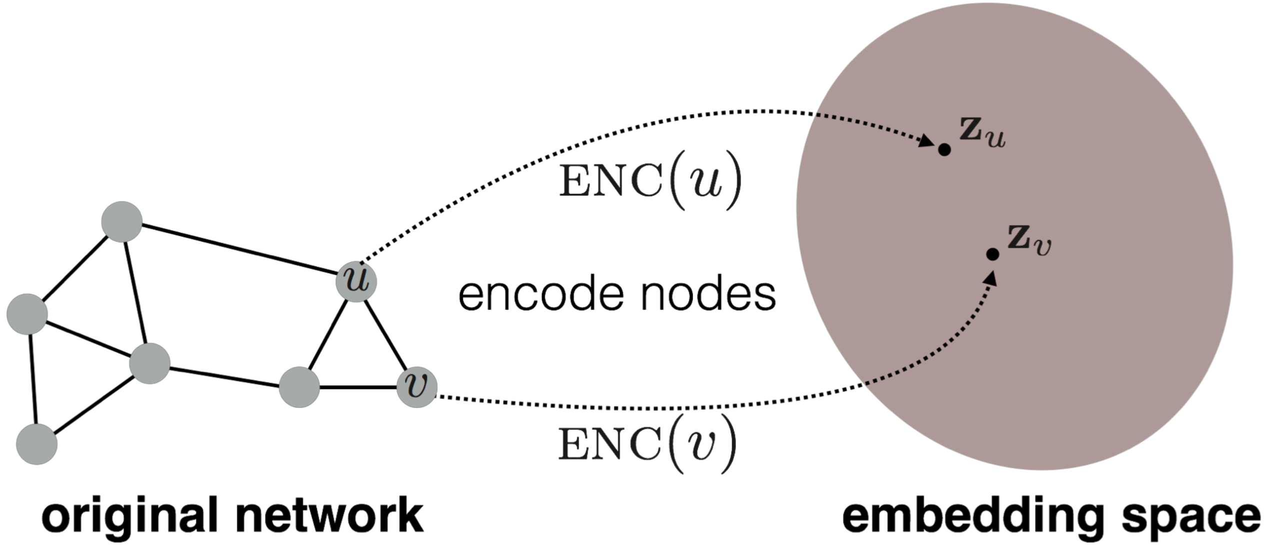
This Talk

- 1) Node embeddings ✓
  - Map nodes to low-dimensional embeddings.
- 2) Graph neural networks ☜
  - Deep learning architectures for graph-structured data
- 3) Generative graph models
  - Learning to generate realistic graph data.

Part 2: Graph Neural Networks

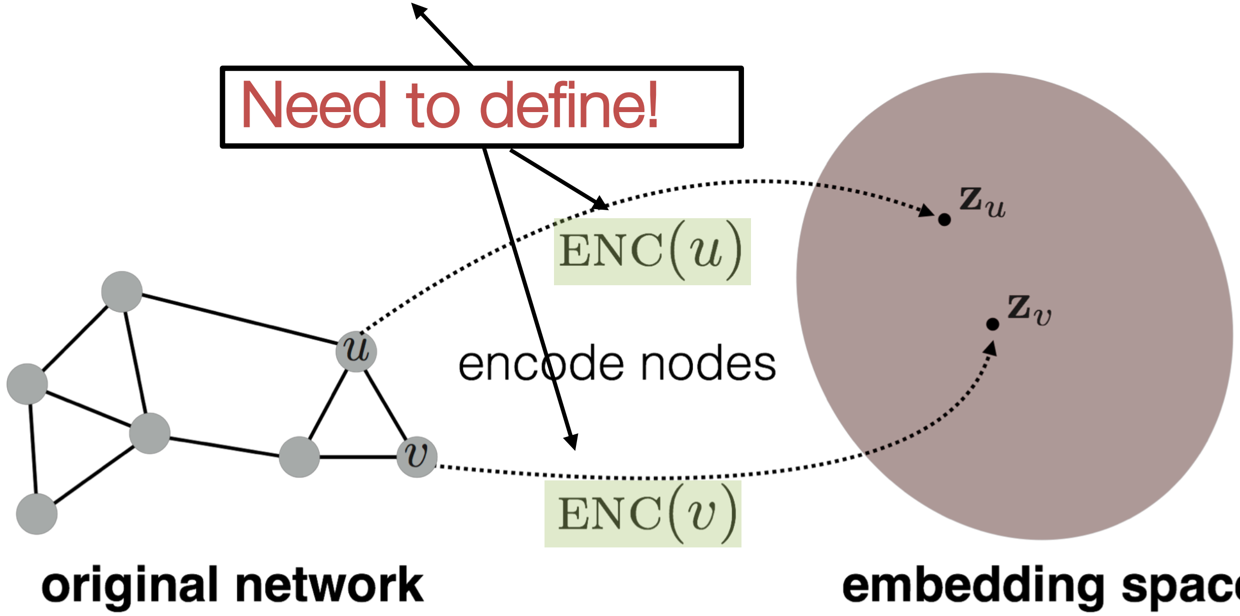
Embedding Nodes

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.



Embedding Nodes

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$



Two Key Components

- Encoder maps each node to a low-dimensional vector.

$$\text{ENC}(v) = \mathbf{z}_v$$

d-dimensional embedding

node in the input graph

- Similarity function specifies how relationships in vector space map to relationships in the original network.

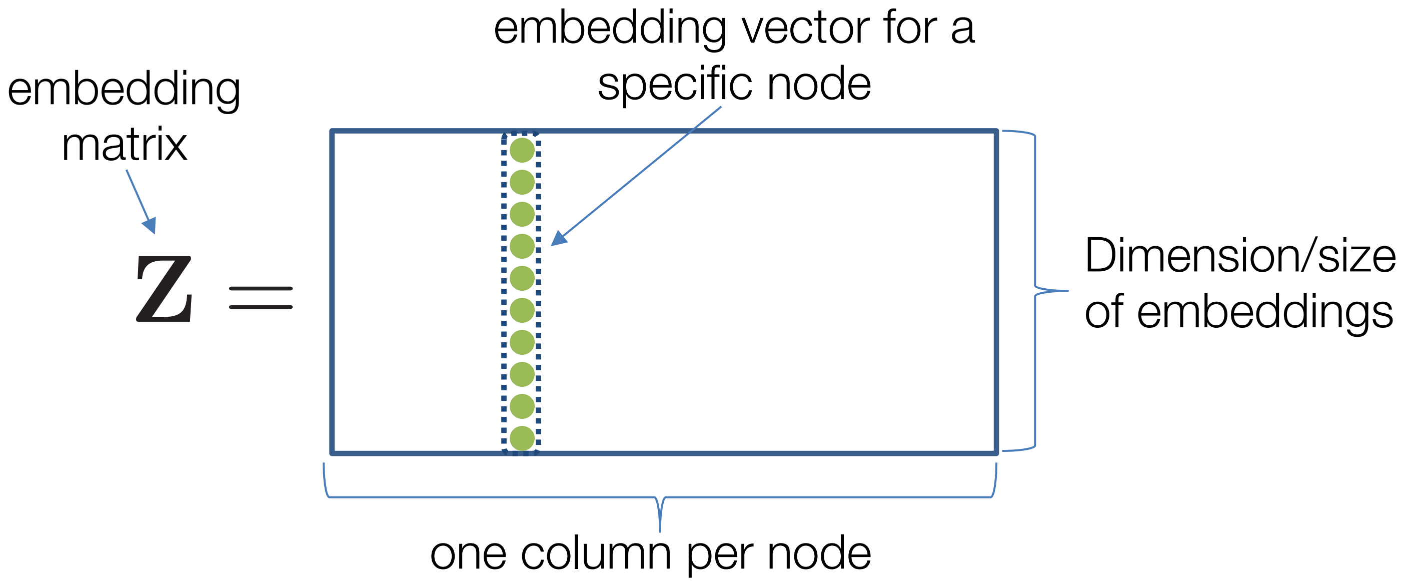
$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

From "Shallow" to "Deep"

- So far we have focused on **"shallow" encoders**, i.e. embedding lookups:


embedding matrix
$\mathbf{Z} =$
embedding vector for a specific node
Dimension/size of embeddings
one column per node

From “Shallow” to “Deep”

- Limitations of shallow encoding:
  - O(|V|) parameters are needed: there no parameter sharing and every node has its own unique embedding vector.
  - Inherently “transductive”: It is impossible to generate embeddings for nodes that were not seen during training.
  - Do not incorporate node features: Many graphs have features that we can and should leverage.

From “Shallow” to “Deep”

- We will now discuss “deeper” methods based on graph neural networks.

$$\text{ENC}(v) =$$ complex function that depends on graph structure.

- In general, all of these more complex encoders can be combined with the similarity functions from the previous section.

