over local network neighborhoods** 局部
- **Inductive capability**: 归纳泛化
  - It is a shared *edge-wise* mechanism 共享同一个α函数
  - It does not depend on the global graph structure 并泛化到新图

Stanford CS224W: GNN Layers in Practice

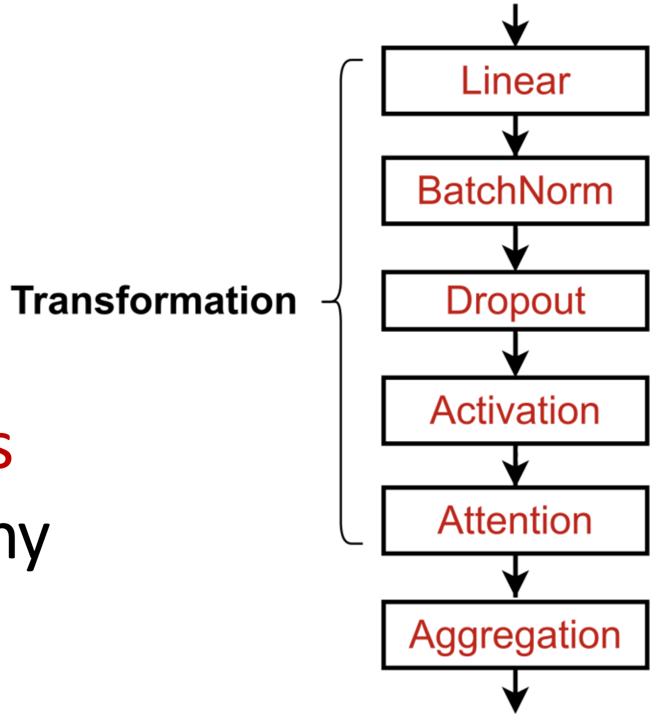
CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

GNN Layer in Practice

通用 GNN 层模板

- 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 模块 modern deep learning modules that proved to be useful in many domains

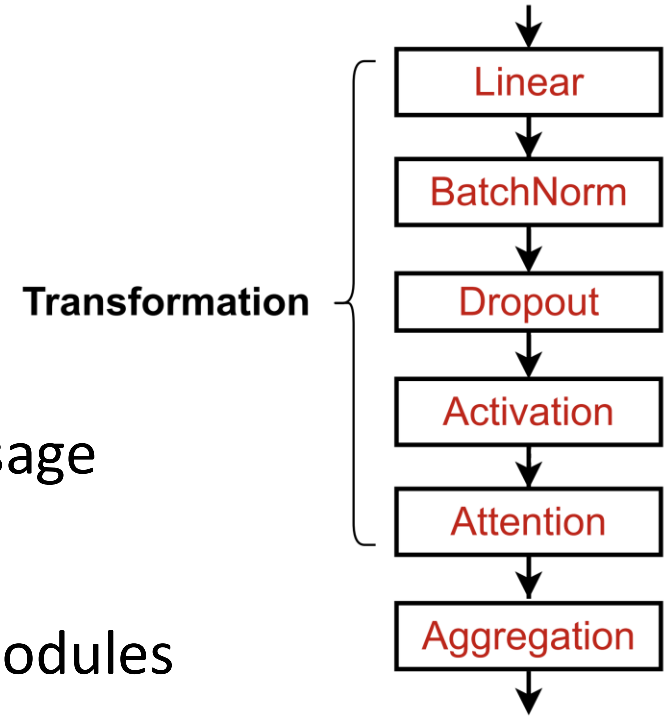
A suggested GNN Layer



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

A suggested GNN Layer



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 在每个维度分别计算N个样本的均值、标准差

N个样本
每个样本D维向量

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

Trainable Parameters:
$\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{D}$ 每个维度有2个恢复参数

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

Step 1:
Compute the mean and variance over N embeddings

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

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

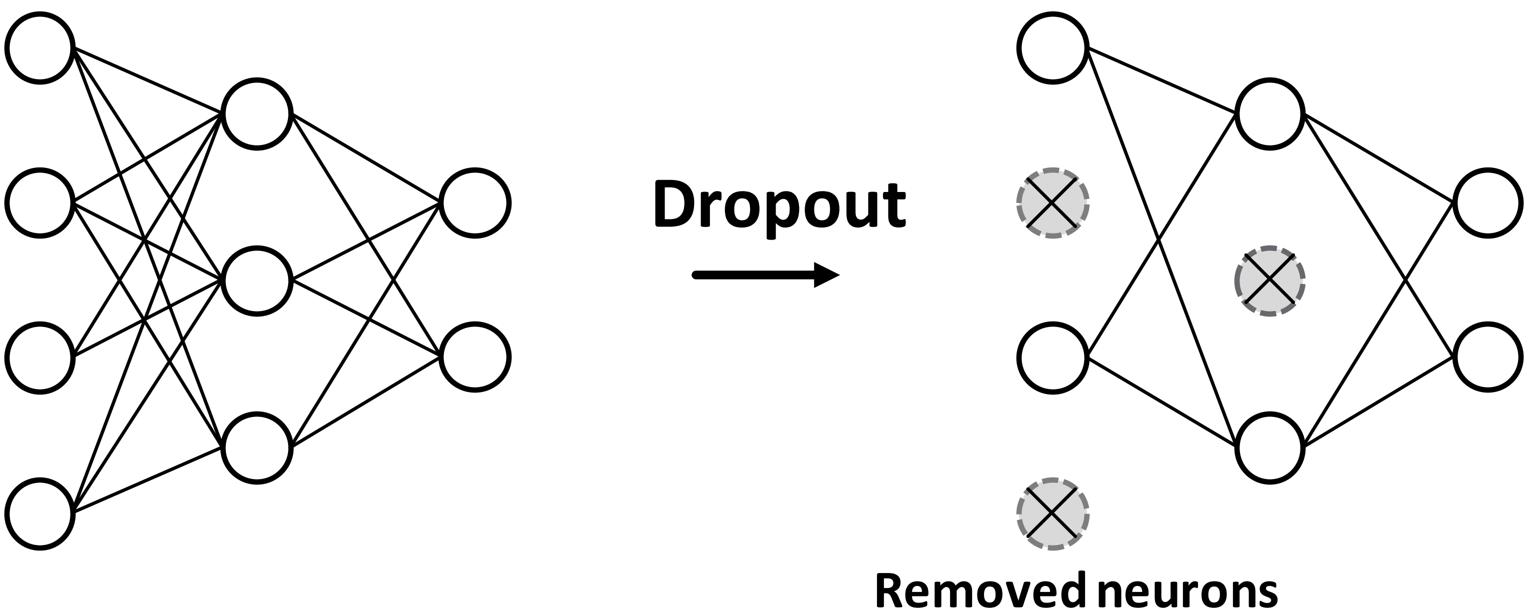
Step 2:
Normalize the feature using computed mean and variance

