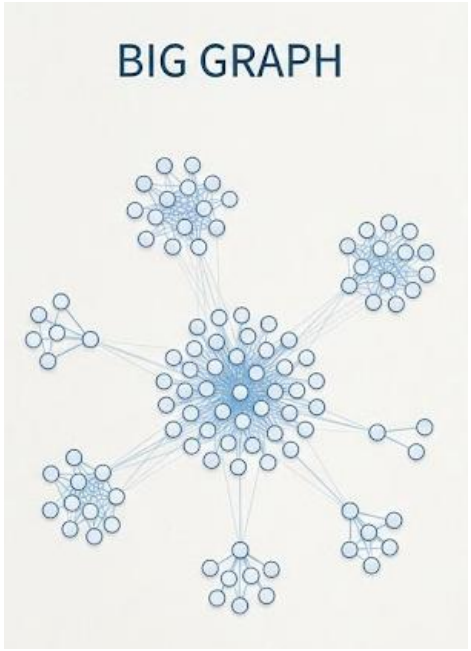
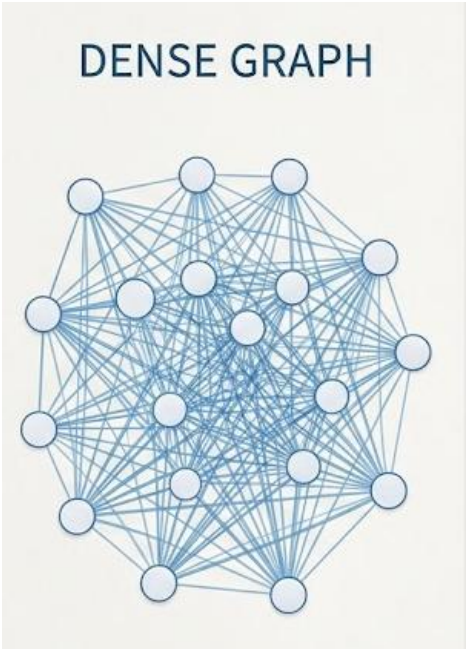
Scalability in GNNs

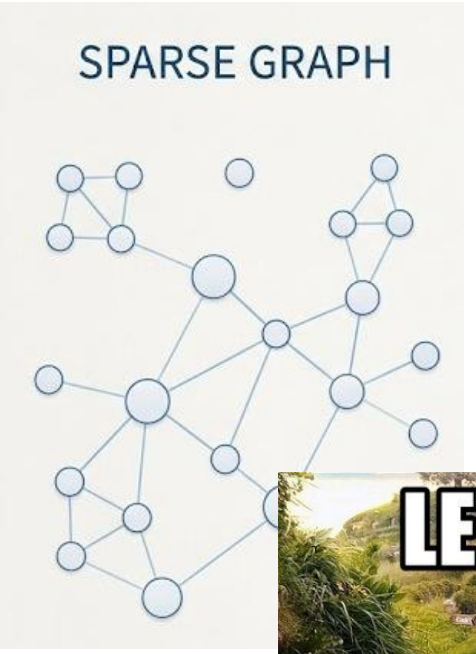
StructML - Tutorial 7

Categorizing Scalabilities in GNNs

We focus on 3 scale challenges in graph topologies







⇒ *how does that limit GNNs?*

Outline

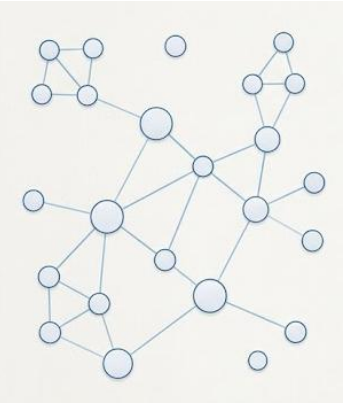
- Sparse Graphs
- Large Graphs
  - Cluster-GCN
- Dense Graphs
  - GraphSAGE
  - Oversmoothing
  - Oversquashing

Outline

- Sparse Graphs
- Large Graphs
  - Cluster-GCN
- Dense Graphs
  - GraphSAGE
  - Oversmoothing
  - Oversquashing

Sparse Graphs

- 💡 We can think of the GNN as passing signals along nodes
  - Sparse graph ⇒ weak signals



- *Risk*: Essential information between nodes will not flow in Message passing
  - Isolated nodes will not receive enough signals
  - Might cause overfitting for centric nodes

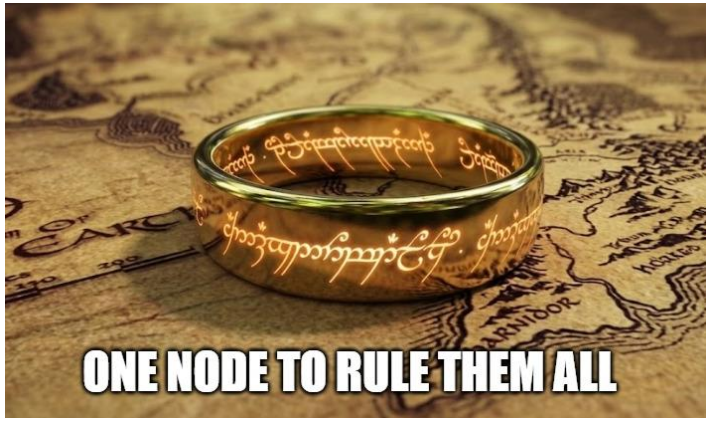
How can we deal with sparse graphs?

Virtual Node for Sparse Graphs

Solution: Adding a virtual master node

✓ Improves connectivity for all nodes

⚠ *Careful*: Adding a virtual node might create unwanted noise

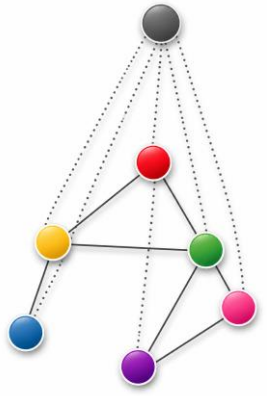


Good for

tasks with a
global perspective
⇒ More graph-level tasks
Example: Molecules – toxic or not

Bad for

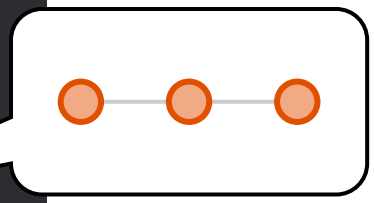
Tasks with a
local perspective
⇒ More node-level tasks
Example: Traffic prediction in road map graph



