Graph Neural Networks

https://web.stanford.edu/class/cs224w/slides/03-GNN1.pdf
https://web.stanford.edu/class/cs224w/slides/04-GNN2.pdf

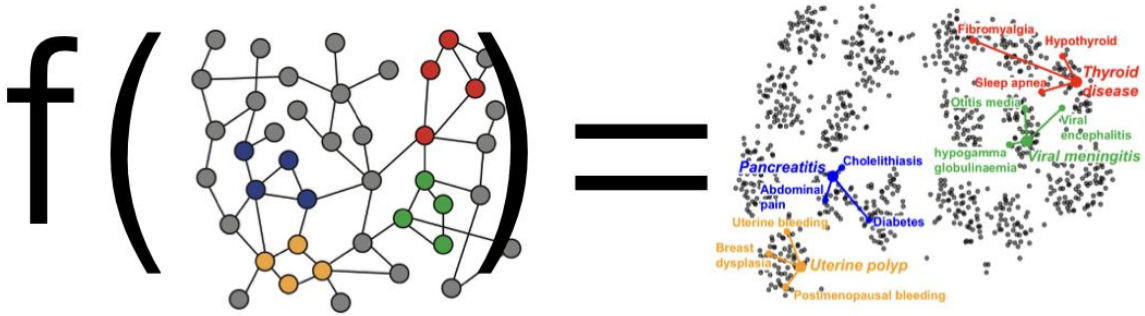
Caltech EAS AI BOOTCAMP

Overview

1. Introduction and Motivation
2. Graph Neural Networks
3. Comparison with ConvNets
4. Deeper Dive

Recap: Node embeddings

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



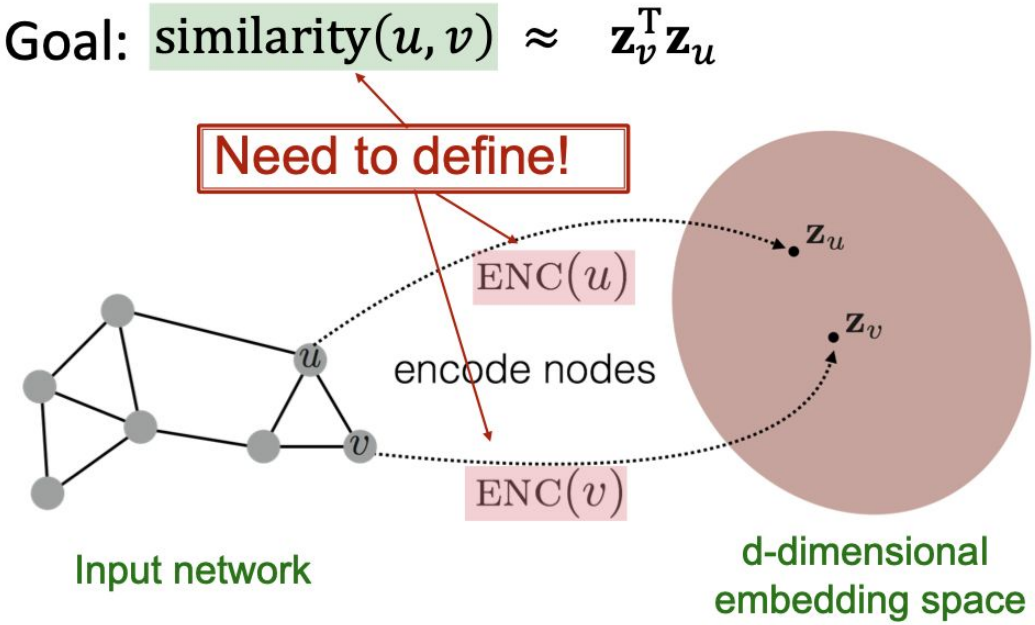
Input graph

2D node embeddings

How to learn mapping function f?

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Recap: Node embeddings



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Recap: Shallow encoders

- **Limitations** of shallow embedding methods:
  - $O(|V|d)$ **parameters are needed:**
    - No sharing of parameters between nodes
    - Every node has its own unique embedding
  - **Inherently "transductive":**
    - Cannot generate embeddings for nodes that are not seen during training
  - **Do not incorporate node features:**
    - Nodes in many graphs have features that we can and should leverage

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Part 1: Motivation

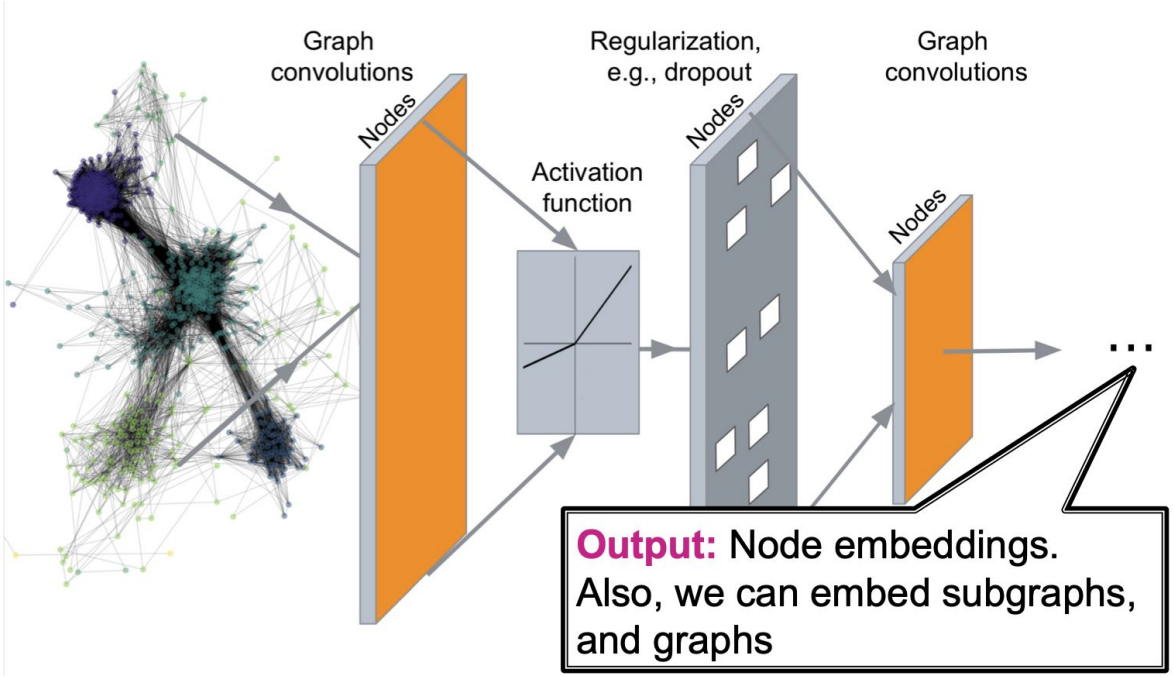
Today: Deep Graph Encoders

- **Today:** We will now discuss deep learnig methods based on **graph neural networks (GNNs):**

$$\mathrm{ENC}(v) =$$ **multiple layers of non-linear transformations based on graph structure**