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

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

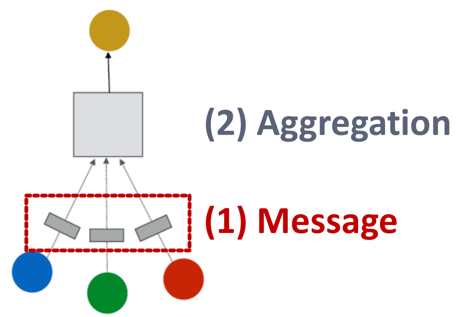
Dropout

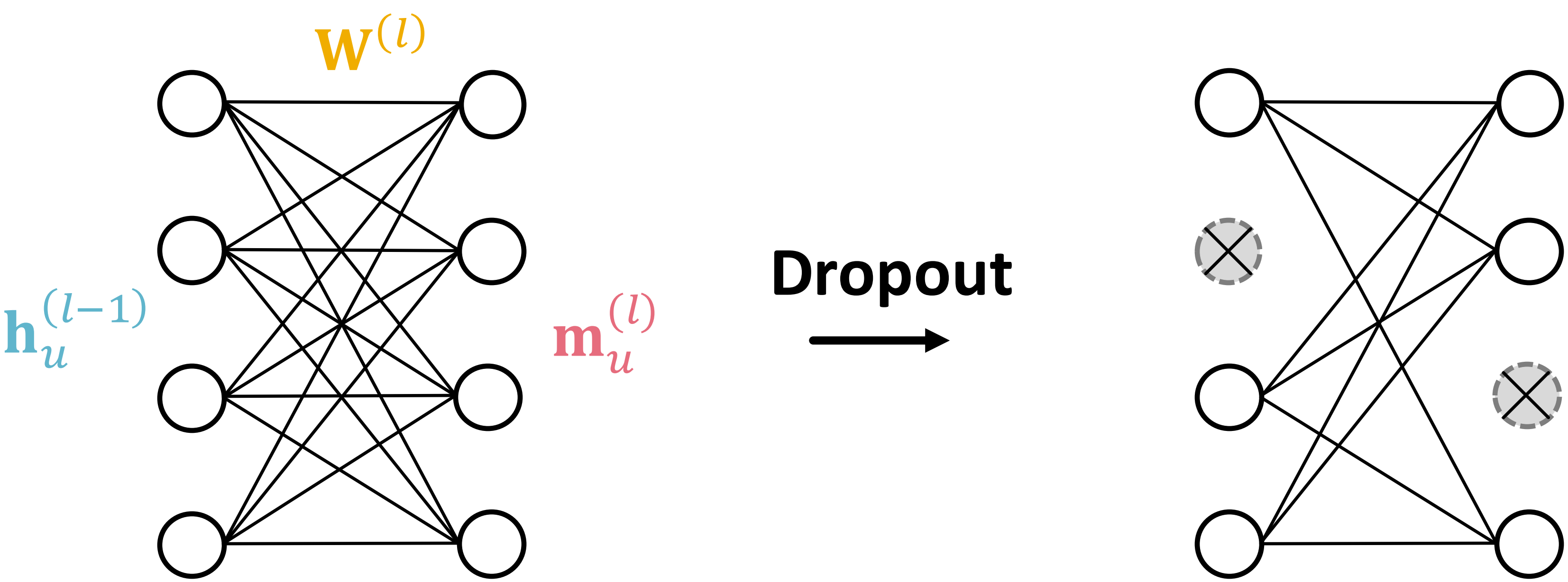
- **Goal**: Regularize a neural net to prevent overfitting.
- **Idea**:
  - **During training**: with some probability p, randomly set neurons to zero (turn off)
  - **During testing:** Use all the neurons for computation



Dropout for GNNs

- In GNN, Dropout is applied to **the <u>linear layer</u> 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

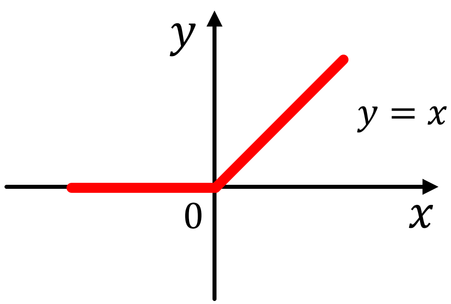
Activation (Non-linearity)

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

- **Rectified linear unit (ReLU)**

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

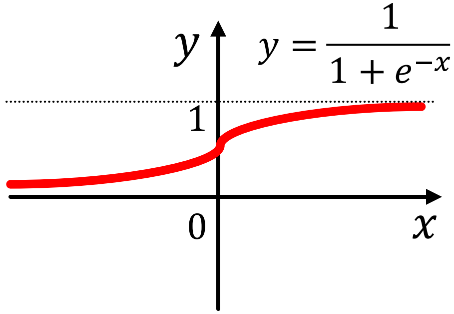
  - Most commonly used



- **Sigmoid**

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

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

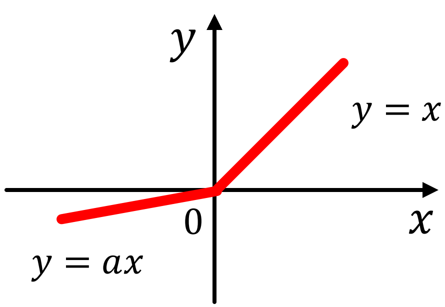


- **Parametric ReLU** Leaky ReLU

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

a_i is a trainable parameter

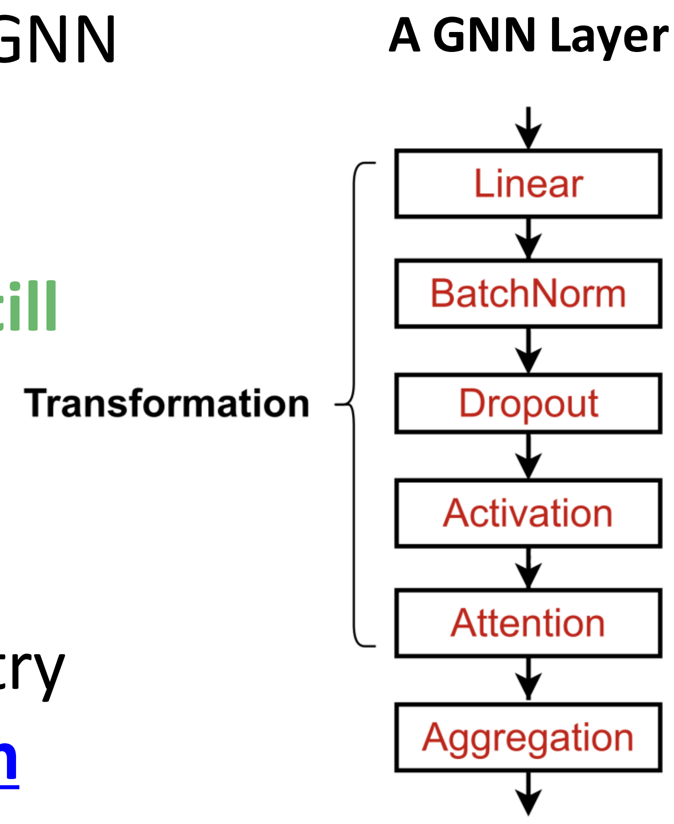
  - Empirically performs better than ReLU