Reference: Neural Message Passing for Quantum Chemistry

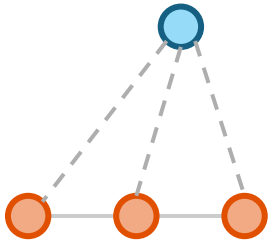
Virtual Node in PyG

```python
import torch
from torch_geometric.data import Data
import torch_geometric.transforms as T

edge_index = torch.tensor([[0, 1, 1, 2],
                           [1, 0, 2, 1]], dtype=torch.long)
x = torch.randn(3, 16) # 3 nodes, 16 features
data = Data(x=x, edge_index=edge_index)

transform = T.VirtualNode()
new_data = transform(data)

print(f"Original nodes: {data.num_nodes}")
print(f"Nodes after transform: {new_data.num_nodes}")
print(f"Original edges: {data.edge_index.size(1)}")
print(f"Edges after transform: {new_data.edge_index.size(1)}")
```



Automatically add a virtual node

```
Original nodes: 3
Nodes after transform: 4
Original edges: 4
Edges after transform: 10
```



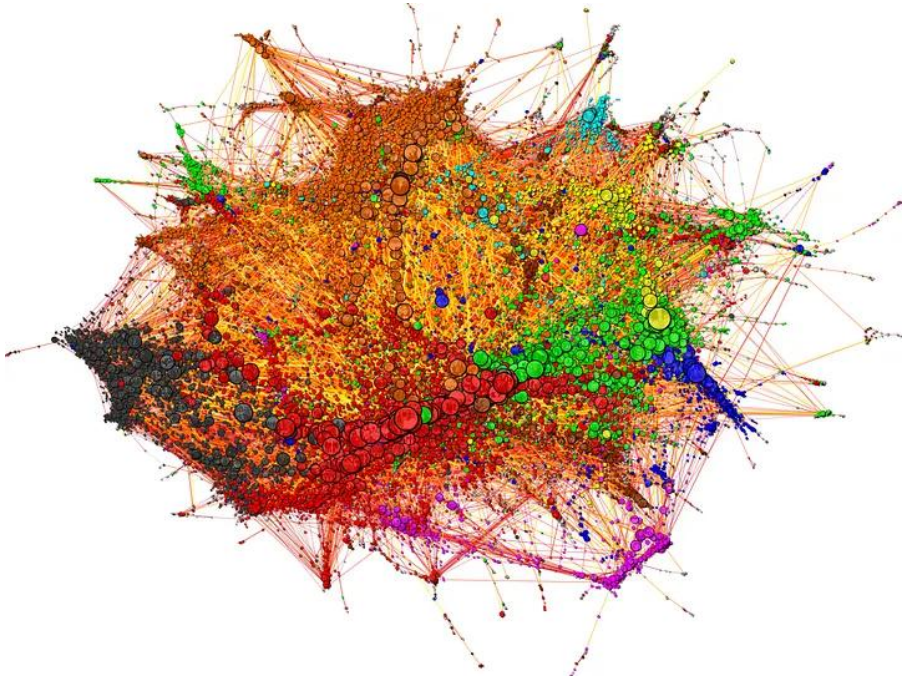
Outline

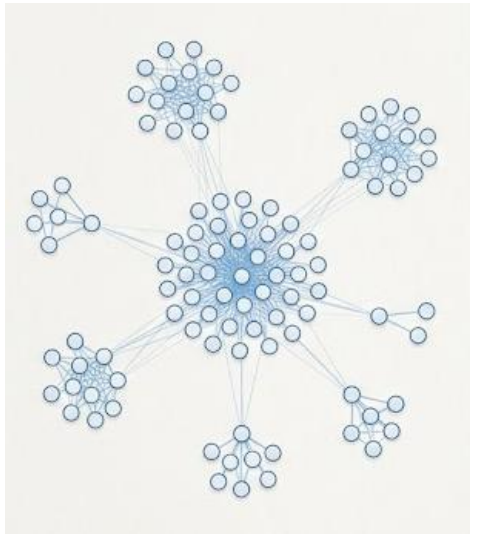
- Sparse Graphs
- Large Graphs
  - Cluster-GCN
- Dense Graphs
  - GraphSAGE
  - Oversmoothing
  - Oversquashing

Large Graphs

- In the real world, graphs are huge. Executing message passing challenges hardware even today.

  Aim: sample a subgraph and apply message passing only on it.





Cluster-GCN

💡 *Idea*: Apply mini-batching to GNNs

Partition: Divide the graph into C clusters

↓

Sample Clusters: Randomly select $q < C$ clusters

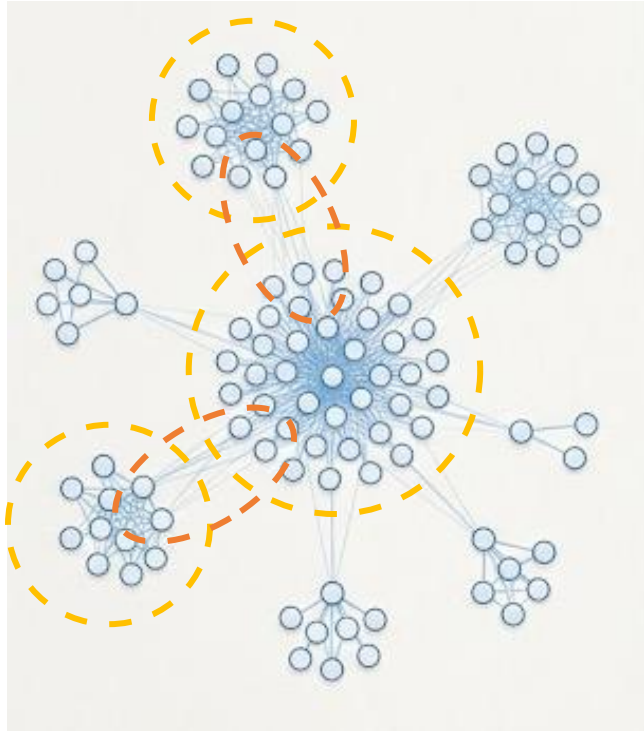
↙ ↖

Generate Subgraph: All nodes from all clusters + edges from the clusters & cross-clusters

