Seminar Series on Graph Neural Networks 06

Towards efficient graph learning

Yong-Min Shin
School of Mathematics and Computing (Computational Science and Engineering)
Yonsei University
2025.05.19

Before going in....

Towards application of graph neural networks

Towards efficient graph learning

Explainable graph neural networks

Fundamental topics on graph neural networks

On the representational power of graph neural networks

A graph signal processing viewpoint of graph neural networks

From label propagation to graph neural networks

On the problem of oversmoothing and oversquashing

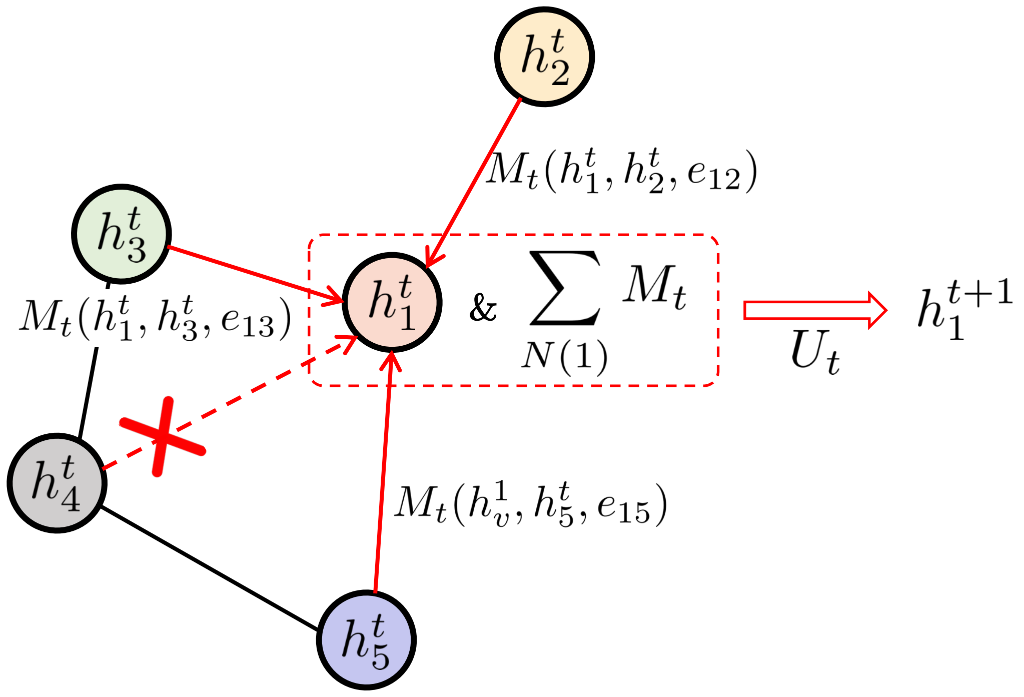
Introduction to graph mining and graph neural networks
(Basic overview to kick things off)

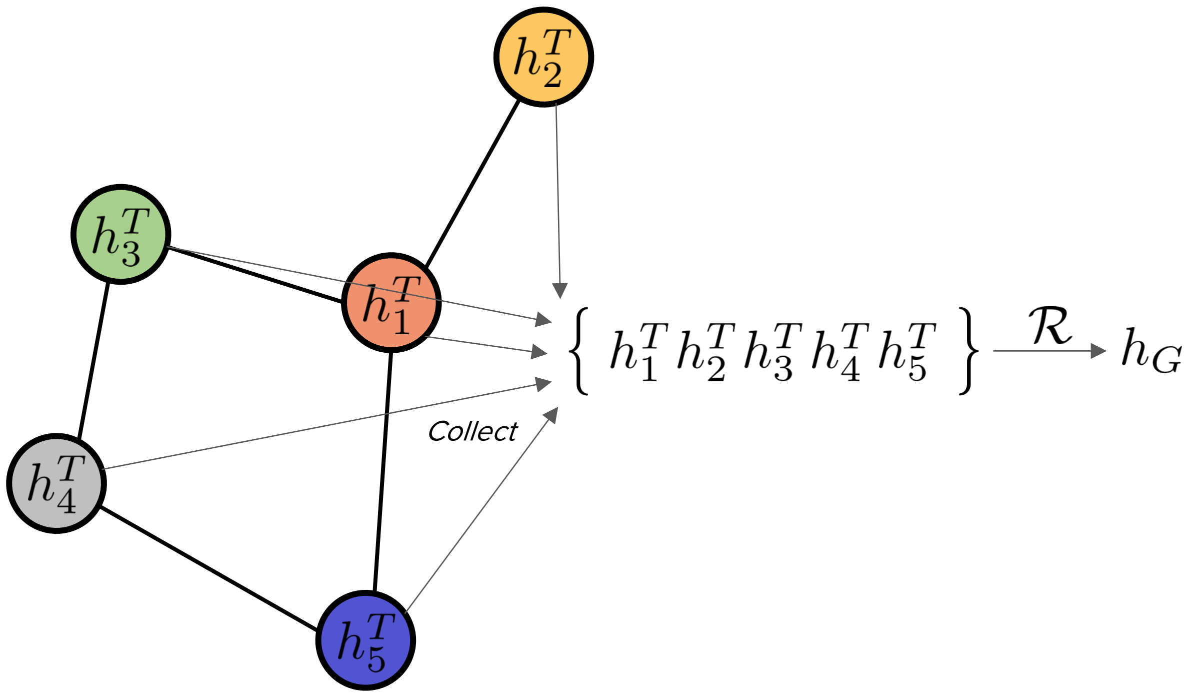
* Presentation slides are available at:
(jordan7186.github.io/presentations/)

Objectives

1. Understanding the **practical limitations** of GNNs in terms of **efficiency**
2. Overview of related field 1: **Simple** GNNs
3. Overview of related field 2: **GNN-to-MLP knowledge distillation**
    - Also check: (Shin et al., Propagate & Distill: Towards effective graph learners using propagation-embracing MLPs, LoG 2023), which is on the same subject ☺

Recap: Message-passing in graph neural networks


h_2^t
$M_t(h_1^t, h_2^t, e_{12})$
h_3^t
$M_t(h_1^t, h_3^t, e_{13})$
h_1^t
$\& \sum_{N(1)} M_t$
U_t
h_1^{t+1}
h_4^t
$M_t(h_v^1, h_5^t, e_{15})$
h_5^t


h_2^T
h_3^T
h_1^T
h_4^T
h_5^T
Collect
$\{ h_1^T h_2^T h_3^T h_4^T h_5^T \} \xrightarrow{\mathcal{R}} h_G$

Definition of message passing (Gilmer)

