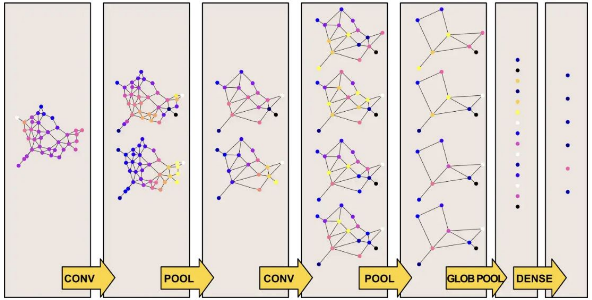


Graph Convolution Networks

Great things are not done by impulse, but by a series of small things brought together.

- Vincent Van Gogh

Jay Urbain, Ph.D.
Electrical Engineering and Computer Science Department
Milwaukee School of Engineering

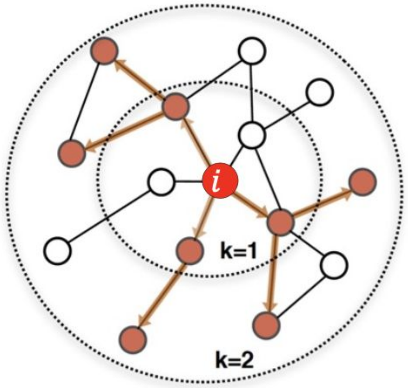
Credits

- **Graph Neural Networks: A review of Methods and Applications, Zhou, 2018.**
- **Graph Representation Learning, Hamilton, 2020.**
- **Graph Convolutional Networks, Kipf and Welling, 2016.**
- Multi-Layer Perceptron as Aggregator, Zaheer, 2017.
- Graph Attention Networks, Velickovic, 2017.
- Gated Graph Neural Networks, Li, 2015.
- DeepFindr
- Michael Bronstein, Oxford, Geometric Deep Learning Course
- Jure Lesvovec, Stanford

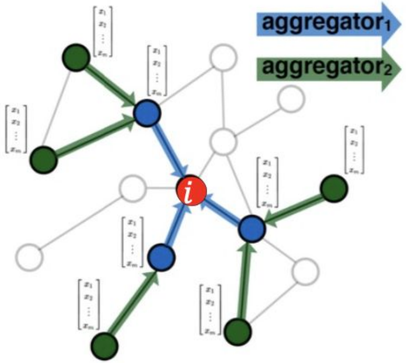
Review: Graph Neural Networks

Idea:

- Node's neighborhood defines a computation graph.
- Propagate information across the graph to compute node features