→

Restricted Message Passing: Run an MPNN over the generated subgraph

$q = 3$



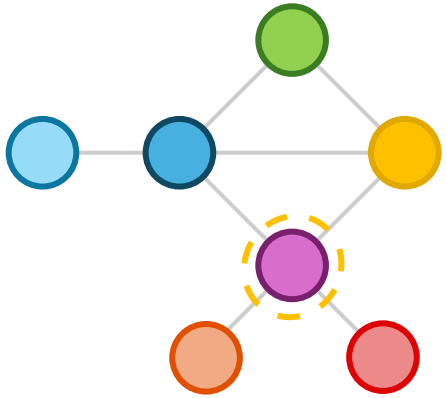
⇒ *Graph-wise sampling*

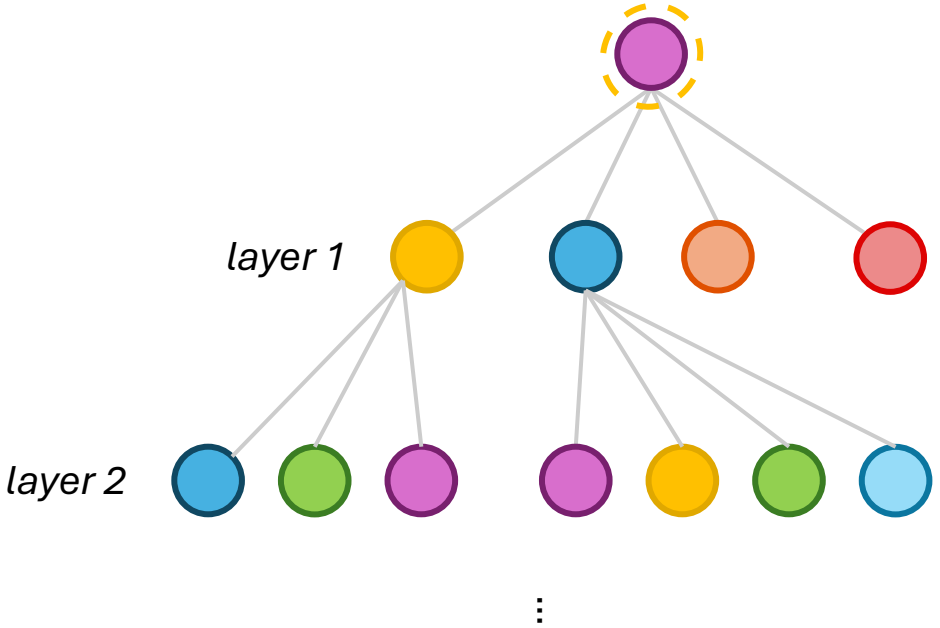
❓ *Why is this good?*

Reference: Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks

Cluster-GCN Advantage

- The computational graph of a node in message passing can be described as a tree
  - Namely the “**receptive field**”





- Receptive fields of nodes in the same cluster highly coincide
  ⇒ Sharing computation in backprop
  ⇒ Improved efficiency ✓

Cluster-GCN in Code

```python
class DeepGCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = GCNConv(dataset.num_node_features, 32)
        self.conv2 = GCNConv(32, 32)
        self.conv3 = GCNConv(32, dataset.num_classes)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1(x, edge_index))
        x = F.dropout(x, training=self.training)
        x = F.relu(self.conv2(x, edge_index))
        x = F.dropout(x, training=self.training)
        x = self.conv3(x, edge_index)
        return F.log_softmax(x, dim=1)
```

How to choose the number of clusters?
→ Sweetspot of runtime vs performance

```python
from torch_geometric.data import ClusterData, ClusterLoader

dataset = Planetoid(root='/tmp/Cora', name='Cora')
data = dataset[0]
...
model_cluster = DeepGCN().to(device)
cluster_data = ClusterData(data, num_parts=128,
                           save_dir=dataset.processed_dir)
train_loader = ClusterLoader(cluster_data, batch_size=20, shuffle=True)

for epoch in range(20):
    total_loss = 0
    for batch in train_loader:
        batch = batch.to(device)
        optimizer_cluster.zero_grad()
        out = model_cluster(batch.x, batch.edge_index)
        loss = F.nll_loss(out[batch.train_mask], batch.y[batch.train_mask])

        loss.backward()
        optimizer_cluster.step()
        total_loss += loss.item()
```

Clustering

Total clusters

Output path

Subsampler

Clusters per batch (q)

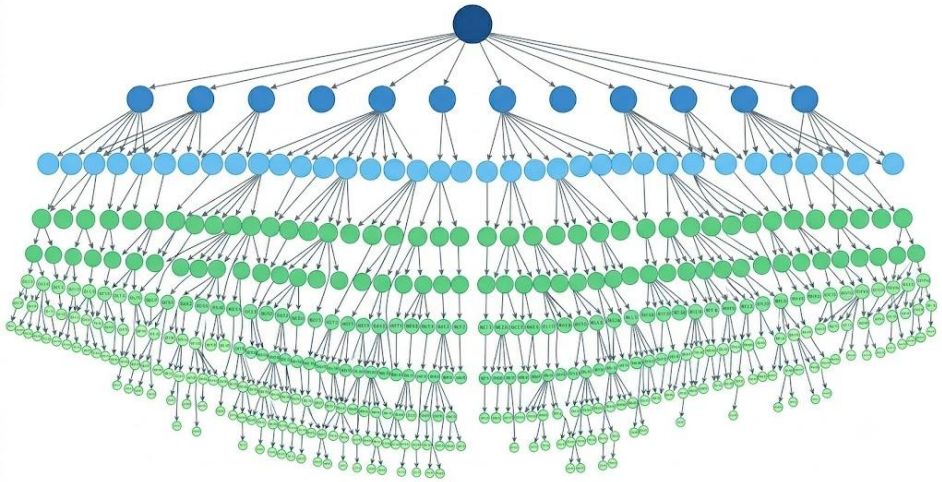
Batch sampling

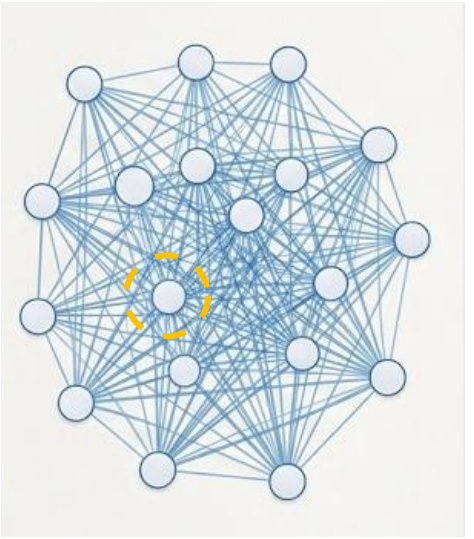
Outline

- Sparse Graphs
- Large Graphs
  - Cluster-GCN
- Dense Graphs
  - GraphSAGE
  - Oversmoothing
  - Oversquashing