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

强烈推荐
见同济子豪兄代码教程



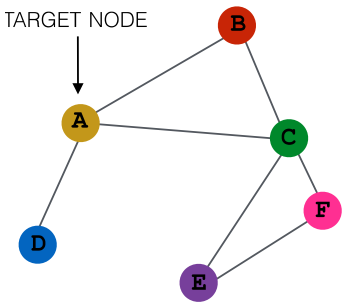
Stanford CS224W: Stacking Layers of a GNN

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

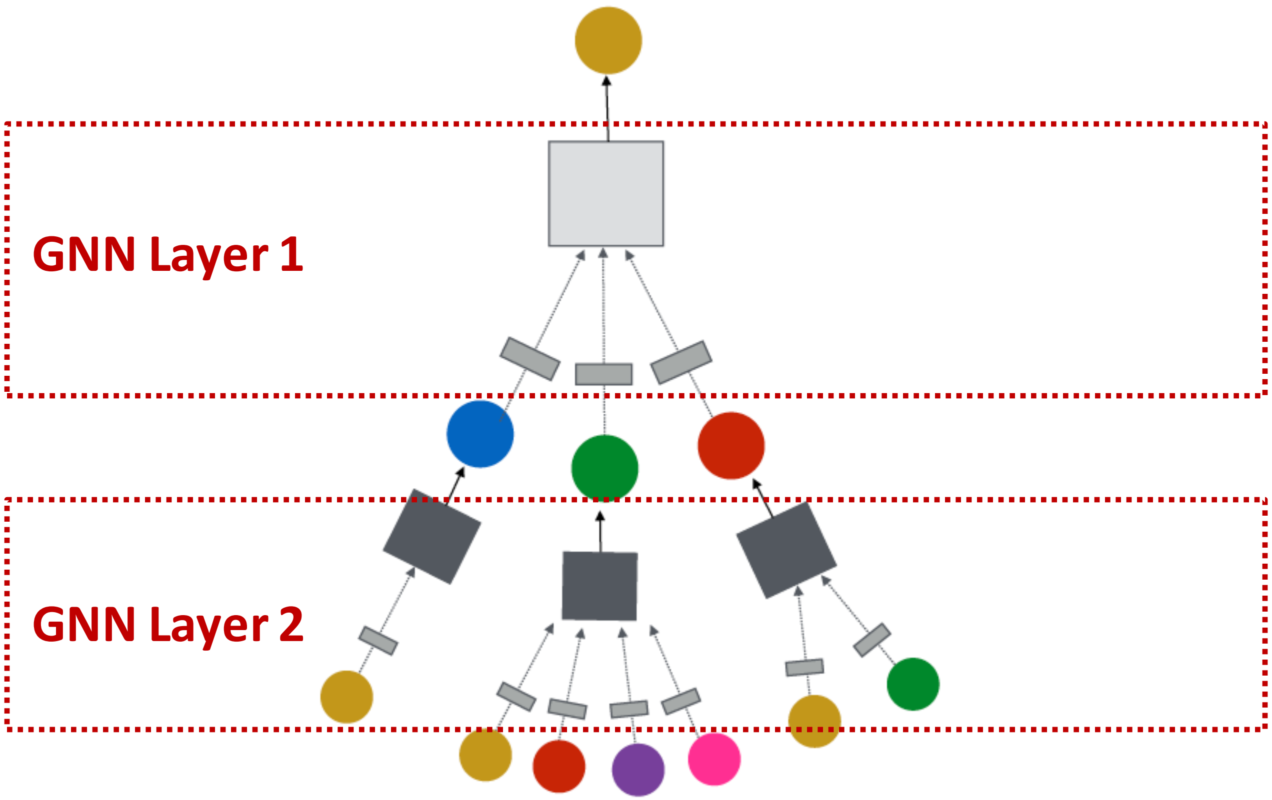
Stacking GNN Layers



INPUT GRAPH

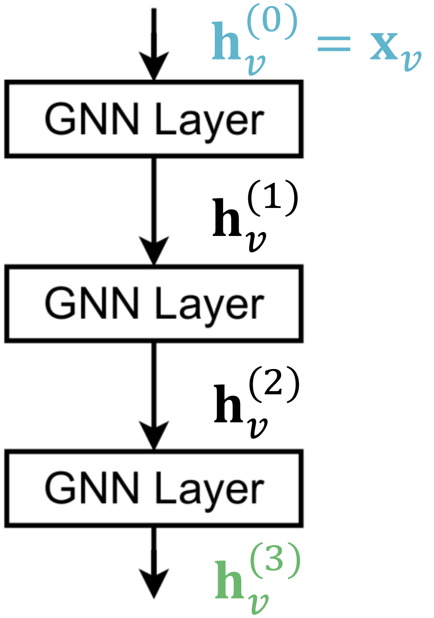
How to connect GNN layers into a GNN?
- **Stack layers sequentially**
- **Ways of adding skip connections**

(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 $\mathbf{x}_v$ 节点属性特征
  - **Output:** Node embeddings $\mathbf{h}_v^{(L)}$ after L GNN layers



The Over-smoothing Problem

GNN层数不能过深

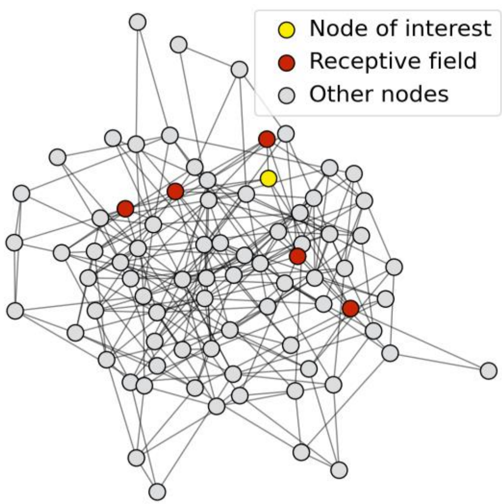
- **The Issue of stacking many GNN layers**
  - GNN suffers from **the over-smoothing problem**
- **The over-smoothing problem: all the node embeddings converge to the same value** 所有节点的Embedding相同
  - This is bad because we **want to use node embeddings to differentiate nodes**
- **Why does the over-smoothing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