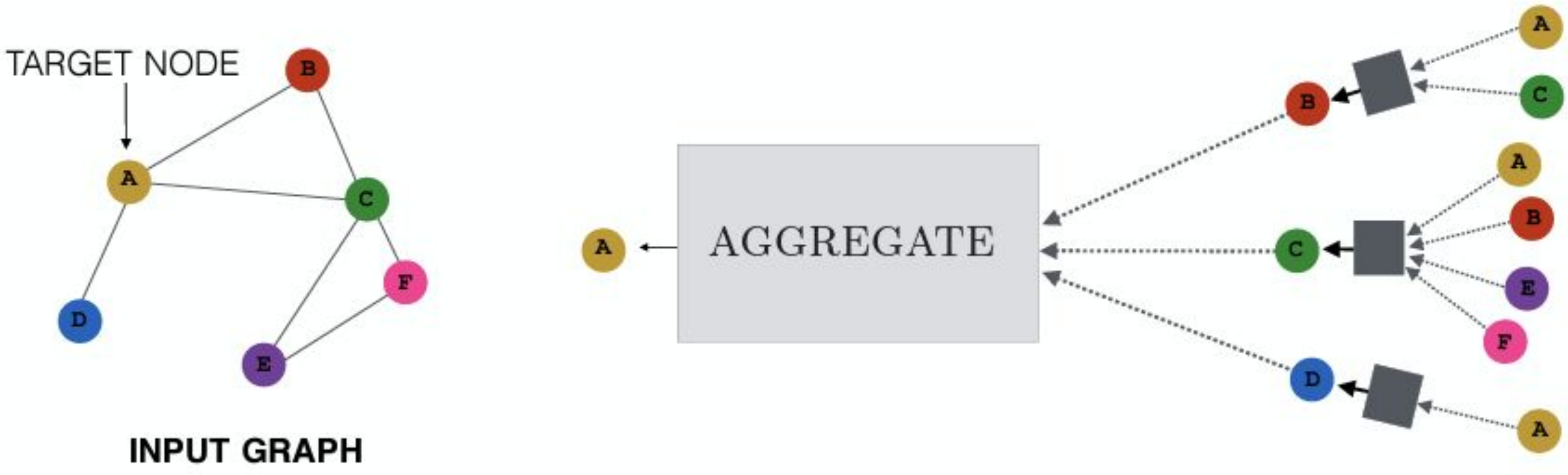
Determine node computation graph



Propagate and transform information

Review: Aggregate from neighbors

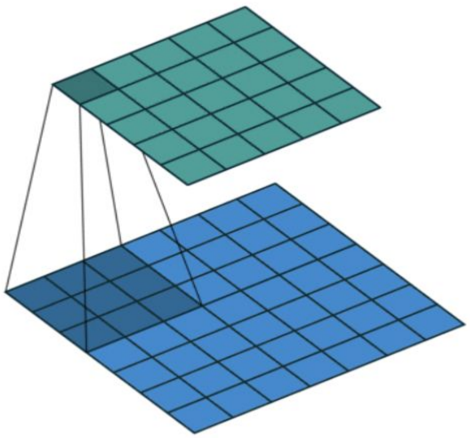
- Nodes aggregate information from their neighbors using neural networks.
- Every node defines a computation graph based on its neighborhood!

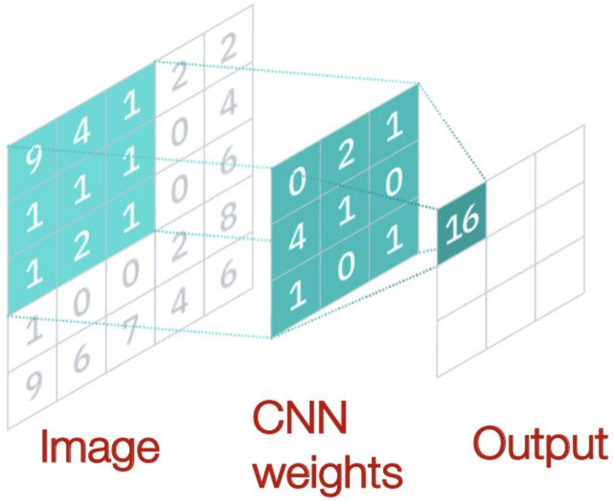


Why GNNs generalize other neural networks

- Convolutional neural network (CNN) layer with 3x3 filter:

Within the same layer, the same filter will be used throughout image, this is referred to as ***weight sharing***.

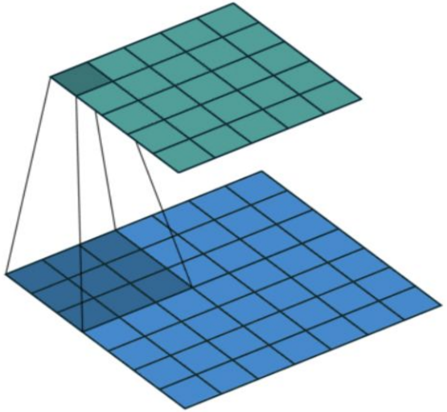


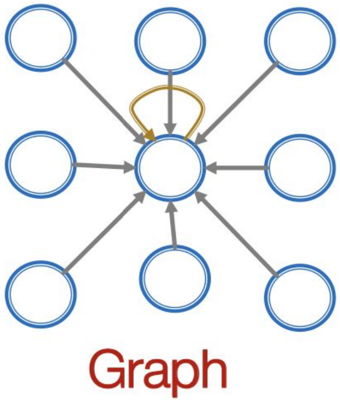


CNN formulation: $h_v^{(l+1)} = \sigma(\sum_{u \in N(v) \cup \{v\}} W_l^u h_u^{(l)}),\ \ \forall l \in \{0, \dots, L-1\}$

GNN versus CNN

- Convolutional neural network (CNN) layer with 3x3 filter:





Node order equivariance: graphs often have no inherent ordering present amongst the nodes.

