Scalability of Graph Neural Networks

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

네트워크 과학 연구실
NETWORK SCIENCE LAB

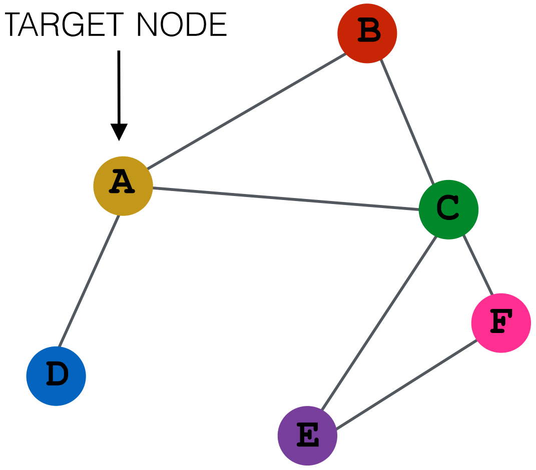
가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

Contents

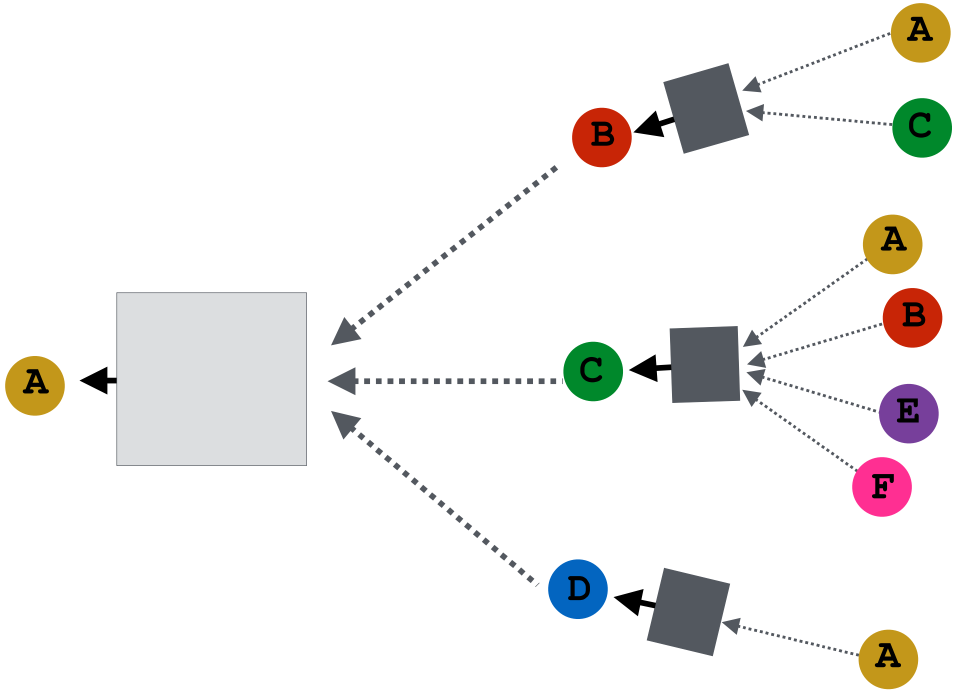
- Issues towards the large-scale GNNs
- Node-wise sampling with GraphSage
- Graph-wise sampling with ClusterGCN
- GraphSAINT

Recap: Neighborhood Aggregation

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



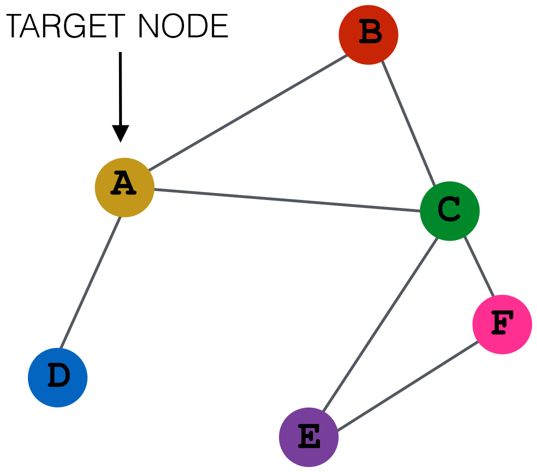
INPUT GRAPH



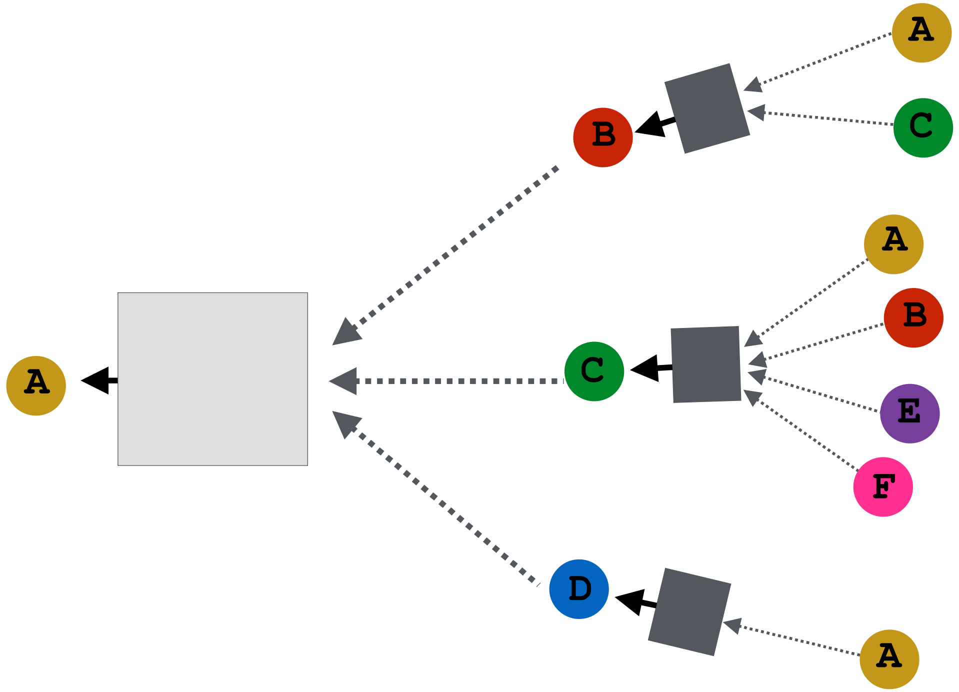
Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Recap: Neighborhood Aggregation

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



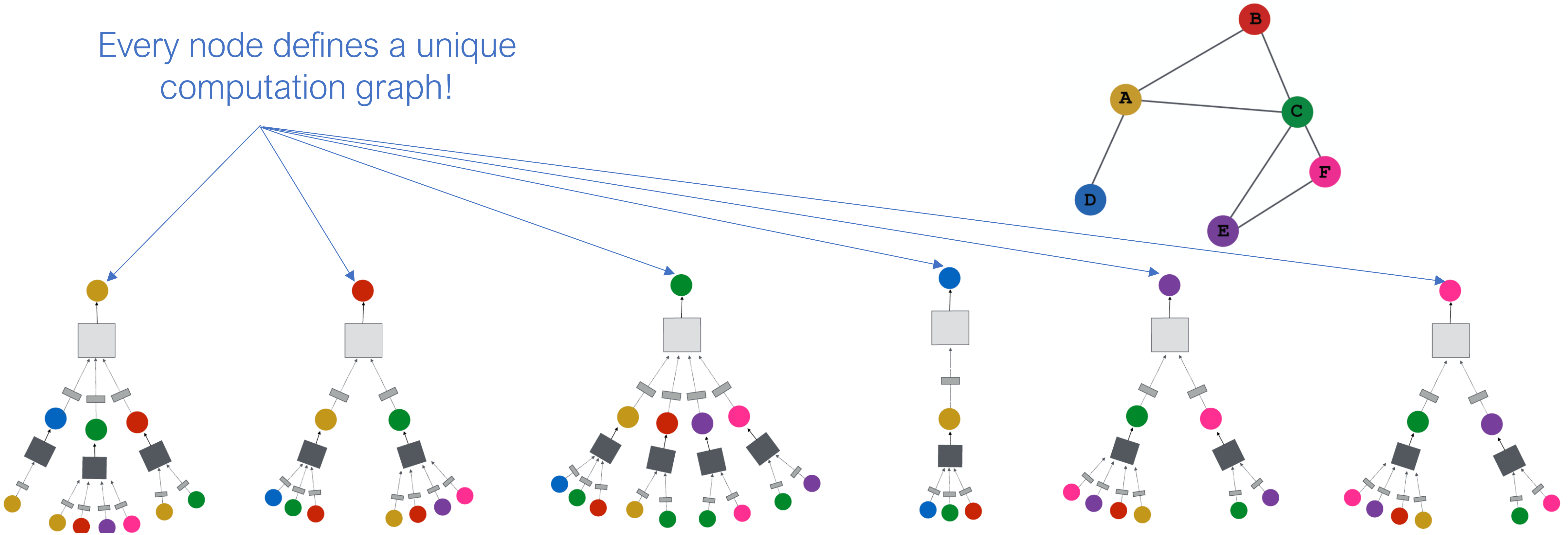
INPUT GRAPH



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Recap: Neighborhood Aggregation

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

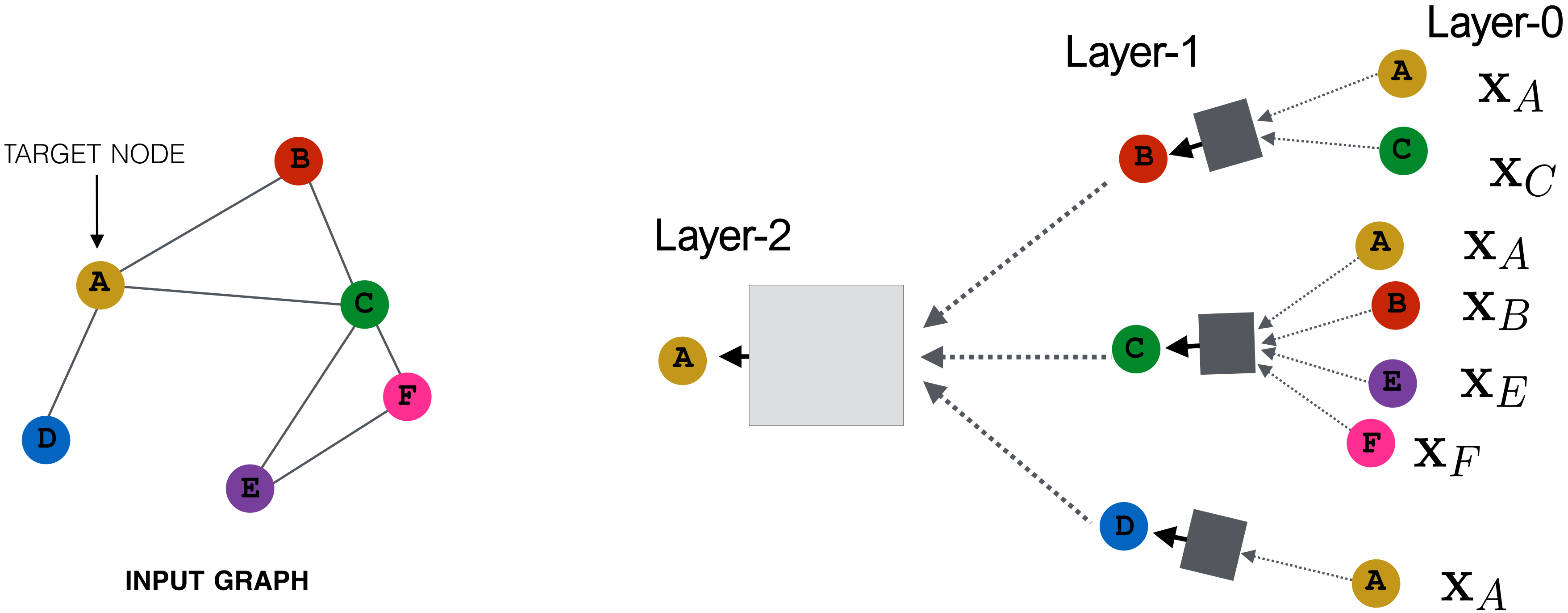


Every node defines a unique computation graph!