1. Message passing phase (Aggregation)

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

2. Update phase (Transformation)

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

3. Readout phase

$$h_G = \mathcal{R}(h_1^T, \cdots, h_{\mathcal{V}}^T)$$

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

Recap: Message-passing in graph neural networks

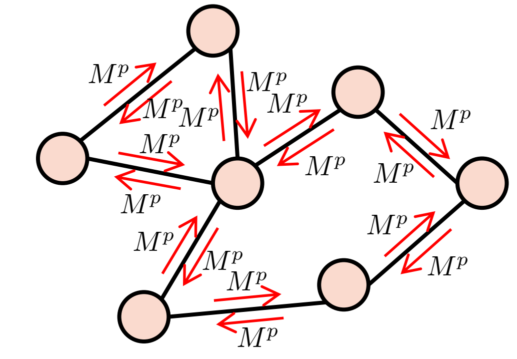
Global view

$$O(L(V + E))$$

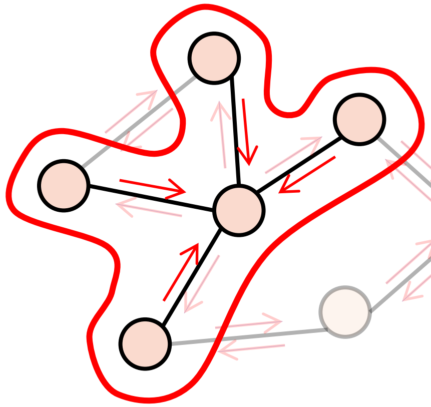
Node view

$$O(r^L)$$

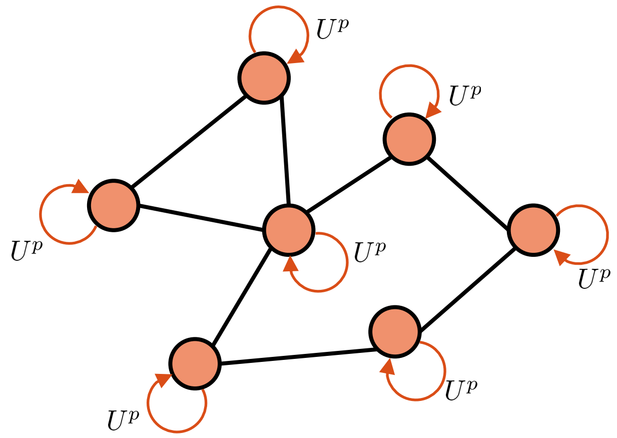
Step 1



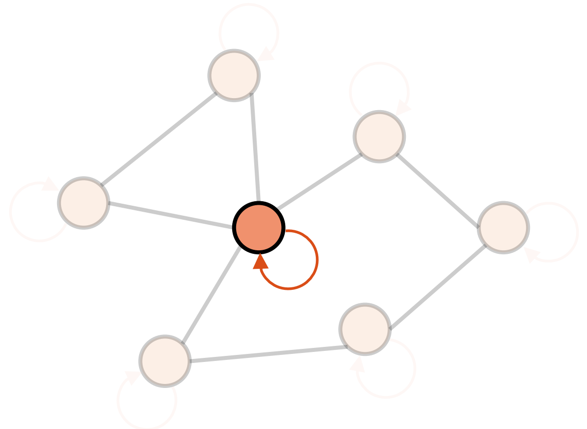
Message function



Step 2



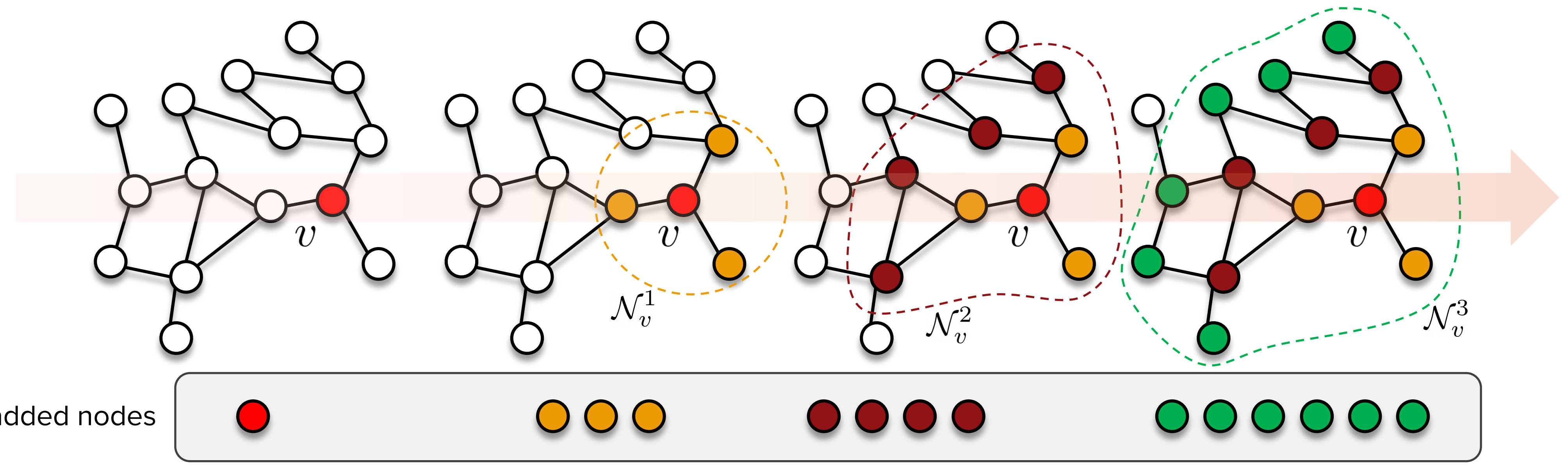
Update function



Update function

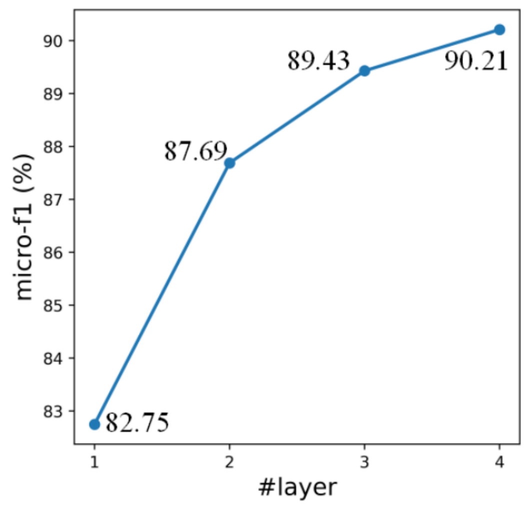
Understanding the practical limitations of GNNs in terms of efficiency

Rethinking GNN time complexity w.r.t. number of layers

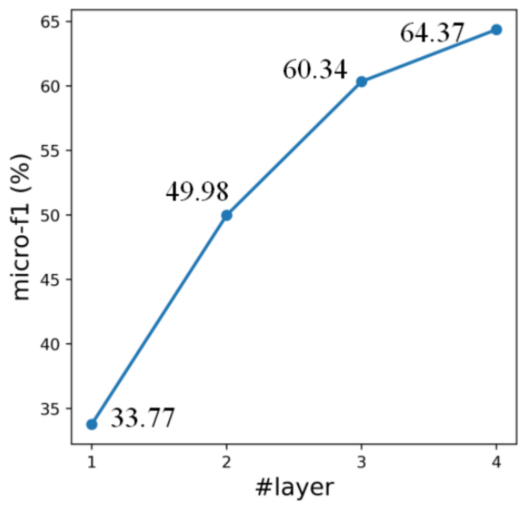


The number of nodes included in the computation (**receptive field**) tends to increase exponentially w.r.t. number of GNN layers.
Intuitive understanding: Assume each node has an average of *d* neighboring nodes, then the total number of nodes will be *d^L* for L GNN layers.

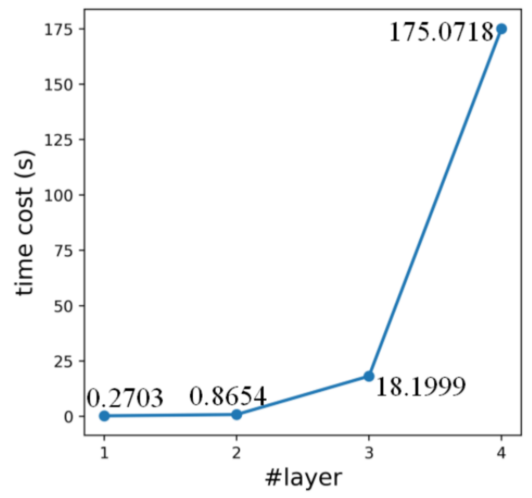
Rethinking GNN time complexity w.r.t. number of layers



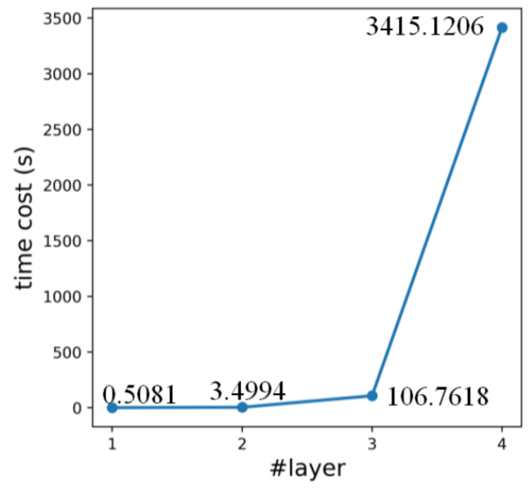
(a) Micro-f1 of GAT on Facebook



(b) Micro-f1 of GAT on AliGraph



(c) Time cost of GAT on Facebook



(d) Time cost of GAT on AliGraph

- (a, b) As the number of layers increase, the number of nodes to consider per nodes increases exponentially.
- (c, d) Although the performance gains are there, the **feed-forward (inference) time also increases exponentially** (here, we are interested in mini-batch or single batch inference)

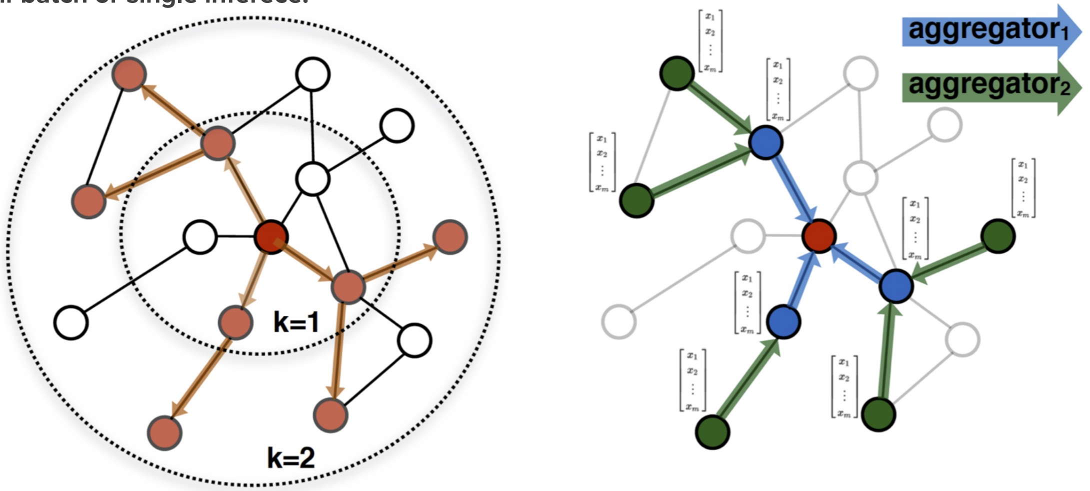
Yan et al., TinyGNN: Learning Efficient Graph Neural Networks, KDD 2020