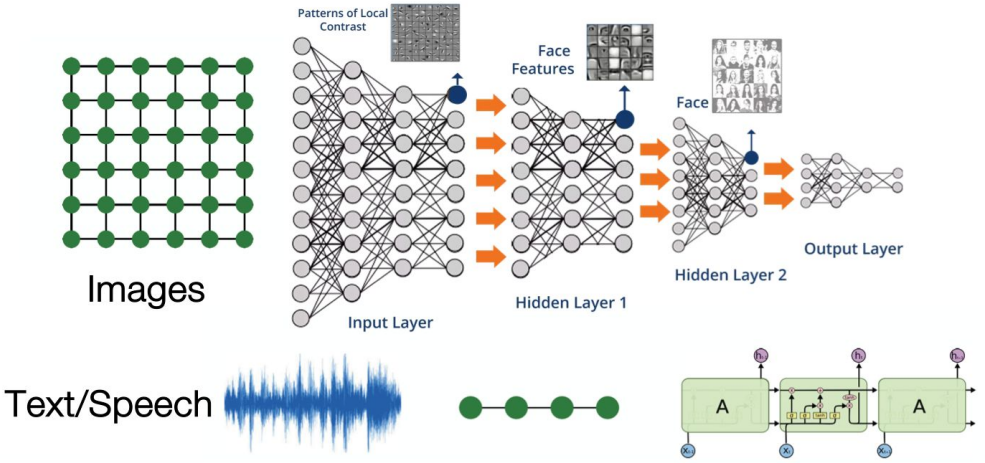
Taken from Stanford CS224W course: http://cs244w.stanford.edu

Deep Graph Encoders



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Modern ML toolbox



Modern deep learning toolbox is designed for simple sequences & grids

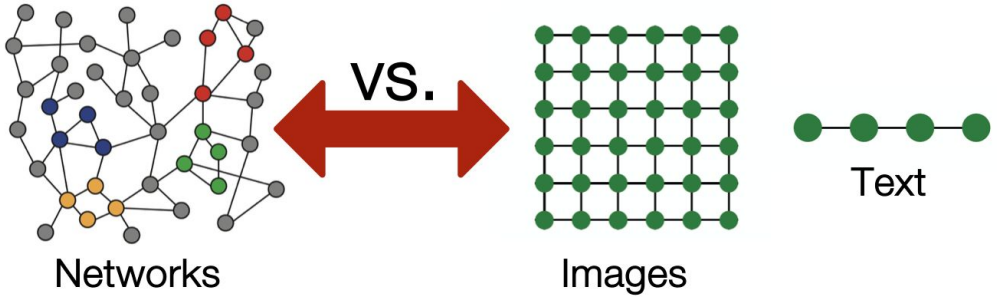
Taken from Stanford CS224W course: http://cs244w.stanford.edu

Caltech EAS AI BOOTCAMP

Why is it hard?

But networks are far more complex!

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



- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Taken from Stanford CS224W course: http://cs244w.stanford.edu

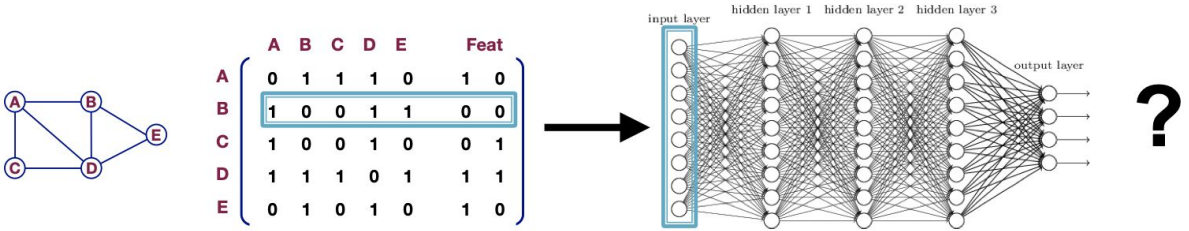
Definitions

- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{|V| \times m}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: User profile, User image
    - Biological networks: Gene expression profiles, gene functional information

Taken from Stanford CS224W course: http://cs244w.stanford.edu

A naive approach

- Join adjacency matrix and features
- Feed them into a deep neural net:

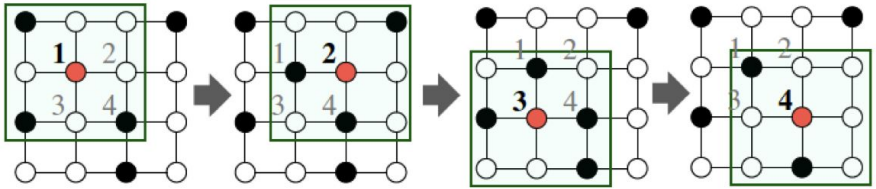


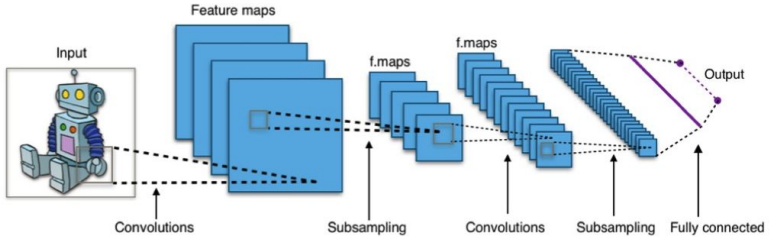
- Issues with this idea:
  - $O(|V|)$ parameters
  - Not applicable to graphs of different sizes
  - Sensitive to node ordering

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Idea: Convolutional Neural Networks

CNN on an image:





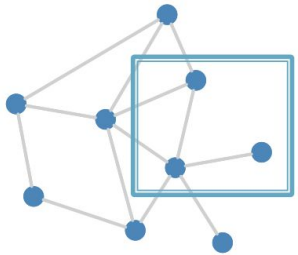
Goal is to generalize convolutions beyond simple lattices

Leverage node features/attributes (e.g., text, images)