Outline for this Section

- We will now discuss “deeper” methods based on graph neural networks.
    1. The Basics
    2. Graph Convolutional Networks
    3. GraphSAGE
    4. Gated Graph Neural Networks
    5. Graph Attention Networks
    6. Subgraph embeddings

The Basics: Graph Neural Networks

Based on material from:

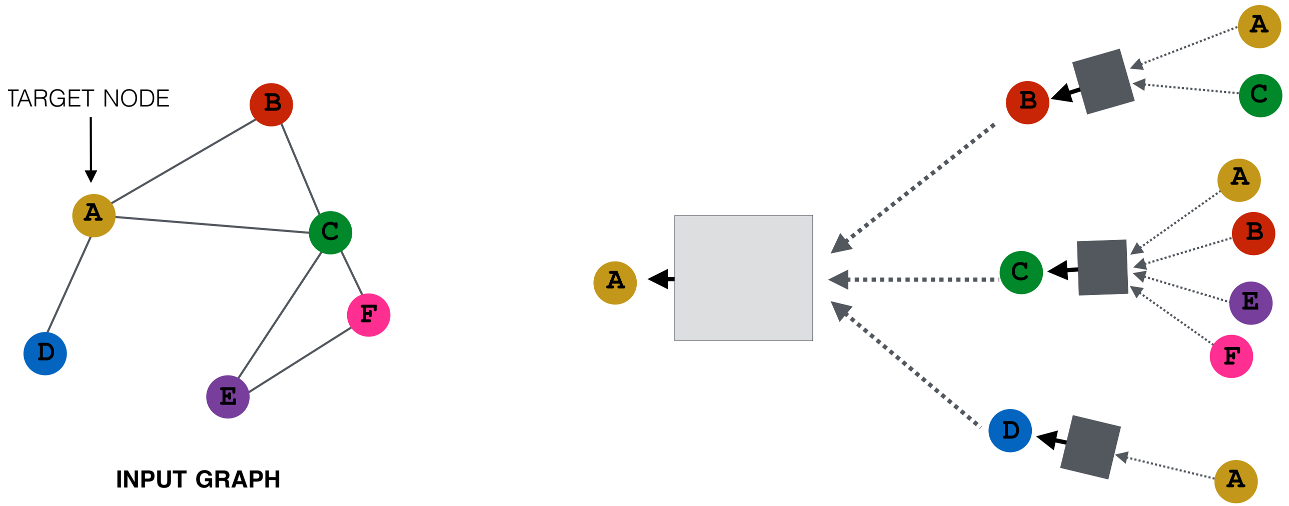
- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
- Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Setup

- Assume we have a graph $\mathbf{G}$:
  - V is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - $\mathrm{X} \in \mathrm{R}^{m \times |V|}$ is a matrix of node features.
    - Categorical attributes, text, image data
      - E.g., profile information in a social network.
    - Node degrees, clustering coefficients, etc.
    - Indicator vectors (i.e., one-hot encoding of each node)

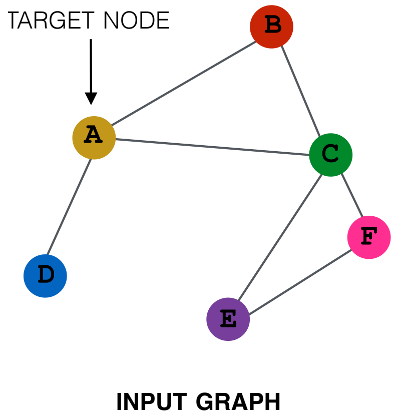
Neighborhood Aggregation

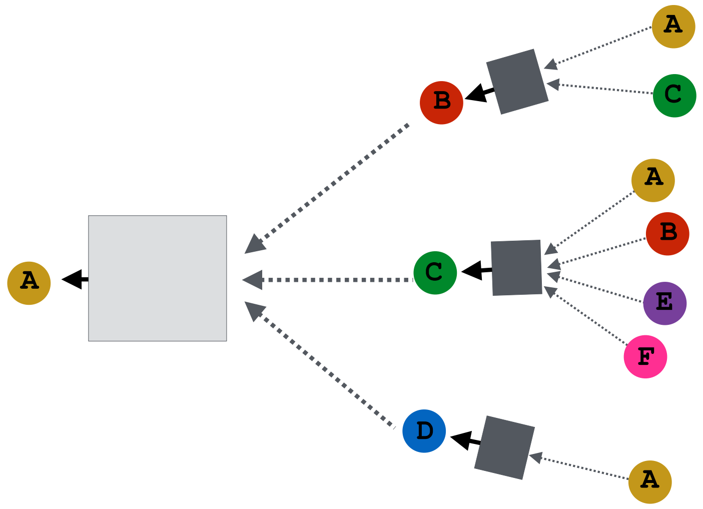
- **Key idea:** Generate node embeddings based on local neighborhoods.



Neighborhood Aggregation

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

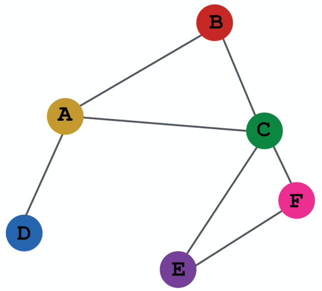




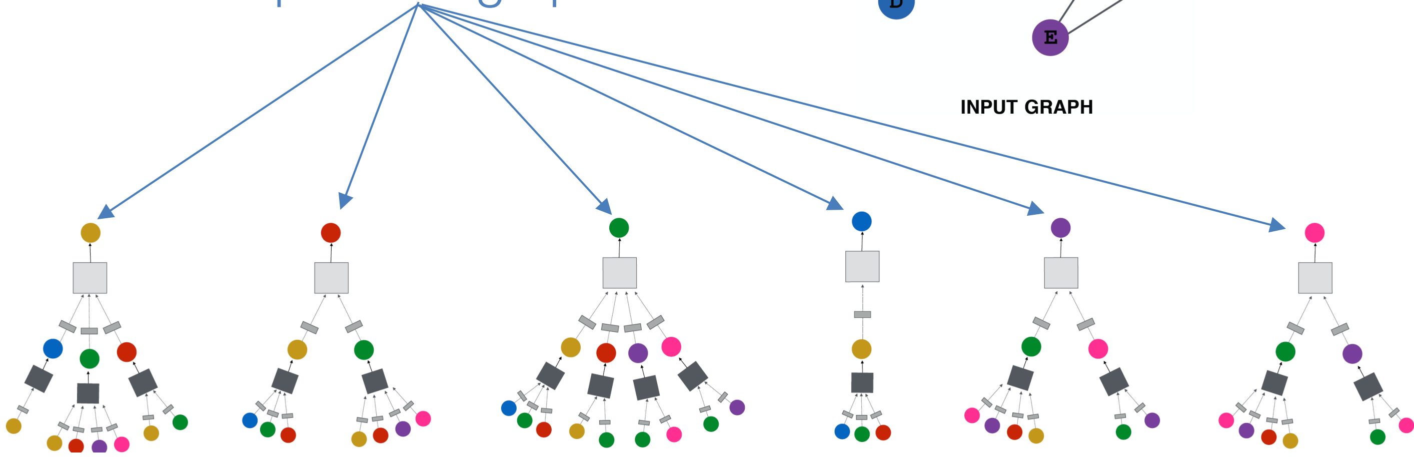
Neighborhood Aggregation

- Intuition: Network neighborhood defines a computation graph

Every node defines a unique computation graph!

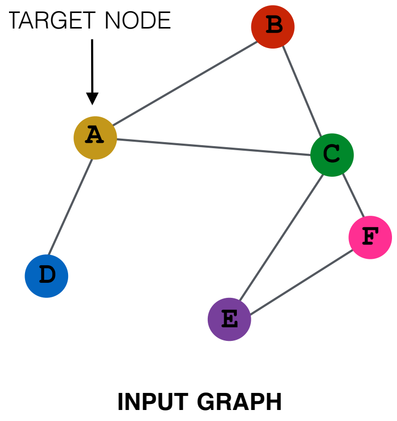


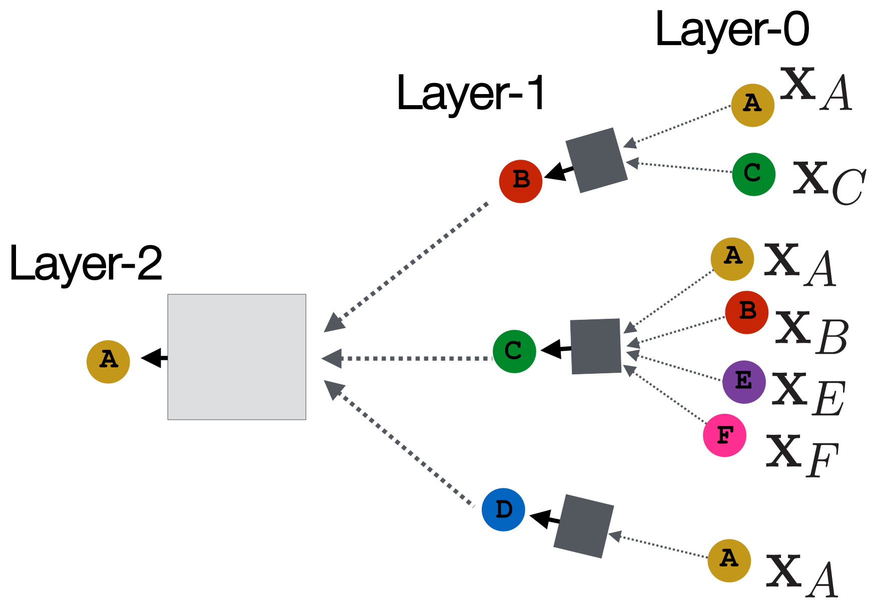
INPUT GRAPH



Neighborhood Aggregation

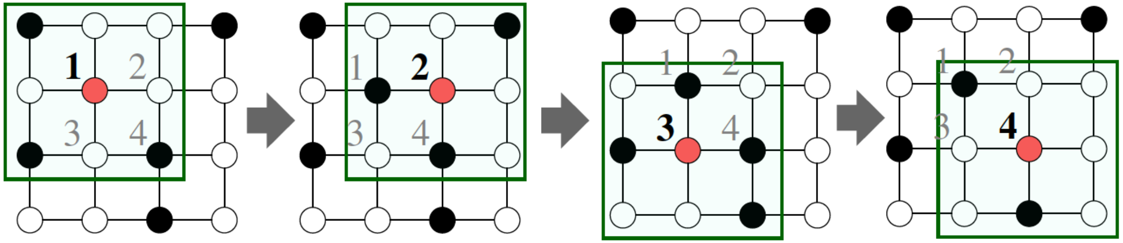
- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- “layer-0” embedding of node $\mathbf{u}$ is its input feature, i.e. $\mathbf{x}_\mathbf{u}$.





Neighborhood "Convolutions"

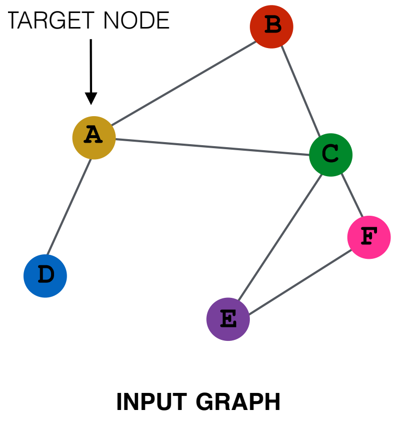
- Neighborhood aggregation can be viewed as a center-surround filter.