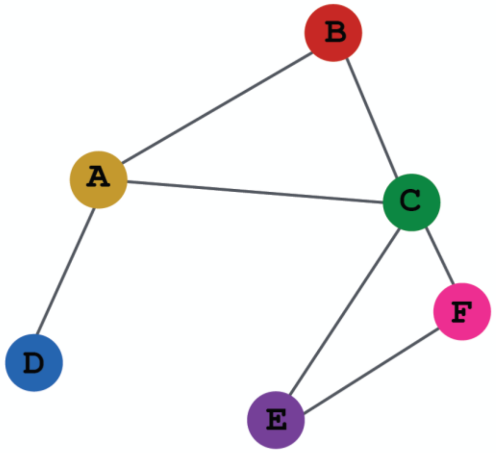
Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Recap: Neighborhood Aggregation

- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- “layer-0” embedding of node A is its input feature, i.e. x_A

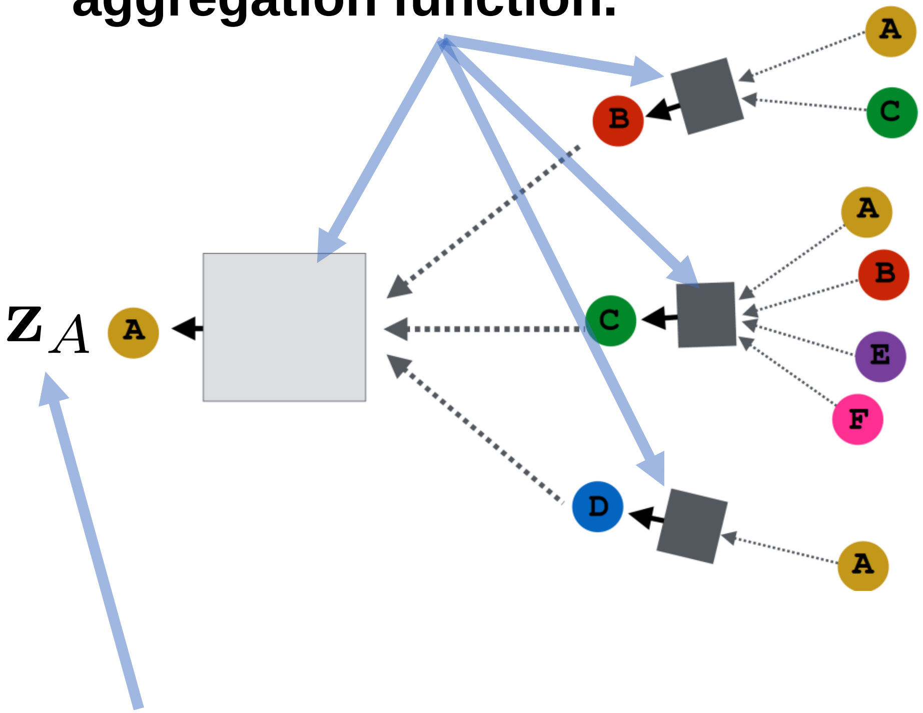


Overview of Model Design



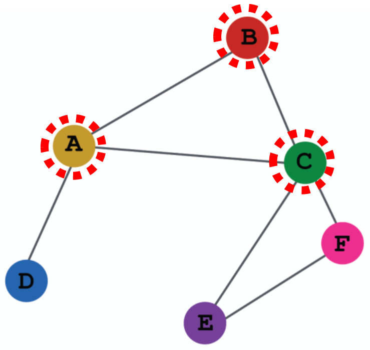
INPUT GRAPH

1) Define a neighborhood aggregation function.



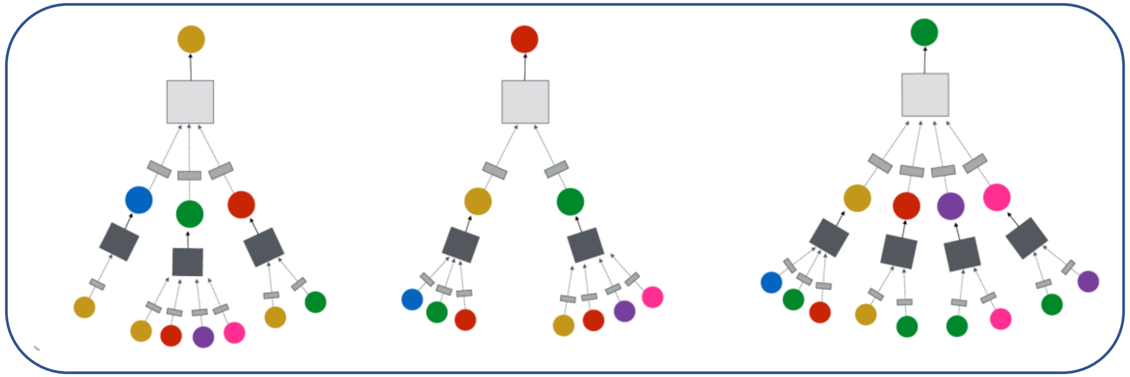
2) Define a loss function on the embeddings, $L(z_u)$

Overview of Model Design



INPUT GRAPH

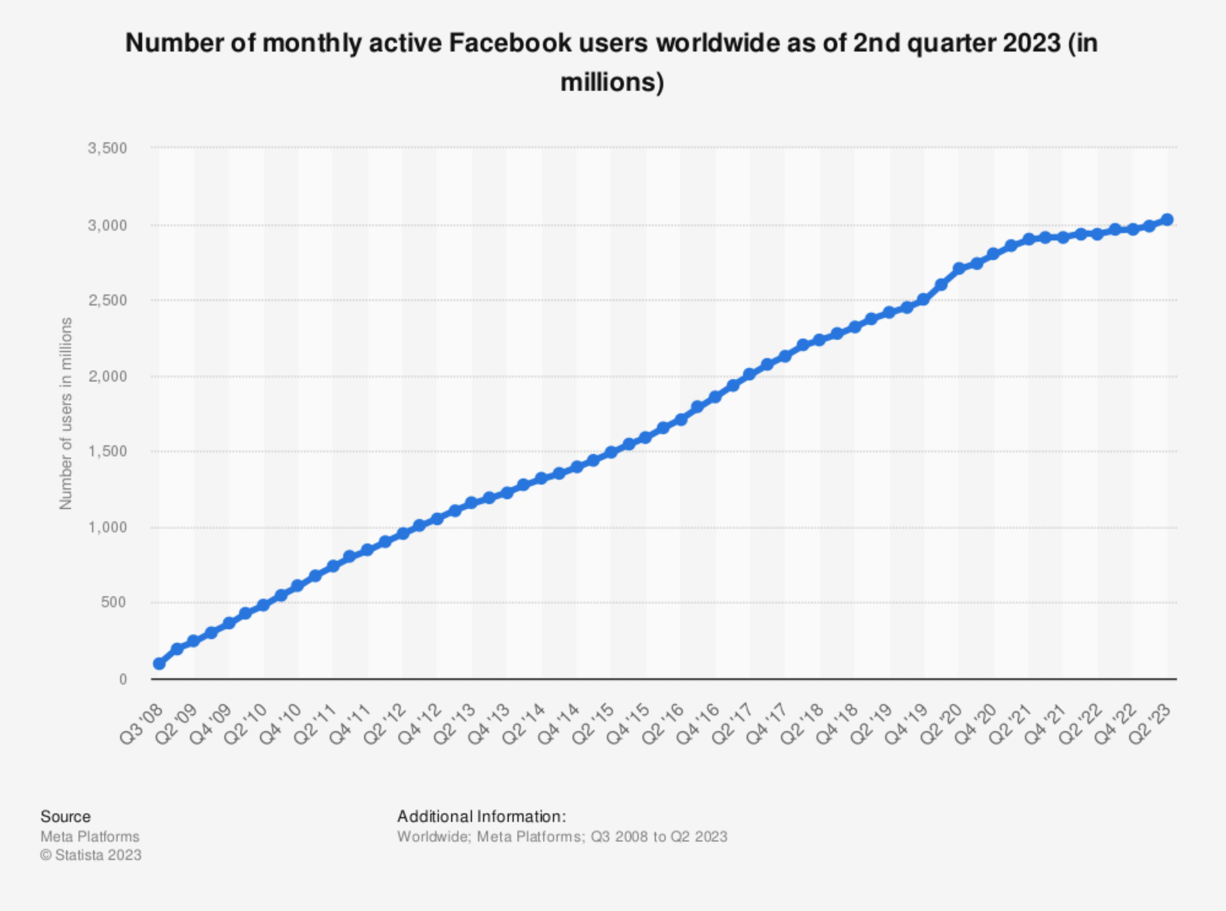
3) Train on a set of nodes, i.e., a batch of compute graphs



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Graphs in the real-world can be very large

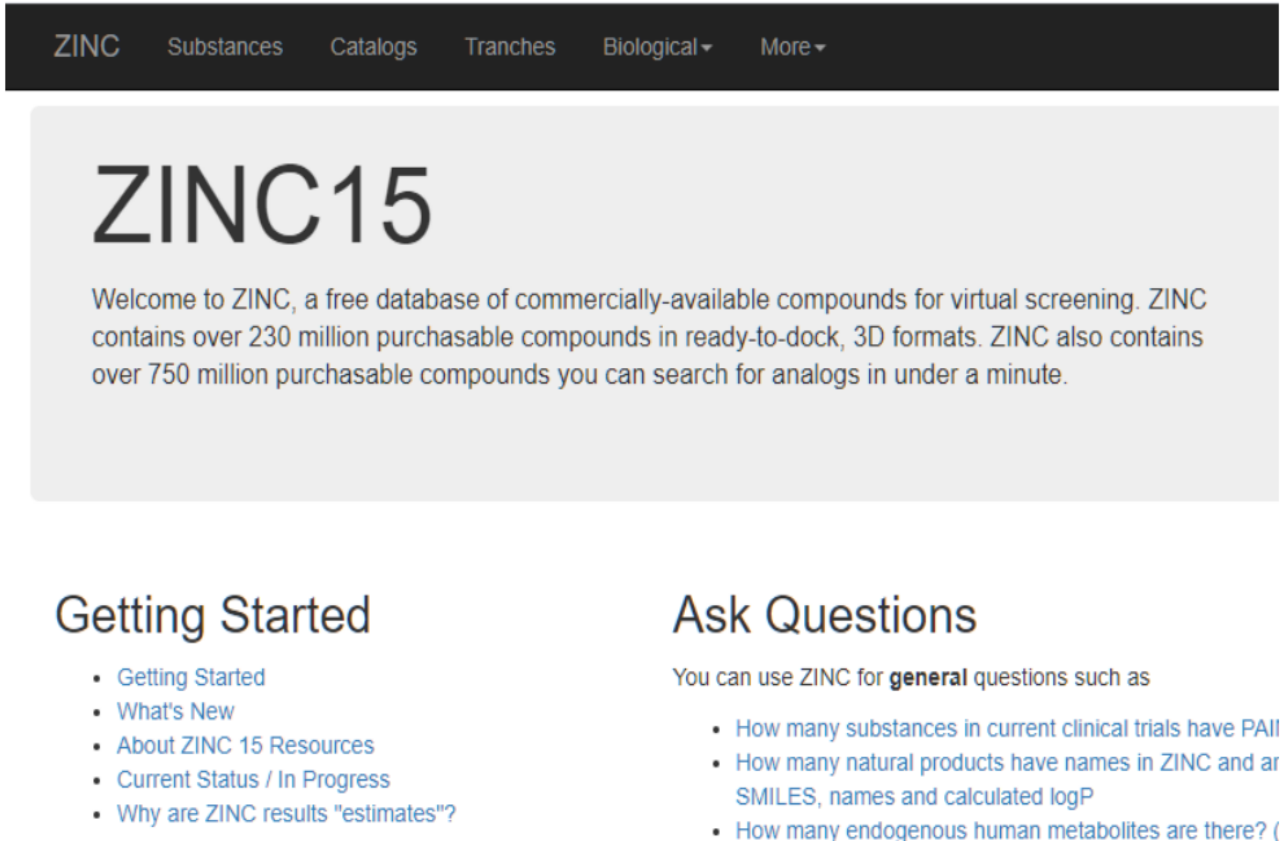
Large scale


Number of monthly active Facebook users worldwide as of 2nd quarter 2023 (in millions)

Source
Meta Platforms
© Statista 2023

Additional Information:
Worldwide; Meta Platforms; Q3 2008 to Q2 2023

Large number



ZINC15

Welcome to ZINC, a free database of commercially-available compounds for virtual screening. ZINC contains over 230 million purchasable compounds in ready-to-dock, 3D formats. ZINC also contains over 750 million purchasable compounds you can search for analogs in under a minute.

Getting Started

- Getting Started
- What's New
- About ZINC 15 Resources
- Current Status / In Progress
- Why are ZINC results "estimates"?