Compare this to images, where every pixel is uniquely determined by its absolute position within the image!

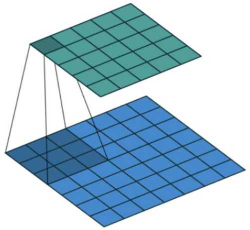
GNN formulation (previous slide): $\mathrm{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u\in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \dots, L-1\}$

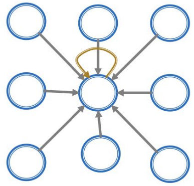
CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u\in \mathrm{N}(v)\cup\{v\}} \mathrm{W}_l^u \mathrm{h}_u^{(l)}), \forall l \in \{0, \dots, L-1\}$

if we rewrite: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u\in \mathrm{N}(v)} \mathbf{W}_l^u \mathrm{h}_u^{(l)} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \dots, L-1\}$

GNN versus CNN

- Convolutional neural network (CNN) layer with 3x3 filter.
- Key difference: We can learn different $\mathbf{W}_l^u$ for each "neighbor" u for pixel v on the image. The reason: we can pick an ordering for the 9 neighbors using relative position to the center pixel: {(-1,-1). (-1,0), (-1, 1), ..., (1, 1)}





GNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \ldots, L-1\}$

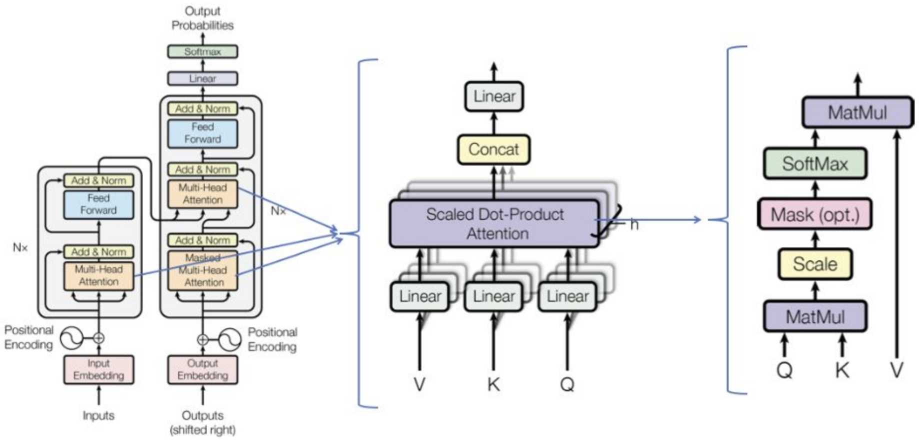
CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v)} \mathbf{W}_l^u \mathrm{h}_u^{(l)} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \ldots, L-1\}$

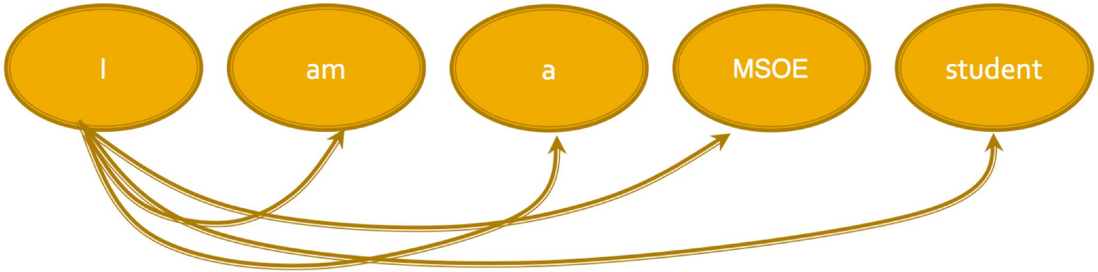
GNN versus CNN

- CNN can be seen as a special GNN with fixed neighbor size and ordering.
- The size of the filter is pre-defined for a CNN.
- The advantage of GNN is it processes arbitrary graphs with different degrees for each node.
- CNN is not permutation invariant:
    - Switching the pixels changes the results.