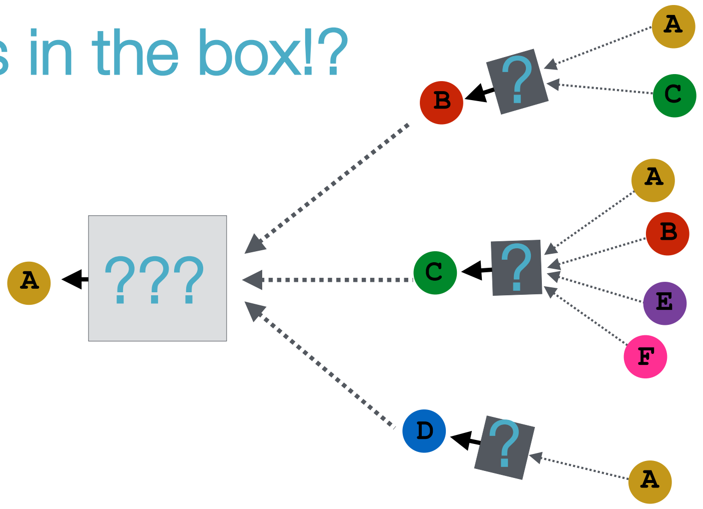
- Mathematically related to spectral graph convolutions (see Bronstein et al., 2017)

Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate information across the layers.

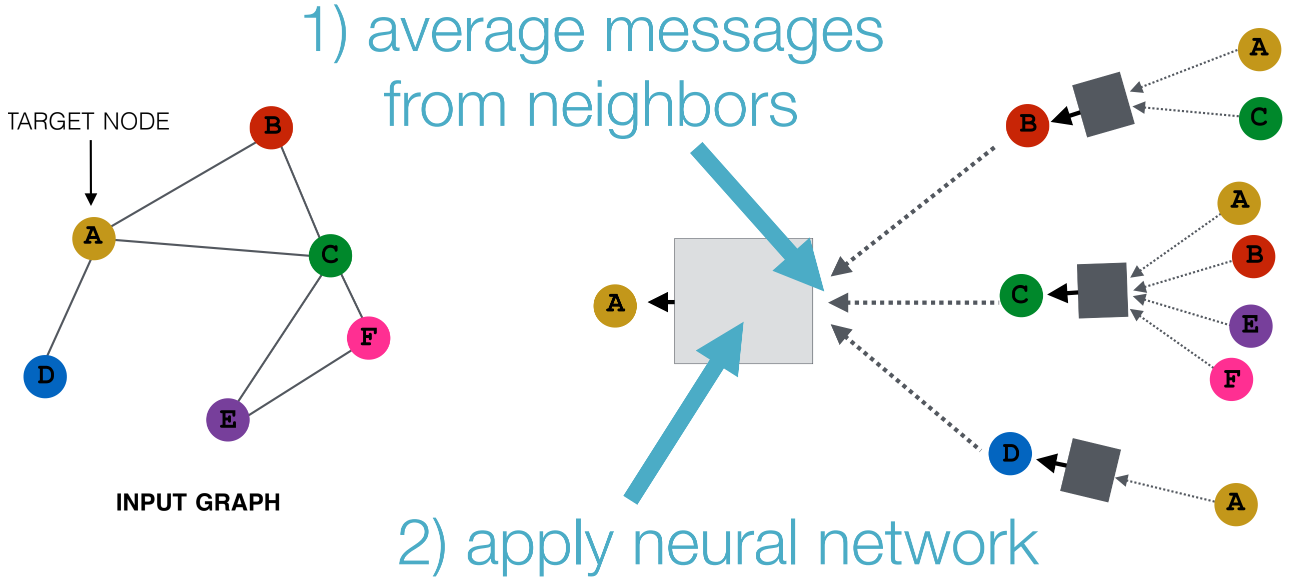


what's in the box!?



Neighborhood Aggregation

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



The Math

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

$$\boxed{\mathbf{h}_v^0 = \mathbf{x}_v}$$

Initial "layer 0" embeddings are equal to node features

$$\boxed{\mathbf{h}_v^k} = \boxed{\sigma}\left(\mathbf{W}_k \boxed{\sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}} + \mathbf{B}_k \boxed{\mathbf{h}_v^{k-1}}\right), \quad \forall k > 0$$

previous layer embedding of v

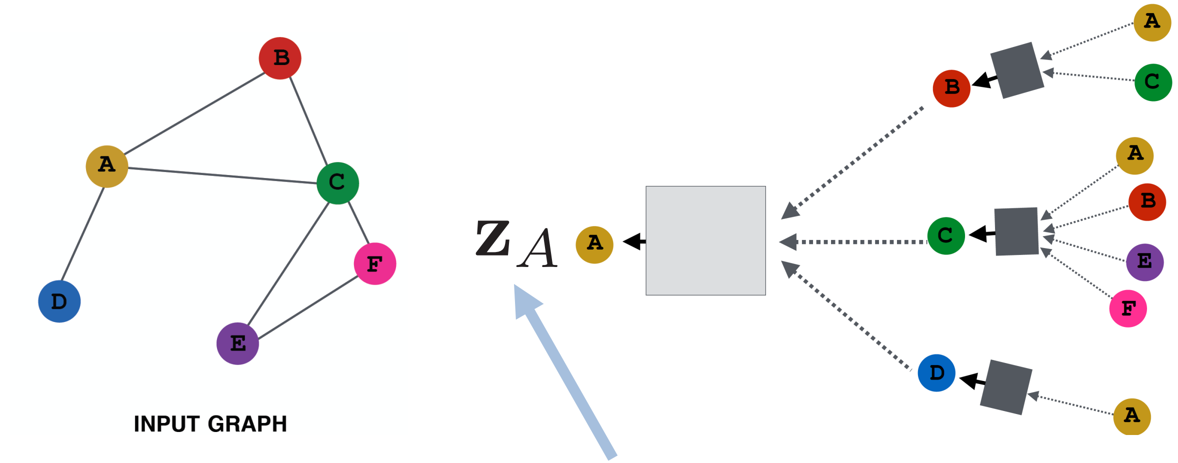
kth layer embedding of v

non-linearity (e.g., ReLU or tanh)

average of neighbor's previous layer embeddings

Training the Model

- How do we train the model to generate "high-quality" embeddings?



Need to define a loss function on the embeddings, $\mathcal{L}(z_u)$!

Training the Model

trainable matrices
(i.e., what we learn)

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

$$\mathbf{h}_v^k = \sigma\left(\boxed{\mathbf{W}_k} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \boxed{\mathbf{B}_k}\mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, \ldots, K\}$$

$$\boxed{\mathbf{z}_v = \mathbf{h}_v^K}$$

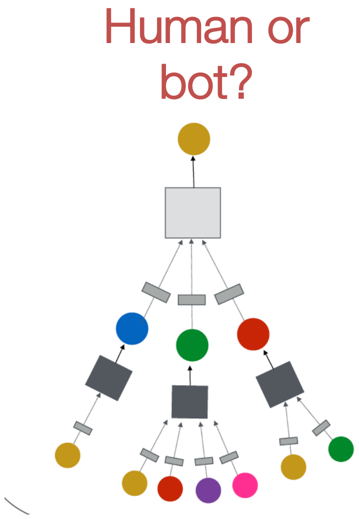
- After K-layers of neighborhood aggregation, we get output embeddings for each node.
- **We can feed these embeddings into any loss function** and run stochastic gradient descent to train the aggregation parameters.

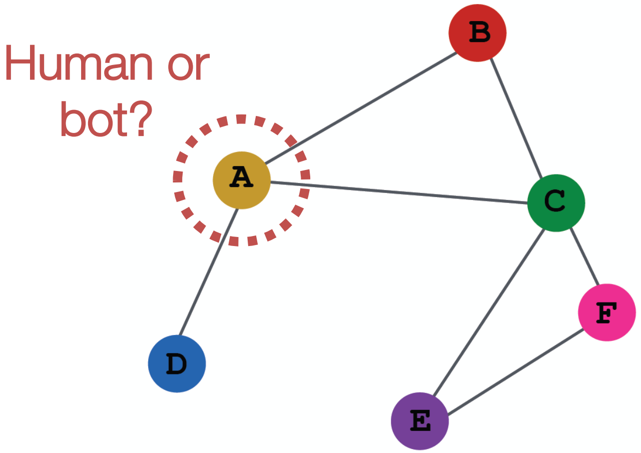
Training the Model

- Train in an **unsupervised manner** using only the graph structure.
- Unsupervised loss function can be anything from the last section, e.g., based on
  - Random walks (node2vec, DeepWalk)
  - Graph factorization
  - i.e., train the model so that "similar" nodes have similar embeddings.