Rethinking GNN time complexity w.r.t. number of layers

Table 1: Time and space complexity of GCN training algorithms. L **is number of layers,** N **is number of nodes,** $\|A\|_0$ **is number of nonzeros in the adjacency matrix, and** F **is number of features. For simplicity we assume number of features is fixed for all layers. For SGD-based approaches,** b **is the batch size and** r **is the number of sampled neighbors per node. Note that due to the variance reduction technique, VR-GCN can work with a smaller** r **than GraphSAGE and FastGCN. For memory complexity,** LF^2 **is for storing** $\{W^{(l)}\}_{l=1}^{L}$ **and the other term is for storing embeddings. For simplicity we omit the memory for storing the graph (GCN) or sub-graphs (other approaches) since they are fixed and usually not the main bottleneck.**

	GCN [9]	Vanilla SGD	GraphSAGE [5]	FastGCN [1]	VR-GCN [2]	Cluster-GCN
Time complexity	$O(L\|A\|_0F + LNF^2)$	$O(d^LNF^2)$	$O(r^LNF^2)$	$O(rLNF^2)$	$O(L\|A\|_0F + LNF^2 + r^LNF^2)$	$O(L\|A\|_0F + LNF^2)$
Memory complexity	$O(LNF + LF^2)$	$O(bd^LF + LF^2)$	$O(br^LF + LF^2)$	$O(brLF + LF^2)$	$O(LNF + LF^2)$	$O(bLF + LF^2)$

*As we have mentioned earlier, the complexity differs when we are talking about (1) full-batch inference and (2) mini-batch or single inferece.



1. Sample neighborhood

2. Aggregate feature information from neighbors

Let's limit the number of neighboring nodes to aggregate per node: $O(r^L) \to O(d^L),\ d < r$

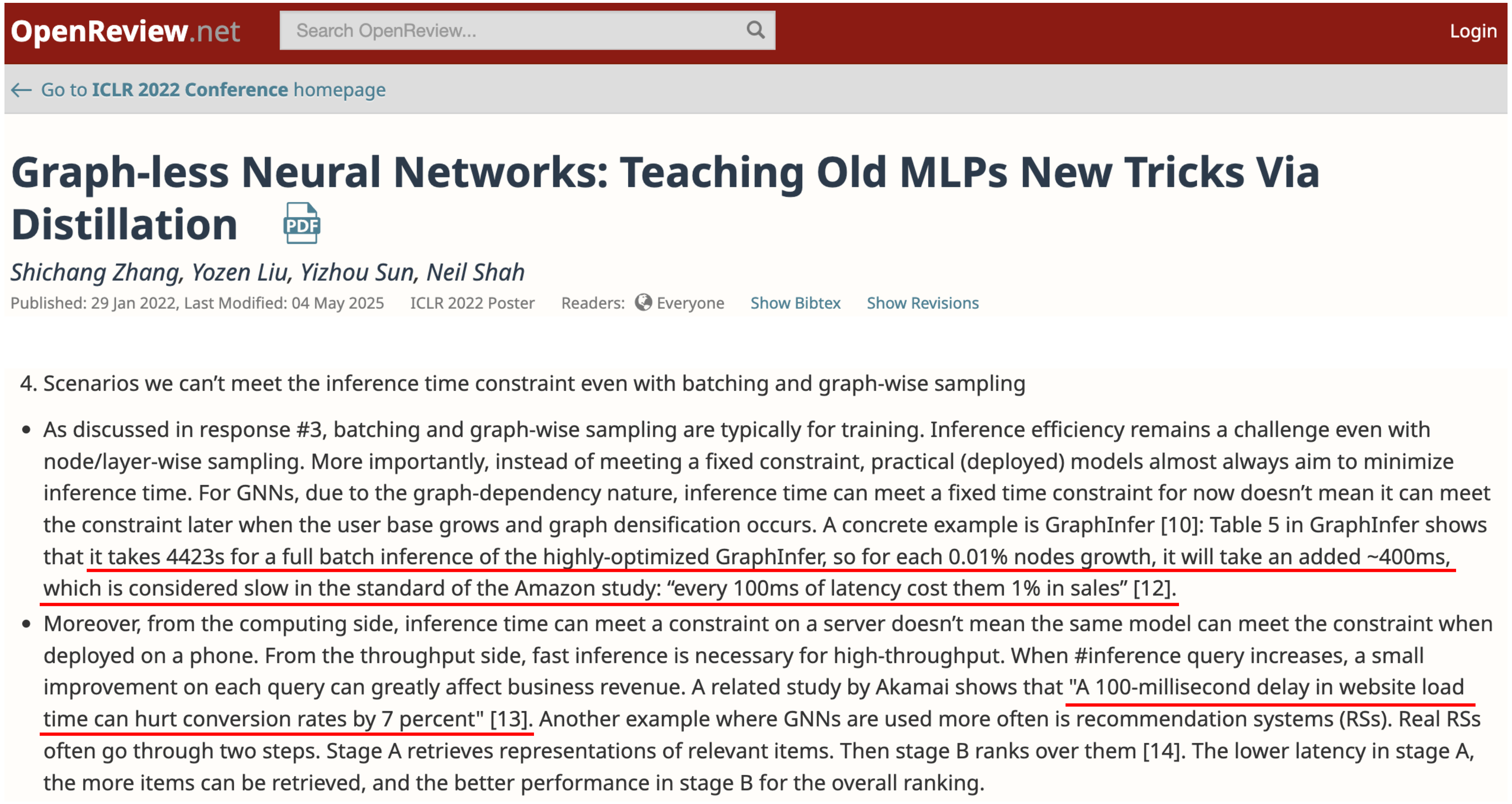
(Top) Chiang et al., Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks, KDD 2019
(Bottom) Hamilton et al., Inductive Representation Learning on Large Graphs, NeurIPS 2017

Why does this matter?

Limited large-scale applications in real world: Despite vast research in academia,
GNNs are rarely used in the industry (expecially in terms of web-service or recommendation systems)

Why does this matter?

Limited large-scale applications in real world: Despite vast research in academia,
GNNs are rarely used in the industry (expecially in terms of web-service or recommendation systems)



Graph-less Neural Networks: Teaching Old MLPs New Tricks Via Distillation

Shichang Zhang, Yozen Liu, Yizhou Sun, Neil Shah

Published: 29 Jan 2022, Last Modified: 04 May 2025 ICLR 2022 Poster Readers: Everyone Show Bibtex Show Revisions

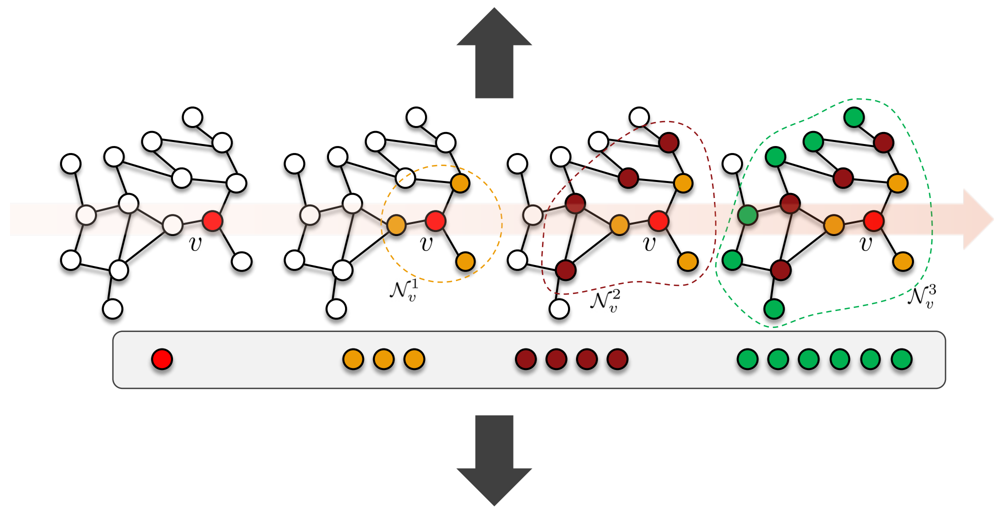
4. Scenarios we can't meet the inference time constraint even with batching and graph-wise sampling

- As discussed in response #3, batching and graph-wise sampling are typically for training. Inference efficiency remains a challenge even with node/layer-wise sampling. More importantly, instead of meeting a fixed constraint, practical (deployed) models almost always aim to minimize inference time. For GNNs, due to the graph-dependency nature, inference time can meet a fixed time constraint for now doesn't mean it can meet the constraint later when the user base grows and graph densification occurs. A concrete example is GraphInfer [10]: Table 5 in GraphInfer shows that it takes 4423s for a full batch inference of the highly-optimized GraphInfer, so for each 0.01% nodes growth, it will take an added ~400ms, which is considered slow in the standard of the Amazon study: "every 100ms of latency cost them 1% in sales" [12].
- Moreover, from the computing side, inference time can meet a constraint on a server doesn't mean the same model can meet the constraint when deployed on a phone. From the throughput side, fast inference is necessary for high-throughput. When #inference query increases, a small improvement on each query can greatly affect business revenue. A related study by Akamai shows that "A 100-millisecond delay in website load time can hurt conversion rates by 7 percent" [13]. Another example where GNNs are used more often is recommendation systems (RSs). Real RSs often go through two steps. Stage A retrieves representations of relevant items. Then stage B ranks over them [14]. The lower latency in stage A, the more items can be retrieved, and the better performance in stage B for the overall ranking.