Dense Graphs

- *Problem*: Lots of neighbors ⇒ Bigger receptive field
  - Exponential growth $O(d^K)$
    - d – average degree, K – number of layers
  - MPNN requires to fetch all receptive fields into memory



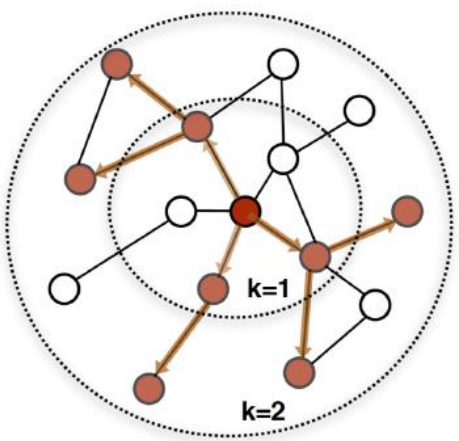


GraphSAGE

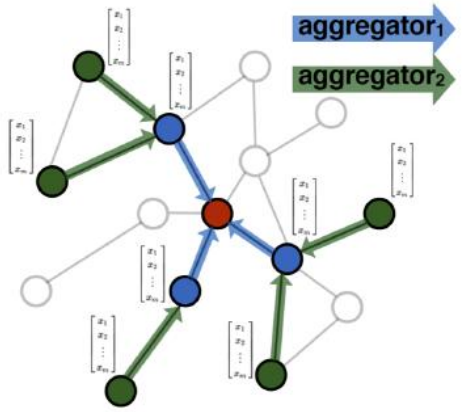
MPNN Architecture - **S**ample & **AG**gregat**E**

💡 *Idea*: Sample neighbors ⇒ node-wise sampling

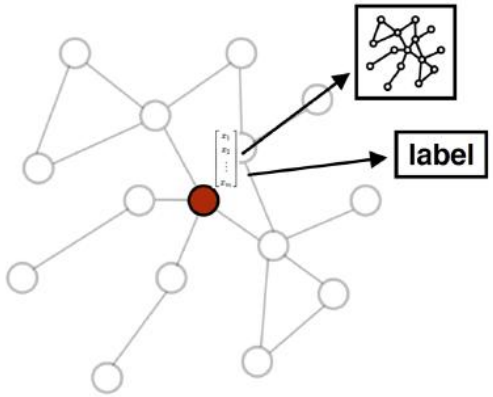
- Also – introduce learnable parameters out of update scope



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

Reference: Inductive Representation Learning on Large Graphs

GraphSAGE

Message Passing NN ⇒ Need to plug in:

Propagation

$$\mathrm{m}_u^{(k)} := \alpha^{(k)}\left(\left\{\left\{ M^{(k)}\left(\mathrm{h}_u^{(k-1)}, \mathrm{h}_v^{(k-1)}\right) \mid v \in N(u) \right\}\right\}\right)$$

Update

$$\mathrm{h}_u^{(k)} := U^{(k)}\left(\mathrm{h}_u^{(k-1)}, m_u^{(k)}\right)$$

$$\mathrm{h}_u^{(k)} = \sigma\left(W^{(k)} \cdot \left[h_u^{(k-1)} \oplus m_u^{(k)}\right]\right)$$

Elementwise average

msg $M^{(k)}\left(\mathrm{h}_u^{(k-1)}, \mathrm{h}_v^{(k-1)}\right) = \mathrm{h}_v^{(k-1)}$

agg $\alpha^{(k)} = \mathrm{MEAN}$

Max-pooling

msg $M^{(k)}\left(\mathrm{h}_u^{(k-1)}, \mathrm{h}_v^{(k-1)}\right) = \sigma\left(W_{pool}\mathrm{h}_v^{(k-1)} + b\right)$

More learnable Parameters in msg

agg $\alpha^{(k)} = \mathrm{MAX}$

LSTM

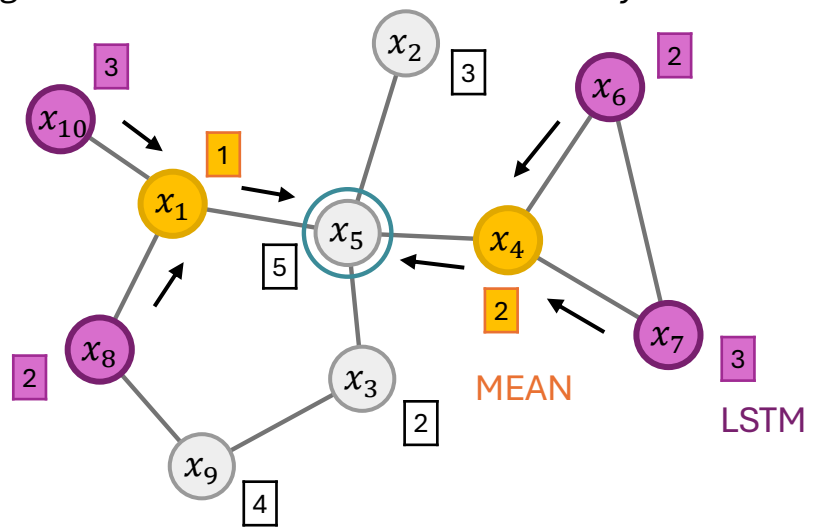
msg $M^{(k)}\left(\mathrm{h}_u^{(k-1)}, \mathrm{h}_v^{(k-1)}\right) = \mathrm{h}_v^{(k-1)}$

agg $\alpha^{(k)} = \mathrm{LSTM}$ → *Sequential architecture with learnable parameters (shuffles nodes)*