Receptive Field of a GNN

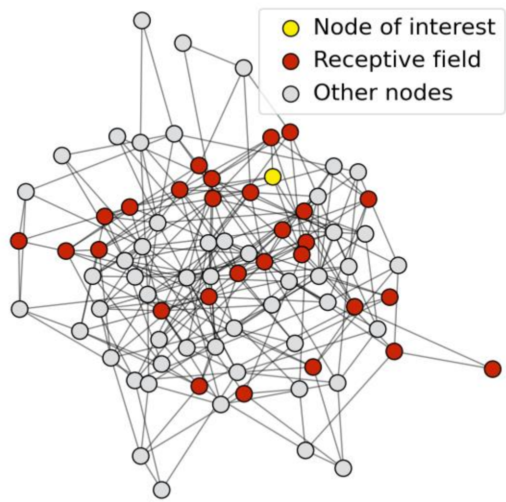
K层GNN的感受野

- **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**

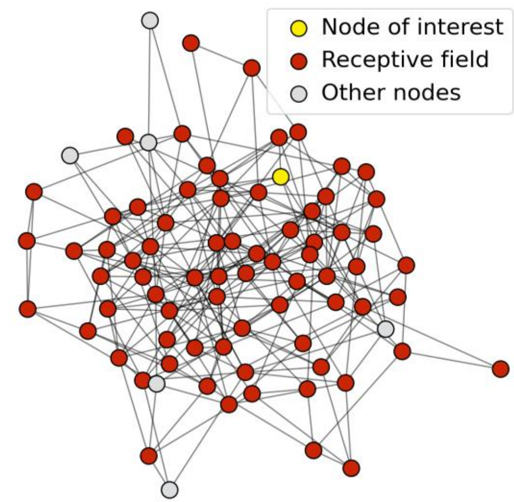
Receptive field for
1-layer GNN



Receptive field for
2-layer GNN



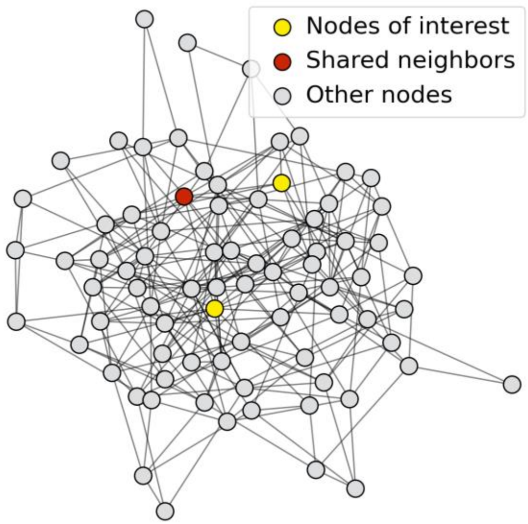
Receptive field for
3-layer GNN



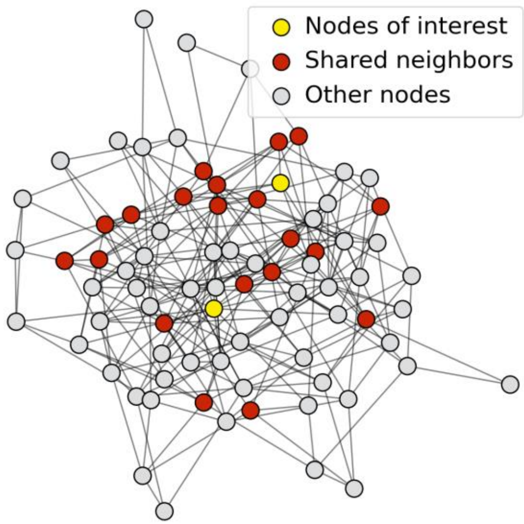
Receptive Field of a GNN

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

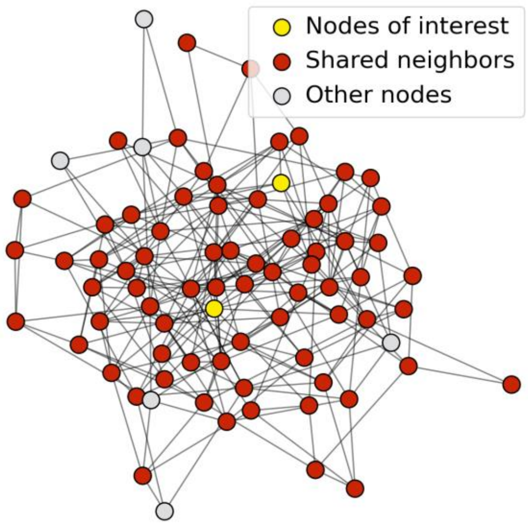
1-hop neighbor overlap
Only 1 node



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** 两个离着很远的节点，也会有相似的计算图 相似的 Embedding
- **Next:** how do we overcome over-smoothing problem?

Design GNN Layer Connectivity

不能无脑堆很多GNN层

- **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 直径 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**!

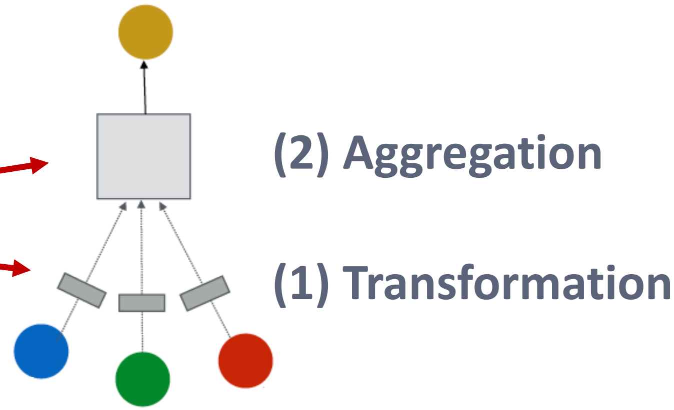
IDEA：用AutoML自动找到最优的L

- **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** GNN层的内部 可以加深度神经网络
  - In our previous examples, each transformation or aggregation function only include one linear layer
  - We can **make aggregation / transformation become a deep neural network**!

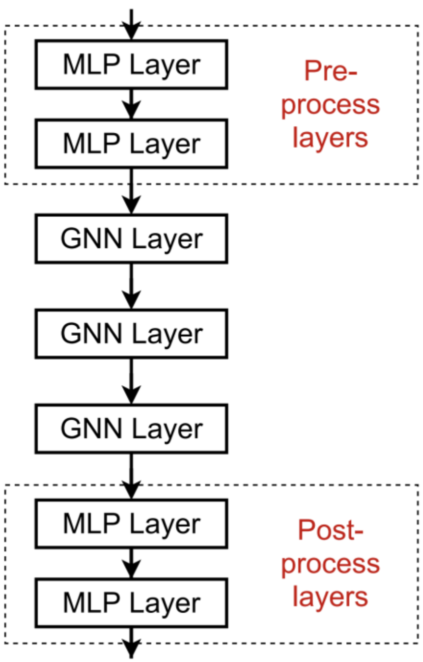
If needed, each box could include a 3-layer MLP



Expressive Power for Shallow GNNs

预处理/后处理层

- **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
处理多模态数据

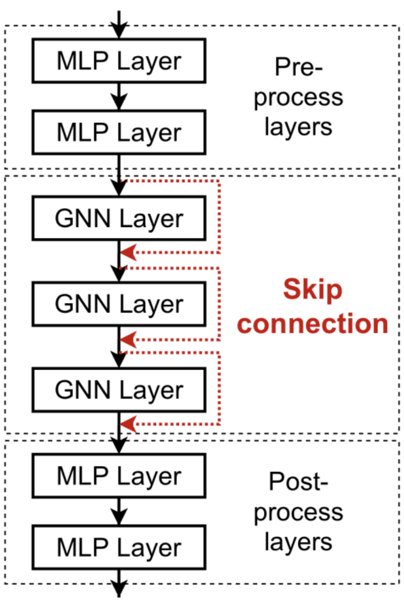
Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs
对节点 Embedding 汇总、推理