https://openreview.net/forum?id=4p6_5HBWPCw

Two major approaches to solve this problem

Simpler GNNs: Making GNN architectures more simple



GNN-to-MLP KD: Training a very good MLP model via knowledge distillation from a GNN teacher

***We will also briefly do an overview of <u>sampling, sparsicification, decoupling, and MLPInit</u> along the way.**

Simple GNN models

(SGC; Wu et al., Simplifying Graph Convolutional Networks, ICML 2019)

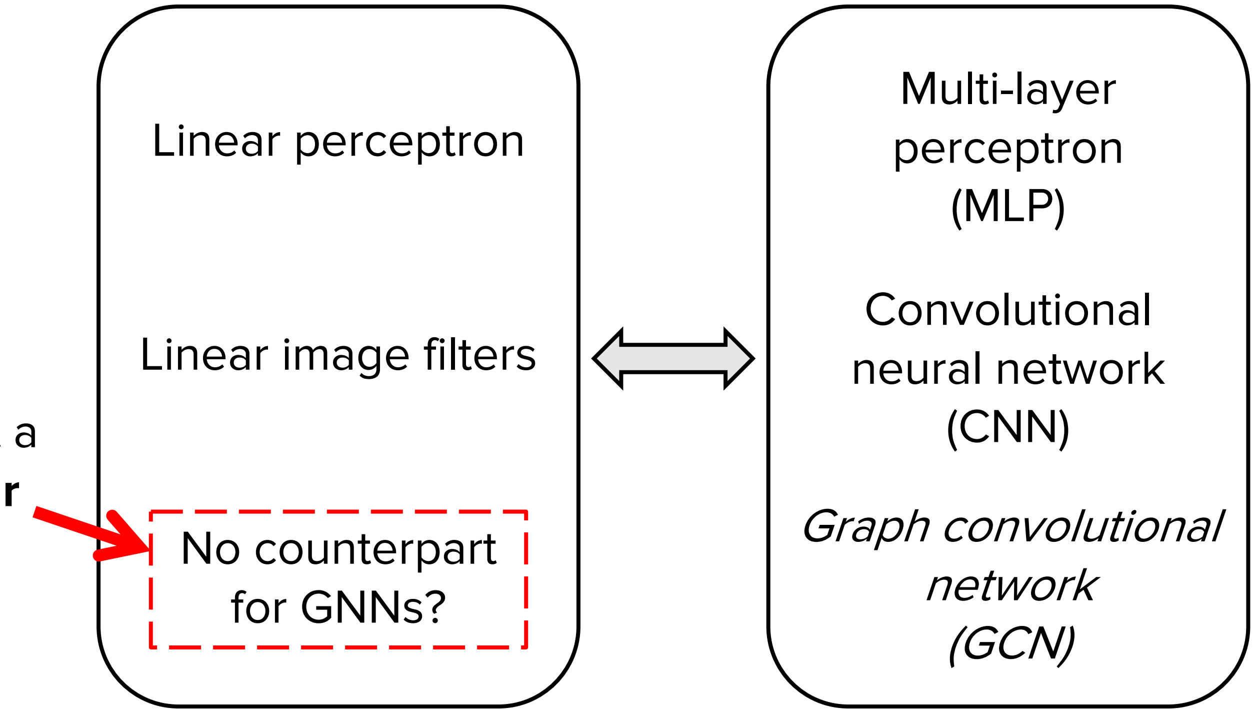
Motivation: Simple counterpart in graph domain

SGC (Wu et al., Simplifying Graph Convolutional Networks, ICML 2019)

"Historically, the development of machine learning algorithms has followed a clear trend from initial simplicity to need-driven complexity."

Motivation to construct a **simple** & **almost linear** version of GNN.

Simple Graph Convolution (SGC)


Linear perceptron
Linear image filters
No counterpart for GNNs?
Multi-layer perceptron (MLP)
Convolutional neural network (CNN)
Graph convolutional network (GCN)

Simple and easy to **interpret**

Introduced due to *insufficiencies in previous models*

Reference model: **GCN** (Kipf, 2017)

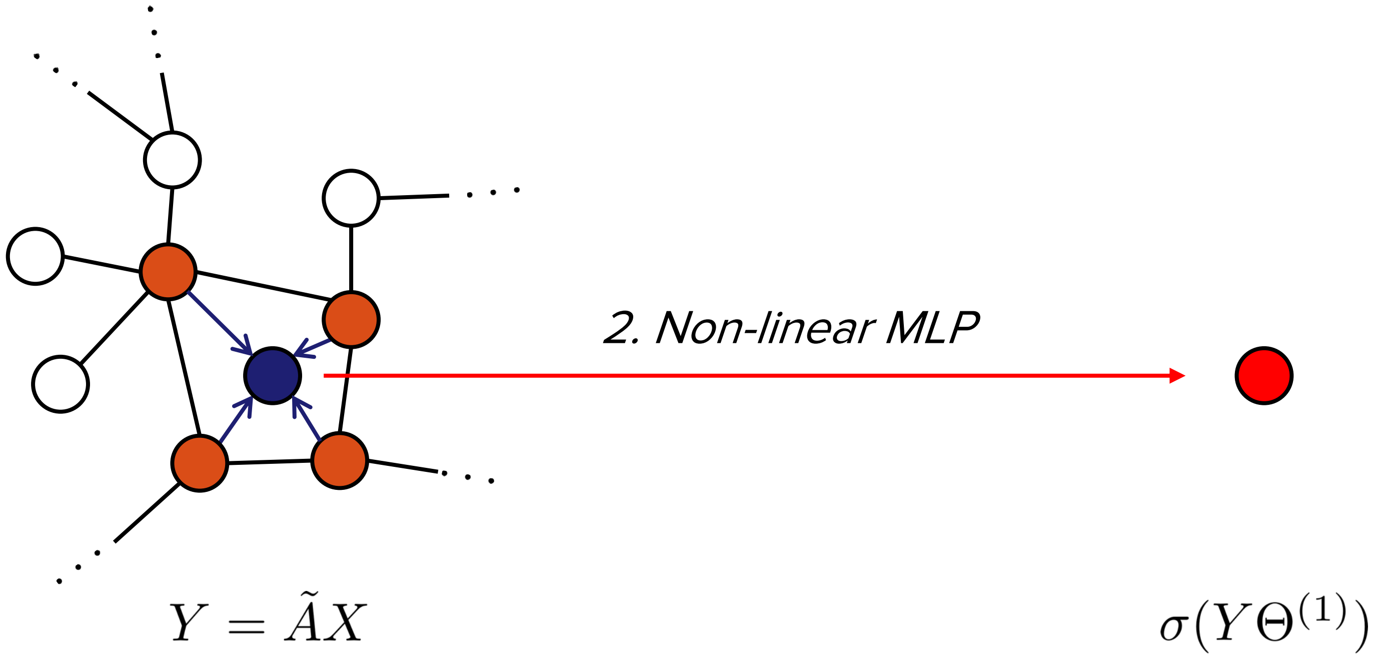
Reference model: GCN

SGC (Wu et al., Simplifying Graph Convolutional Networks, ICML 2019)

One layer of GCN

$$f_1(\tilde{A}, X) = \sigma(\tilde{A} X \Theta^{(1)})$$

N	Number of nodes
$\tilde{A} \in R^{N \times N}$	Normalized adjacency matrix
$X \in R^{N \times D}$	D-dimensional feature matrix
$\Theta^{(k)}$	Learnable weight matrix at *kth* layer
$\sigma(\cdot)$	Non-linear activation function
f_k	*kth* GCN layer