Ask Questions

You can use ZINC for **general** questions such as

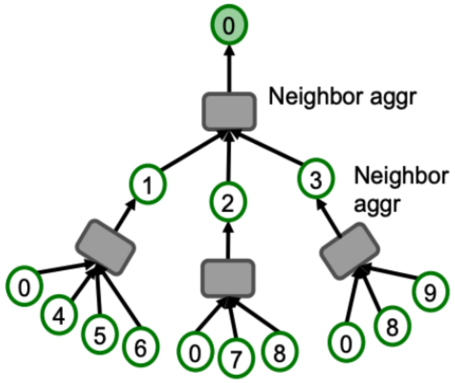
- How many substances in current clinical trials have PAII
- How many natural products have names in ZINC and ar SMILES, names and calculated logP
- How many endogenous human metabolites are there? (

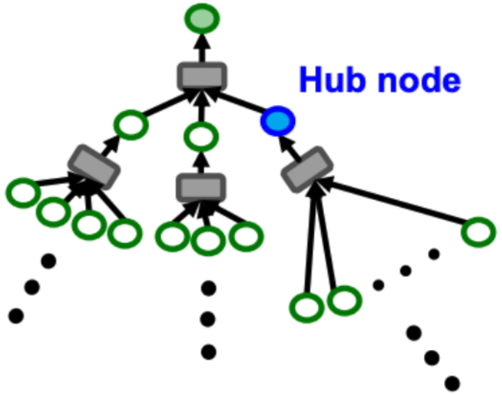
https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

Graph Convolutional Network Limitations

- **Computationally Expensive:**
  - We need to generate the complete K-hop neighborhood computational graph and then need to aggregate plenty of information from its surroundings.
  - As we go deeper into the neighborhood computation graph becomes exponentially large.

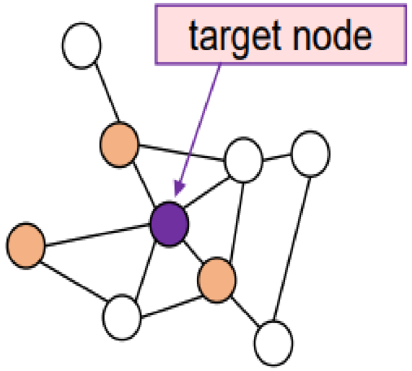
  → problem while fitting these computational graphs inside GPU memory.
- **The curse of Hub nodes or Celebrity nodes:**
  - Hub nodes are those nodes which are very high degree nodes in the graph



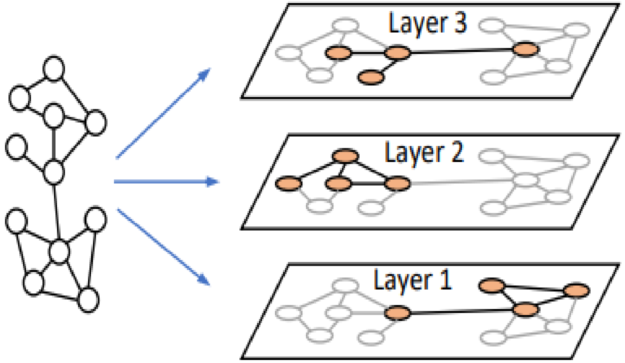


Three paradigms toward large-scale GNNs

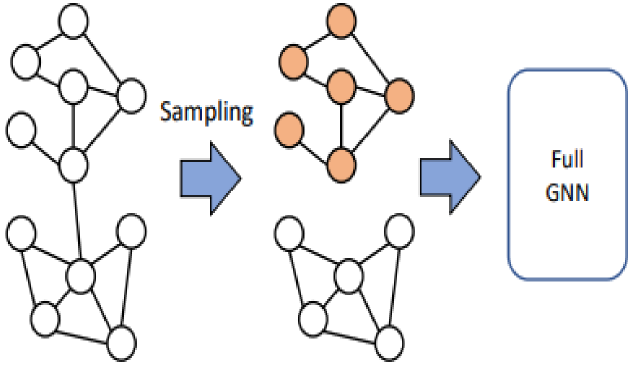
- **Why the original GNN fails on large graph?**
  - Large memory requirement.
  - Inefficient gradient update.
- **Three paradigms toward large-scale GNN:**
  - Node-wise sampling
  - Layer-wise sampling
  - Graph-wise sampling



Node-wise Sampling



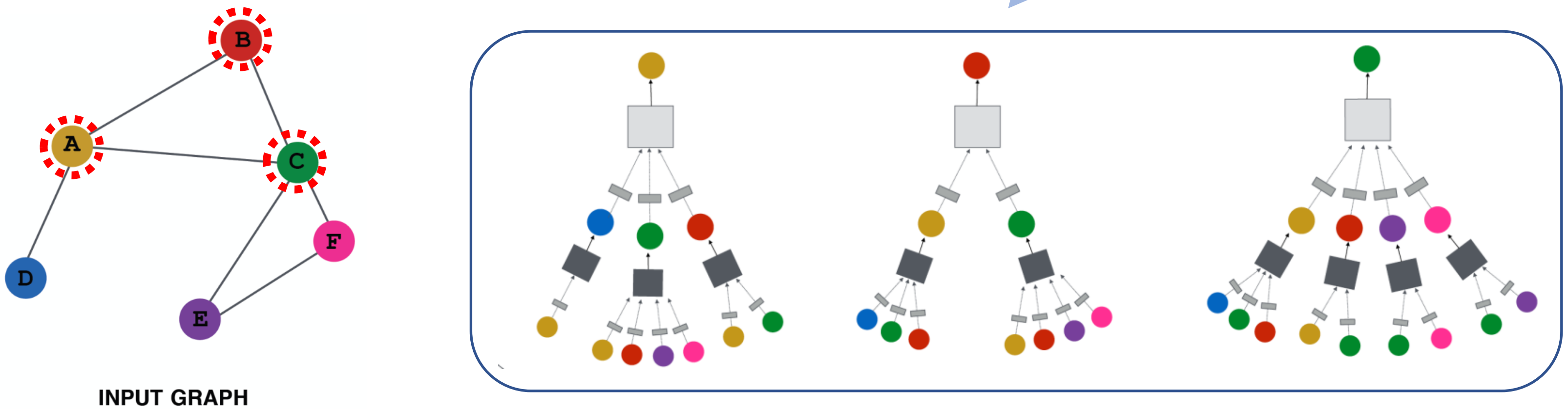
Layer-wise Sampling



Graph-wise Sampling

Two issues towards the large-scale GNNs

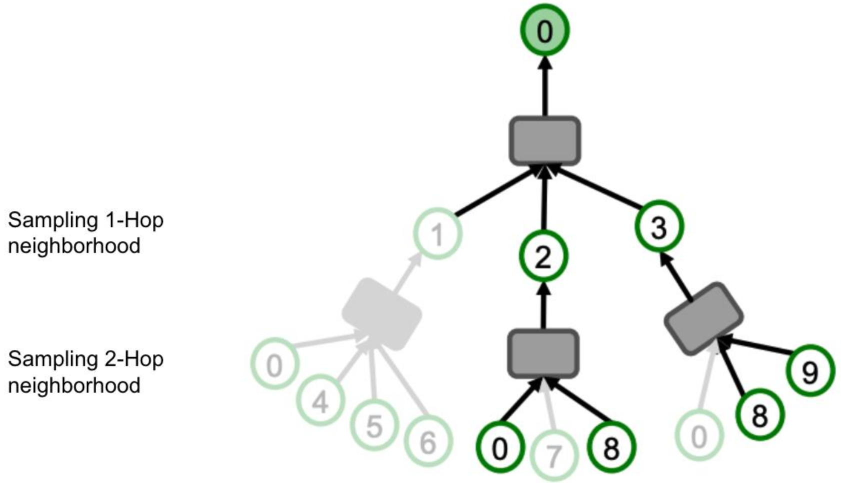
- How to design efficient sampling algorithm?
- How to guarantee the sampling quality?



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

GraphSAGE Idea

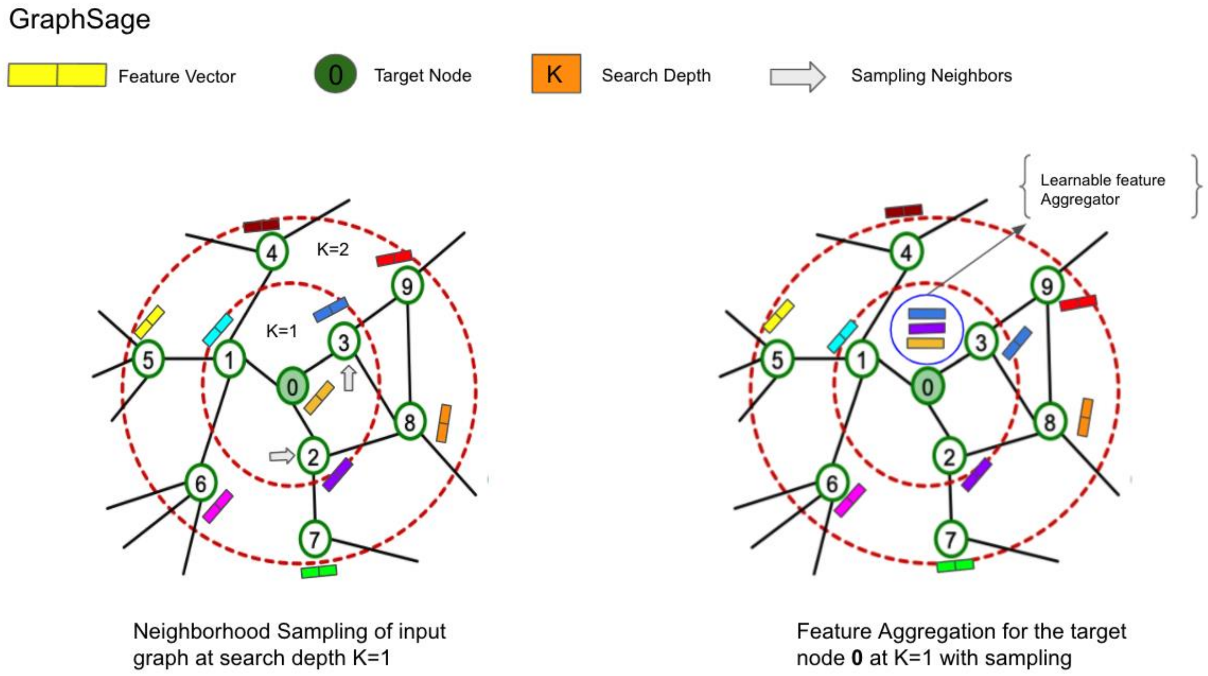
- So far, we have aggregated the neighbor messages by taking their (weighted) average, can we do better?
- **The idea:** not take the entire K-hop neighborhood of a target node but select few nodes at random from the K-hop neighborhood in order to generate computational graph.
  - This process is known as neighborhood sampling which provides the GraphSage algorithm its unique ability of scaling up to billions of nodes in the graph.



Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." *Advances in neural information processing systems* 30 (2017).