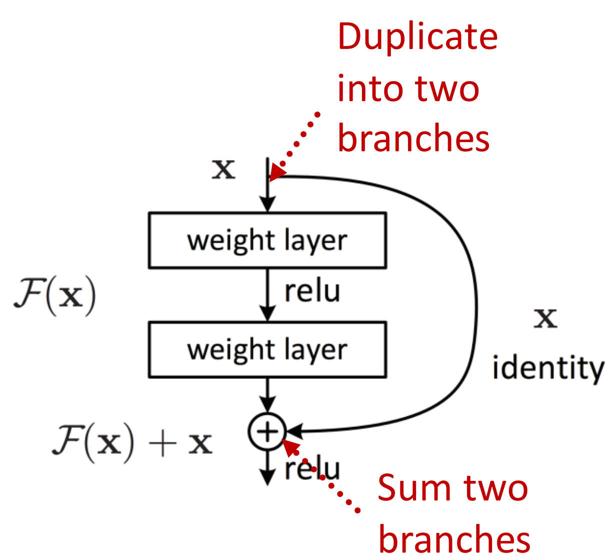
In practice, adding these layers works great!

He et al. Deep Residual Learning for Image Recognition, CVPR 2015

Design GNN Layer Connectivity

- **What if my problem still requires many GNN layers?**
- **Lesson 2: Add skip connections in GNNs** 前面感受野较小的层
  - **Observation from over-smoothing:** Node embeddings in earlier GNN layers can sometimes better differentiate (区分) 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:

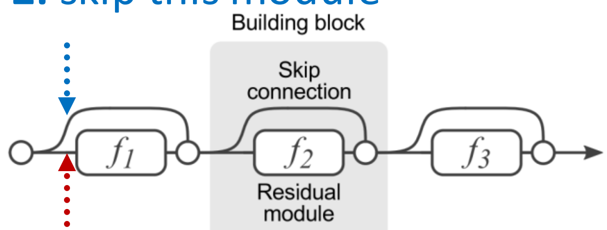
$$\boldsymbol{F}(\mathbf{x}) + \mathbf{x}$$

见同济子豪兄ResNet论文精读

Idea of Skip Connections

- **Why do skip connections work?** 见同济子豪兄ResNet论文精读
  - **Intuition:** Skip connections create **a mixture of models**
  - N skip connections → 2^N possible paths 潜在的模型集成
  - Each path could have up to N modules N：残差模块个数
  - We automatically get **a mixture of shallow GNNs and deep GNNs**

Path 2: skip this module



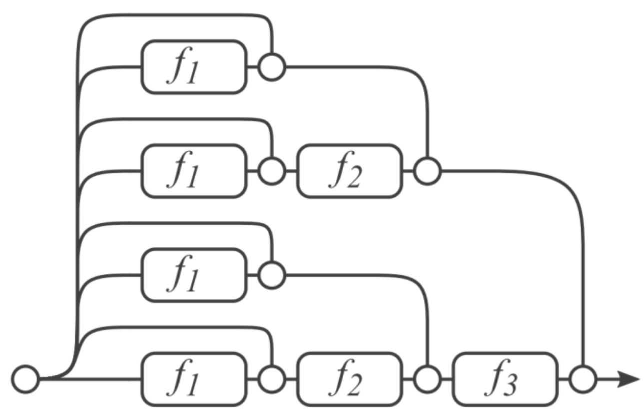
Path 1: include this module

(a) Conventional 3-block residual network

=

All the possible paths:

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



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

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

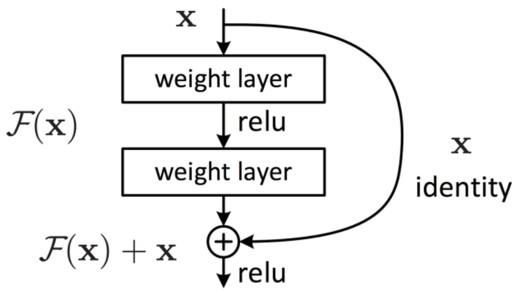
- **A GCN layer with skip connection**
- $\mathbf{h}_v^{(l)} = \sigma\left(\boxed{\sum_{u\in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}} + \boxed{\mathbf{h}_v^{(l-1)}}\right)$

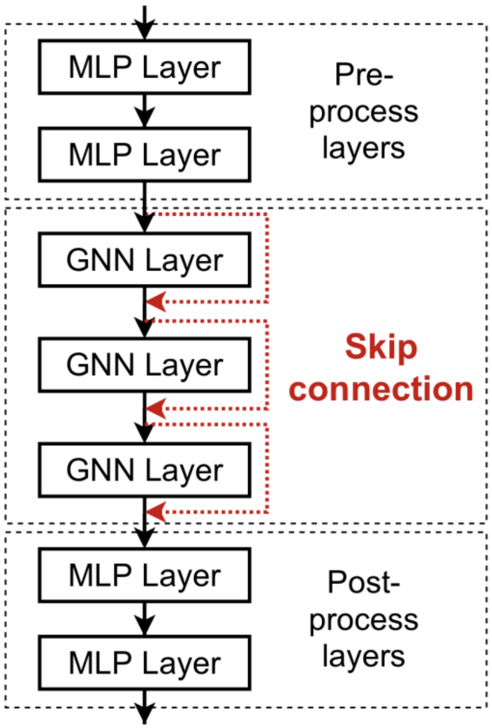
$F(\mathbf{x})$ + $\mathbf{x}$

残差

恒等映射

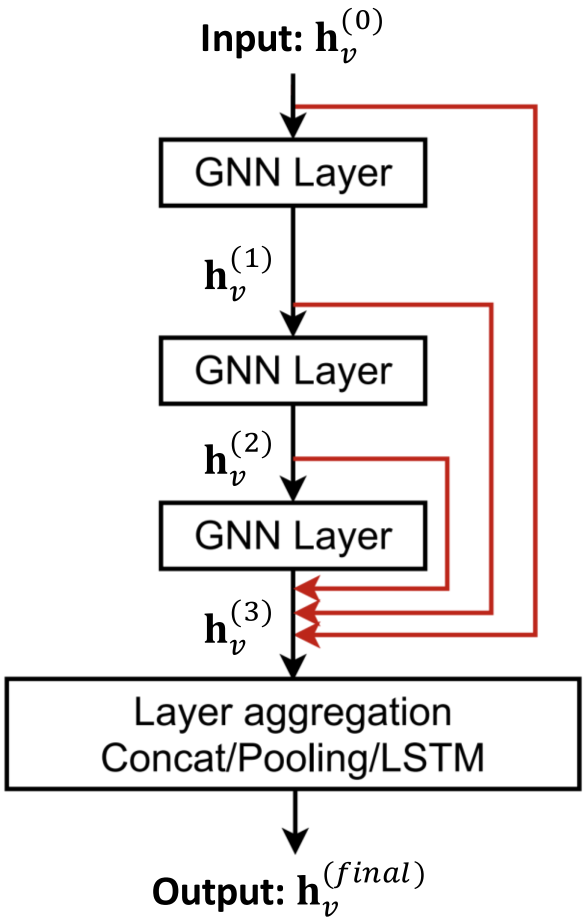
skip connection





Other Options of Skip Connections

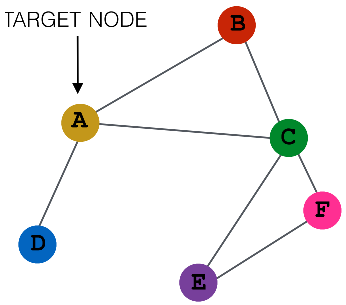
- **Other options:** Directly skip to the last layer
  - The final layer directly **aggregates from the all the node embeddings** in the previous layers