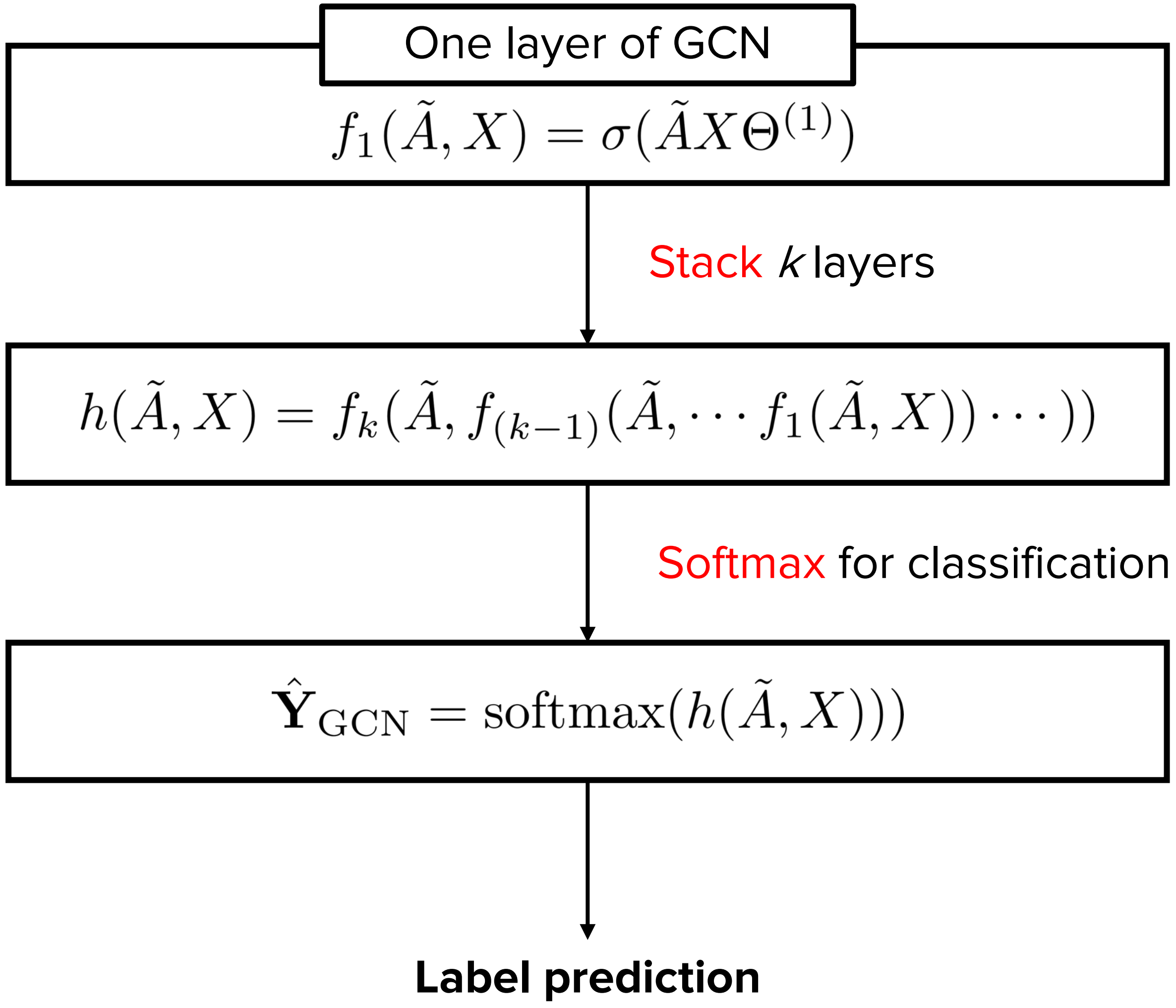
1. Local averaging

Reference model: GCN

SGC (Wu et al., Simplifying Graph Convolutional Networks, ICML 2019)

Overall architecture of GCN for node classification

N	Number of nodes
$\tilde{A} \in R^{N \times N}$	Normalized adjacency matrix
$X \in R^{N \times D}$	D-dimensional feature matrix
$\Theta^{(k)}$	Learnable weight matrix at *kth* layer
$\sigma(\cdot)$	Non-linear activation function
f_k	*kth* GCN layer


One layer of GCN
$f_1(\tilde{A}, X) = \sigma(\tilde{A} X \Theta^{(1)})$
Stack *k* layers
$h(\tilde{A}, X) = f_k(\tilde{A}, f_{(k-1)}(\tilde{A}, \cdots f_1(\tilde{A}, X)) \cdots))$
Softmax for classification
$\hat{\mathbf{Y}}_{\text{GCN}} = \text{softmax}(h(\tilde{A}, X)))$
Label prediction

Model architecture for Simplified Graph Convolution

SGC (Wu et al., Simplifying Graph Convolutional Networks, ICML 2019)

One layer of GCN

$$f_1(\tilde{A}, X) = \sigma(\tilde{A} X \Theta^{(1)})$$

Stack *k* layers

$$h(\tilde{A}, X) = f_k(\tilde{A}, f_{(k-1)}(\tilde{A}, \cdots f_1(\tilde{A}, X))\cdots))$$

Softmax for classification

$$\hat{\mathbf{Y}}_{\mathrm{GCN}} = \mathrm{softmax}(h(\tilde{A}, X)))$$

Label prediction

One layer of **SGC**

$$f_1(\tilde{A}, X) = \tilde{A} X \Theta^{(1)}$$

Remove the non-linear activation function for linearity

$$h(\tilde{A}, X) = \boxed{\tilde{A}\tilde{A}\cdots\tilde{A}} X \cdots \boxed{\Theta^{(1)} \cdots \Theta^{(k-1)} \Theta^{(k)}}$$

Collapse

$$\Theta^{(1)} \cdots \Theta^{(k-1)} \Theta^{(k)} \to \Theta$$

Collapse

$$h(\tilde{A}, X) = \boxed{\tilde{A}^k} X \cdots \boxed{\Theta}$$

$$\hat{\mathbf{Y}}_{\mathrm{SGC}} = \mathrm{softmax}(\tilde{A}^k X \Theta)$$

Label prediction

Empirical performance of Simplified Graph Convolution

SGC (Wu et al., Simplifying Graph Convolutional Networks, ICML 2019)

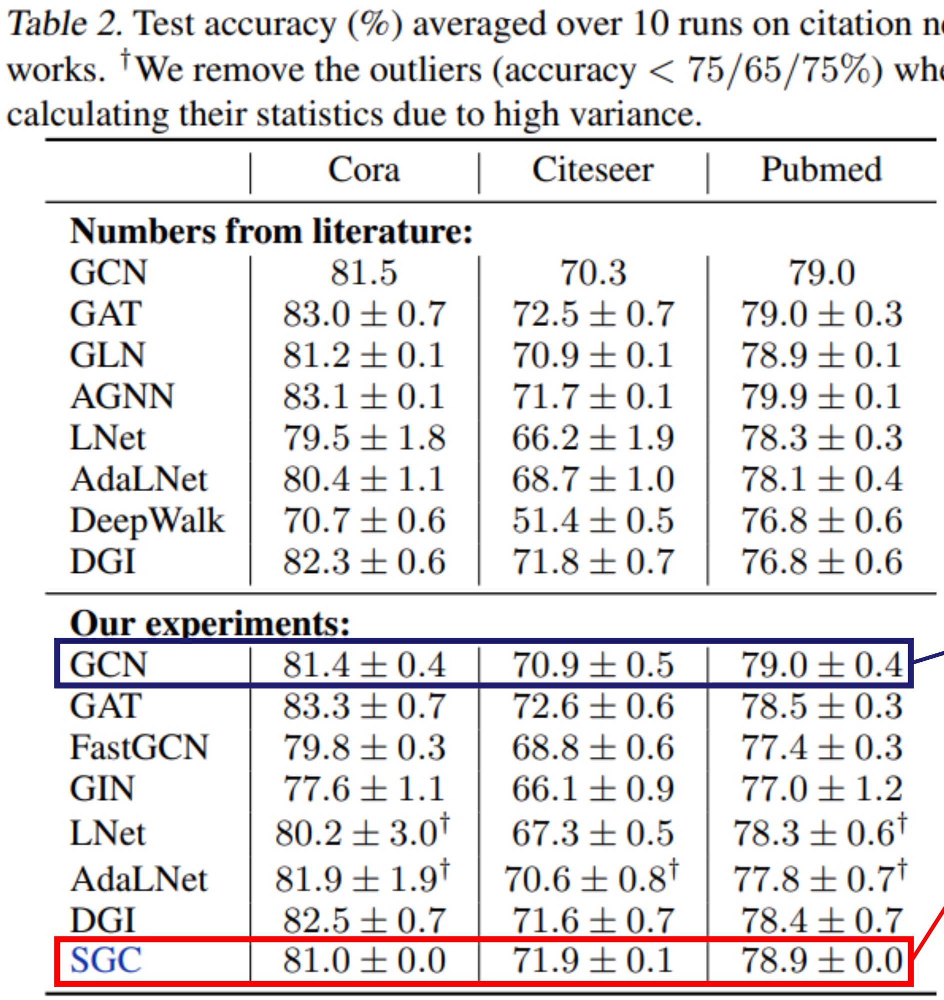


Table 2. Test accuracy (%) averaged over 10 runs on citation networks. †We remove the outliers (accuracy < 75/65/75%) when calculating their statistics due to high variance.