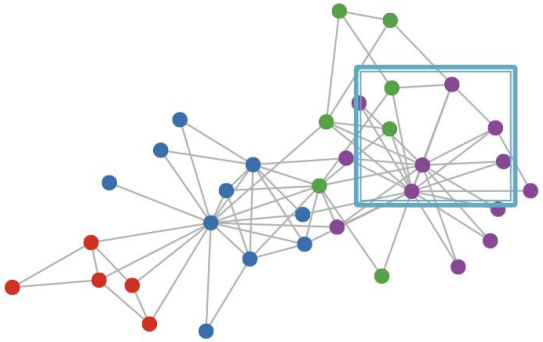
Taken from Stanford CS224W course: http://cs244w.stanford.edu

Real-World Graphs

But our graphs look like this:



or this:

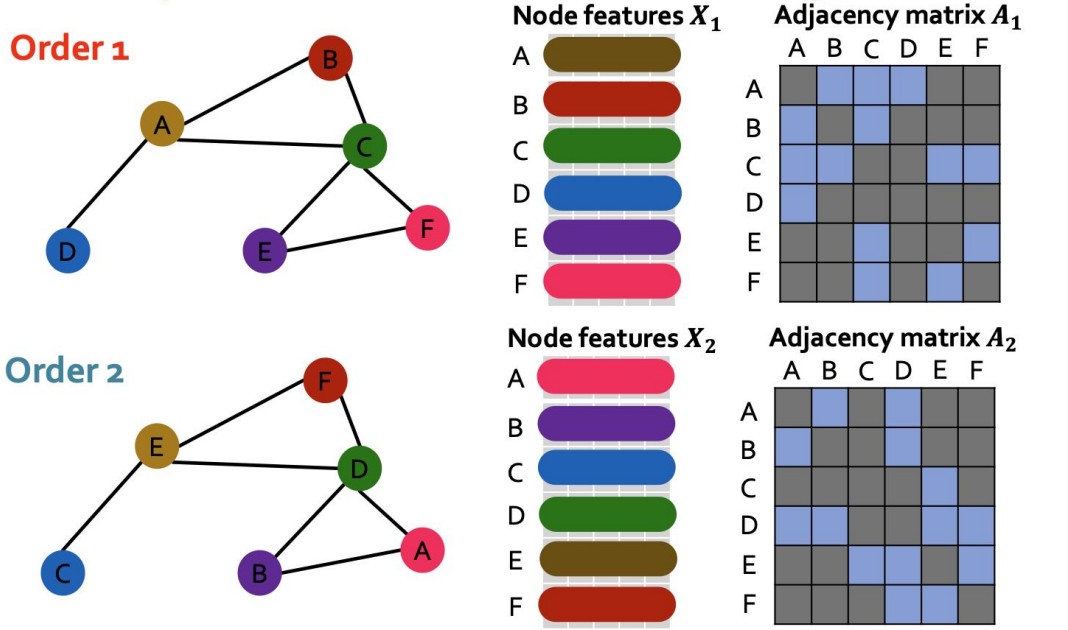


- There is no fixed notion of locality or sliding window on the graph
- Graph is **permutation invariant**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Permutation Invariance

- **Graph does not have a canonical ordering of the nodes!**



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Caltech EAS AI BOOTCAMP

Permutation Invariance

What do we mean by "graph representation is the same for two orderings"?

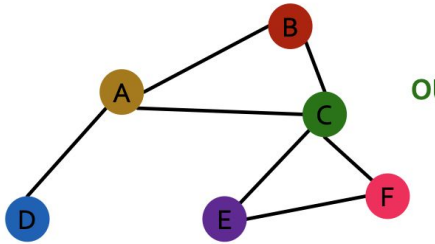
In other words, f maps a graph to a d-dim embedding

- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a vector $\mathbb{R}^d$ then

$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) = f(\boldsymbol{A}_2, \boldsymbol{X}_2)$$

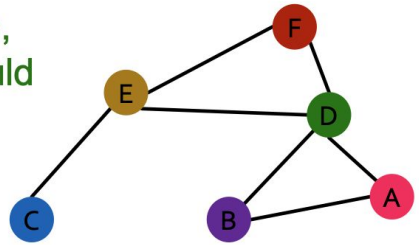
$\boldsymbol{A}$ is the adjacency matrix
$\boldsymbol{X}$ is the node feature matrix

Order 1: $\boldsymbol{A}_1, \boldsymbol{X}_1$



For two orders, output of f should be the same!

Order 2: $\boldsymbol{A}_2, \boldsymbol{X}_2$



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Invariance vs. Equivariance

- **Permutation-invariant**

$$f(A, X) = f(PAP^T, PX)$$

Permute the input, the output stays the same. (map a graph to a vector)

- **Permutation-equivariant**

$$Pf(A, X) = f(PAP^T, PX)$$

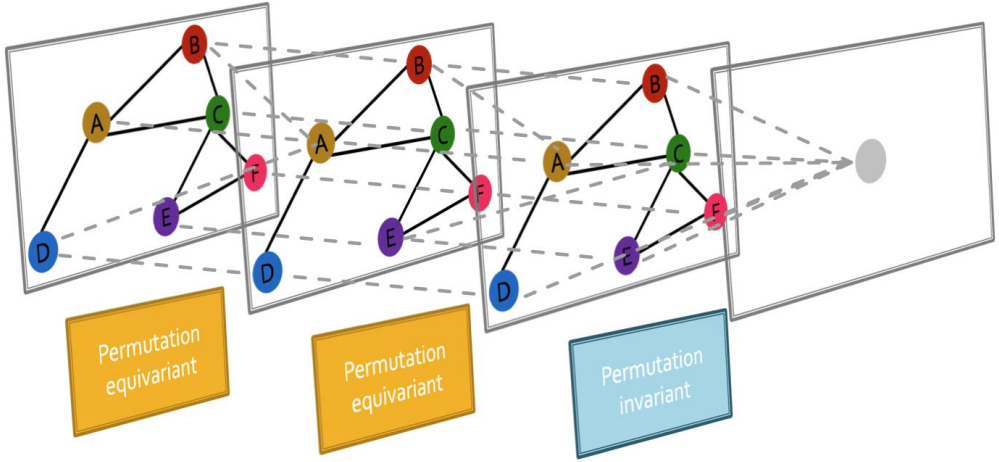
Permute the input, output also permutes accordingly. (map a graph to a matrix)

- **Examples:**
  - $f(A, X) = 1^T X$: Permutation-**invariant**
    - Reason: $f(PAP^T, PX) = 1^T PX = 1^T X = f(A, X)$
  - $f(A, X) = X$: Permutation-**equivariant**
    - Reason: $f(PAP^T, PX) = PX = Pf(A, X)$