Stanford CS224W: Graph Manipulation in GNNs

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

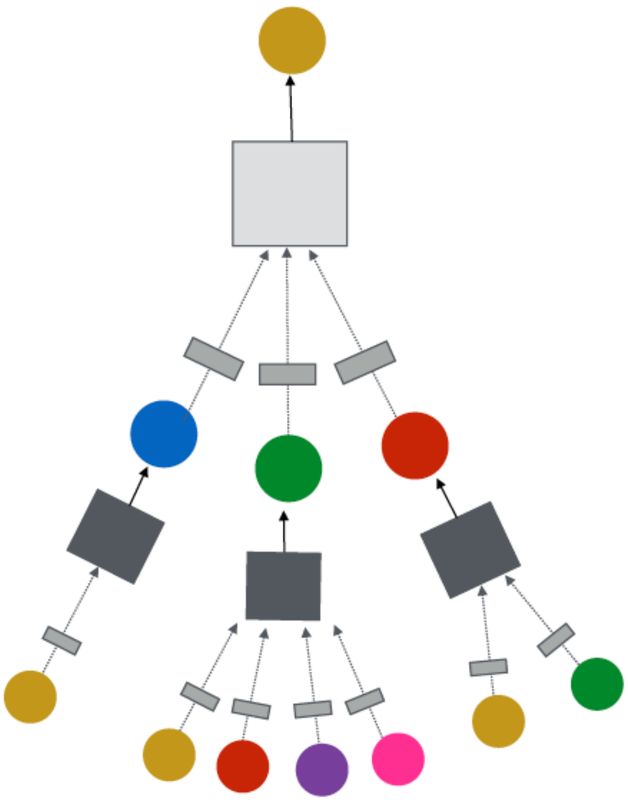
General GNN Framework



INPUT GRAPH

Idea: Raw input graph $\neq$ computational graph 计算图增强

- **Graph feature augmentation** 节点属性特征增强
- **Graph structure manipulation** 图结构增强



(4) Graph manipulation

Why Manipulate Graphs

Our assumption so far has been

- **Raw input graph = computational graph**

Reasons for breaking this assumption

- **Feature level:** 节点特征层面
  - The input graph **lacks features** → feature augmentation
- **Structure level:** 图结构层面
  - The graph is **too sparse** → inefficient message passing 社恐
  - The graph is **too dense** → message passing is too costly 大V
  - 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

原始的图不太可能恰好就是最优的计算图

Graph Manipulation Approaches

- **Graph Feature manipulation**
  - The input graph **lacks features** → **feature augmentation**
- **Graph Structure manipulation**
  - The graph is **too sparse** → **Add virtual 虚拟 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

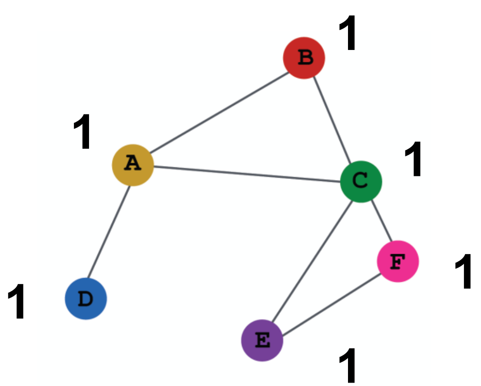
只有邻接矩阵

- **Standard approaches:**
- **a) Assign constant values to nodes**

常数

也可以是固定长度的常数向量

[1 1 1 1 1 1]

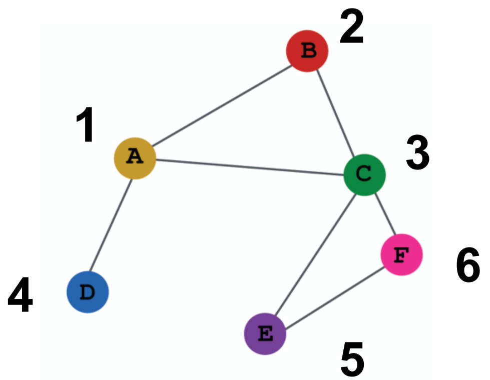


INPUT GRAPH

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** 唯一ID编号
  - These IDs are converted into **one-hot vectors**



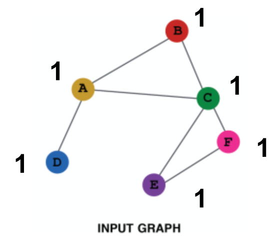
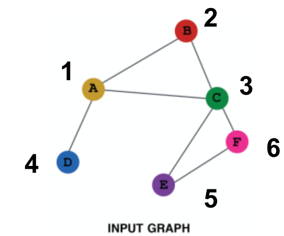
One-hot vector for node with ID=5

ID = 5

$$\underbrace{[0, 0, 0, 0, 1, 0]}_{\text{Total number of IDs} = 6}$$

Feature Augmentation on Graphs

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

	Constant node feature	One-hot node feature
		
Expressive power	**Medium**. All the nodes are 相同 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 无法泛化到新节点
Computational cost	**Low**. Only 1 dimensional feature 标量或固定长度的常数向量	**High**. High dimensional feature, cannot apply to large graphs one-hot
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

(Jiaxuan 2023)

Finished here.

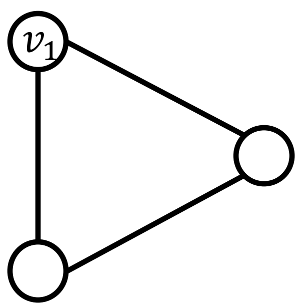
Good lecture overall. The slides for GAT is slightly repetitive, we could shrink 1 slide on the content there.