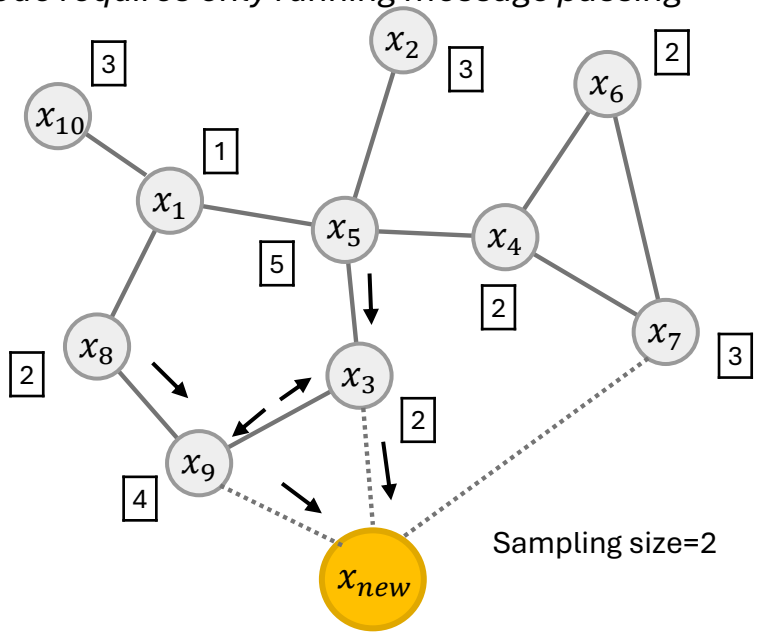
⚠ *Not perm. invariant* ←

GraphSAGE

Different aggregations can be combined in different layers:



Adding a node requires only running message passing



Loss

Unsupervised (negative sampling)

$$J_{\mathcal{G}}(z_u) = -\log\left(\sigma(z_u^\top z_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma\left(-z_u^\top z_{v_n}\right)\right)$$

- v – a node that co-occurs near u on fixed-length random walk
- P_n - negative sampling distribution
- σ – nonlinearity
- Q – number of negative samples

Supervised – using a classification task (Cross Entropy)

$$\mathcal{L}_{\text{sup}} = -\sum_{u \in \mathcal{V}_{train}} y_u \log(\hat{y}_u)$$

$$\hat{y}_u = \text{softmax}(\mathbf{W_{cls}} \mathbf{z_u})$$

GraphSAGE

The good:

✅ *Inductive*

New nodes don't require a whole graph recalculation

✅ *Scalable*

Nodes sampling makes the architecture more scalable

✅ *Flexible*

Multiple aggregation functions to choose from

The bad:

❌ *Information Loss*

Inherent node sampling reduces information

❌ *Non-deterministic*

Randomized sampling makes message passing non-deterministic

❌ *Exponential*

Even with sampling – receptive field growth is exponential 🔎

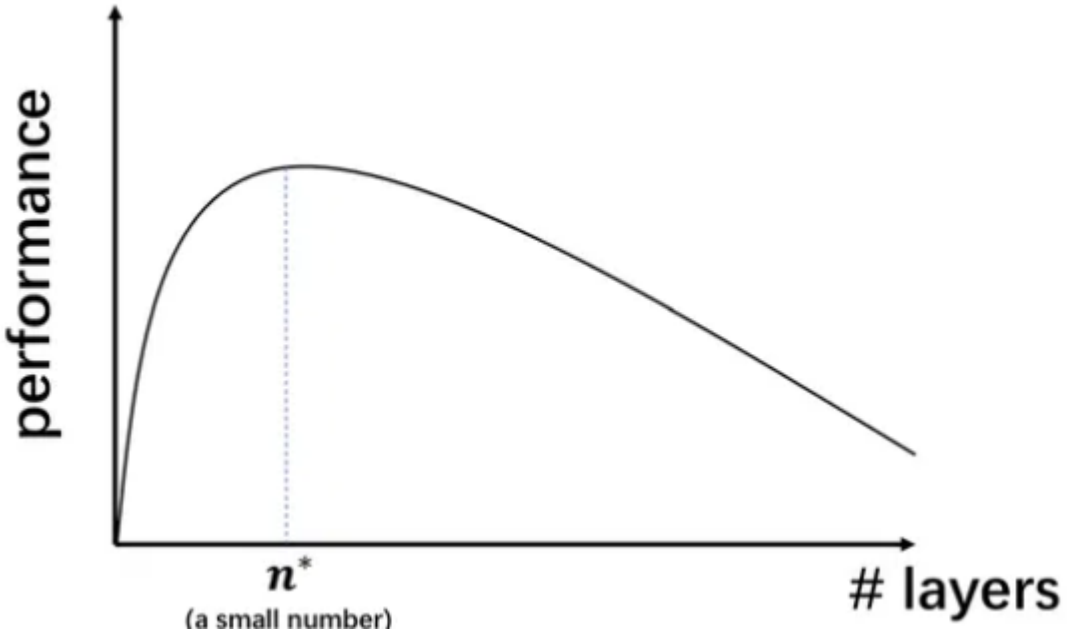
How Deep can GNNs be?

Domain	Model	Layers	Primary Resource
Vision	**ResNet-152**	152	*Deep Residual Learning for Image Recognition* (He et al., 2015)
	EfficientNet-B7	813	*EfficientNet: Rethinking Model Scaling for ConvNets* (Tan & Le, 2019)
Language	**BERT-Large**	24	*BERT: Pre-training of Deep Bidirectional Transformers* (Devlin et al., 2018)
	GPT-3	96	*Language Models are Few-Shot Learners* (Brown et al., 2020)
Graph	**GCN**	2–3	*Semi-Supervised Classification with GCNs* (Kipf & Welling, 2017)
	GraphSAGE	2–3	*Inductive Representation Learning on Large Graphs* (Hamilton et al., 2017)
	GAT	2–3	*Graph Attention Networks* (Veličković et al., 2018)
	GIN	5	*How Powerful are Graph Neural Networks?* (Xu et al., 2019)

Why so little layers for GNNs?

How Deep can GNNs be?

The phenomenon as it's seen in experiments:



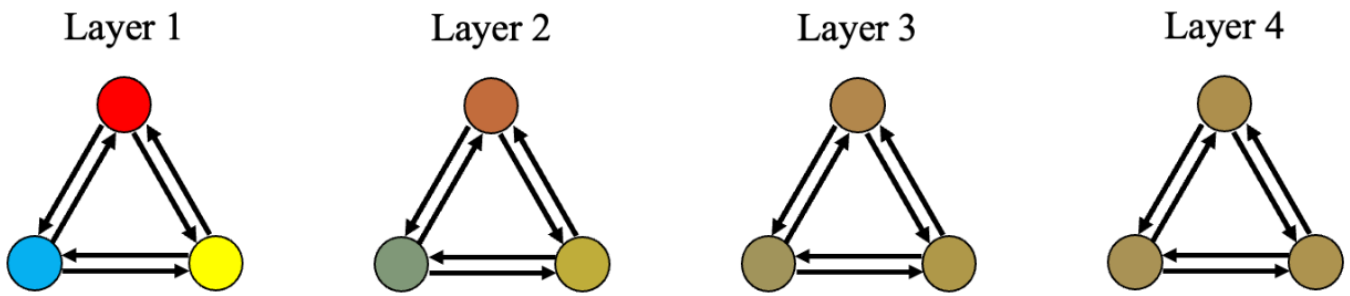
Reference: Demystifying Oversmoothing in Attention-Based Graph Neural Networks