Training the Model

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





e.g., an online social network

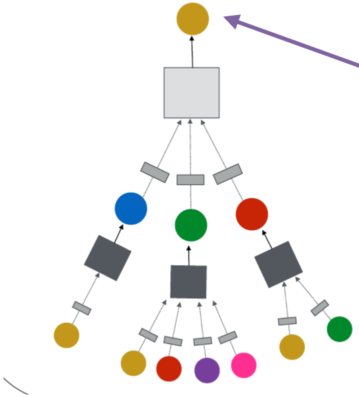
Training the Model

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

Human or bot?

classification weights

$$\mathcal{L} = \sum_{v \in V} \boxed{y_v} \log(\sigma(\boxed{\mathbf{z}_v}^\top \boxed{\theta})) + (1 - \boxed{y_v}) \log(1 - \sigma(\boxed{\mathbf{z}_v}^\top \boxed{\theta}))$$

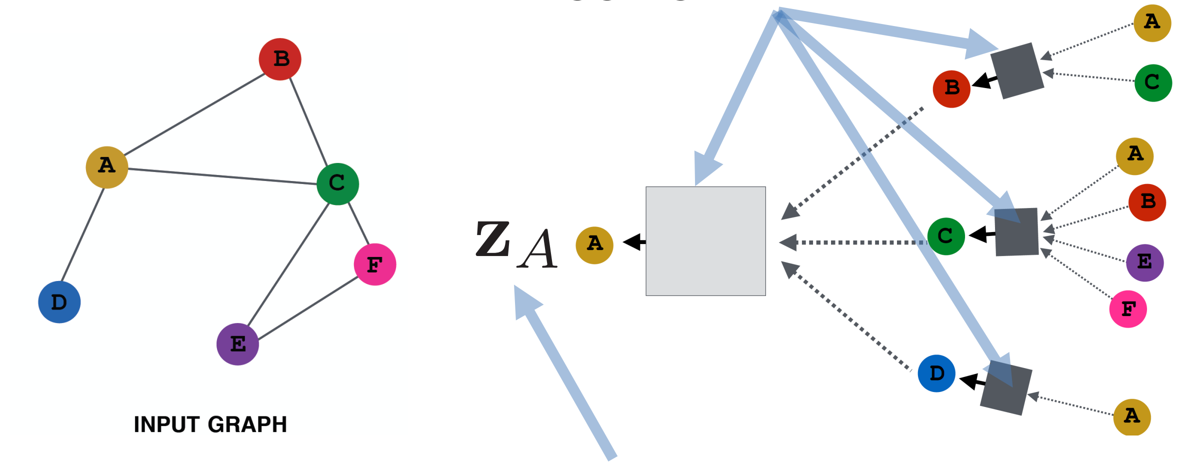


output node embedding

node class label

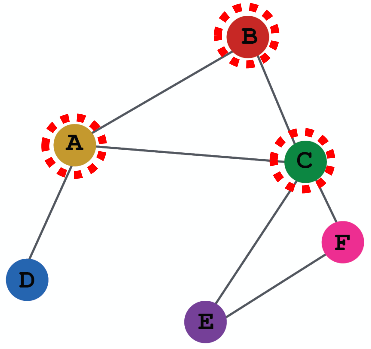
Overview of Model

1) Define a neighborhood aggregation function.



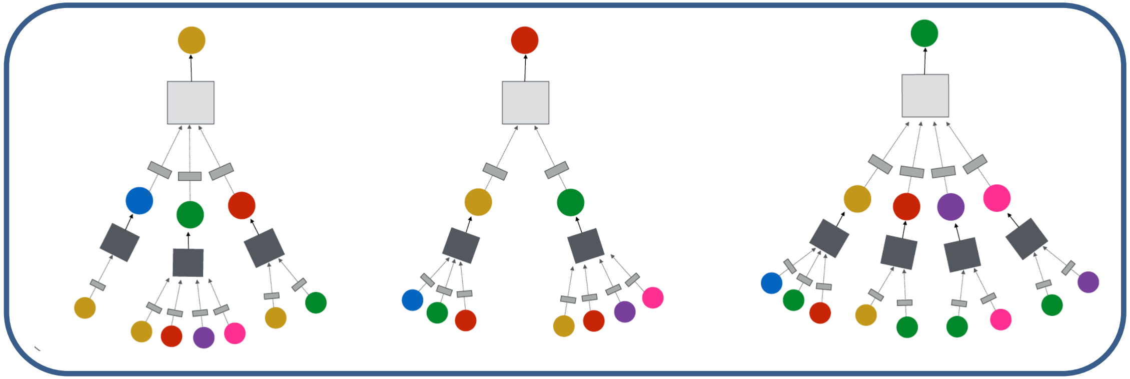
2) Define a loss function on the embeddings, $\mathcal{L}(\mathbf{z}_u)$

Overview of Model

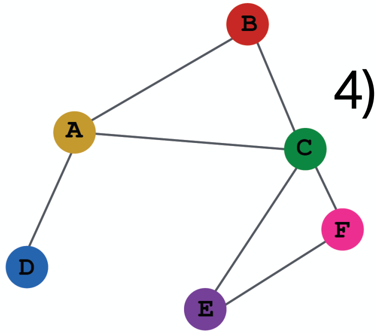


INPUT GRAPH

3) Train on a set of nodes, i.e., a batch of compute graphs



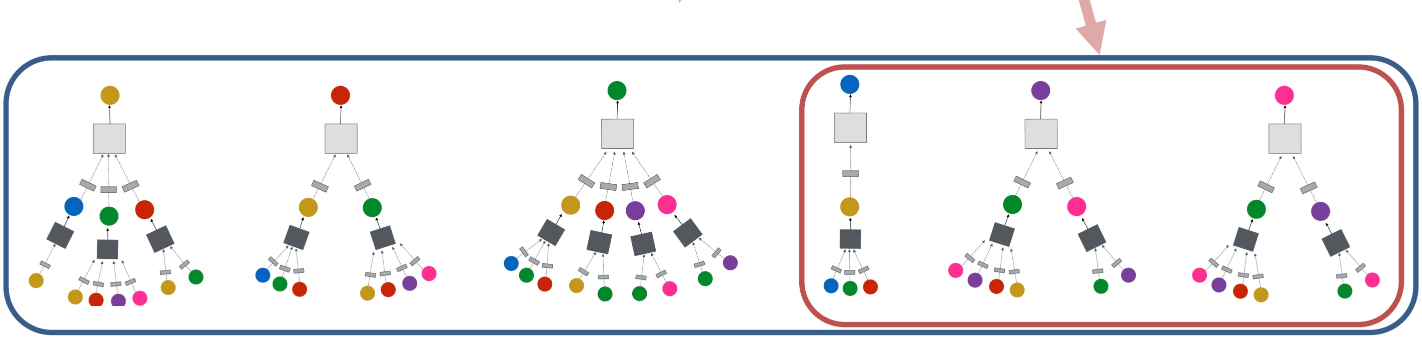
Overview of Model



INPUT GRAPH

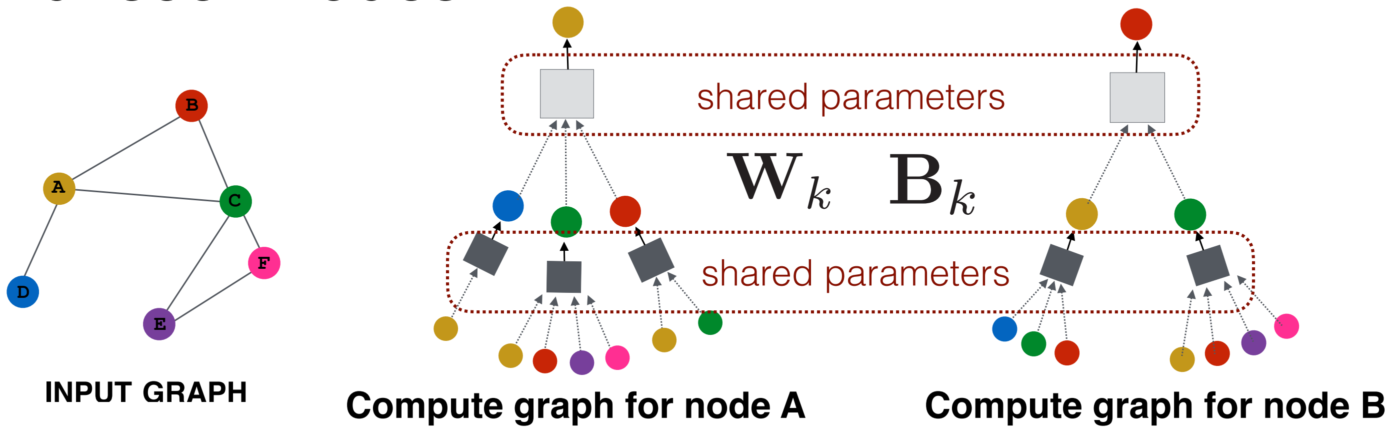
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

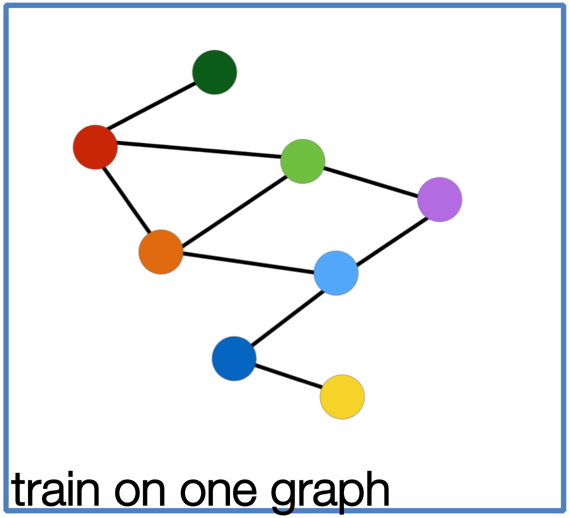


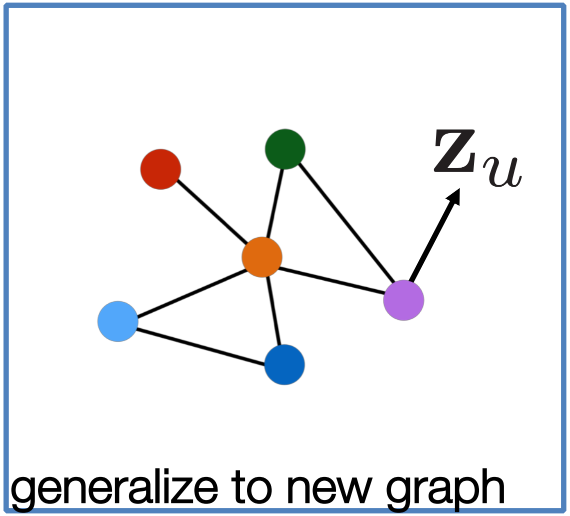
Inductive Capability

- 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!