Feature Augmentation on Graphs

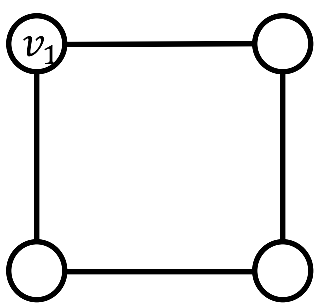
Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- **Example:** Cycle count feature 数节点所在环的长度 (分子结构分析)
  - Can GNN learn the length of a cycle that v_1 resides in?
  - **Unfortunately, no**

v_1 resides in a cycle with length 3



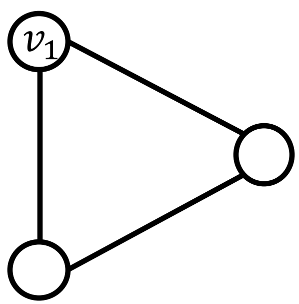
v_1 resides in a cycle with length 4



Feature Augmentation on Graphs

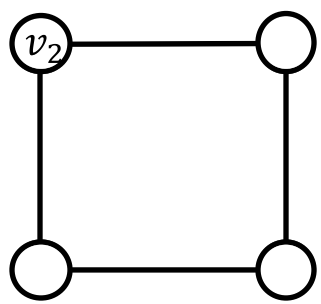
- v_1 **cannot differentiate which graph it resides in**
  - Because all the nodes in the graph have degree of 2 环中所有节点同形
    除非用节点属性特征区分
  - The computational graphs will be the same binary tree

v_1 resides in a cycle with length 3



三个节点同形

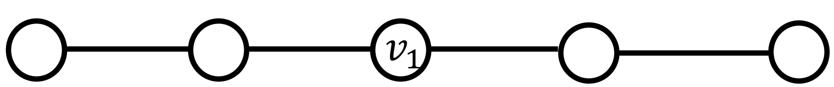
v_1 resides in a cycle with length 4



四个节点同形

v_1 resides in a cycle with infinite length

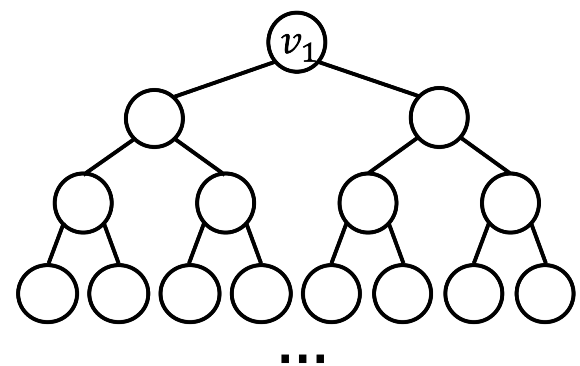
…



…

计算图完全相同(节点颜色相同)

The computational graphs for node v_1 are always the same



下一讲: GNN的表达能力

Feature Augmentation on Graphs

Why do we need feature augmentation?

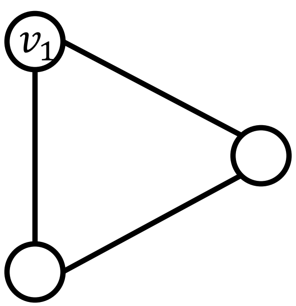
- **(2) Certain structures are hard to learn by GNN**
- **Solution:**
  人为补充信息至节点属性特征
  - We can use cycle count as augmented node features

We start from cycle with length 0

Augmented node feature for v_1

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

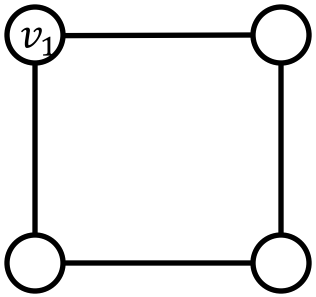
v_1 resides in a cycle with length 3



Augmented node feature for v_1

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

v_1 resides in a cycle with length 4



Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- Other commonly used augmented features:
  - **Clustering coefficient**
  - **PageRank**
  - **Centrality**
  - **...**

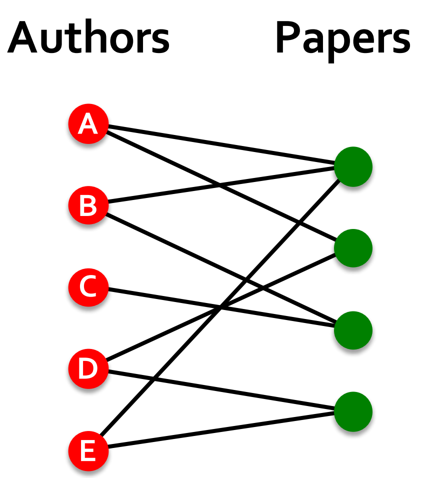
  Node Degree
- Any **feature we have introduced in Lecture 2 can be used!**

Add Virtual Nodes / Edges

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

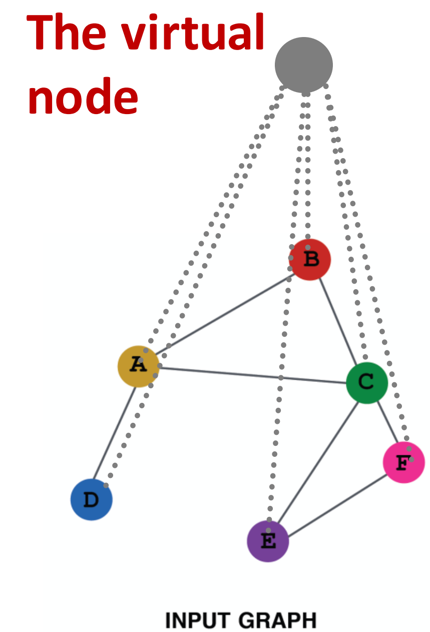
邻居 (AA^T)
邻居的邻居

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



Add Virtual Nodes / Edges

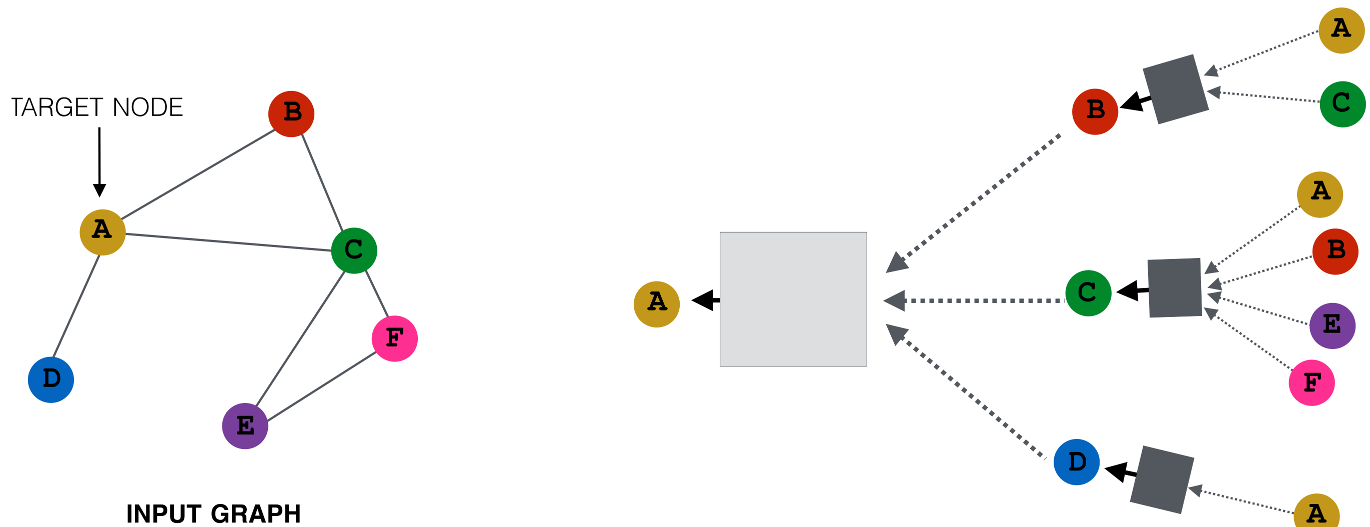
- **Motivation:** Augment sparse graphs
- **(2) Add virtual nodes**
  - The virtual node will connect to all the nodes in the graph
    - Suppose in a sparse graph, two nodes have shortest path distance of 10
    - After adding the virtual node, **all the nodes will have a distance of 2**
      - **Node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