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Invariance and Equivariance defining characteristics of Graphs

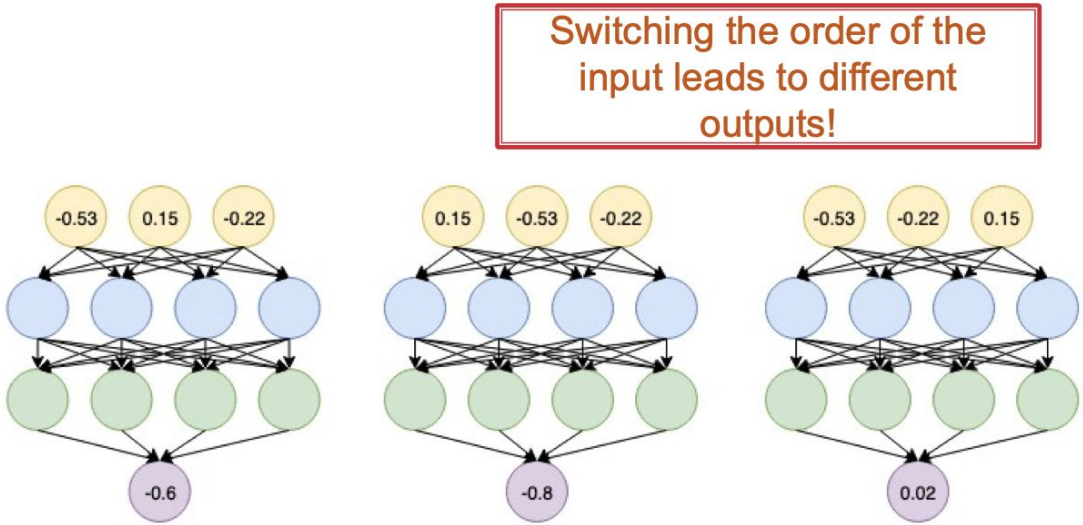
- **Graph neural networks consist of multiple permutation equivariant / invariant functions.**



Taken from Stanford CS224W course: http://cs244w.stanford.edu

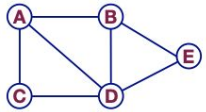
Are MLPs invariant/equivariant?

- **No.**

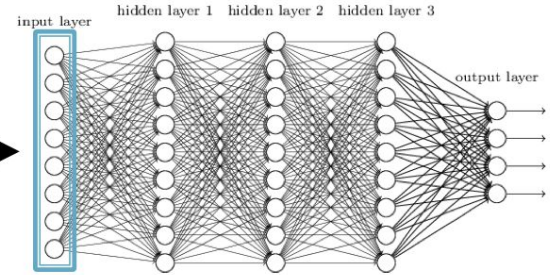


Taken from Stanford CS224W course: http://cs244w.stanford.edu

Are MLPs invariant/equivariant?



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



?

This explains why **the naïve MLP approach fails for graphs**!

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Caltech EAS AI BOOTCAMP

Are MLPs invariant/equivariant?

Next: Design graph neural networks that are permutation invariant / equivariant by passing and aggregating information from neighbors!

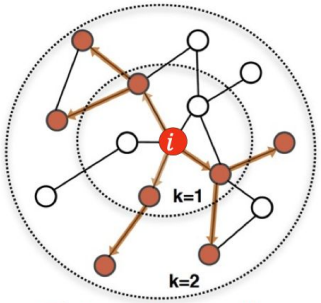
Taken from Stanford CS224W course: http://cs244w.stanford.edu

Part 2: Graph Neural Networks (GNNs)

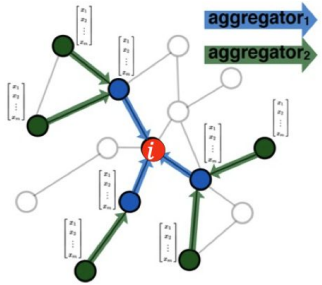
Graph Convolutional Neural Networks (GCNs)

Graph Neural Networks

Idea: Node's neighborhood defines a computation graph



Determine node computation graph



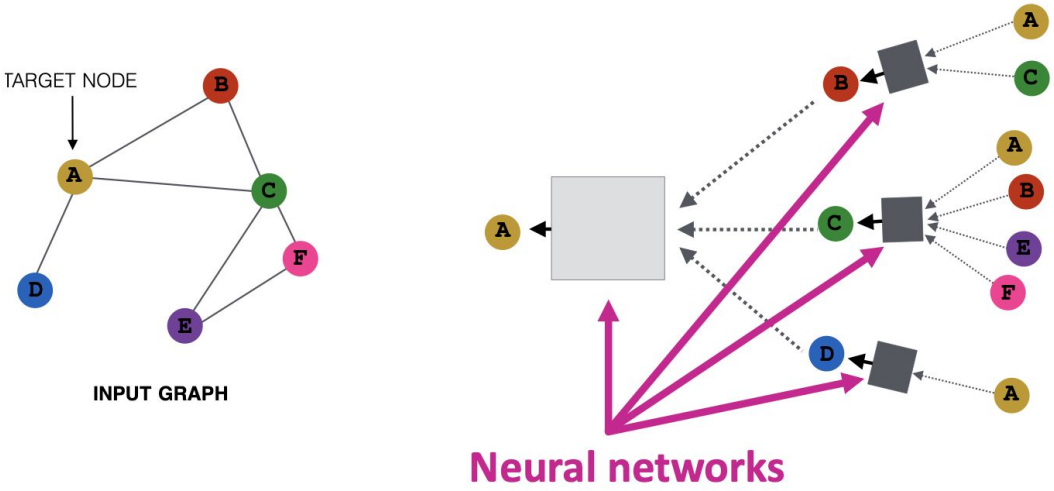
Propagate and transform information

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

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Idea: Aggregate Neighbors

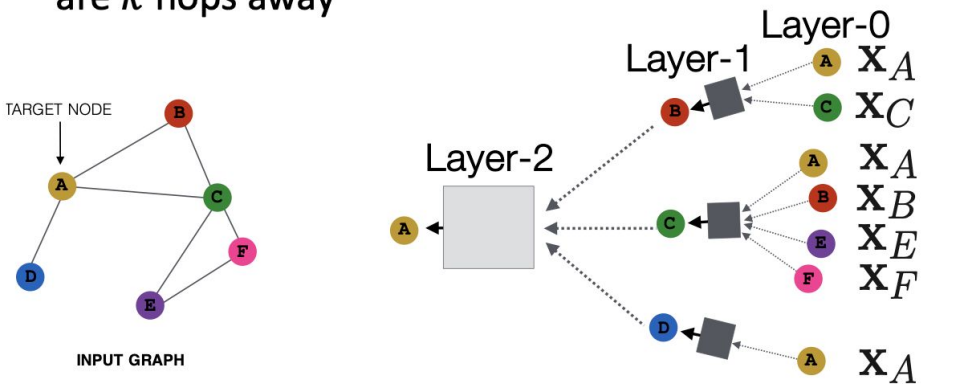
- **Intuition:** Nodes aggregate information from their neighbors using neural networks