Inductive Capability

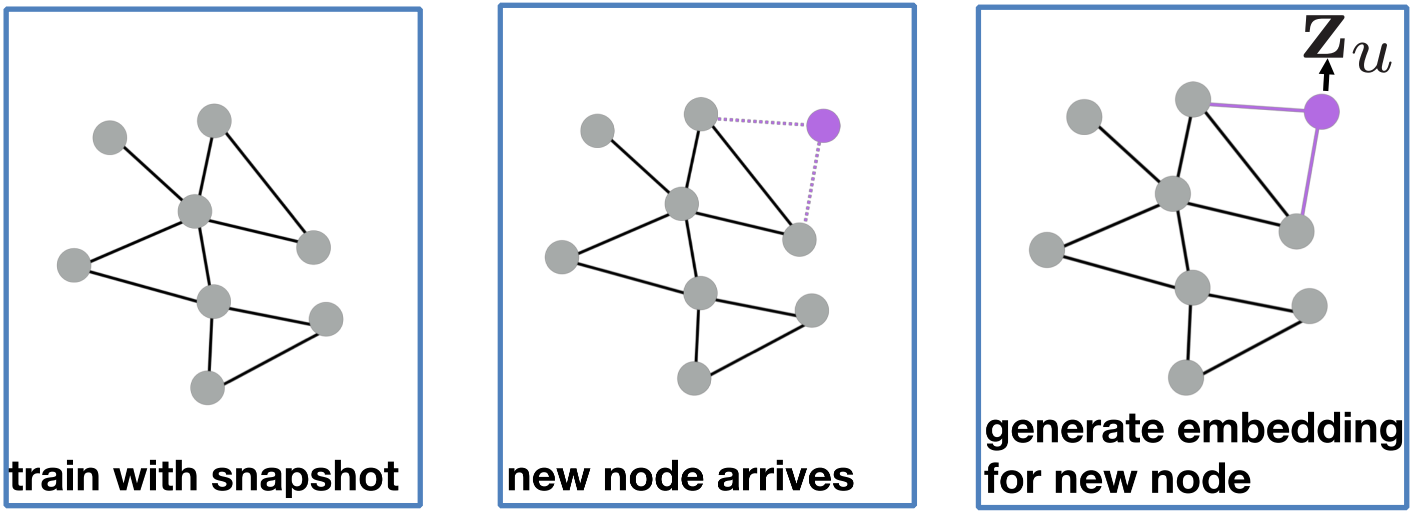




Inductive node embedding $\rightarrow$ generalize to entirely unseen graphs

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

Inductive Capability



Many application settings constantly encounter previously unseen nodes.

e.g., Reddit, YouTube, GoogleScholar, ….

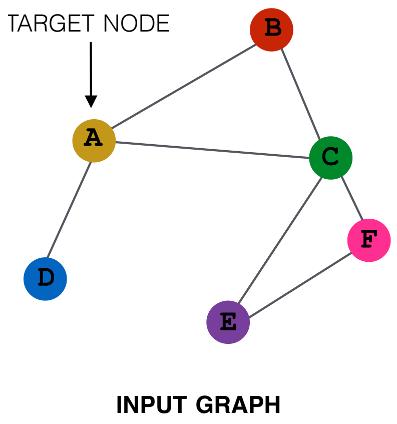
Need to generate new embeddings "on the fly"

Quick Recap

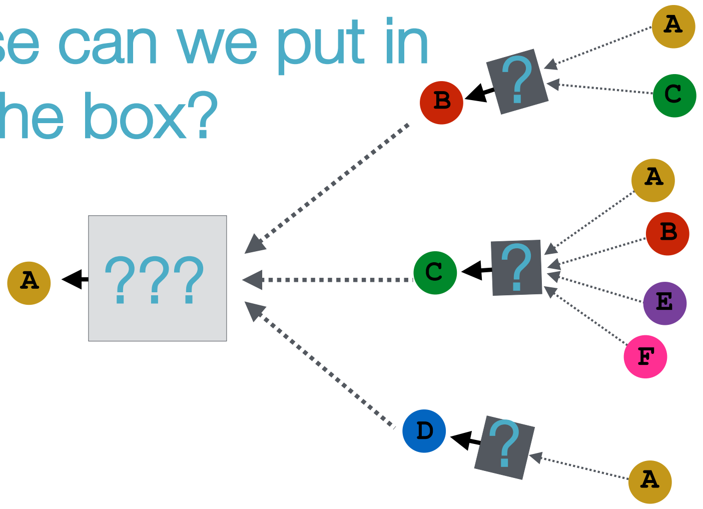
- Recap: Generate node embeddings by aggregating neighborhood information.
  - Allows for parameter sharing in the encoder.
  - Allows for inductive learning.
- We saw a **basic variant of this idea**… now we will cover some state of the art variants from the literature.

Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages



What else can we put in the box?



Graph Convolutional Networks

Based on material from:

- Kipf et al., 2017. Semisupervised Classification with Graph Convolutional Networks. *ICLR*.

Graph Convolutional Networks

- Kipf et al.'s **Graph Convolutional Networks (GCNs)** are a slight variation on the neighborhood aggregation idea:

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

Graph Convolutional Networks

Basic Neighborhood Aggregation

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

VS.

GCN Neighborhood Aggregation

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

same matrix for self and neighbor embeddings

per-neighbor normalization

Graph Convolutional Networks

- Empirically, they found this configuration to give the best results.
  - More parameter sharing.
  - Down-weights high degree neighbors.

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

use the same transformation matrix for self and neighbor embeddings

instead of simple average, normalization varies across neighbors

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE ☜
4. Gated Graph Neural Networks
5. Graph Attention Networks
6. Subgraph Embeddings

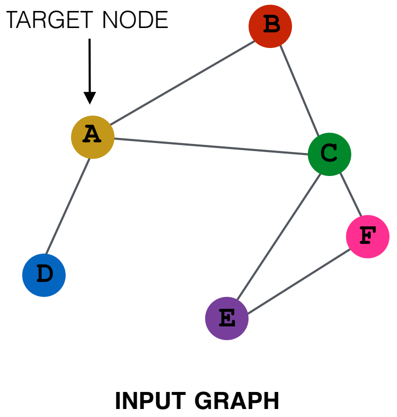
GraphSAGE

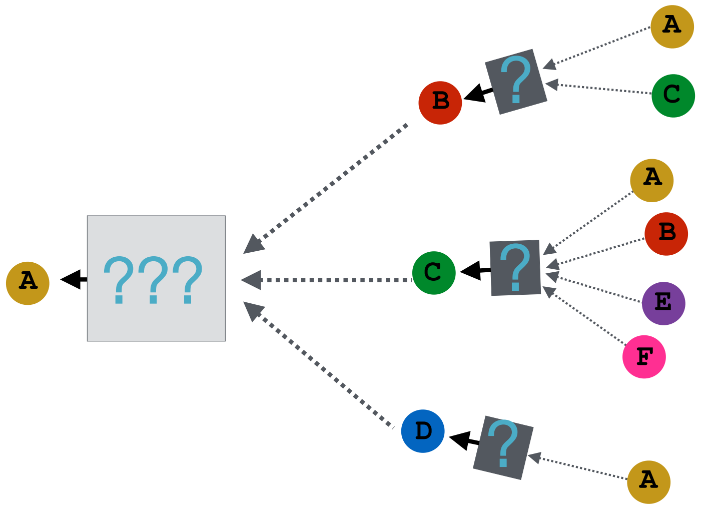
Based on material from:

- Hamilton et al., 2017. Inductive Representation Learning on Large Graphs. *NIPS.*

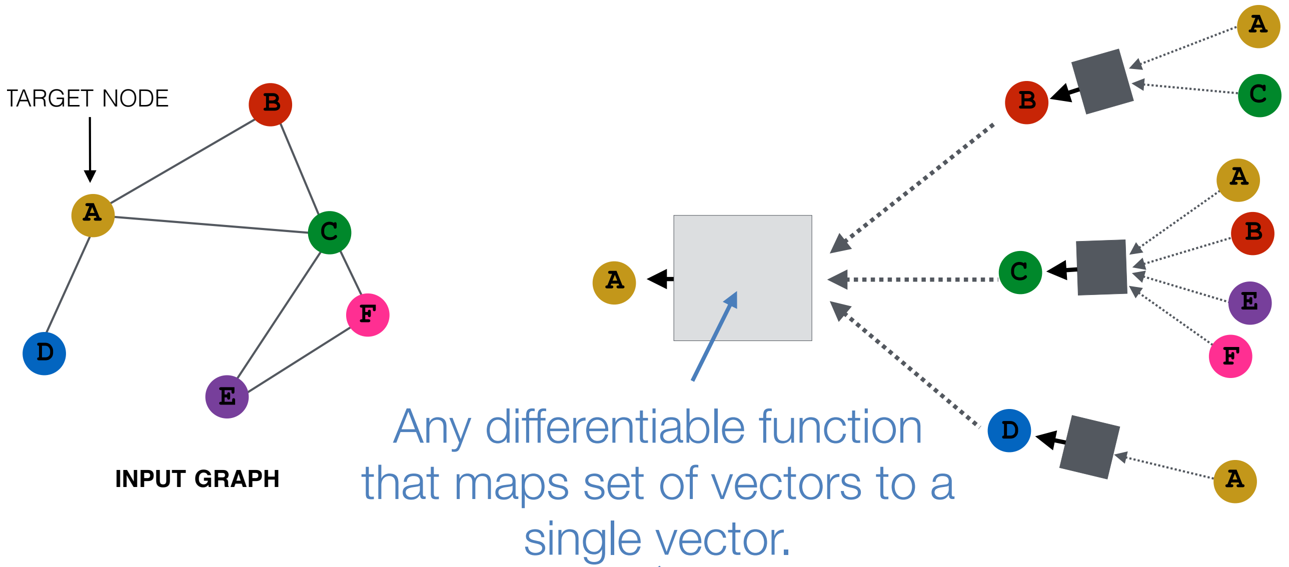
GraphSAGE Idea

- So far we have aggregated the neighbor messages by taking their (weighted) average, can we do better?