GraphSage is Inductive Representation Learning algorithm

- GraphSage is an inductive version of GCNs which implies that it does not require the whole graph structure during learning, and it can generalize well to the unseen nodes.
- We don't need to learn the embeddings for each node.
- Learning an aggregation function (MEAN, POOLING, or LSTM) which when given an information (or features) from the local neighborhood of a node then it knows how to aggregate those features (learning takes place via stochastic gradient descent)



Neighborhood Sampling of input graph at search depth K=1

Feature Aggregation for the target node **0** at K=1 with sampling

Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

GraphSAGE Idea

- GraphSAGE (SAmple and aggreGatE)
  - Instead of training individual embeddings for each node, generates embeddings by sampling features from neighborhoods

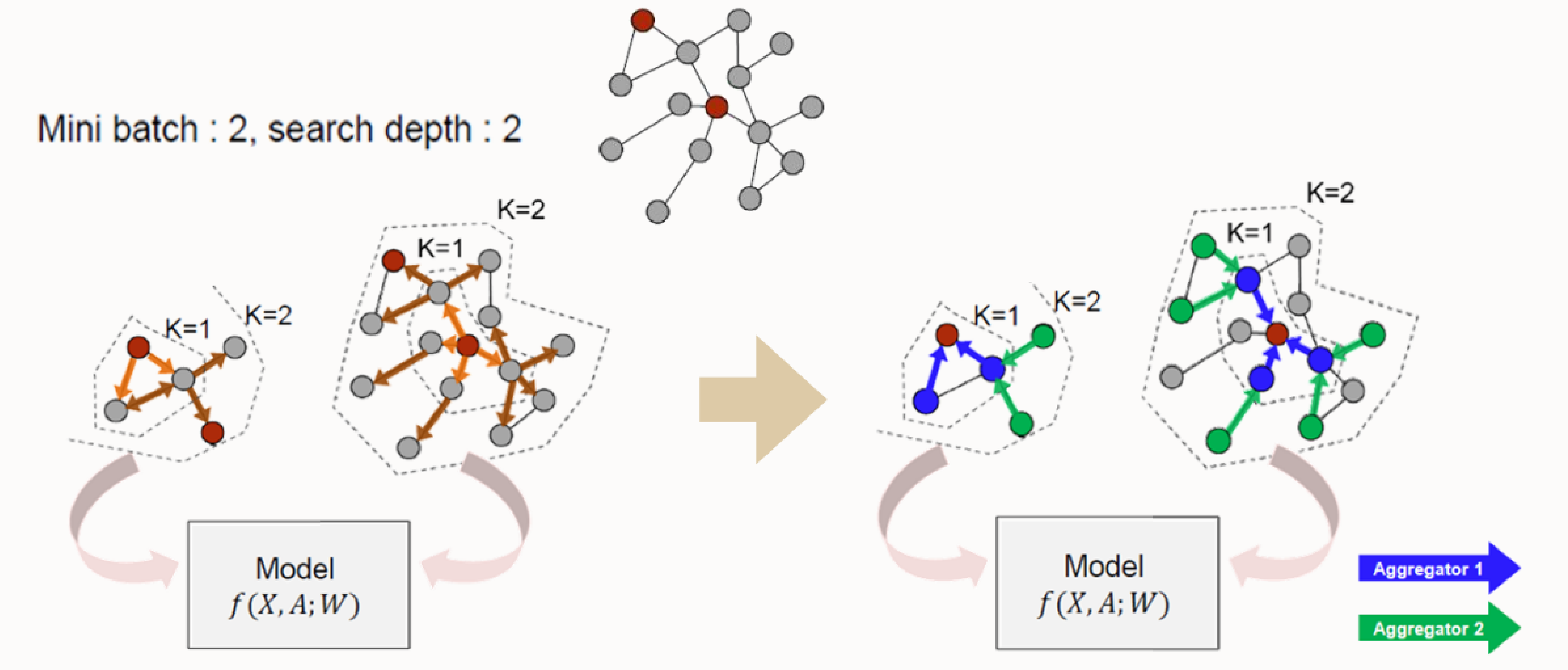
  ➔ **Mini batch**

  - Train a set of aggregator functions that learn to aggregate feature information a node's local neighborhood

  ➔ **Aggregating**

GraphSAGE: Mini batch

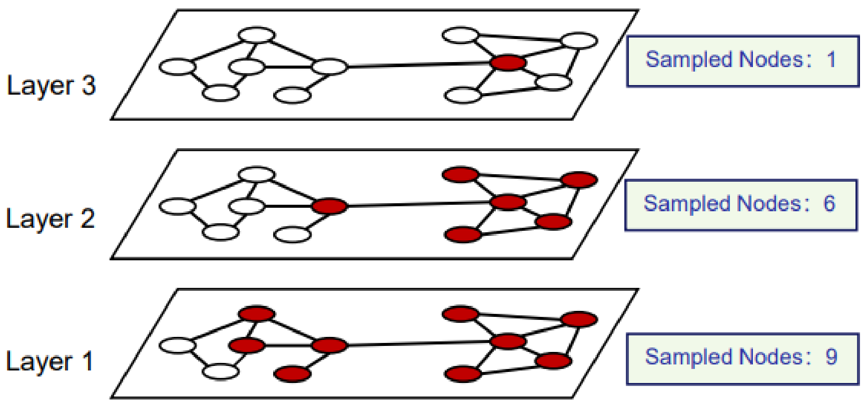
- **Mini batch:** There are three steps.
  - Sample neighbourhood
  - Aggregate feature information from neighbours
  - Predict graph context and label using aggregated information



GraphSAGE

- Towards large-scale **GraphSAGE**:
  - Sampling mini-batch (sample target nodes as a mini-batch)
  - Sampling a fixed size set for each target nodes

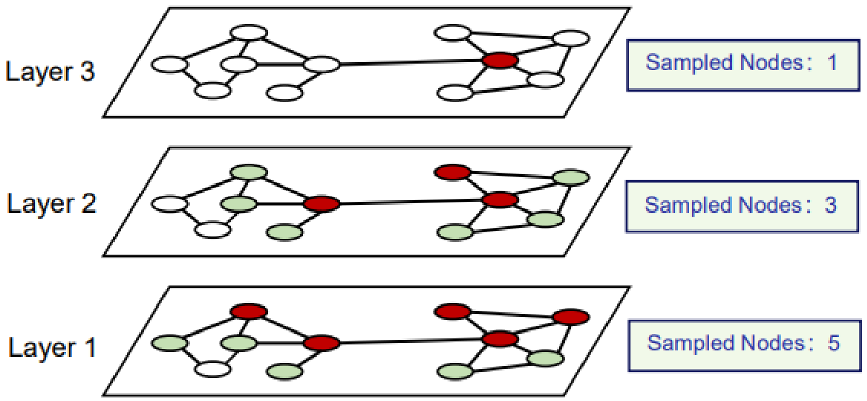
Mini-batch training



Mini-batch training, Batch Size=1

- Sample target nodes as a mini-batch;
- Only consider the nodes that used by computing the representation in the batch.

Fixed-size neighbor sampling

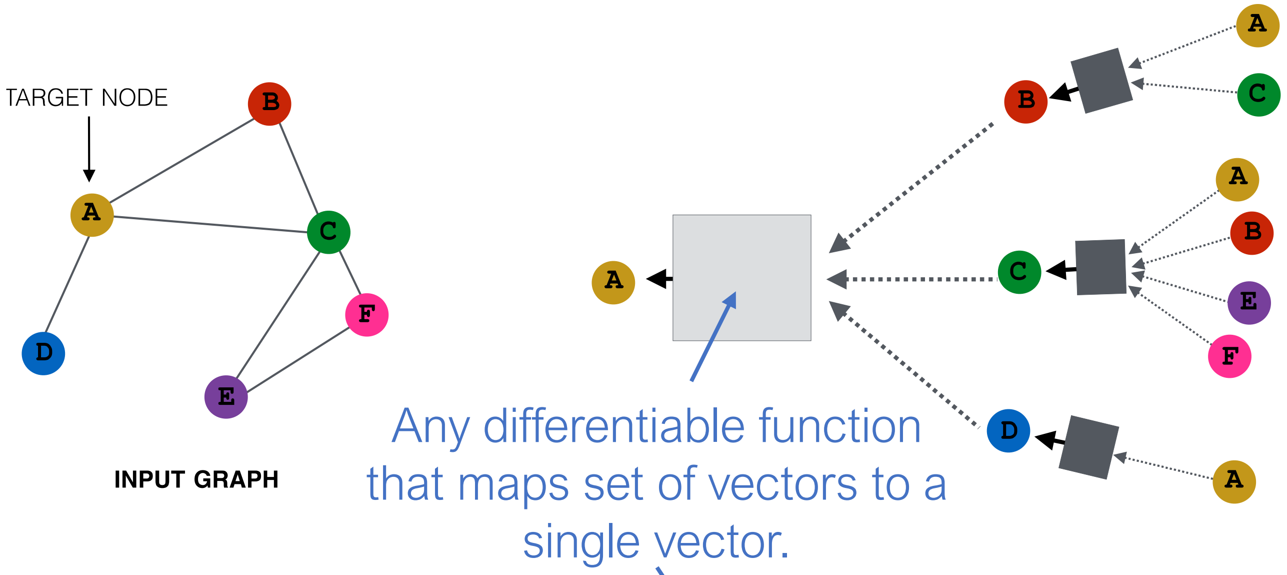


Fix-size neighbor sampling S=2

- Sample fixed-size (S_i for layer i) set of neighbors for computing.
- Number of nodes at input layer: $O(|V|^K) \rightarrow O(\prod_{i=1}^{K} S_i)$

GraphSAGE

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



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

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

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

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

generalized aggregation

Neighborhood sampling

GraphSAGE: Aggregation functions

- How to aggregate information from neighbourhood

Concatenate neighbor embedding and self 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

- Mean:

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

- Pooling: Transform neighbor vectors and apply symmetric vector function.

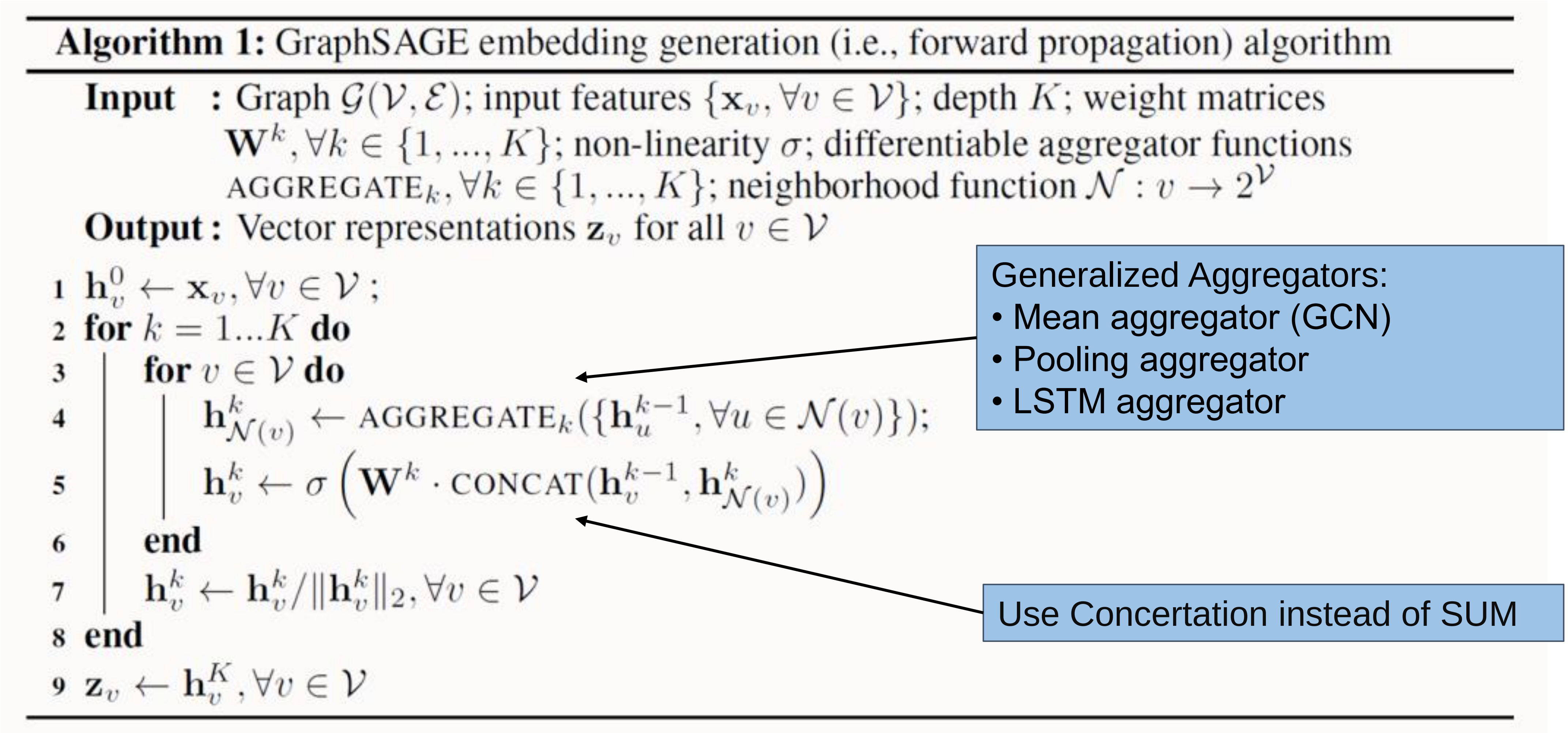
Element-wise mean/max

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

- LSTM: random permutation of neighbors

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

GraphSAGE: Algorithm



```
Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm
Input  : Graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices
         $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity $\sigma$; differentiable aggregator functions
         $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$
Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$
$\mathbf{h}^0_v \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
for $k = 1...K$ do
    for $v \in \mathcal{V}$ do
        $\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}^{k-1}_u, \forall u \in \mathcal{N}(v)\})$;
        $\mathbf{h}^k_v \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}^{k-1}_v, \mathbf{h}^k_{\mathcal{N}(v)})\right)$
    end
    $\mathbf{h}^k_v \leftarrow \mathbf{h}^k_v / \|\mathbf{h}^k_v\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}^K_v, \forall v \in \mathcal{V}$
```

Generalized Aggregators:

- Mean aggregator (GCN)
- Pooling aggregator
- LSTM aggregator

Use Concertation instead of SUM

Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." *Advances in neural information processing systems* 30 (2017).