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Deep Model: Many Layers

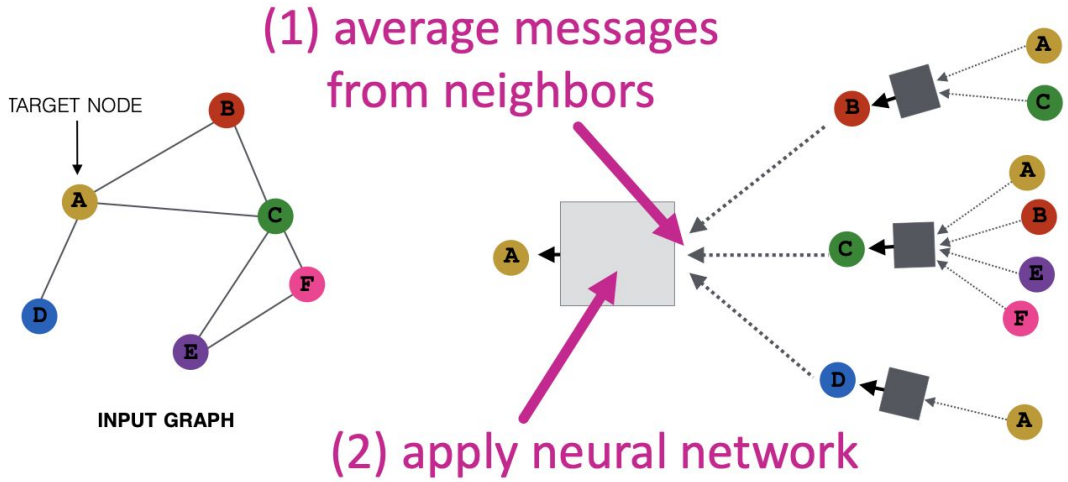
- Model can be of arbitrary depth:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node v is its input feature, x_v
  - Layer-k embedding gets information from nodes that are k hops away



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Neighborhood Aggregation

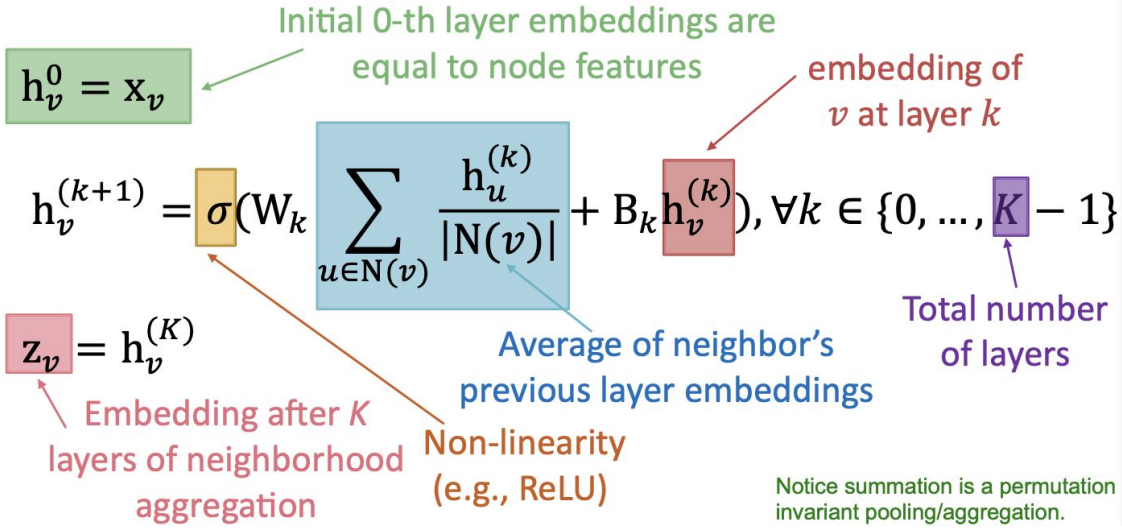
- **Basic approach:** Average information from neighbors and apply a neural network



Taken from Stanford CS224W course: http://cs244w.stanford.edu

The Math: Deep Encoder of a GNN

- **Basic approach:** Average neighbor messages and apply a neural network



Taken from Stanford CS224W course: http://cs244w.stanford.edu

How to train a GNN

- Node embedding z_v is a function of input graph
- **Supervised setting**: We want to minimize loss $\mathcal{L}$:

$$\min_{\Theta} \mathcal{L}(y, f_\Theta(z_v))$$

  - y: node label
  - $\mathcal{L}$ could be L2 if y is real number, or cross entropy if y is categorical
- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Unsupervised Training

- **One possible idea: "Similar" nodes have similar embeddings:**

$$\min_{\Theta} \mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}(y_{u,v}, \mathrm{DEC}(z_u, z_v))$$

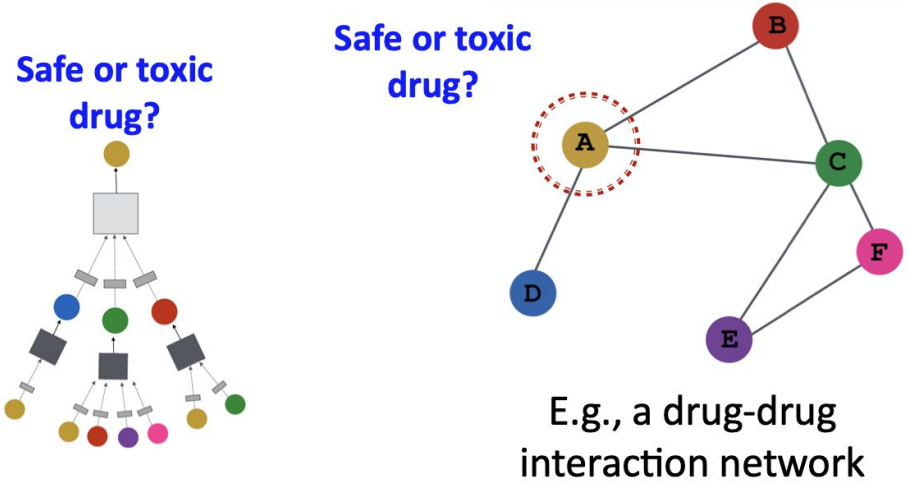
  - where $y_{u,v} = 1$ when node u and v are **similar**
  - $z_u = f_\Theta(u)$ and $\mathrm{DEC}(\cdot,\cdot)$ is the dot product
- CE is the cross entropy loss:
  - $\mathrm{CE}(\boldsymbol{y}, f(\boldsymbol{x})) = -\sum_{i=1}^{C}(y_i \log f_\Theta(x)_i)$
    - y_i and $f_\Theta(x)_i$ are the **actual** and **predicted** values of the i-th class.
    - **Intuition:** the lower the loss, the closer the prediction is to one-hot