GraphSAGE Idea



Any differentiable function that maps set of vectors to a single vector.

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

GraphSAGE Differences

- Simple neighborhood aggregation:

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

- GraphSAGE:

concatenate self embedding and neighbor embedding

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

generalized aggregation

GraphSAGE Variants

- Mean:

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

- Pool
  - Transform neighbor vectors and apply symmetric vector function.

element-wise mean/max

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

- LSTM:
  - Apply LSTM to random permutation of neighbors.

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

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE ✓
4. Gated Graph Neural Networks ☜
5. Graph Attention Networks
6. Subgraph Embeddings

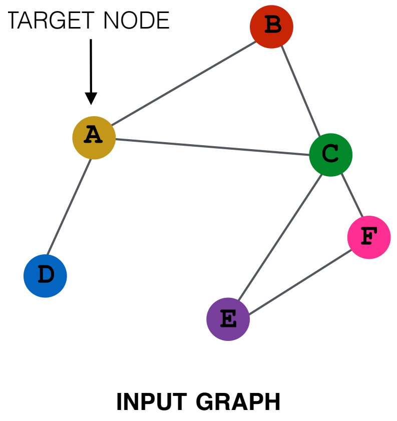
Gated Graph Neural Networks

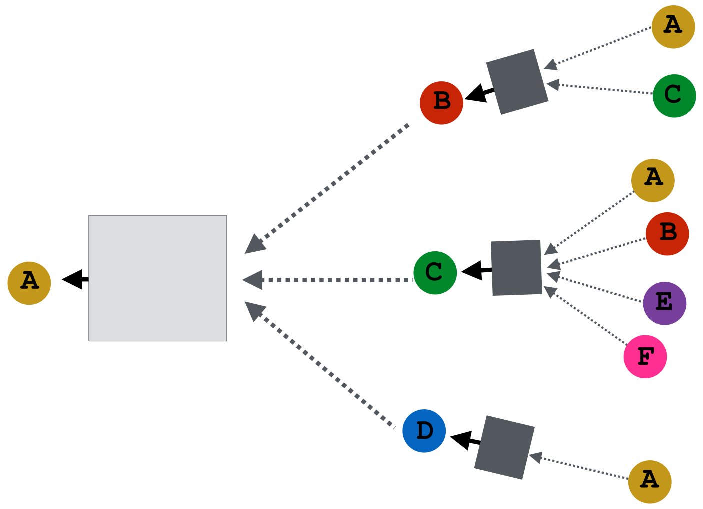
Based on material from:

- Li et al., 2016. Gated Graph Sequence Neural Networks. *ICLR.*
- Gilmer et al., 2017. Neural Message Passing for Quantum Chemistry. *ICML.*

Neighborhood Aggregation

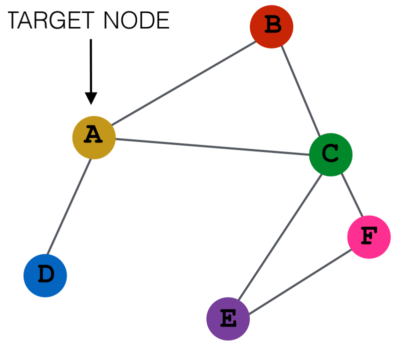
- Basic idea: Nodes aggregate “messages” from their neighbors using neural networks



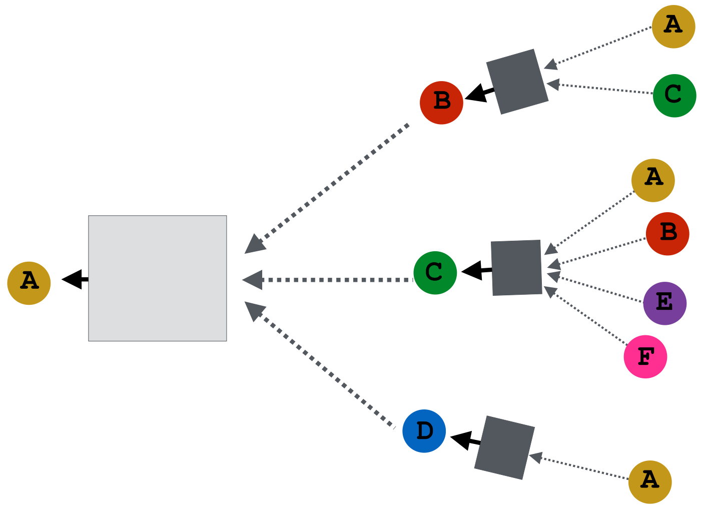


Neighborhood Aggregation

- GCNs and GraphSAGE **generally only 2-3 layers deep.**

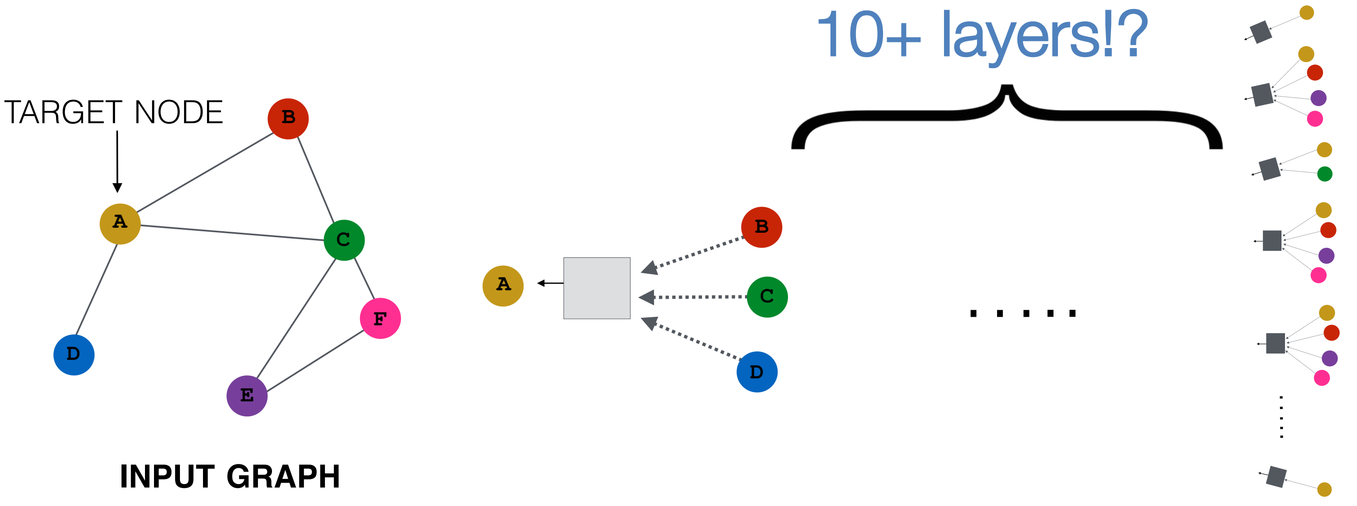


INPUT GRAPH



Neighborhood Aggregation

- But what if we want to go deeper?

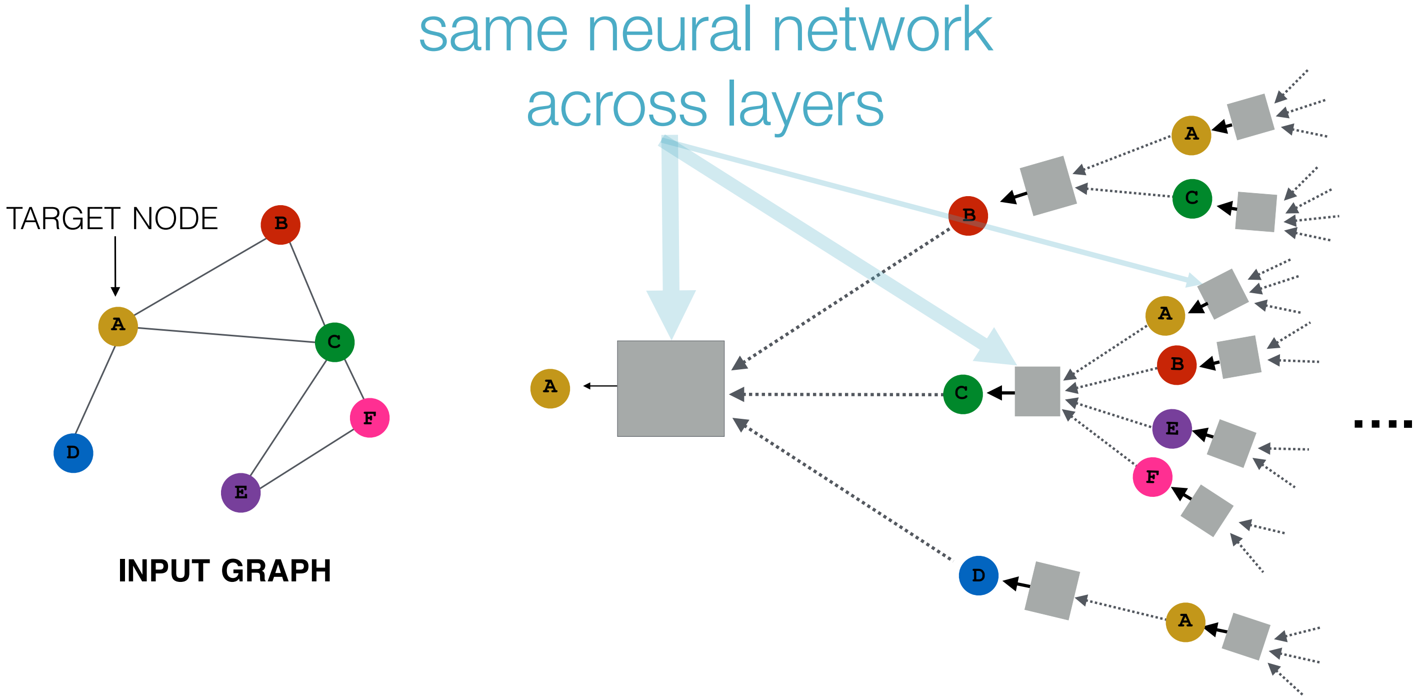


Gated Graph Neural Networks

- How can we build models with many layers of neighborhood aggregation?
- Challenges:
    - Overfitting from too many parameters.
    - Vanishing/exploding gradients during backpropagation.
- **Idea:** Use techniques from modern recurrent neural networks!