Outline

- Sparse Graphs
- Large Graphs
  - Cluster-GCN
- Dense Graphs
  - GraphSAGE
  - Oversmoothing
  - Oversquashing

Oversmoothing

- Highly connected components “average away” the message
  - Every layer makes the node representations more similar
- After a few layers, the unique messages “vanish”
  - Exponentially



https://julballa.github.io/6s898-final-blog/blog/

Measuring Oversmoothing

A function $\mu: \mathbb{R}^{|V| \times d} \longrightarrow \mathbb{R}_{\geq 0}$

- Smaller value ⇒ more similarity
  - $\mu(X) = 0 \Leftrightarrow X_i = c \quad \forall i \in V$
- Triangular inequality
  - $\mu(X + Y) \leq \mu(X) + \mu(Y)$

Oversmoothing:

$$\mu\left(X^{(k)}\right) \leq C_1 e^{-C_2 k}, \text{ for } k \in \{0, \ldots, N\} \text{ with some constants } C_1, C_2 > 0.$$

Reference: A Survey on Oversmoothing in Graph Neural Networks

Node Similarity Candidates

MAD – Mean Average Distance

$$\mu_{MAD}\left(X^{(k)}\right) = \frac{1}{|V|}\sum_{u\in V}\sum_{v\in N(u)} 1 - \underbrace{\frac{{z_u^{(k)}}^{\top} z_v^{(k)}}{|z_u^{(k)}||z_v^{(k)}|}}_{\text{Cosine similarity}}$$

- ✓ Scale stability due to normalization
- x Doesn't satisfy triangle inequality
- x Less efficient – due to denominator

Dirichlet Energy

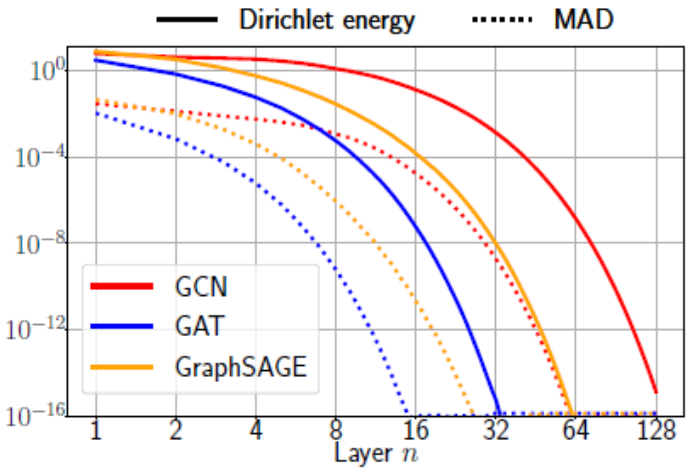
$$\mu_D\left(X^{(k)}\right) = \frac{1}{|V|}\sum_{u\in V}\sum_{v\in N(u)} |z_u^{(k)} - z_v^{(k)}|_2^2$$

- ✓ More efficient – no denominator
- ✓ Satisfies requirements from oversmoothing measurement
- x Scale instability

Oversmoothing in Experiments

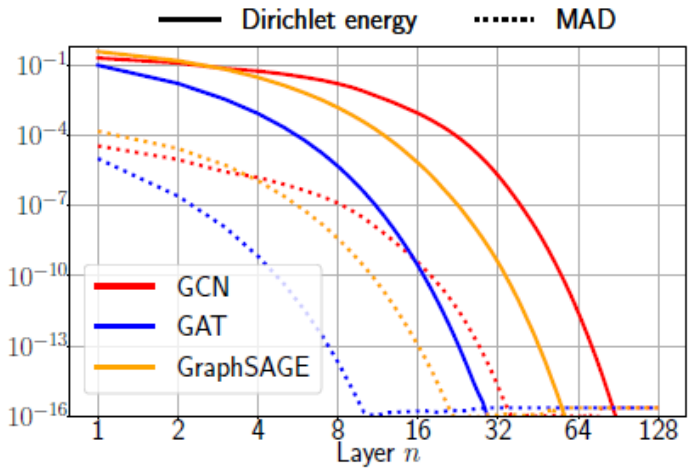
- Evaluating oversmoothing over different datasets and measures

Small scale



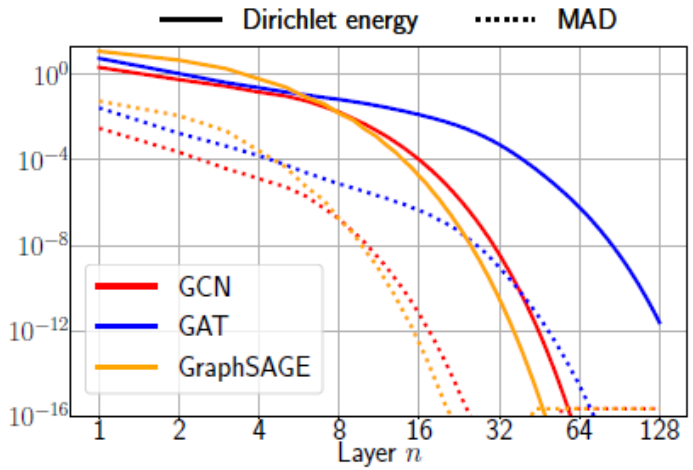
Dataset: Texas graph

Medium scale



Dataset: Cora

Large scale



Dataset: Facebook network (Cornell5)

Reference: A Survey on Oversmoothing in Graph Neural Networks

Mitigating Oversmoothing

DropEdge

Randomly remove edges at each training epoch

$$A_{drop} = A - A_{mask}$$

- A - adjacency matrix
- A_{mask} - random masking of A

⇒ reduces density ⇒ reduces oversmoothing

Skip Connection

Keep initial features for further layers

$$X^{(k+1)} = \sigma\left(\left((1-\alpha)\hat{A}X^{(k)} + \alpha X^{(0)}\right)W^{(k)}\right)$$

- $X^{(k)}$ - node features at layer k
- $\hat{A} = \widehat{D}^{-\frac{1}{2}}(A+I)\widehat{D}^{-\frac{1}{2}}$
- α – tuning hyperparameter (not learnable)
- $W^{(k)}$ – the learnable matrix at layer k
- σ – nonlinearity