Node similarity: for example random walk or matrix factorization or node proximity

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Supervised Training

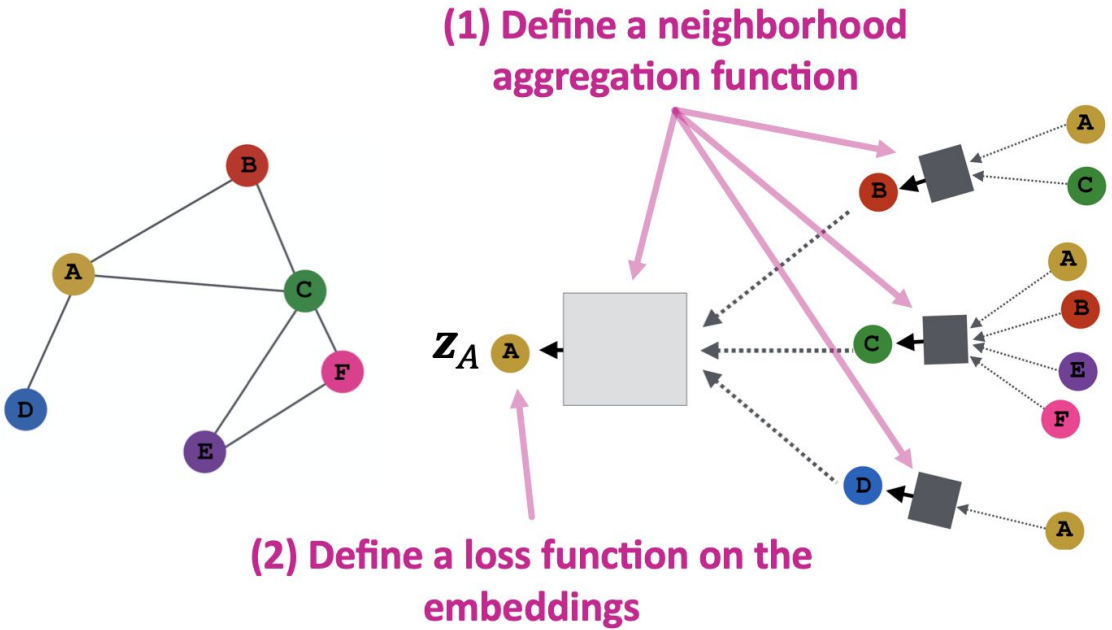
Directly train the model for a supervised task (e.g., **node classification**)



E.g., a drug-drug interaction network

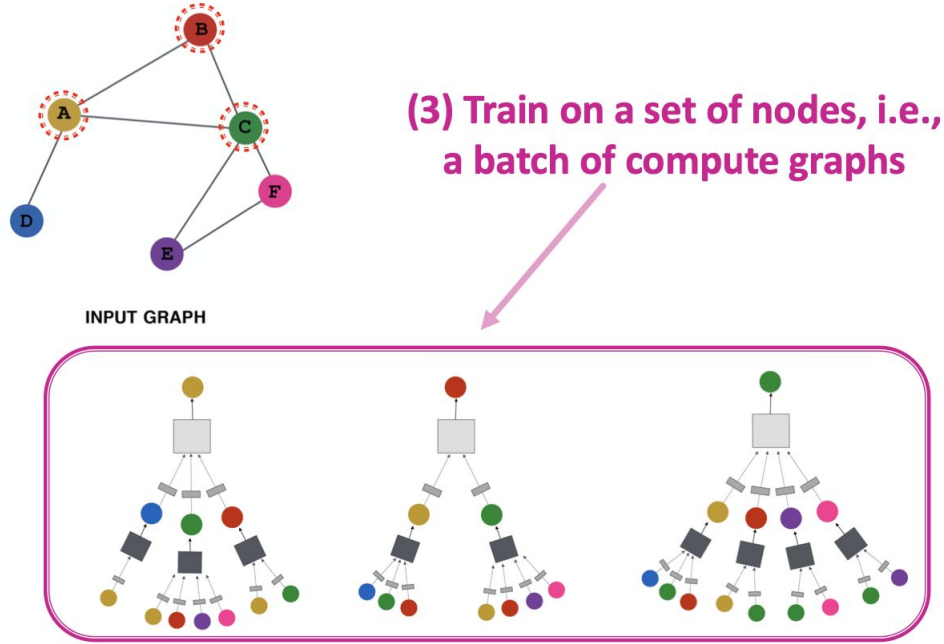
Taken from Stanford CS224W course: http://cs244w.stanford.edu