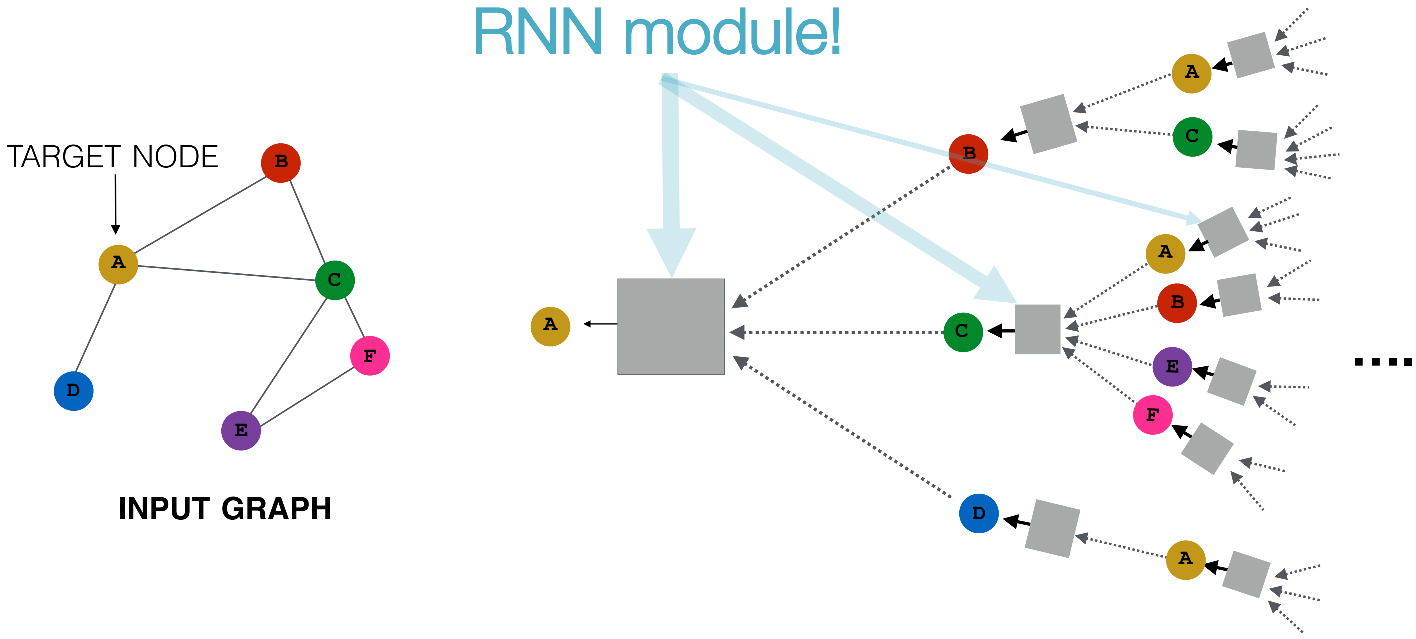
Gated Graph Neural Networks

- **Idea 1:** Parameter sharing across layers.



Gated Graph Neural Networks

- **Idea 2:** Recurrent state update.



The Math

- **Intuition:** Neighborhood aggregation with RNN state update.
  1. Get "message" from neighbors at step k:

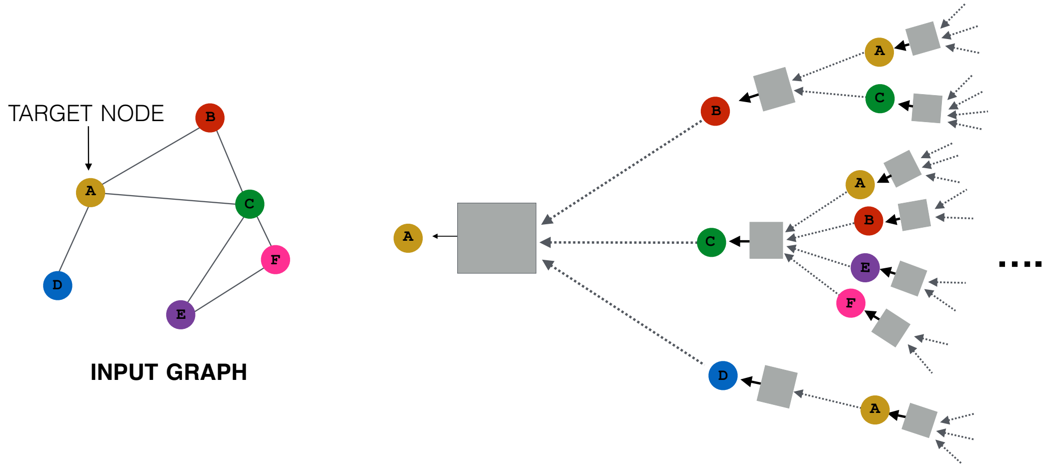
$$\mathbf{m}_v^k = \boxed{\mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}}$$

aggregation function does not depend on $\mathbf{k}$

  2. Update node "state" using Gated Recurrent Unit (GRU). New node state depends on the old state and the message from neighbors:

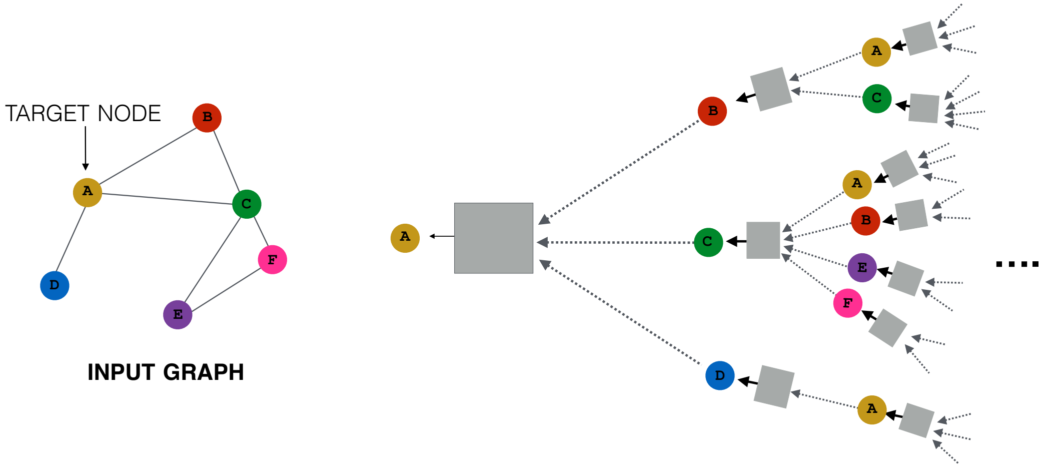
$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

Gated Graph Neural Networks



- Can handle models with >20 layers.
- Most real-world networks have small diameters (e.g., less than 7).
- Allows for complex information about global graph structure to be propagated to all nodes.

Gated Graph Neural Networks



- Useful for complex networks representing:
  - Logical formulas.
  - Programs.

Message-Passing Neural Networks

- **Idea:** We can generalize the gated graph neural network idea:
  1. Get "message" from neighbors at step k:

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

Can incorporate edge features.

Generic "message" function (e.g., sum or MLP).

  2. Update node "state":

$$\mathbf{h}_v^k = U(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

Generic update function (e.g., LSTM or GRU)

Message-Passing Neural Networks

- This is a general conceptual framework that subsumes most GNNs.
  1. Get "message" from neighbors at step k:

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

  2. Update node "state":

$$\mathbf{h}_v^k = U(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

- Gilmer et al., 2017. Neural Message Passing for Quantum Chemistry. *ICML.*

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE ✓
4. Gated Graph Neural Networks ✓
5. Graph Attention Networks ☜
6. Subgraph Embeddings

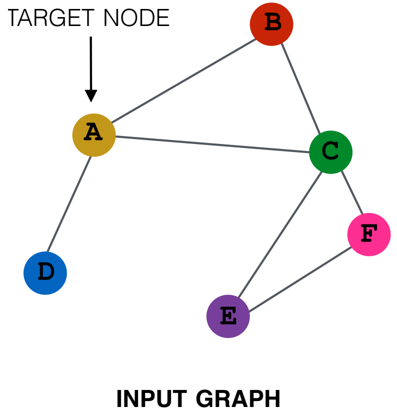
Graph Attention Networks

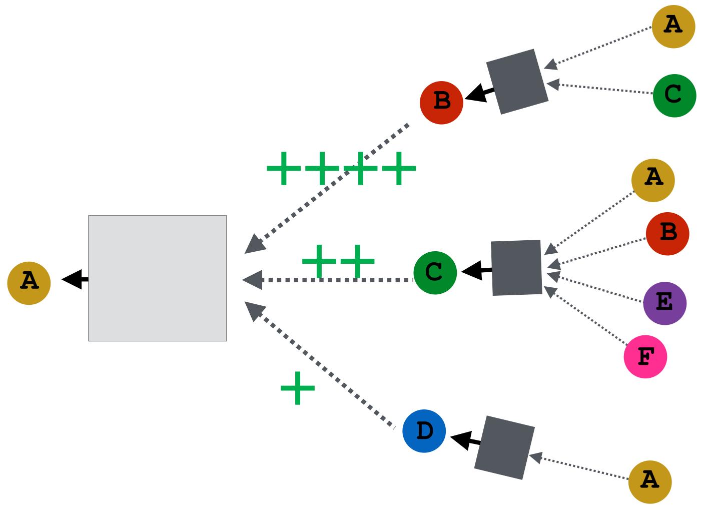
Based on material from:

- Velickovic et al., 2018. [Graph Attention Networks]. *ICLR.*

Neighborhood Attention

- What if some neighbors are more important than others?