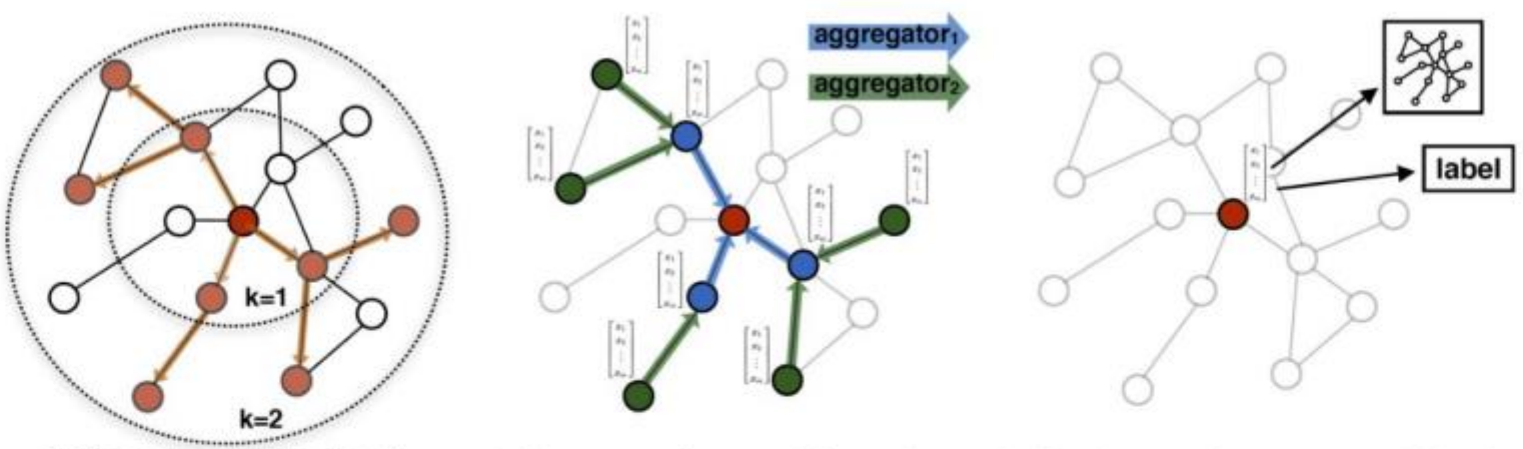
GraphSAGE: Weighting factor

- Weighting factor in GraphSAGE
  - α_{uv} (importance) is defined explicitly based on the structure properties of graph.
  - All neighbors $u \in N(v)$ are equally important to node v

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

Weighting factor

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



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Node-wise Sampling: GraphSAGE

- Pros:
  - Generalized aggregator.
  - Mini-batch training and fixed-size neighbor sampling.
- Cons:
  - Neighborhood expansion on deeper GNNs.
  - No guarantees for the sampling quality.

GraphSAGE: Sample code

- DataLoader uses **NeighborSampler** to create mini-batches.
- Each mini-batch contains a node index and local graph information about that index. The key here is to sample local graph information for each mini-batch.
- **For example:**
  - Sampling neighboring nodes in each layer. sizes=[10, 5] means that 10 and 5 neighboring nodes are sampled in each layer.
  - We also use batch_size=32 to process 32 nodes in each batch.

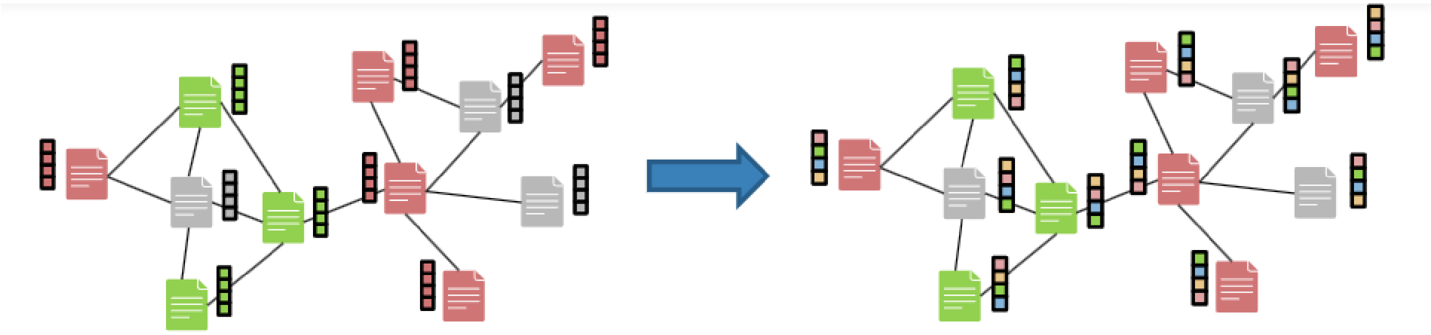
```
from torch_geometric.loader import NeighborSampler

loader = NeighborSampler(data.edge_index, sizes=[10, 5], batch_size=32,
                shuffle=True, num_nodes=data.num_nodes)
```

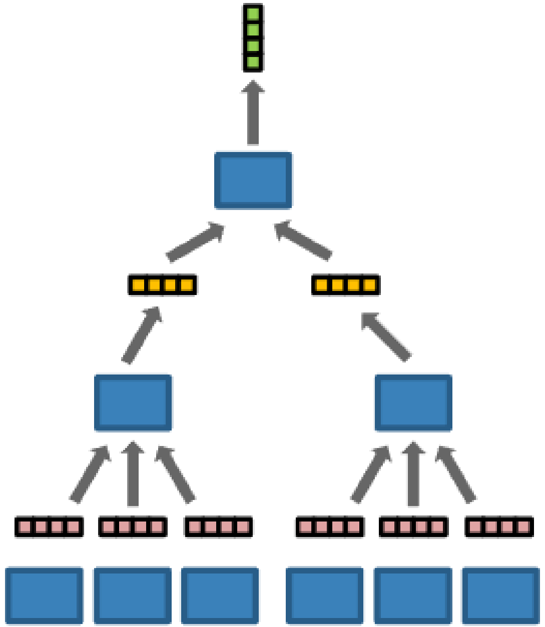
Lets do some example codes in the Sample code file

Graph-wise Sampling: ClusterGCN

- **GCN is not trivial:**
  - In standard neural networks (e.g., CNN), loss function can be decomposed as $\sum_{i=0}^{N} \boldsymbol{loss}(x_i, y_i)$
  - In GCN, loss on a node not only depends on itself but all its neighbors.
  - This dependency brings difficulties when performing **SGD on GCN.**



What we expect

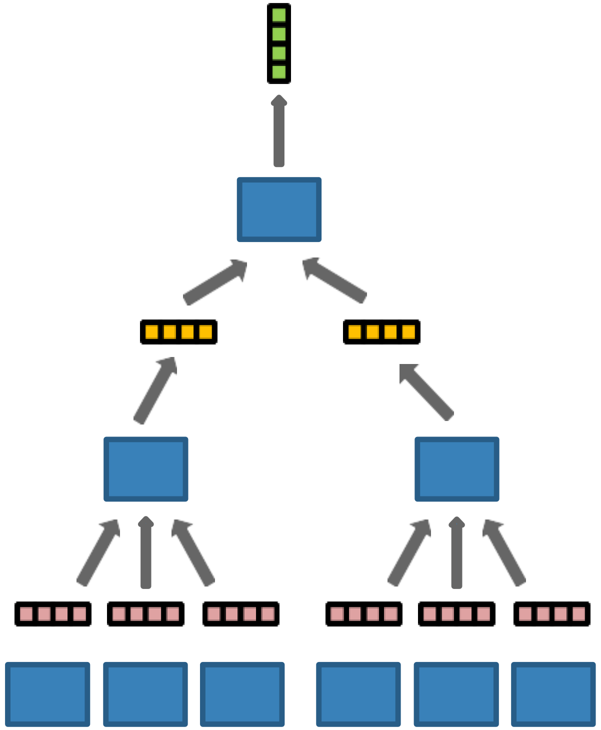


What truly happen

Chiang, Wei-Lin, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks."

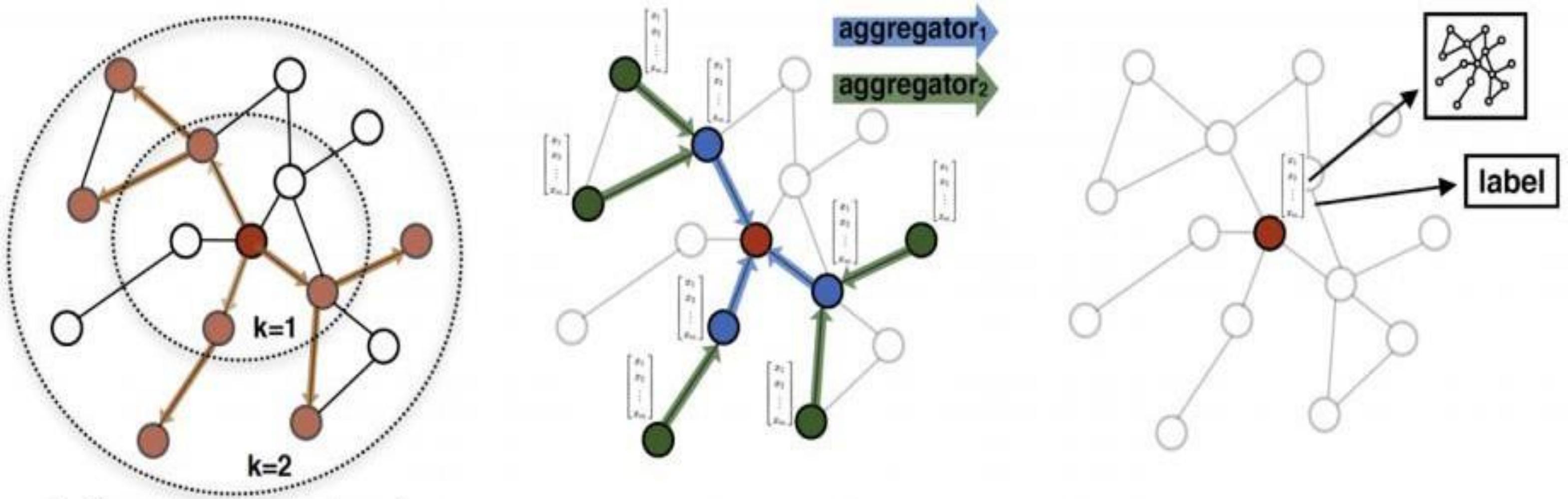
Graph-wise Sampling: ClusterGCN

- Issues come from **high computation costs.**
- Suppose we desire to calculate a target node's loss with a 2-layer GCN.
- To obtain its final representation, needs all node embeddings in its 2-hop neighborhood.
- **For example:** 9 nodes' embeddings needed but only get 1 loss (low utilization).



Graph-wise Sampling: ClusterGCN

- **Idea**: subsample a smaller number of neighbors
  - For example, GraphSAGE (NeurIPS’17) considers a subset of neighbors per node
  - But it still suffers from **recursive neighborhood expansion**.



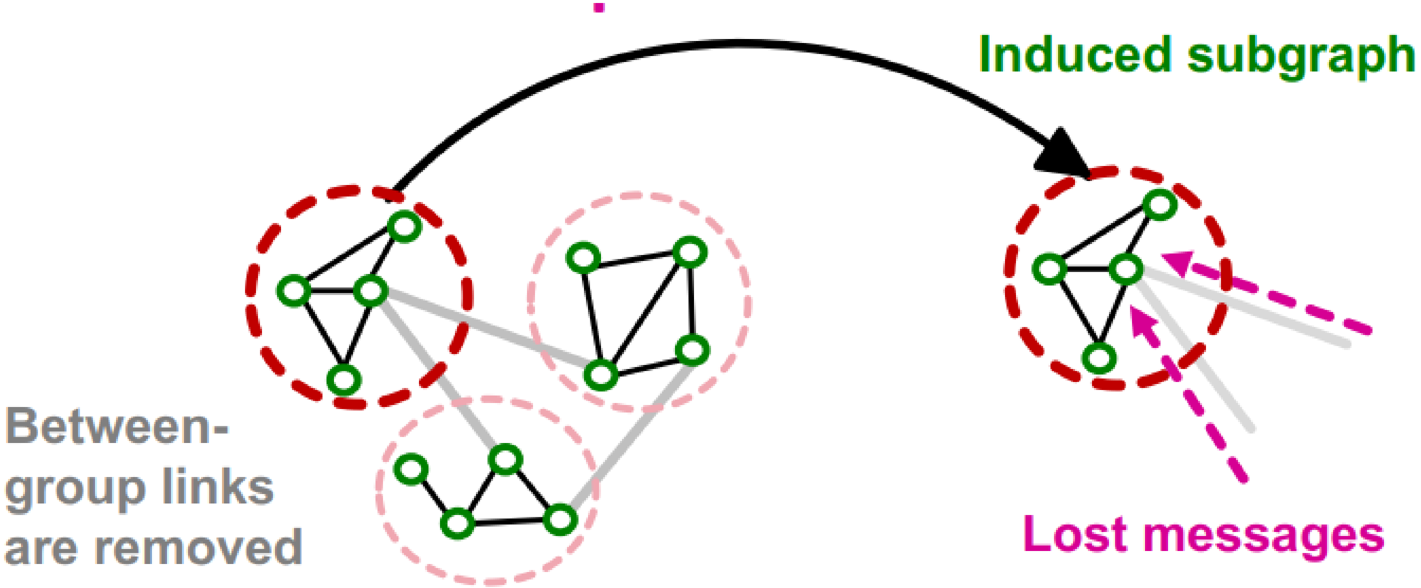
1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Graph-wise Sampling: ClusterGCN

- **Problems**:
  - The induced subgraph removes between group links.
  - As a result, messages from other groups will be lost during message passing, which could hurt the GNN’s performance.