见同济子豪兄GraphSAGE论文精读

Node Neighborhood Sampling

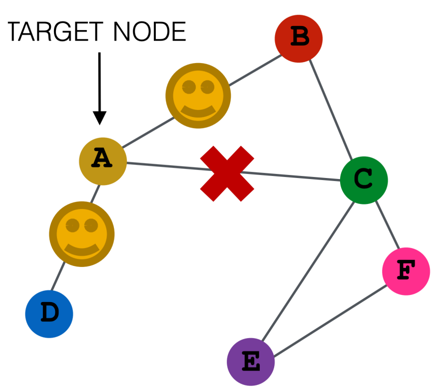
- **Previously:**
  - All the nodes are used for message passing



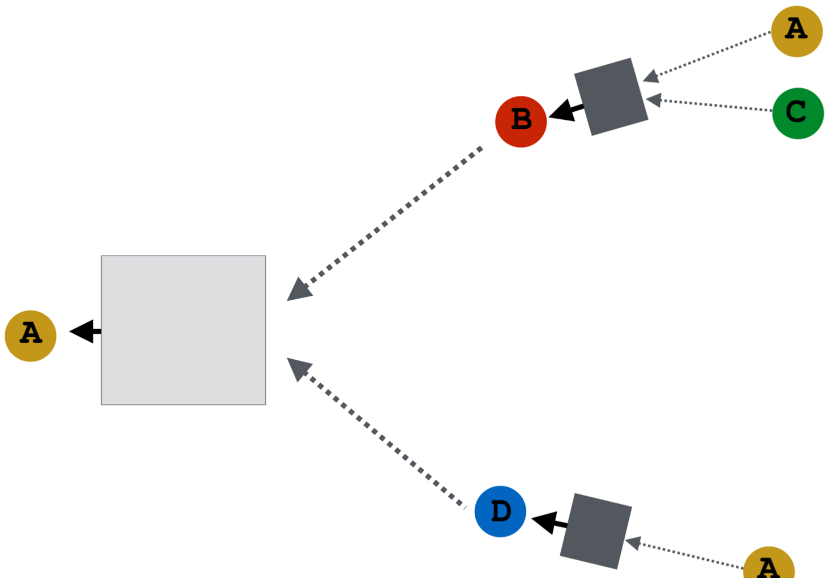
- **New idea:** (Randomly) sample a node's neighborhood for message passing

Neighborhood Sampling Example

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

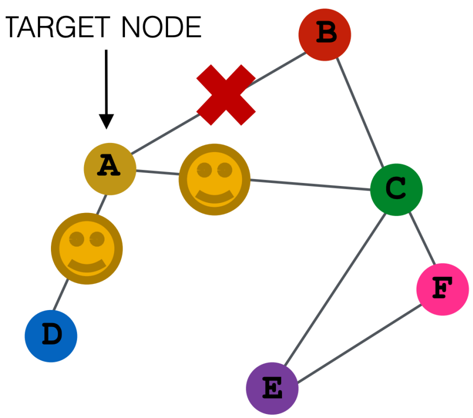


INPUT GRAPH

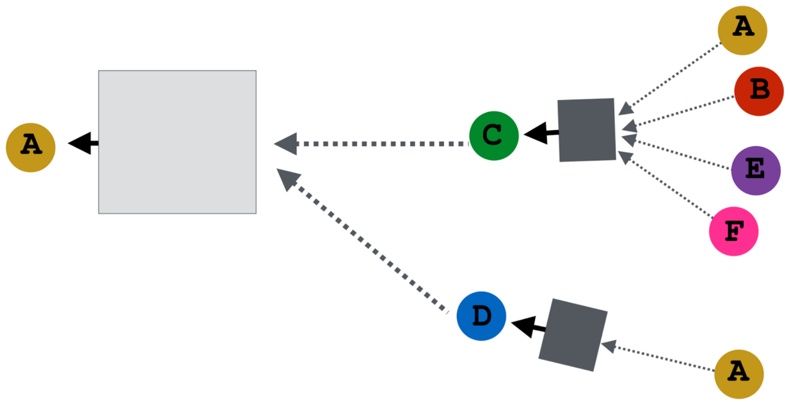


Neighborhood Sampling Example

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



INPUT GRAPH



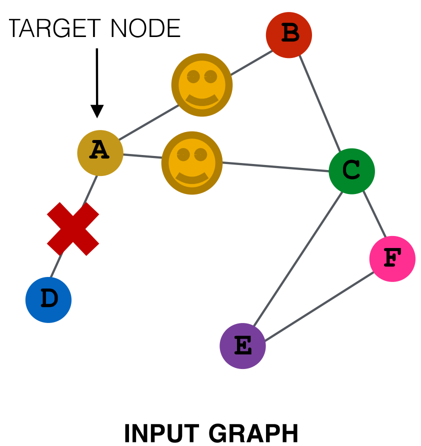
Neighborhood Sampling Example

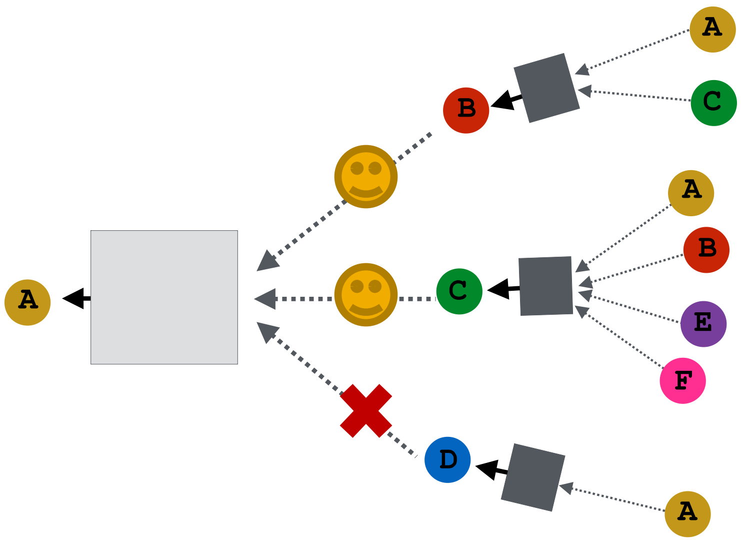
每个Epoch采样不同的邻域

期望

- 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!

扩展到大规模图(推荐系统)





Summary of the lecture

- **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
- **Next:** GNN objectives, GNN in practice