Graph Attention Networks

- Augment basic graph neural network model with attention.

$$\mathbf{h}_v^k = \boxed{\sigma}\Big(\boxed{\sum_{u \in N(v) \cup \{v\}}} \boxed{\alpha_{v,u}} \mathbf{W}^k \mathbf{h}_u^{k-1}\Big)$$

Non-linearity

Sum over all neighbors (and the node itself)

Learned attention weights!

Attention weights

- Various attention models are possible.
- The original GAT paper uses:

$$\alpha_{v,u} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{Qh}_v, \mathbf{Qh}_u]\right)\right)}{\sum_{u' \in N(v)\cup\{v\}} \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{Qh}_v, \mathbf{Qh}_{u'}]\right)\right)}$$

- Achieved SOTA in 2018 on a number of standard benchmarks.

Attention in general

- Various attention mechanisms can be incorporated into the "message" step:
  1. Get "message" from neighbors at step k:

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

  2. Update node "state":

$$\mathbf{h}_v^k = U(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

Incorporate attention here.

Recent advances in graph neural nets (not covered in detail here)

- Generalizations based on spectral convolutions:
  - Geometric Deep Learning (Bronstein et al., 2017)
  - Mixture Model CNNs (Monti et al., 2017)
- Speed improvements via subsampling:
  - FastGCNs (Chen et al., 2018)
  - Stochastic GCNs (Chen et al., 2017)
- And much more!!!

So what is SOTA?

- No consensus…
- Standard benchmarks ~2017-2018
  - Cora, CiteSeer, PubMed
  - Semi-supervised node classification.
  - **Extremely noisy evaluation and basic GNN/GCNs are very strong…**
- Attention, gating, and other modifications have shown improvements in specific settings (e.g., molecule classification, recommender systems).

Outline for this Section

1. The Basics ✓
2. Graph Convolutional Networks ✓
3. GraphSAGE ✓
4. Gated Graph Neural Networks ✓
5. Graph Attention Networks ✓
6. Subgraph Embeddings ☜

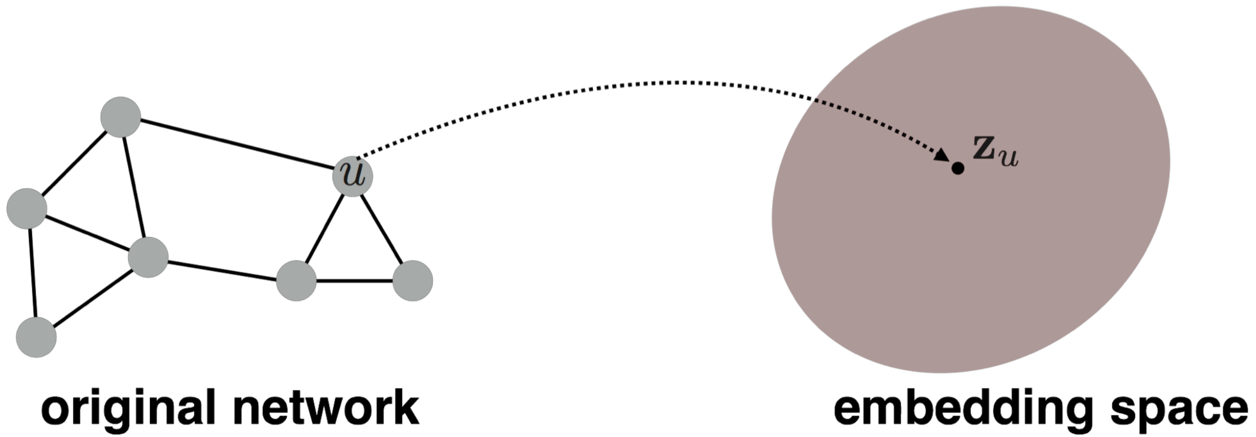
(Sub)graph Embeddings

Based on material from:

- Duvenaud et al. 2016. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *ICML.*
- Li et al. 2016. Gated Graph Sequence Neural Networks. *ICLR.*
- Ying et al, 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *NeurIPS*.

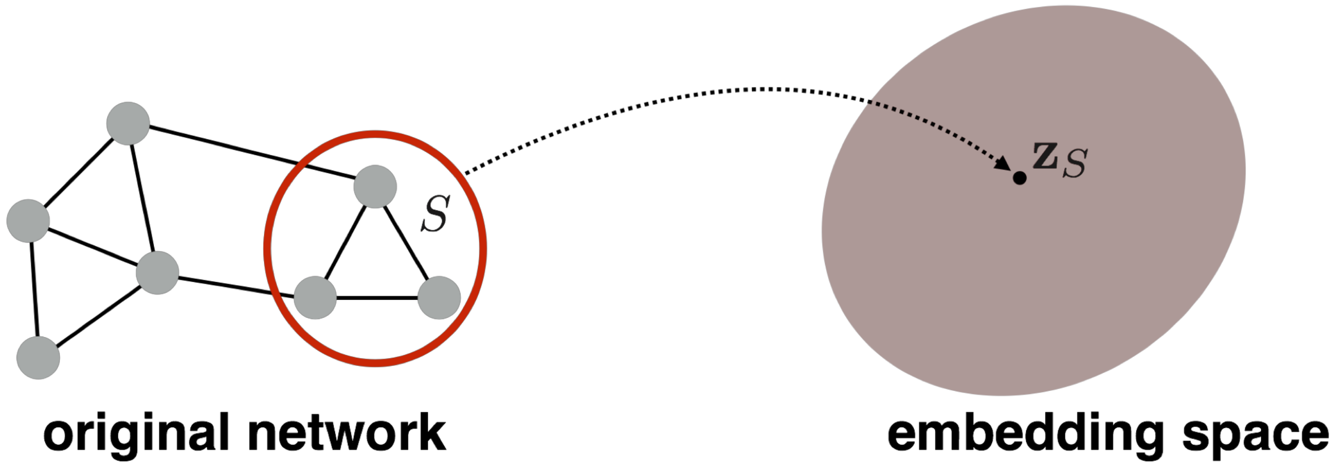
(Sub)graph Embeddings

- So far we have focused on node-level embeddings…



(Sub)graph Embeddings

- But what about subgraph embeddings?



Approach 1

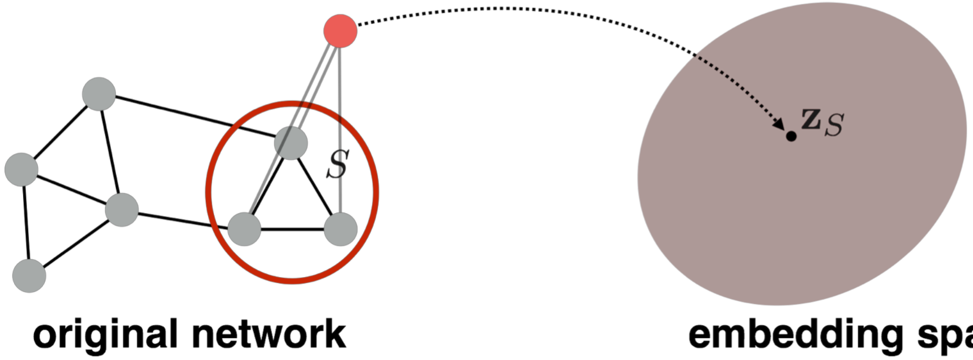
- **Simple idea:** Just sum (or average) the node embeddings in the (sub)graph

$$\mathbf{z}_S = \sum_{v \in S} \mathbf{z}_v$$

- Used by [Duvenaud et al., 2016]() to classify molecules based on their graph structure.

Approach 2

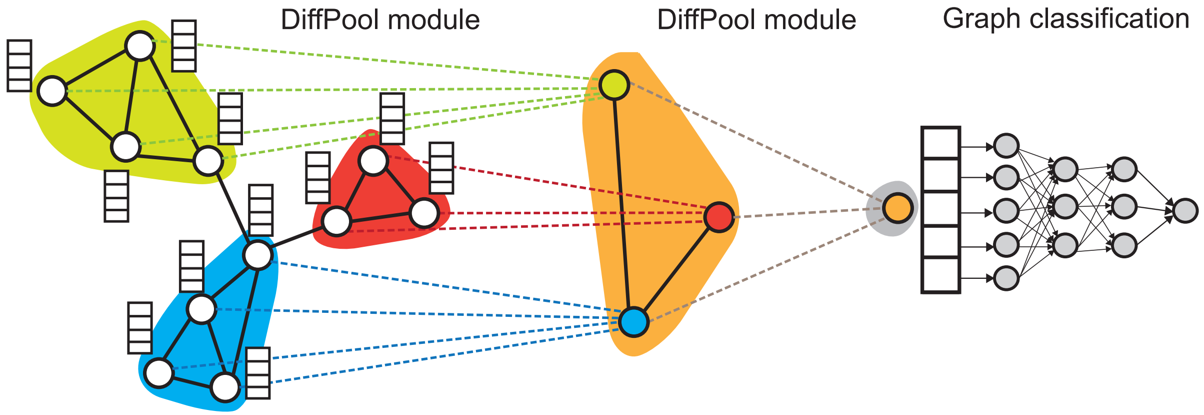
- **Idea:** Introduce a "virtual node" to represent the subgraph and run a standard graph neural network.



- Proposed by [Li et al., 2016]() as a general technique for subgraph embedding.

Approach 3

- **Idea:** Learn how to hierarchically cluster the nodes.



- First proposed by [Ying et al., 2018](#) and currently SOTA(?).

Approach 3

- **Idea:** Learn to hierarchically cluster the nodes.
- Basic overview:
  1. Run GNN on graph and get node embeddings.
  2. Cluster the node embeddings together to make a "coarsened" graph.
  3. Run GNN on "coarsened" graph.
  4. Repeat.
- Different approaches to clustering:
  - Soft clustering via learned softmax weights (Ying et al., 2018)
  - Hard clustering (Cangea et al., 2018 and Gao et al., 2018)