Chiang, Wei-Lin, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks."

Graph-wise Sampling: ClusterGCN

- **Ways to improve the Embedding Utilization**:
  - If considering all losses at one time (full-batch), 9 nodes' embedding used and got 9 losses.

$$GCN_{2-layer}(A, X) = A\sigma(AXW^{(0)})W^{(1)}$$

  - Embedding Utilization: optimal.
  - The key is to re-use nodes' embeddings as many as possible
  - Focus on dense parts of the graph.

Chiang, Wei-Lin, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks."

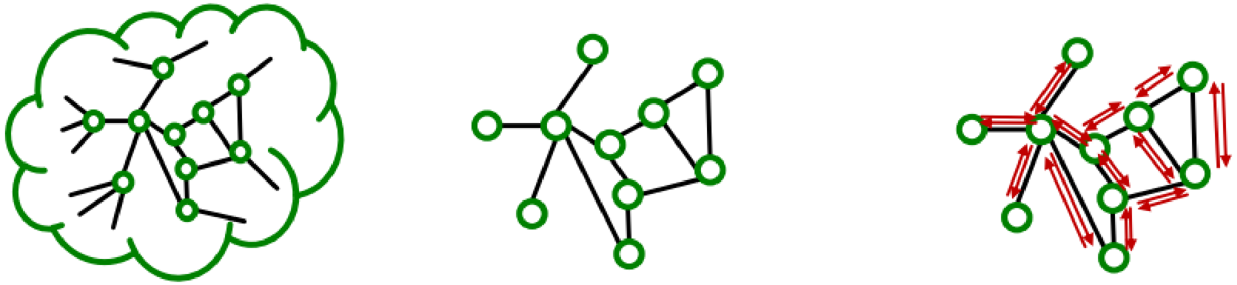
Graph-wise Sampling: ClusterGCN

- **Idea**: apply graph clustering algorithm (e.g., METIS) to identify dense subgraphs.

Large graph

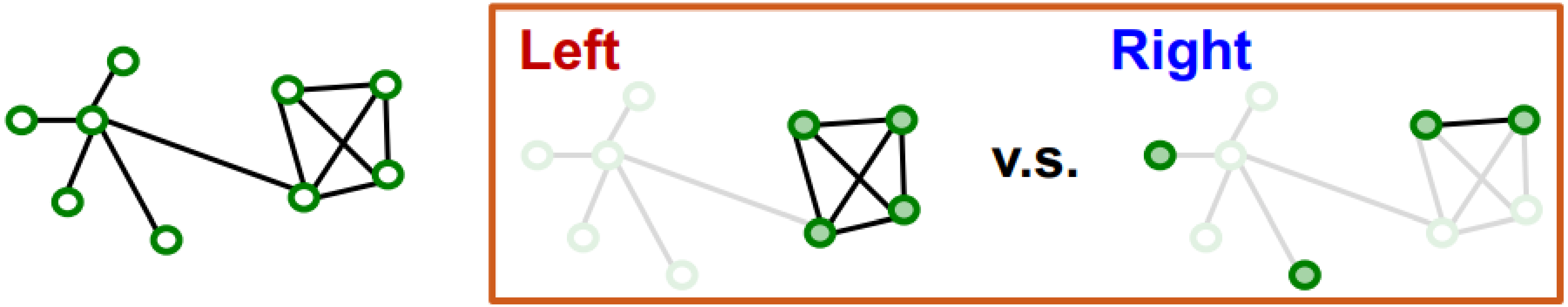
Sampled subgraph (small enough to be put on a GPU)

Layer-wise node embeddings update on the GPU



Original graph

Subgraphs (both 4-node induced subgraph)

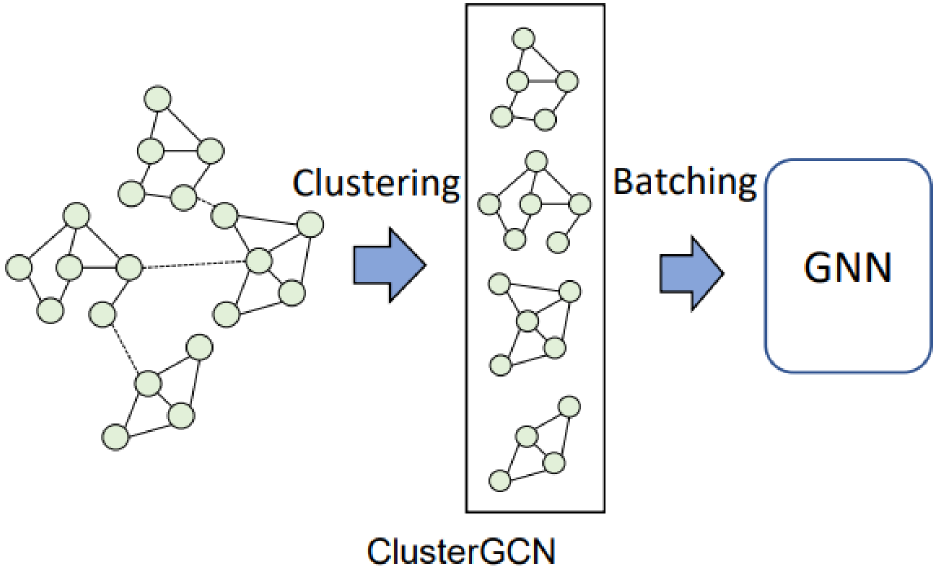


Graph-wise Sampling: ClusterGCN

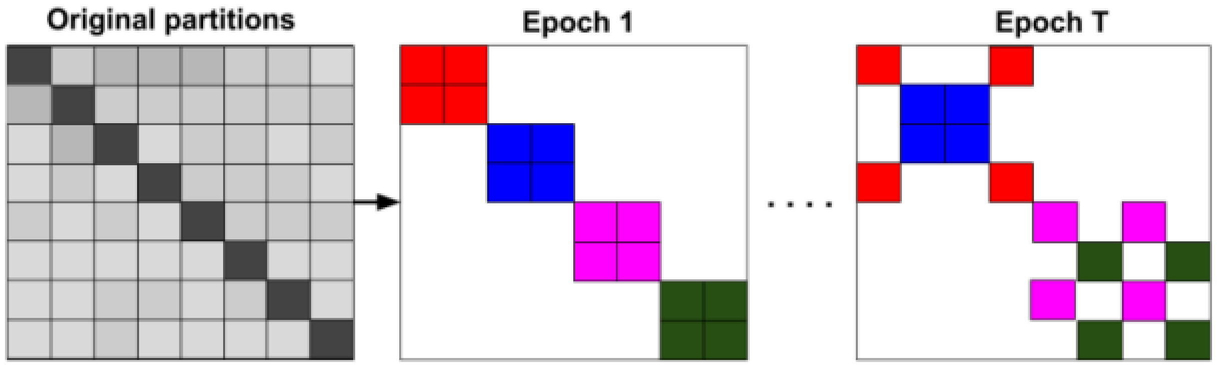
- Extract small clusters based efficient clustering algorithms.

$$\bar{G} = [G_1, \cdots, G_c] = [\{\mathcal{V}_1, \mathcal{E}_1\}, \cdots, \{\mathcal{V}_c, \mathcal{E}_c\}],$$

$$\bar{A} = \begin{bmatrix} A_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_{cc} \end{bmatrix}, \Delta = \begin{bmatrix} 0 & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & 0 \end{bmatrix},$$

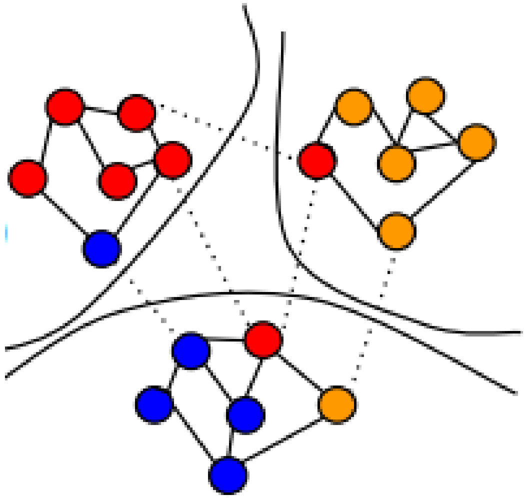


- Random batching at the subgraph level.



Graph-wise Sampling: ClusterGCN

- **Idea**: apply graph clustering algorithm (e.g., METIS) to identify dense subgraphs.
- **Cluster-GCN**
  - Partition the graph into several clusters, remove between-cluster edges
  - Each subgraph is used as a mini-batch in SGD
  - Embedding utilization is optimal because nodes' neighbors stay within the cluster

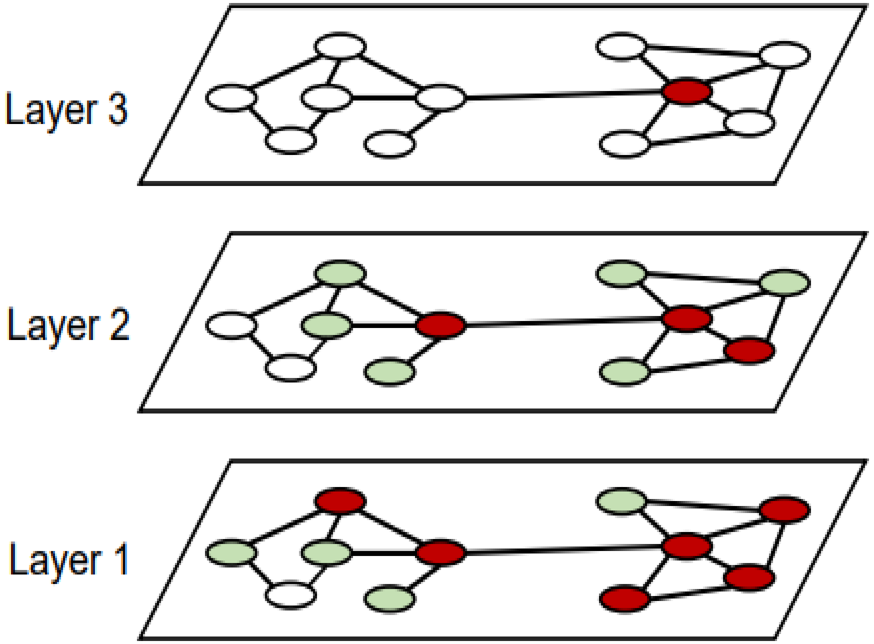


Chiang, Wei-Lin, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks."