	Cora	Citeseer	Pubmed
Numbers from literature:			
GCN	81.5	70.3	79.0
GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
GLN	81.2 ± 0.1	70.9 ± 0.1	78.9 ± 0.1
AGNN	83.1 ± 0.1	71.7 ± 0.1	79.9 ± 0.1
LNet	79.5 ± 1.8	66.2 ± 1.9	78.3 ± 0.3
AdaLNet	80.4 ± 1.1	68.7 ± 1.0	78.1 ± 0.4
DeepWalk	70.7 ± 0.6	51.4 ± 0.5	76.8 ± 0.6
DGI	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
Our experiments:			
GCN	81.4 ± 0.4	70.9 ± 0.5	79.0 ± 0.4
GAT	83.3 ± 0.7	72.6 ± 0.6	78.5 ± 0.3
FastGCN	79.8 ± 0.3	68.8 ± 0.6	77.4 ± 0.3
GIN	77.6 ± 1.1	66.1 ± 0.9	77.0 ± 1.2
LNet	$80.2 \pm 3.0^{\dagger}$	67.3 ± 0.5	$78.3 \pm 0.6^{\dagger}$
AdaLNet	$81.9 \pm 1.9^{\dagger}$	$70.6 \pm 0.8^{\dagger}$	$77.8 \pm 0.7^{\dagger}$
DGI	82.5 ± 0.7	71.6 ± 0.7	78.4 ± 0.7
SGC	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0

Task: **semi-node classification**

Dataset: Citation networks (Cora / Citeseer / Pubmed)

The performance of SGC is very competitive w.r.t. GCN and other GNNs.

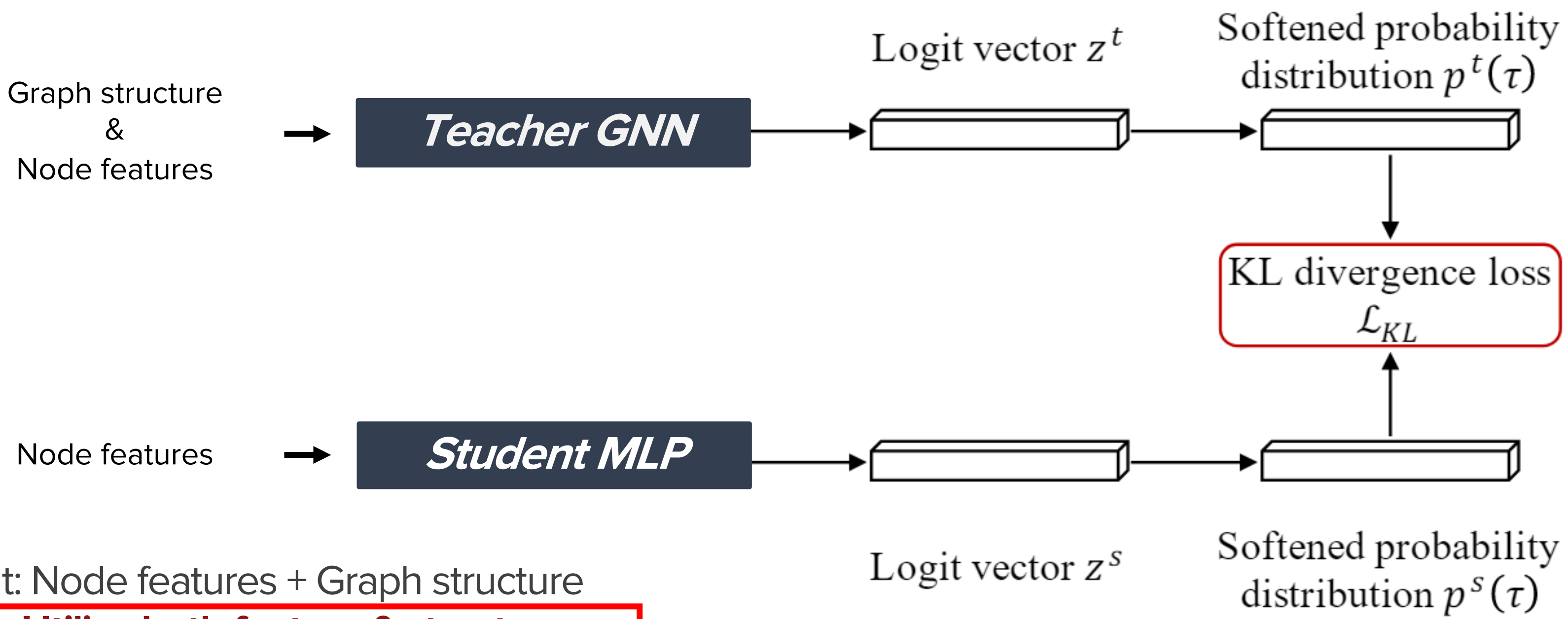
It is worth noting that SGCs *already* have the upper hand in terms of model complexity.

GNN-to-MLP knowledge distillation

(GLNN; Zhang et al., Graph-less Neural Networks: Teaching Old MLPs New Tricks via Distillation, ICLR 2022)

Overview of the solution proposed by GLNN

GLNN (Zhang et al., Graph-less Neural Networks: Teaching Old MLPs New Tricks via Distillation, ICLR 2022)


Graph structure
&
Node features
Teacher GNN
Logit vector z^t
Softened probability distribution $p^t(\tau)$
KL divergence loss $\mathcal{L}_{KL}$
Node features
Student MLP
Logit vector z^s
Softened probability distribution $p^s(\tau)$

- **GNN**
  - Input: Node features + Graph structure
  - Pros: **Utilize both feature & structure**
  - Cons: **Aggregation is slow**
- **MLP**
  - Input: Node feature (no graph structure)
  - Pros: **Blazingly fast**
  - Cons: **No access to graph structure**

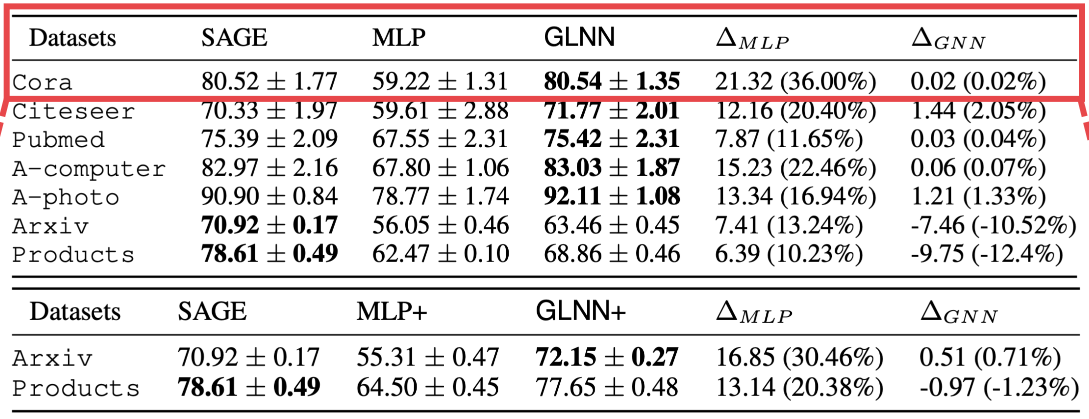
- **GLNN (Zheng et al., ICLR 2022)**
  - Main architecture: **MLP** (Fast!)
  - **Knowledge distillation (KD)** from **teacher GNN** (performance ↑)
  - Other follow-up studies (Zhang et al., 2022; Tian et al., 2022; Hu et al., 2021; Shin et al., 2023).

Hinton et al., Distilling the knowledge in a neural network, arXiv 2015
[Illustration] (Modified) Kim et al., Comparing Kullback-Leibler divergence and mean squared error loss in knowledge distillation, IJCAI 2021
Zheng et al., Cold Brew: Distilling graph node representations with incomplete or missing neighborhoods, ICLR 2022
Tian et al., Learning MLPs on Graphs: A Unified View of Effectiveness, Robustness, and Efficiency, ICLR 2023
Hu et al., Graph-MLP: Node classification without message passing in graph, arXiv 2021
Shin et al., Propagate & Distill: Towards effective graph learners using propagation-embracing MLPs, LoG 2023

Performance of GLNN

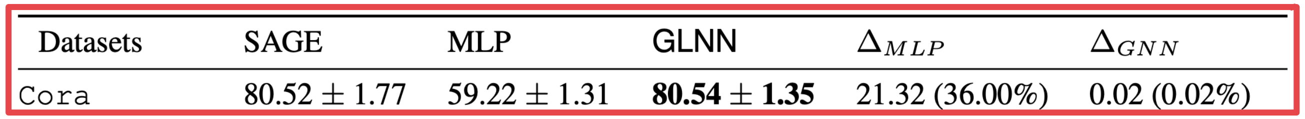
GLNN (Zhang et al., Graph-less Neural Networks: Teaching Old MLPs New Tricks via Distillation, ICLR 2022)

Performance: Sometimes even exceeds the teacher GNN



Datasets	SAGE	MLP	GLNN	Δ_{MLP}	Δ_{GNN}
Cora	80.52 ± 1.77	59.22 ± 1.31	**80.54 ± 1.35**	21.32 (36.00%)	0.02 (0.02%)
Citeseer	70.33 ± 1.97	59.61 ± 2.88	**71.77 ± 2.01**	12.16 (20.40%)	1.44 (2.05%)
Pubmed	75.39 ± 2.09	67.55 ± 2.31	**75.42 ± 2.31**	7.87 (11.65%)	0.03 (0.04%)
A-computer	82.97 ± 2.16	67.80 ± 1.06	**83.03 ± 1.87**	15.23 (22.46%)	0.06 (0.07%)
A-photo	90.90 ± 0.84	78.77 ± 1.74	**92.11 ± 1.08**	13.34 (16.94%)	1.21 (1.33%)
Arxiv	**70.92 ± 0.17**	56.05 ± 0.46	63.46 ± 0.45	7.41 (13.24%)	-7.46 (-10.52%)
Products	**78.61 ± 0.49**	62.47 ± 0.10	68.86 ± 0.46	6.39 (10.23%)	-9.75 (-12.4%)