Transformer

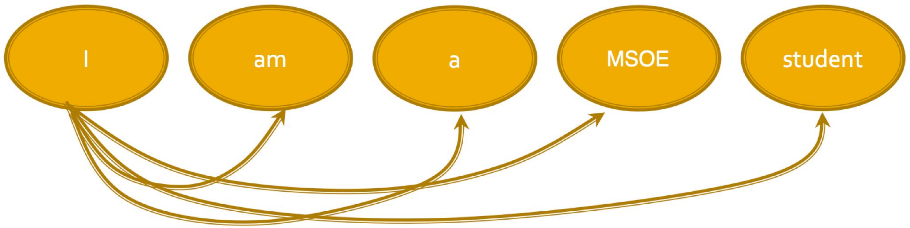
- Transformer is one of the most popular architectures that achieves great performance in many sequence modeling tasks.
- Key component: ***self-attention***.
    - Every word “attends” to every other word via matrix calculation.

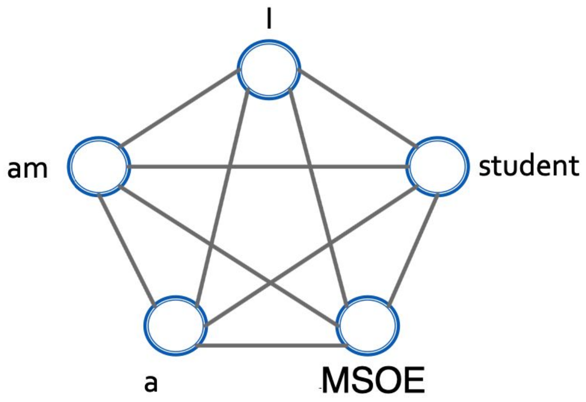




Transformer

- Transformer layer can be seen as a special GNN that runs on a fully connected word graph.
- Since each word attends to all the other words, the computation graph of a transformer layer is identical to that of a GNN on the fully-connected word graph.

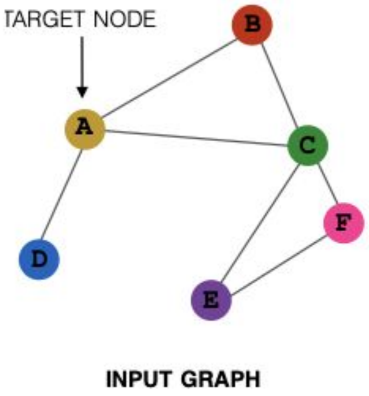


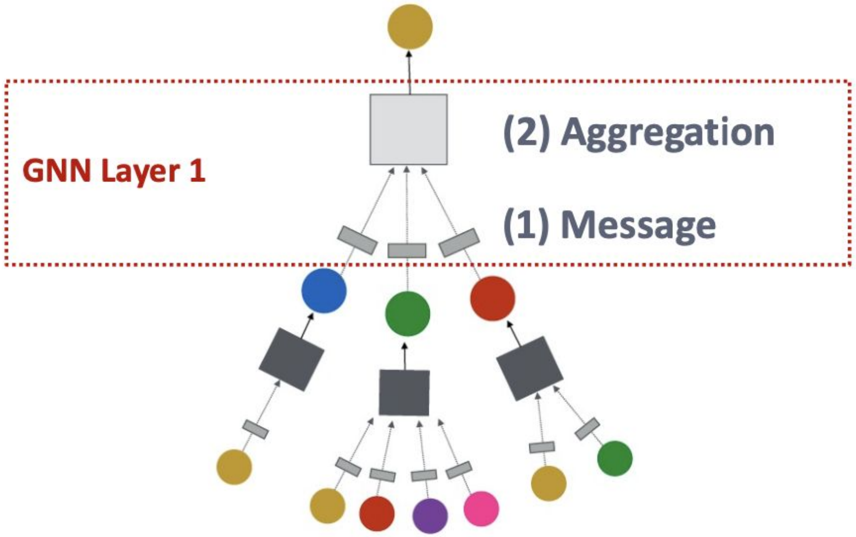


(Complete) Graph

A GNN Layer

- GNN Layer = Message + Aggregation
- Different instantiations under this perspective: GCN, GraphSage, GAT, etc.