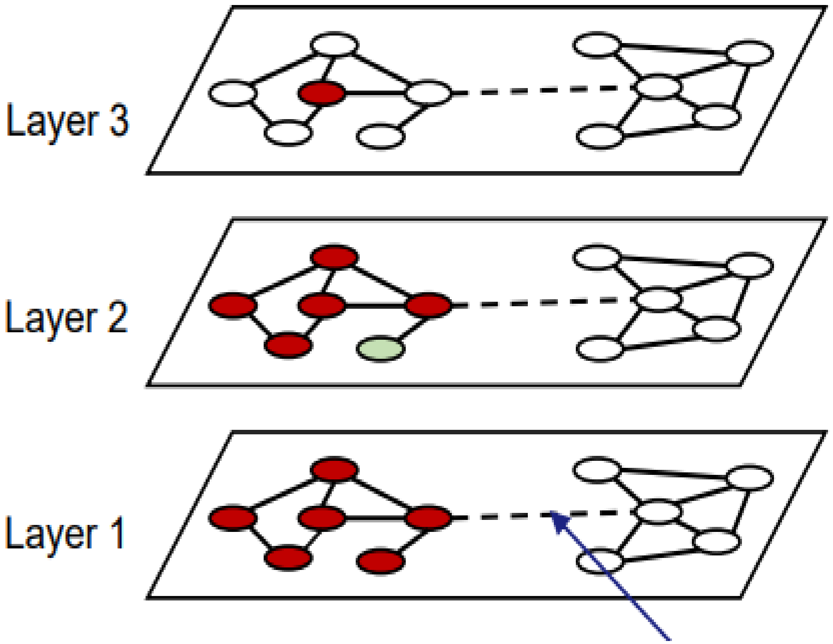
Graph-wise Sampling: ClusterGCN

- Neighbor expansion control:

Only consider the nodes in the same clusters



Fix-size neighbor sampling S=2



Only sample the nodes in the clusters

Graph-wise Sampling: ClusterGCN

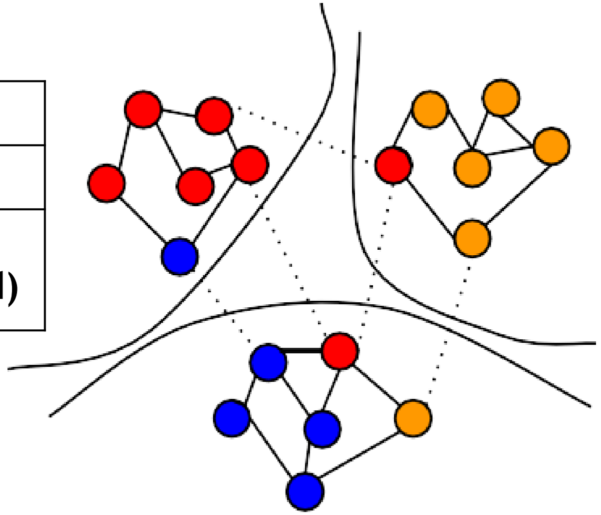
- **Pros:**
  - Good performance / Good memory usage.
  - Alleviate the neighborhood expansion problem in traditional mini-batch training.
- **Cons:**
  - Empirical results without analyzing the sampling quality.

Chiang, Wei-Lin, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks."

Graph-wise Sampling: ClusterGCN

- Partition the graph into several clusters, remove between-cluster edges.
- Each subgraph is used as a mini-batch in SGD.
- Embedding utilization is optimal because nodes' neighbors stay within the cluster.
- Even though 20% edges are removed, the accuracy of GCN model remains similar.

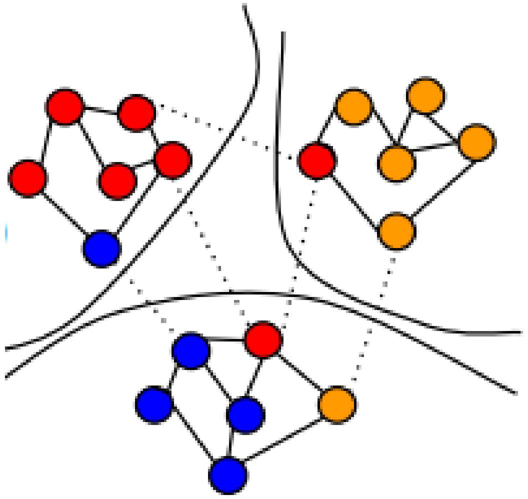
CiteSeer	Random partitioning	Graph partitioning
1 (no partitioning)	72.0	72.0
100 partitions	46.1	71.5 (~20% edges removed)



Chiang, Wei-Lin, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks."

Graph-wise Sampling: ClusterGCN

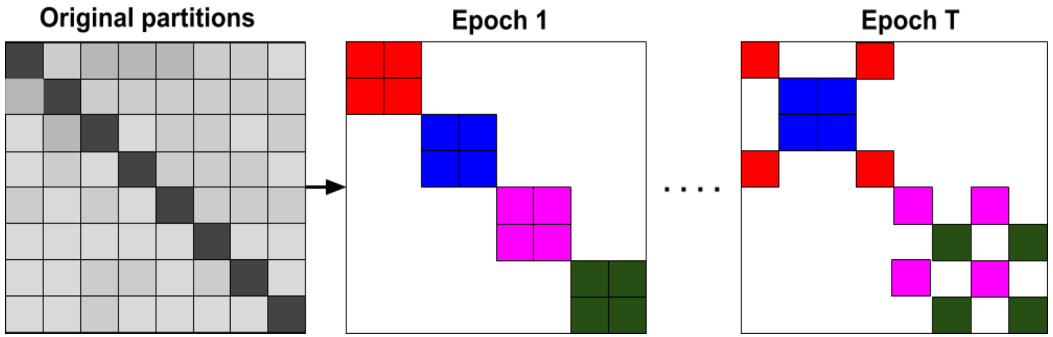
- **Issues**: imbalanced label distribution
  - nodes with similar labels are clustered together.
  - Hence the label distribution within a cluster could bedifferent from the original data.
  - Leading to a biased SGD.

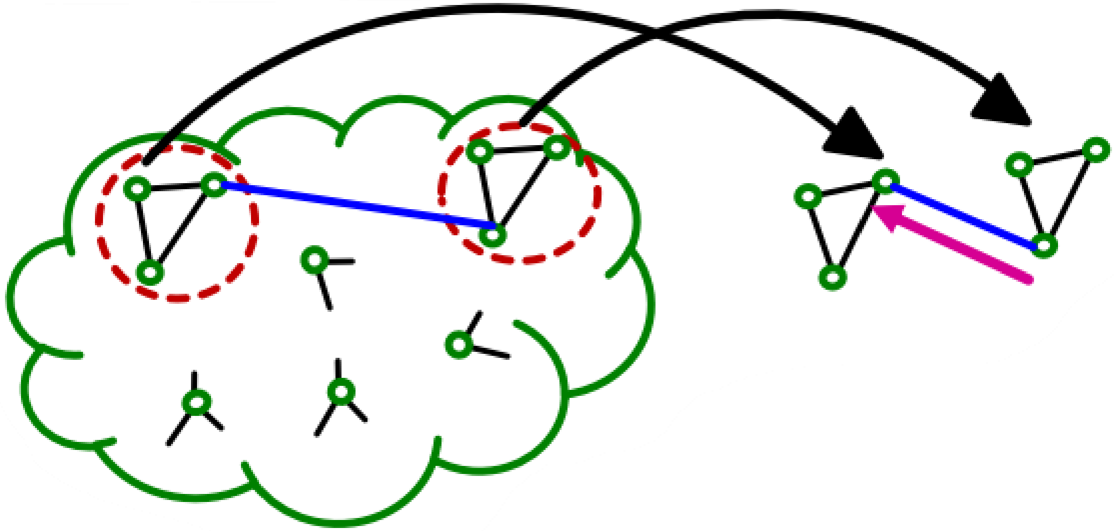


Chiang, Wei-Lin, et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks."

Graph-wise Sampling: ClusterGCN

- **Multiple Clusters:**
  - A randomly select multiple clusters as a batch has been proposed.
- **Two advantages:**
  - Balance label distribution within a batch
  - Recover some missing edges between-cluster

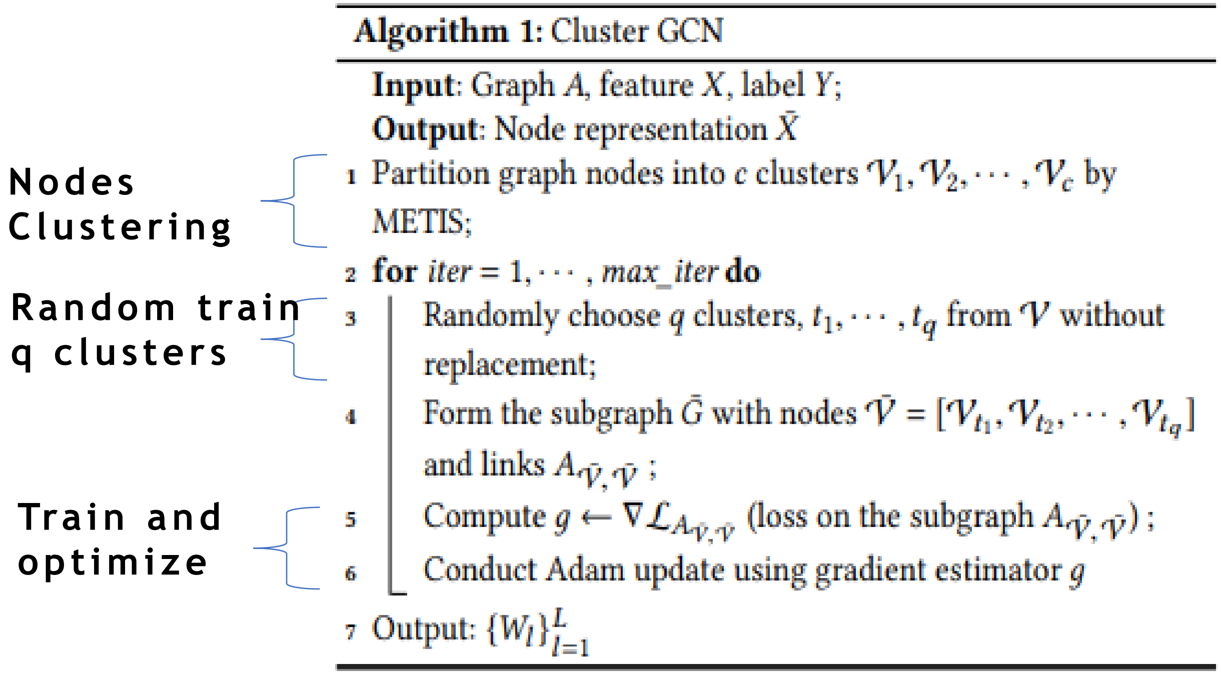




ClusterGCN: algorithm

Mini-batch training:

- For each mini-batch, **randomly sample a set of q node groups:** $\{V_{t_1}, \dots, V_{t_q}\} \subset \{V_1, \dots, V_C\}$.
- **Aggregate all nodes across the sampled node groups**: $V_{aggr} = V_{t_1} \cup \cdots \cup V_{t_q}$
- Extract the **induced subgraph** $G_{aggr} = (V_{aggr}, E_{aggr})$, where $E_{aggr} = \{(u, v) \mid u, v \in V_{aggr}\}$
  - E_{aggr} **also includes between-group edges!**



Nodes Clustering

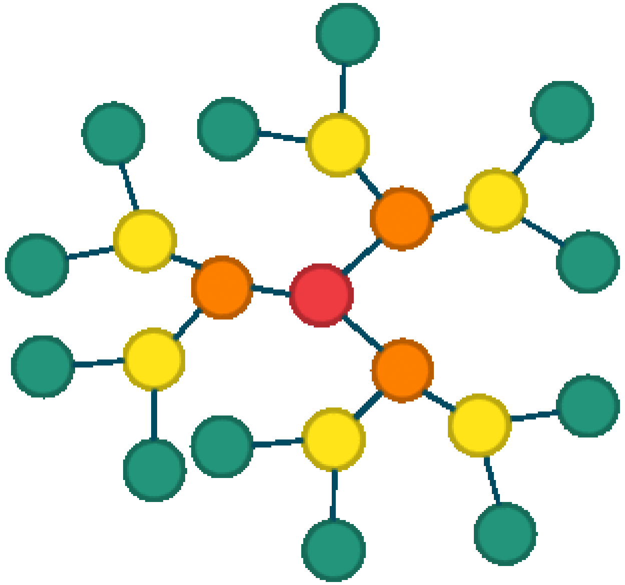
Random train q clusters

Train and optimize

```
Algorithm 1: Cluster GCN
Input: Graph $A$, feature $X$, label $Y$;
Output: Node representation $\bar{X}$
Partition graph nodes into $c$ clusters $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_c$ by METIS;
for $iter = 1, \cdots, max\_iter$ do
    Randomly choose $q$ clusters, $t_1, \cdots, t_q$ from $\mathcal{V}$ without replacement;
    Form the subgraph $\bar{G}$ with nodes $\bar{\mathcal{V}} = [\mathcal{V}_{t_1}, \mathcal{V}_{t_2}, \cdots, \mathcal{V}_{t_q}]$ and links $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$;
    Compute $g \leftarrow \nabla \mathcal{L}_{A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}}$ (loss on the subgraph $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$);
    Conduct Adam update using gradient estimator $g$
Output: $\{W_l\}_{l=1}^{L}$
```

GraphSAINT: Problem Statements

- "Neighbour explosion", GCNs rely on aggregating neighbor information to update nodes.
- It is very intuitive that the more layers of GCNs, the more neighbors need to be considered when updating nodes; while the number of layers is fixed
- The higher the average degree, the more neighbors need to be considered.