⇒ prevents “washing away” initial distinct features

Reference: DropEdge: Towards Deep Graph Convolutional Networks on Node Classification

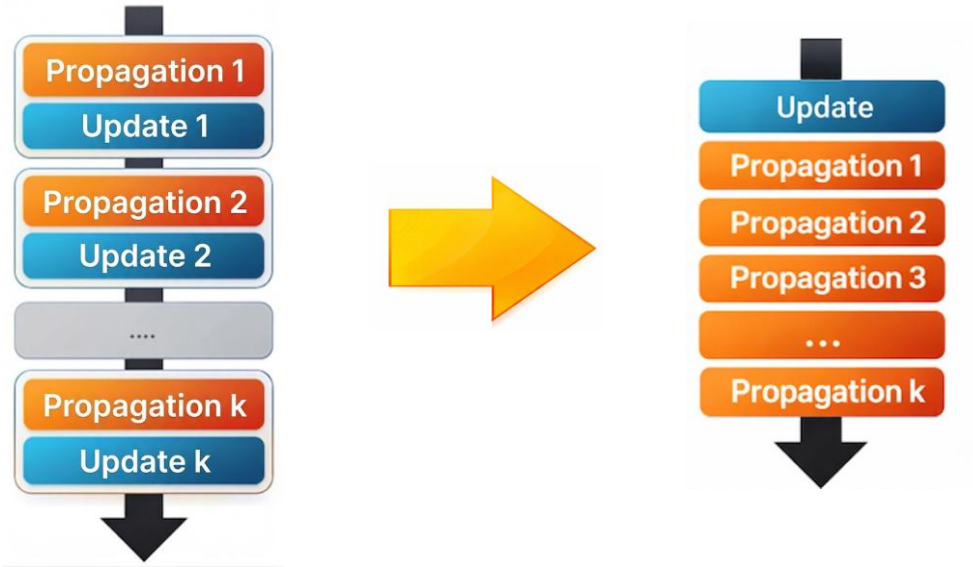
Reference: Simple and Deep Graph Convolutional Networks

Reference: Residual Networks Behave Like Ensembles of Relatively Shallow Networks

Deep Adaptive Graph Neural Networks (DAGNN)

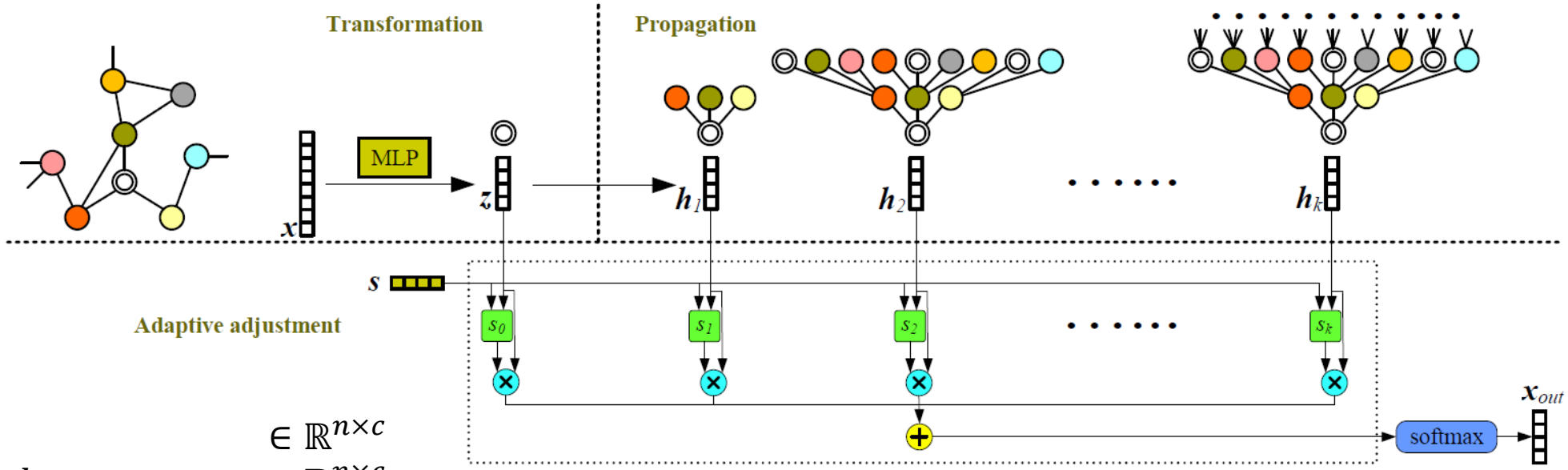
One step further: Decouple propagation from update

- Collect all receptive fields at once and take the best
- This way – it won't forget the original information due to oversmoothing



Reference: Towards Deeper Graph Neural Networks

Deep Adaptive Graph Neural Networks (DAGNN)