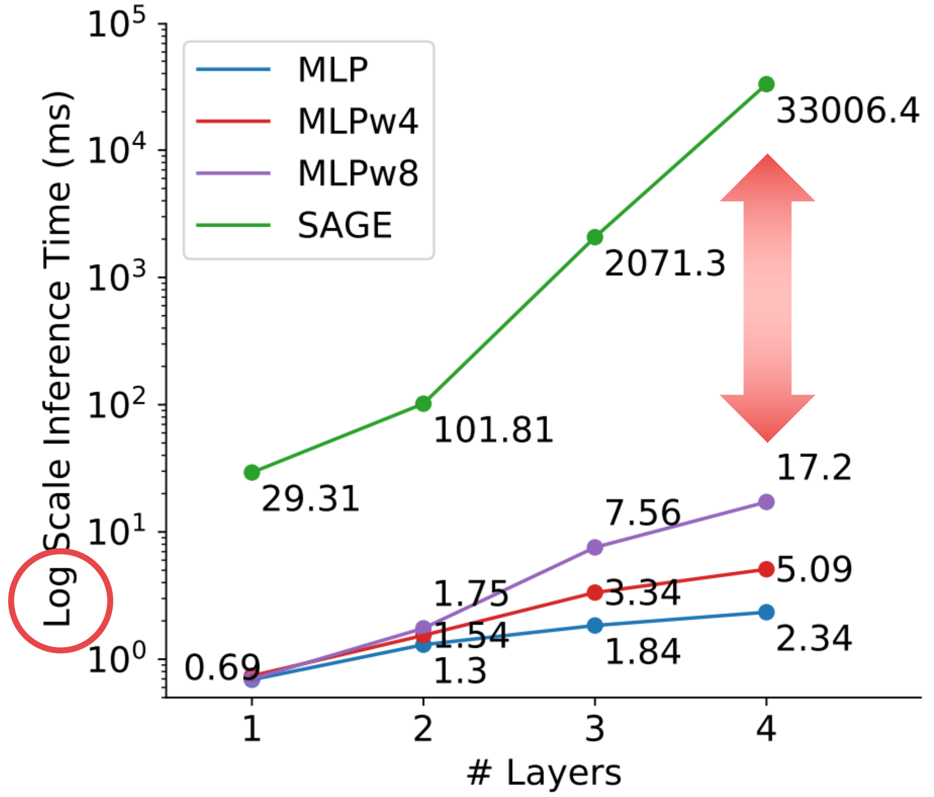
Datasets	SAGE	MLP+	GLNN+	Δ_{MLP}	Δ_{GNN}
Arxiv	70.92 ± 0.17	55.31 ± 0.47	**72.15 ± 0.27**	16.85 (30.46%)	0.51 (0.71%)
Products	**78.61 ± 0.49**	64.50 ± 0.45	77.65 ± 0.48	13.14 (20.38%)	-0.97 (-1.23%)



Datasets	SAGE	MLP	GLNN	Δ_{MLP}	Δ_{GNN}
Cora	80.52 ± 1.77	59.22 ± 1.31	**80.54 ± 1.35**	21.32 (36.00%)	0.02 (0.02%)

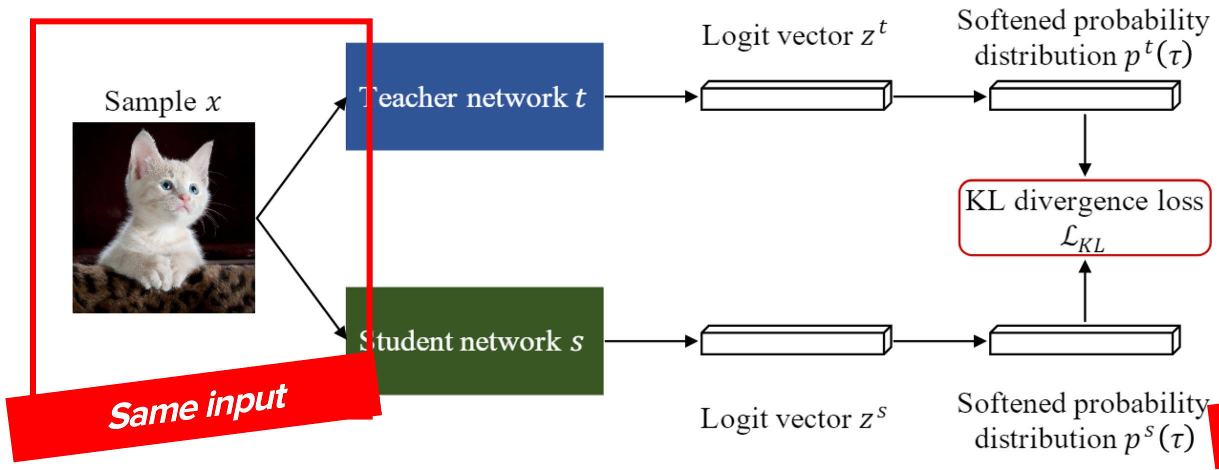
Why is this surprising? ***MLPs*** *have* ***no input related to graph structure***

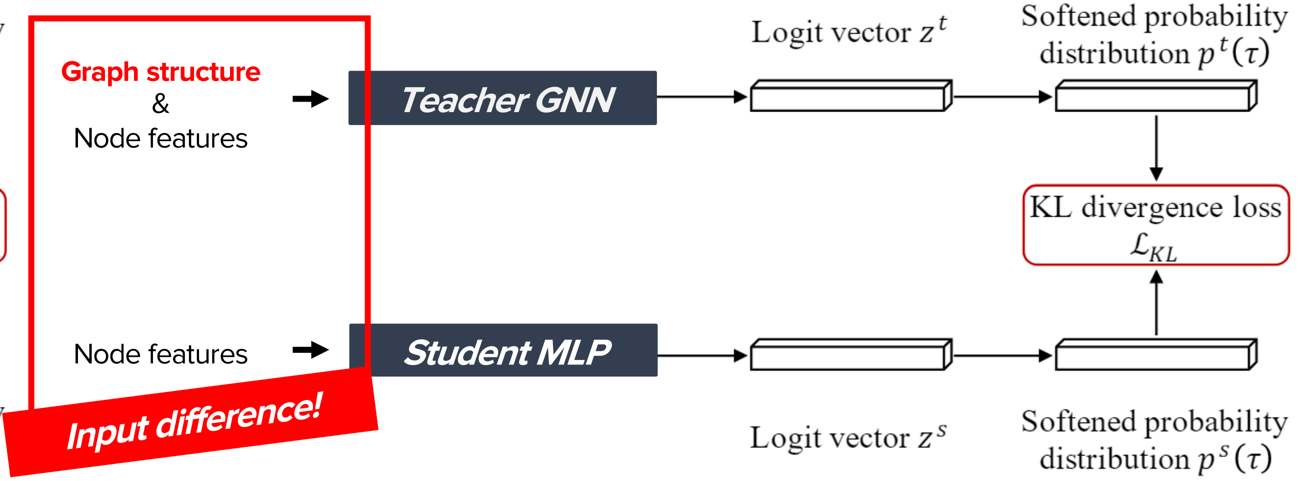
Inference speed: MLPs are just significantly faster
(this is rather obvious)



Unique benefit in GNN-to-MLP KD compared to traditional KD

GLNN (Zhang et al., Graph-less Neural Networks: Teaching Old MLPs New Tricks via Distillation, ICLR 2022)


Sample x
Teacher network t
Student network s
Logit vector z^t
Softened probability distribution $p^t(\tau)$
KL divergence loss $\mathcal{L}_{KL}$
Logit vector z^s
Softened probability distribution $p^s(\tau)$
Same input


Graph structure & Node features
Teacher GNN
Logit vector z^t
Softened probability distribution $p^t(\tau)$
KL divergence loss $\mathcal{L}_{KL}$
Node features
Student MLP
Logit vector z^s
Softened probability distribution $p^s(\tau)$
Input difference!

Knowledge distillation in other domains (e.g., CV)

Knowledge distillation in GNN-to-MLP

- Traditional KD scenarios: Model architectures are more or less the same
- GNN-to-MLP: Vastly different architectures (GNN vs. MLP) and input (Node features + graph structure vs. Node features only)
- The vast architectural difference brings **unique benefit and challenges** compared to traditional KD scenarios.

Why does this work?