Training Overview



Taken from Stanford CS224W course: http://cs244w.stanford.edu

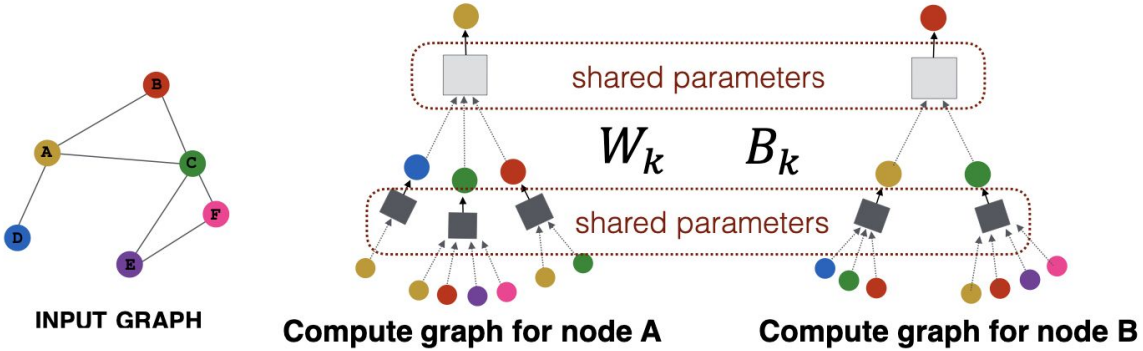
Training Overview



Taken from Stanford CS224W course: http://cs244w.stanford.edu

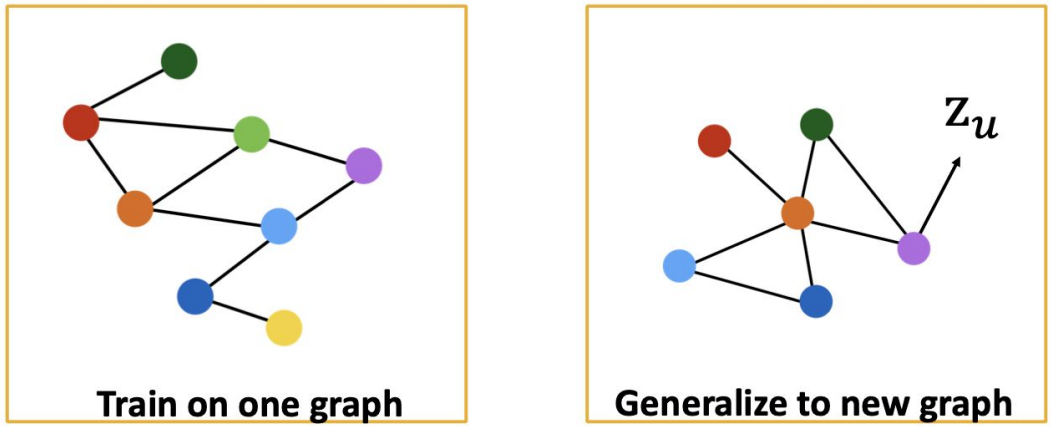
Inductive capability

- **The same aggregation parameters are shared for all nodes:**
  - The number of model parameters is sublinear in $|V|$ and we can **generalize to unseen nodes**!



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Inductive capability: New Graphs

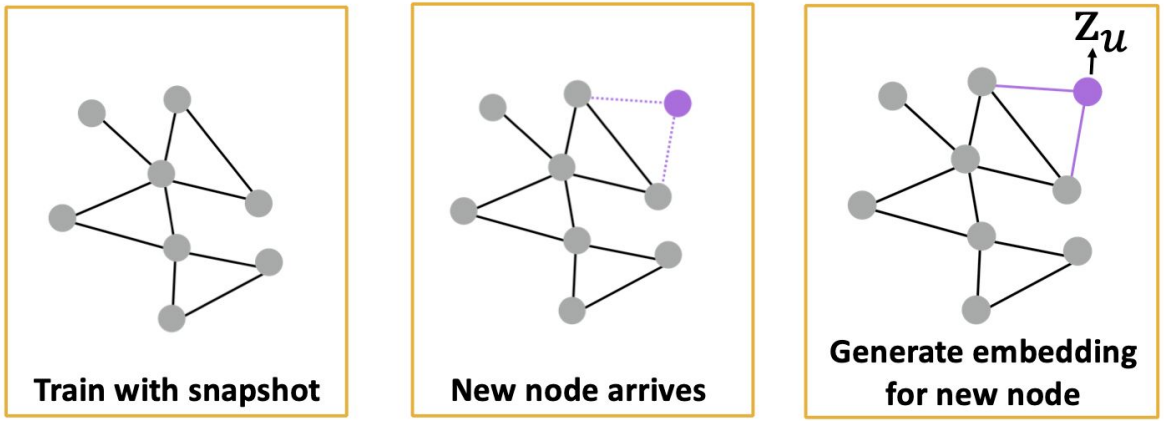


Inductive node embedding → Generalize to entirely unseen graphs

E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Inductive capability: New Nodes



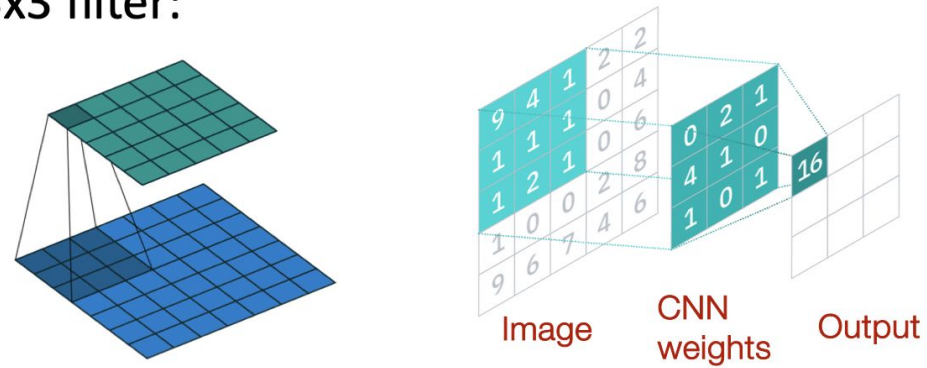
- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

Taken from Stanford CS224W course: http://cs244w.stanford.edu

Part 3: Comparison with ConvNets

Convolutional Neural Network

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



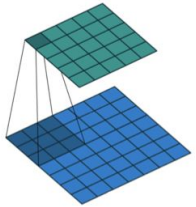
CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v) \cup \{v\}} \mathrm{W}_l^u \mathrm{h}_u^{(l)}), \quad \forall l \in \{0, \ldots, L-1\}$

$N(v)$ represents the 8 neighbor pixels of v.

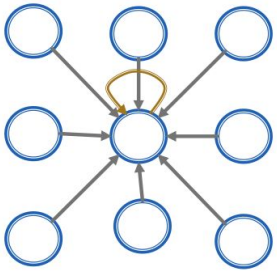
Taken from Stanford CS224W course: http://cs244w.stanford.edu

CNN vs. GNN

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



Image



Graph

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

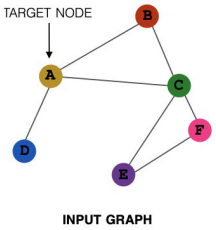
CNN formulation: $\mathrm{h}_v^{(l+1)} = \sigma(\sum_{u \in \mathrm{N}(v)} \mathbf{W}_l^u \mathrm{h}_u^{(l)} + \mathrm{B}_l \mathrm{h}_v^{(l)}), \forall l \in \{0, \ldots, L-1\}$

Key difference: We can learn different W_l^u for different "neighbor" u for pixel v on the image. The reason is we can pick an order for the 9 neighbors using **relative position** to the center pixel: {(-1,-1). (-1,0), (-1, 1), …, (1, 1)}