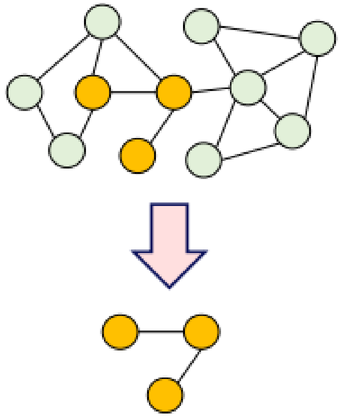


Zeng, Hanqing, et al. "GraphSAINT: Graph Sampling Based Inductive Learning Method."

Graph-wise Sampling: GraphSAINT

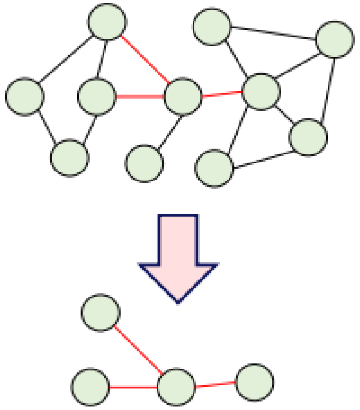
- Directly sample a subgraph for mini-batch training according to subgraph sampler.
- Sampler construction

Node sampler



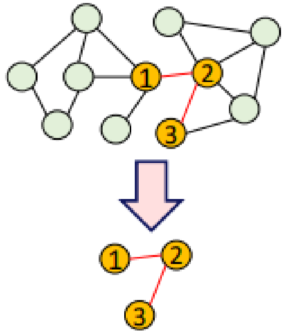
Uniformly sample nodes.

Edge sampler



Sample edge with probably $p_{u,v} \propto 1/d_u + 1/d_v$

Random walk sampler

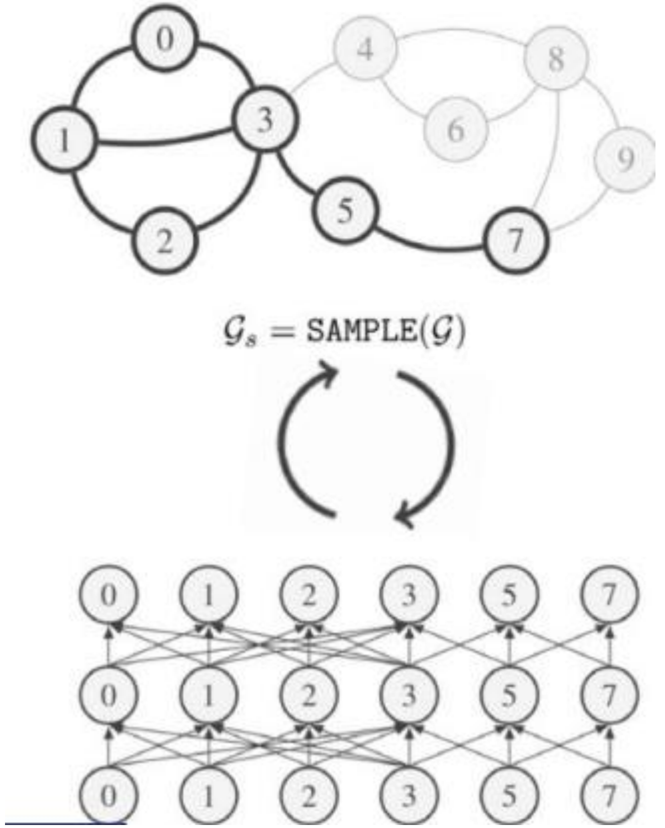


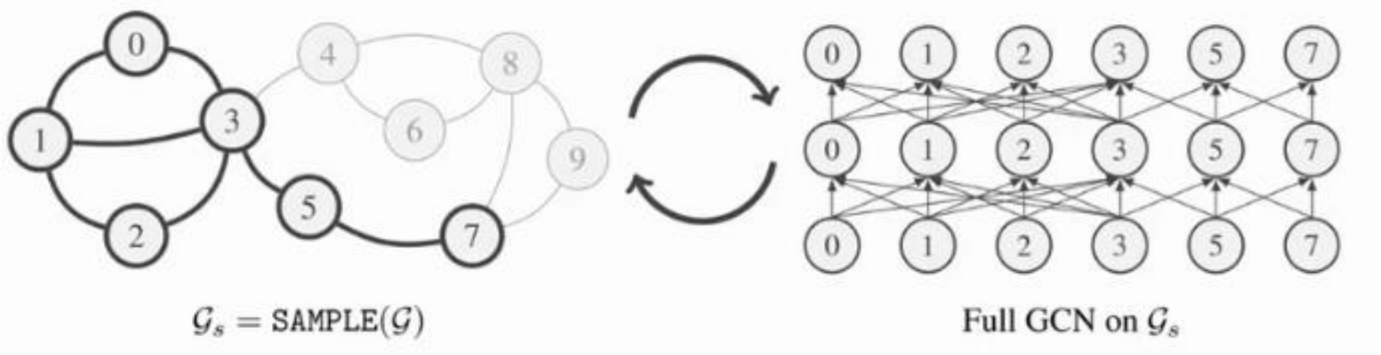
Sample edge with probably $p_{u,v} \propto \boldsymbol{B}_{u,v} + \boldsymbol{B}_{v,u}$

- $B_{u,v}$: the probability of a random walk to start at u and end at v in L hops.

Graph-wise Sampling: GraphSAINT

- ➢ Sample a small subgraph, then build a complete GNN
- ➢ Constant neighborhood size
- ➢ Not an I.I.D data sampler
- ➢ **Why?**
  - ➢ Popular users will be sampled more frequently, i.e., influencers, …
- ➢ **How?**
  - ➢ Normalize aggregration and mini batch loss by edge/node sampling frequency.




0
1
2
3
4
5
6
7
8
9
0 1 2 3 5 7
0 1 2 3 5 7
0 1 2 3 5 7
$\mathcal{G}_s = \text{SAMPLE}(\mathcal{G})$
Full GCN on $\mathcal{G}_s$

Graph-wise Sampling: GraphSAINT

- How to eliminate the **bias** introduce by the sampler?
  - Normalize minibatch loss by probability of sampling each node:

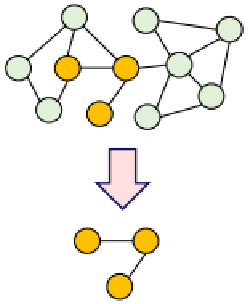
$$\mathcal{L}_{\text{batch}} = \sum_{v \in G_s} L_v / \lambda_v \, , \lambda_v = |V| p_v .$$

p_v: the probability of a node $v \in V$ being sampled.
p_{uv}: the probability of an edge $u, v \in E$ being sampled.

  - Normalize neighbor aggregation by probability of sampling each edge :

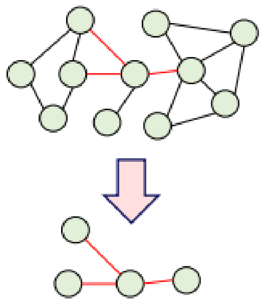
$$a(u, v) = p_{u,v} / p_v$$

Node sampler



Uniformly sample nodes.

Edge sampler

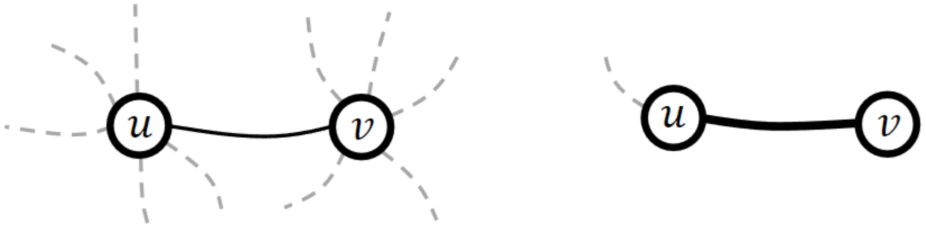


Sample edge with probably $p_{u,v} \propto 1/d_u + 1/d_v$

Graph-wise Sampling: GraphSAINT

- Another issue: Need to preserve connectivity → **Variance reduction**.
- How?
  - Important” neighbors to be sampled more frequently.
  - Independent edge sampling: optimal edge probability for variance minimization

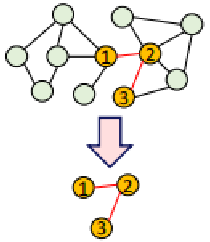
$$p_{u,v} \propto \frac{1}{d_u} + \frac{1}{d_v}$$



Sample “━━” more
Sample “——” less

- Extension
  - Multi-layer version of random edge sampling → Random walk sampler.
  - Variations of random walk sampler → Multi-dimensional RW, etc.

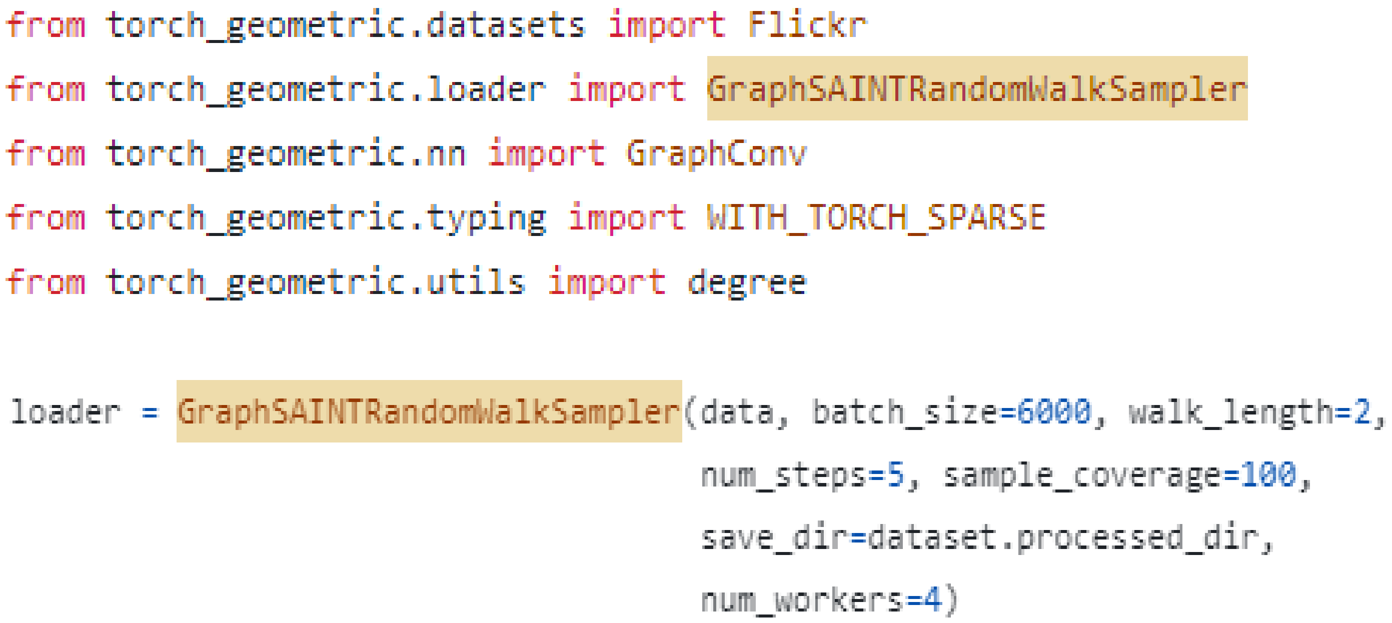
Random walk sampler



Sample edge with probably $p_{u,v} \propto \boldsymbol{B}_{u,v} + \boldsymbol{B}_{v,u}$

- $B_{u,v}$: the probability of a random walk to start at u and end at v in L hops.

GraphSAINT: Sample code



```
from torch_geometric.datasets import Flickr
from torch_geometric.loader import GraphSAINTRandomWalkSampler
from torch_geometric.nn import GraphConv
from torch_geometric.typing import WITH_TORCH_SPARSE
from torch_geometric.utils import degree

loader = GraphSAINTRandomWalkSampler(data, batch_size=6000, walk_length=2,
                                     num_steps=5, sample_coverage=100,
                                     save_dir=dataset.processed_dir,
                                     num_workers=4)
```

Given a graph, this class samples nodes and constructs subgraphs that can be processed in a mini – batch fashion.

Q A