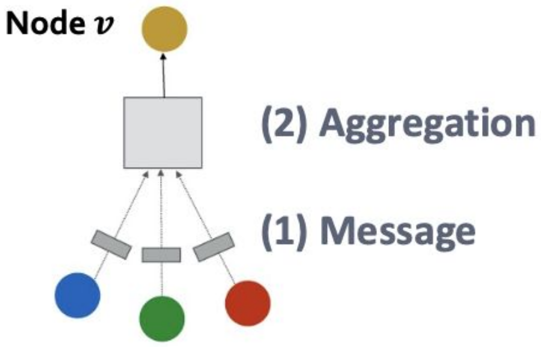


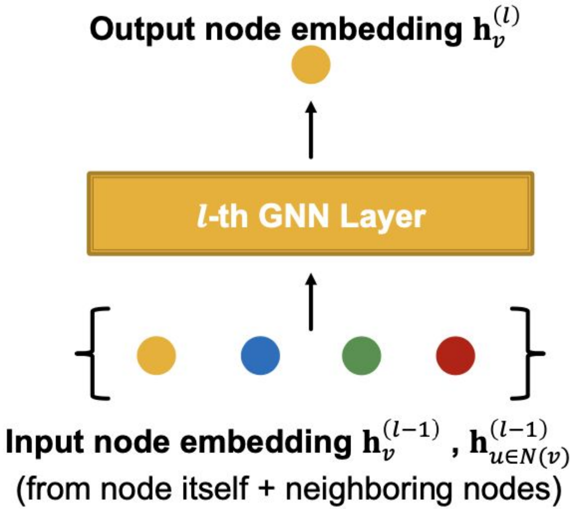


A GNN Layer

Idea of a GNN Layer:

- Compress a set of vectors into a single vector
- Two-step process: **Message** and **Aggregation**





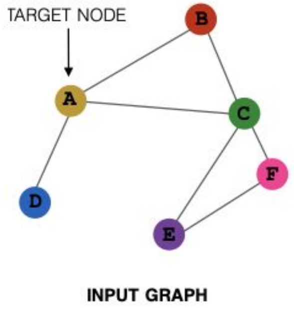
Message Computation

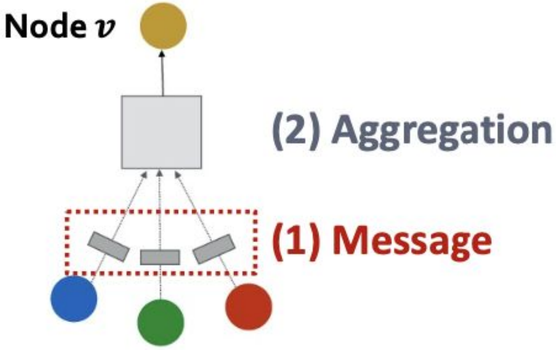
- Each node will create a message, which will be sent to other nodes later

$$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$$

- Example: Linear layer multiply node features with weight matrix.

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





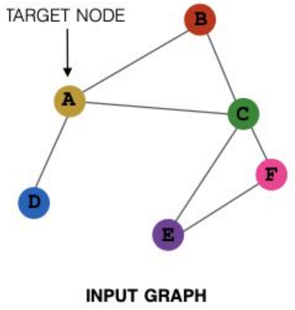
Message Aggregation

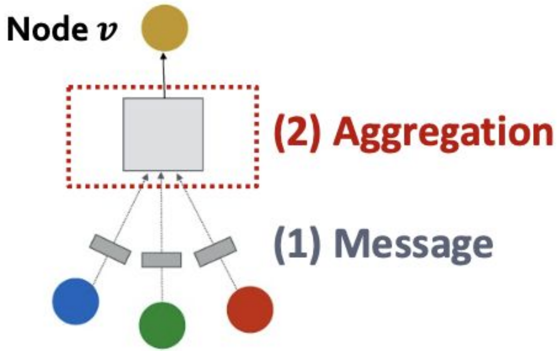
- Each node will aggregate the messages from node v's neighbors

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

- Example: Sum(·), Mean(·) or Max(·) aggregator.