Taken from Stanford CS224W course: http://cs244w.stanford.edu

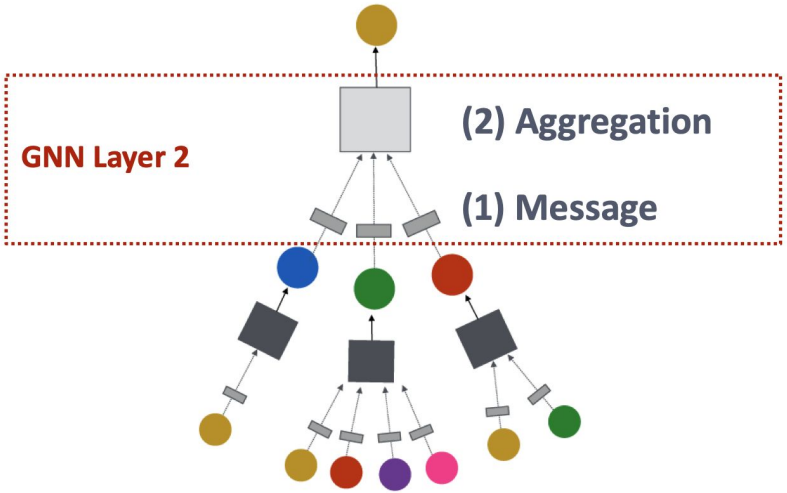
Part 4: General GNNs

A GNN Layer



GNN Layer = Message + Aggregation

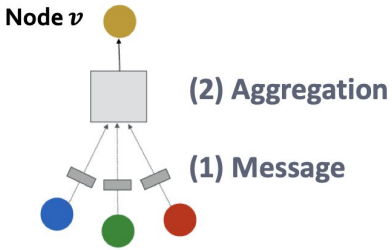
- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**

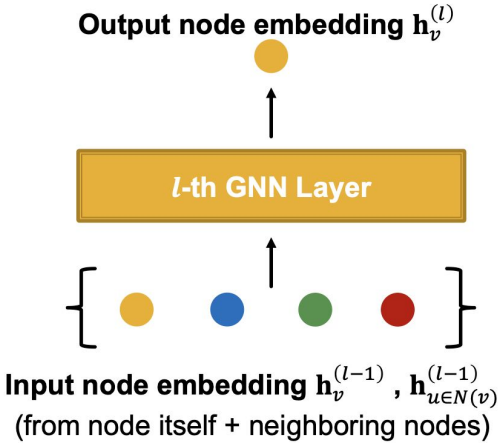


Taken from Stanford CS224W course: http://cs244w.stanford.edu

A GNN Layer

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

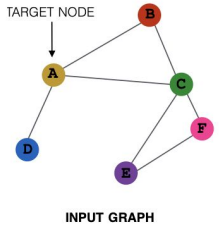


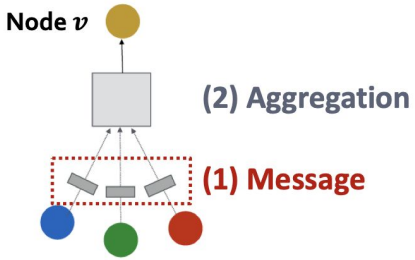


Taken from Stanford CS224W course: http://cs244w.stanford.edu

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





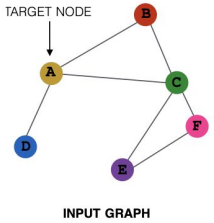
Taken from Stanford CS224W course: http://cs244w.stanford.edu

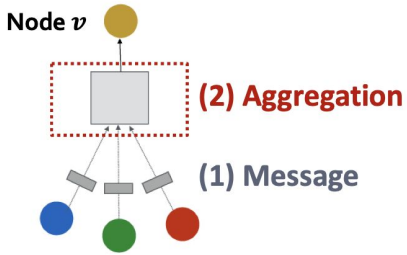
Message aggregation

- **(2) Aggregation**
  - **Intuition:** Node v will aggregate the messages from its neighbors u:

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

  - **Example:** $\text{Sum}(\cdot)$, $\text{Mean}(\cdot)$ or $\text{Max}(\cdot)$ aggregator
    - $\mathbf{h}_v^{(l)} = \text{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$





Taken from Stanford CS224W course: http://cs244w.stanford.edu

Message aggregation

- **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

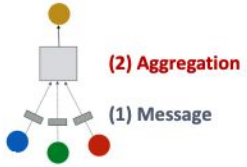
$$\mathbf{h}_v^{(l)} = \text{CONCAT}\left(\text{AGG}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right), \mathbf{m}_v^{(l)}\right)$$

First aggregate from neighbors

Taken from Stanford CS224W course: http://cs244w.stanford.edu

A Single Layer of GNN

- **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 $\sigma(\cdot)$. Examples: $\mathrm{ReLU}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, ...
    - Can be added to **message or aggregation**



Taken from Stanford CS224W course: http://cs244w.stanford.edu

Correspondence to classic GNN

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

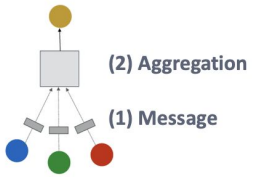
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

- **How to write this as Message + Aggregation?**

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

Message

Aggregation

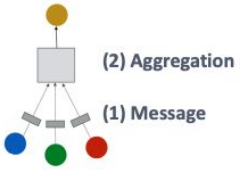


Taken from Stanford CS224W course: http://cs244w.stanford.edu

Correspondence to classic GNN

- **(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)

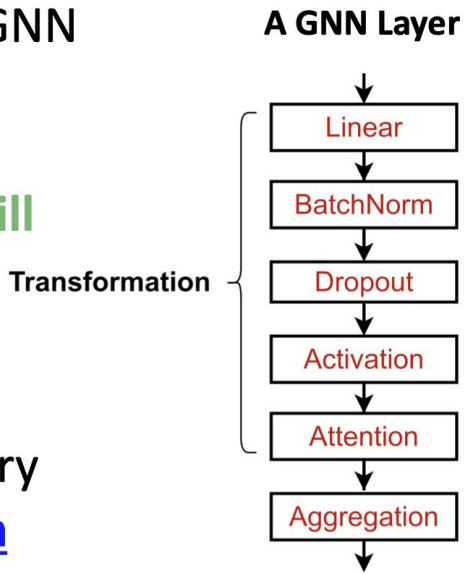
- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\text{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

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

Taken from Stanford CS224W course: http://cs244w.stanford.edu

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**



Taken from Stanford CS224W course: http://cs244w.stanford.edu