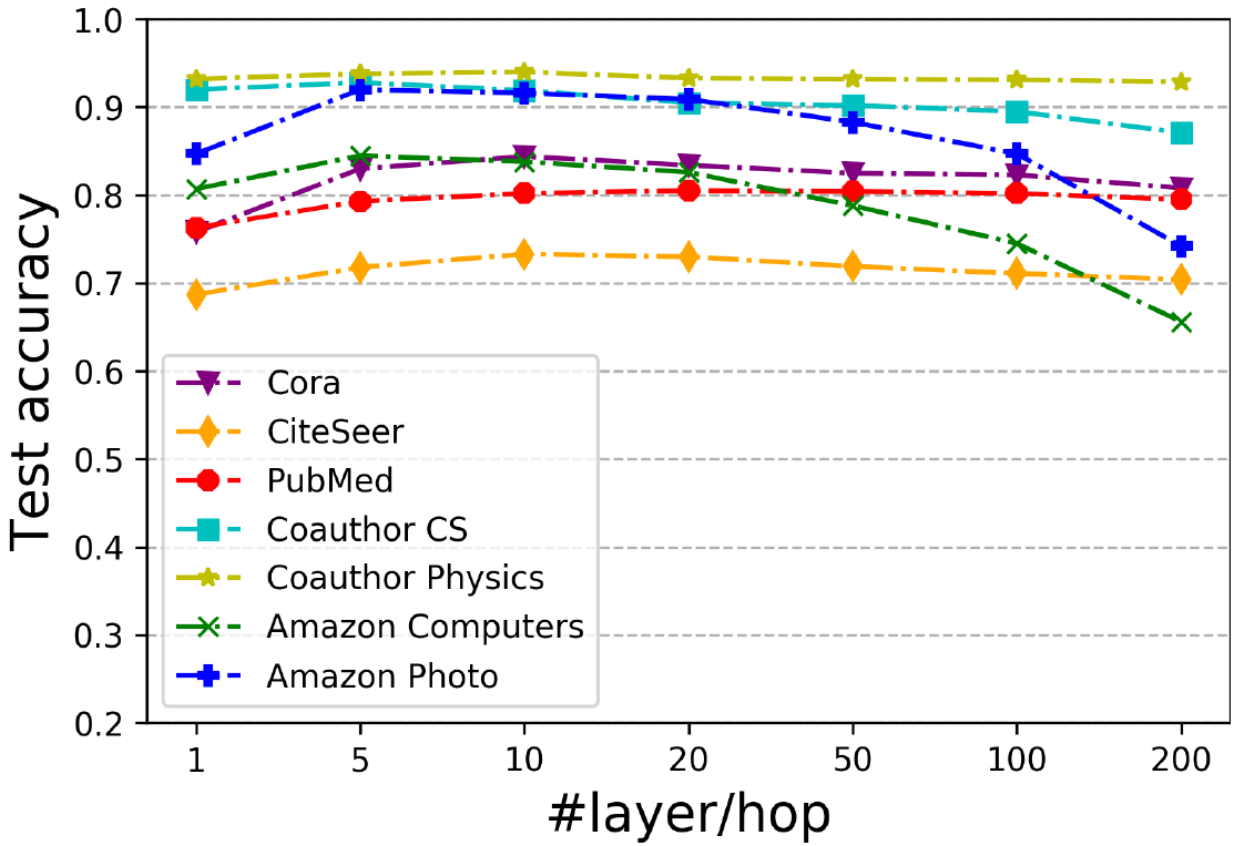


$$Z = \text{MLP}(X) \qquad \in \mathbb{R}^{n \times c}$$
$$H_\ell = \hat{A}^\ell Z, \ell = 1,2,\dots,k \qquad \in \mathbb{R}^{n \times c}$$
$$H = \text{stack}(Z, H_1, \dots, H_k) \qquad \in \mathbb{R}^{n \times (k+1) \times c}$$
$$S = \sigma(Hs) \qquad \in \mathbb{R}^{n \times (k+1) \times 1}$$
$$\tilde{S} = \text{reshape}(S) \qquad \in \mathbb{R}^{n \times 1 \times (k+1)}$$
$$X_{out} = \text{softmax}\left(\text{squeeze}\left(\tilde{S}H\right)\right) \qquad \in \mathbb{R}^{n \times c}$$

$\hat{A} = \hat{D}^{-\frac{1}{2}}(A + I)\hat{D}^{-\frac{1}{2}}$

$s \in \mathbb{R}^{c \times 1}$ - trainable projection vector

Deep Adaptive Graph Neural Networks (DAGNN)



- Drop in Amazon datasets – due to high density
  ⇒ Dense graphs are more sensitive to oversmoothing

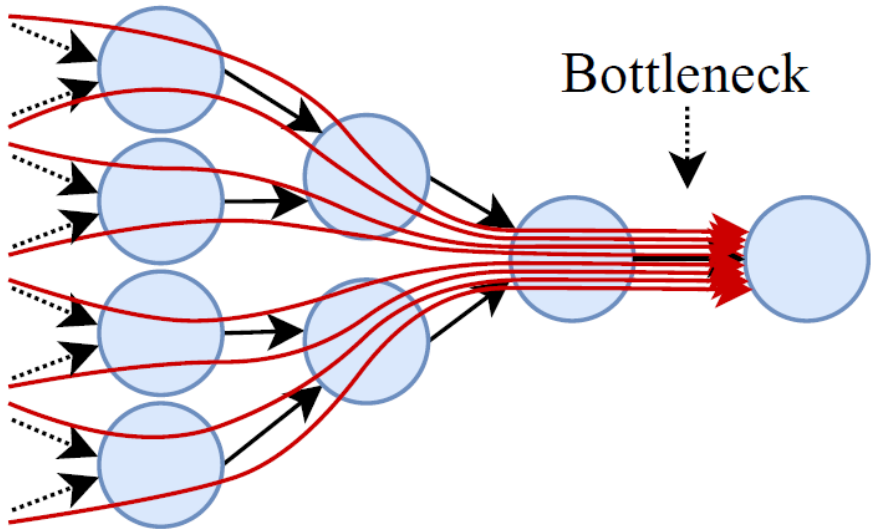
Outline

- Sparse Graphs
- Large Graphs
  - Cluster-GCN
- Dense Graphs
  - GraphSAGE
  - Oversmoothing
  - Oversquashing

Oversquashing

Problem: Exponentially aggregated information needs to be encoded in a fixed embedding size

- Long-distant nodes make less impact on a node, even though maybe they should



Mitigating Oversquashing

Method 1: Virtual Node

Add a virtual node connecting all nodes in the graph
⇒ reduces information bottleneck

⇒

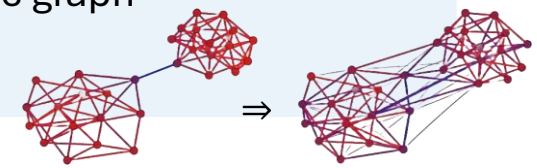
Method 2: Fully Adjacent Layer

Add connectivity – but only in last layer
⇒ And not with intermediate node – but fully connect all nodes to each other

⇒

Method 3: Smarter Rewires

Use more sophisticated algorithms to rewire graph



$S \subseteq \mathcal{V} \quad \text{vol}(S) = \sum_{i \in S} d_i$

$\partial S \triangleq \{(u, v) \in E \mid u \in S, v \in \mathcal{V} \setminus S\}$

$h_G(S) = |\partial S| / \min\big(\text{vol}(S), \text{vol}(\mathcal{V} \setminus S)\big)$

$$\boxed{h_G \triangleq \min_S h_G(S)}$$

measures oversquashing

Reference: On the Bottleneck of Graph Neural Networks and its Practical Implications

Reference: Understanding over-squashing and bottlenecks on graphs via curvature

More to Explore

- Mitigating exponential blowup
  - Layer-wise sampling – sample fixed size subgraph per layer
- Mitigating Large Graphs
  - Transfer learning: generalize from smaller graphs to bigger graphs
- Handling temporal graphs
  - Graphs change over time
  - New approach: represent graphs as series of events
    - $G = \{x(t_1), x(t_2), \ldots\}$, where $x(t)$ is:
      - $v_i(t)$ - Adding node
      - $e_{ij}(t)$ - Adding edge (interaction)