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





Message Aggregation Issues

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 $\mathbf{h}_v^{(l-1)}$ computing $\mathbf{h}_v^{(l)}$

- Message: compute message from node v itself
- Aggregation: After aggregating from neighbors, we can aggregate the message from node v itself

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

Single GNN Layer Summary

Message: each node computes a message

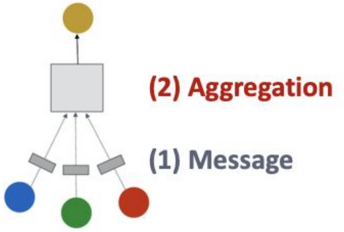
$$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)\ \mathbf{h}_v^{(l-1)}$$

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/non-linear learning

- Can be added to message or aggregation
- $\sigma(\cdot)$: ReLU($\cdot$), Sigmoid($\cdot$) , ...



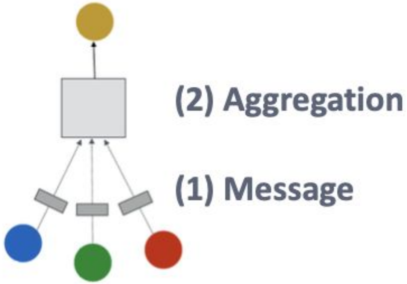
Classic GCN Layer - 1

Graph Convolutional Networks (GCN)

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

Aggregation: aggregate messages from neighbors

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



Classic GCN Layer - how to write as message

Graph Convolutional Networks (GCN)

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

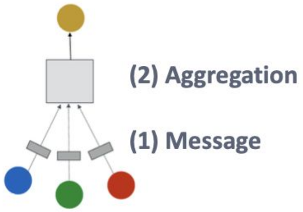
Message: each neighbor normalizes by node degree

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

Aggregation: sum over messages from neighbors, then apply activation.

GCN graph assumes self-edges that are included in calculation.

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



GraphSAGE

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

Message is computed within the AGG()

Two-stage aggregation

- Stage 1: Aggregate from node neighbors

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

- Stage 2: Further aggregate over the node itself

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

GraphSAGE Neighbor Aggregation

Mean: take weighted average of neighbors

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

Aggregation

Message computation

Pool: Transform neighbor vectors and apply symmetric vector function Mean(·) or Max(·)

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

Aggregation

Message computation

LSTM: Apply LSTM to reshuffled neighbors

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

Aggregation

GraphSAGE L2 Normalization

- Without L2 normalization, the embedding vectors have different scales.
- In some cases (not always), normalization of embedding results in performance improvement.
- After L2 normalization, all vectors will have the same L2-norm.

$$\mathbf{h}_v^{(l)} \leftarrow \frac{\mathbf{h}_v^{(l)}}{\left\|\mathbf{h}_v^{(l)}\right\|_2} \;\forall v \in V \text{ where } \|u\|_2 = \sqrt{\textstyle\sum_i u_i^2} \;\; (\ell_2\text{-norm})$$

Adding Attention

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

Weighting factor (importance) of node u's message to node v. Node degree.

$$\alpha_{vu} = \frac{1}{|N(v)|}$$

Graph Attention Network

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

Attention is inspired by cognitive attention.

The attention $\boldsymbol{\alpha}_{vu}$ focuses on the important parts of the input data and fades out the rest.

- Idea: the NN should devote more computing power on that small but important part of the data.
- Which part of the data is more important depends on the context and is learned through training.

Graph Attention Network

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

Attention weights

Can we do better than simple neighborhood aggregation?

Can we let weighting factors α_{vu} be learned?

Goal: Learn importance to different neighbors of each node in the graph based on context.

Compute embedding h 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

Compute attention coefficients ***e*** across pairs of nodes ***u, v*** based on their messages.

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

e_{vu} indicates the importance of u's message to node v

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