GLNN (Zhang et al., Graph-less Neural Networks: Teaching Old MLPs New Tricks via Distillation, ICLR 2022)

1) There is **always the optimal weight parameter for any given problem**

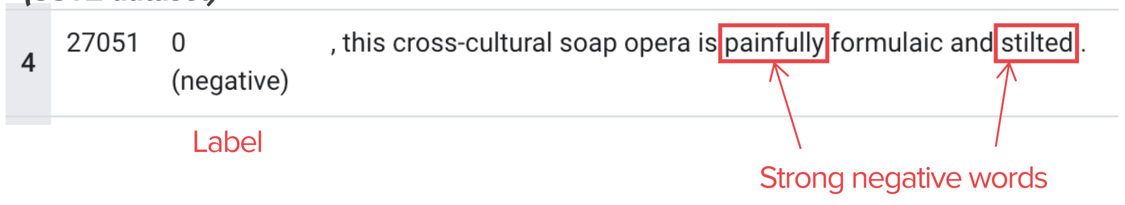
- (Theoretical analysis in the paper) concludes that **GNNs are more expressive than MLPs** due to the **architectural differences.**
- **Empirically**, however, **the gap makes little difference** when |X| is large.
- In real applications, **node features can be high dimensional like bag-of-words**, or even word embeddings, thus making |X| enormous.
- These point that the **node features should be informative & correlated to the graph structure**, which naturally connects to the next point…

(Personal note)

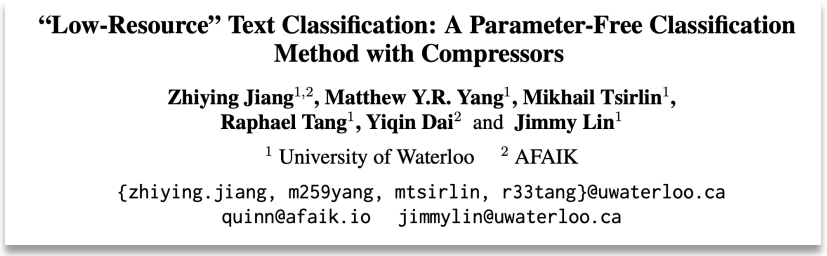
2) The problem (i.e., node classification) may be **easier than previously thought**.

1. Sentiment classification in NLP

(SST2 dataset)


4	27051	0 (negative)	, this cross-cultural soap opera is painfully formulaic and stilted .

2. Text classification (Jiang et al., ACL 2022)


"Low-Resource" Text Classification: A Parameter-Free Classification Method with Compressors

Zhiying Jiang[1,2], **Matthew Y.R. Yang**[1], **Mikhail Tsirlin**[1], **Raphael Tang**[1], **Yiqin Dai**[2] and **Jimmy Lin**[1]

[1] University of Waterloo [2] AFAIK

{zhiying.jiang, m259yang, mtsirlin, r33tang}@uwaterloo.ca quinn@afaik.io jimmylin@uwaterloo.ca

Another noteworthy study: MLPInit

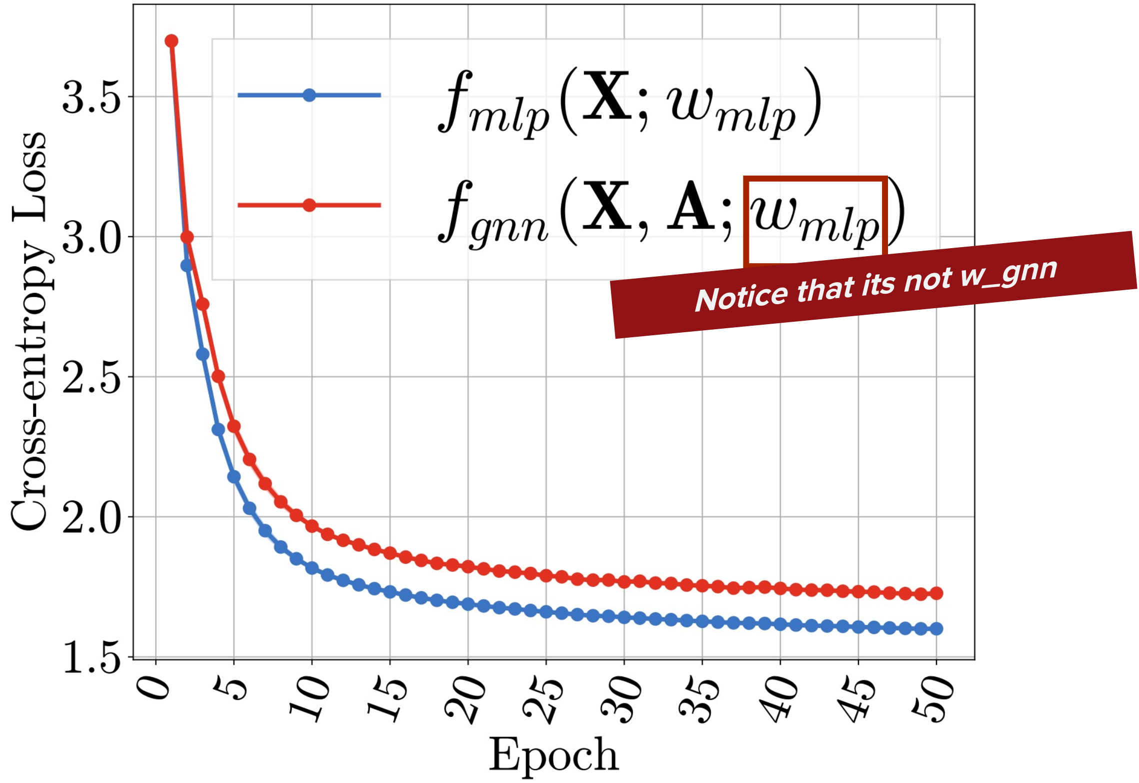
MLPInit (Han et al., MLPInit: Embarrassingly simple GNN training acceleration with MLP initialization, ICLR’23)

$$\textbf{GNN: } \mathbf{H}^l = \sigma(\mathbf{A}\mathbf{H}^{l-1}\boxed{\mathbf{\Theta}^l}) \qquad \textbf{MLP: } \mathbf{H}^l = \sigma(\mathbf{H}^{l-1}\boxed{\mathbf{\Theta}^l})$$

Computation speed: Training MLPs are MUCH faster than GNNs

Performance: Naïvely replacing the weights of GNN to those of a trained MLP immediately provide benefits.

Operation		Yelp	
#Nodes		716847	
#Edges		13954819	
	Forward	Backward	Total
~~*Feature transformation*~~ $Z = WX$	1.58	4.41	5.99
$H = AZ$ *Message-passing*	9.74	19157.17	19166.90
			$\boxed{3199\times}$



Another noteworthy study: MLPInit

MLPInit (Han et al., MLPInit: Embarrassingly simple GNN training acceleration with MLP initialization, ICLR'23)

Algorithm 1 PyTorch-style Pseudocode of MLPInit

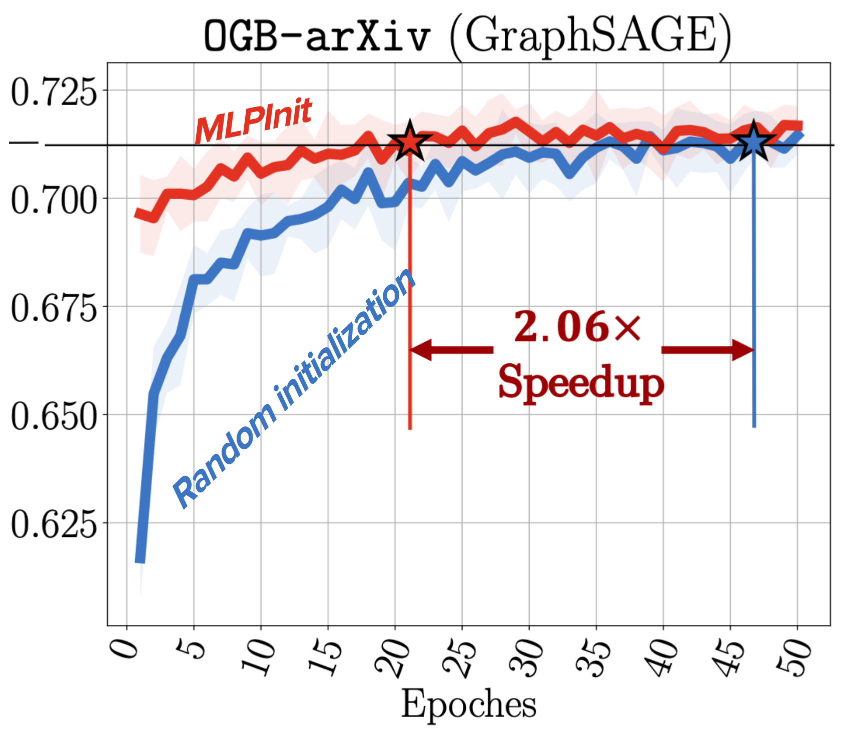
```
# f_gnn: graph neural network model
# f_mlp: PeerMLP of f_gnn

# Train PeerMLP for N epochs
for X, Y in dataloader_mlp:
    P = f_mlp(X)
    loss = nn.CrossEntropyLoss(P, Y)
    loss.backward()
    optimizer_mlp.step()

# Initialize GNN with MLPInit
torch.save(f_mlp.state_dict(), "w_mlp.pt")
f_gnn.load_state_dict("w_mlp.pt")

# Train GNN for n epochs
for X, A, Y in dataloader_gnn:
    P = f_gnn(X, A)
    loss = nn.CrossEntropyLoss(P, Y)
    loss.backward()
    optimizer_gnn.step()
```

Results in 1) training speed benefits, 2) performance benefits and others



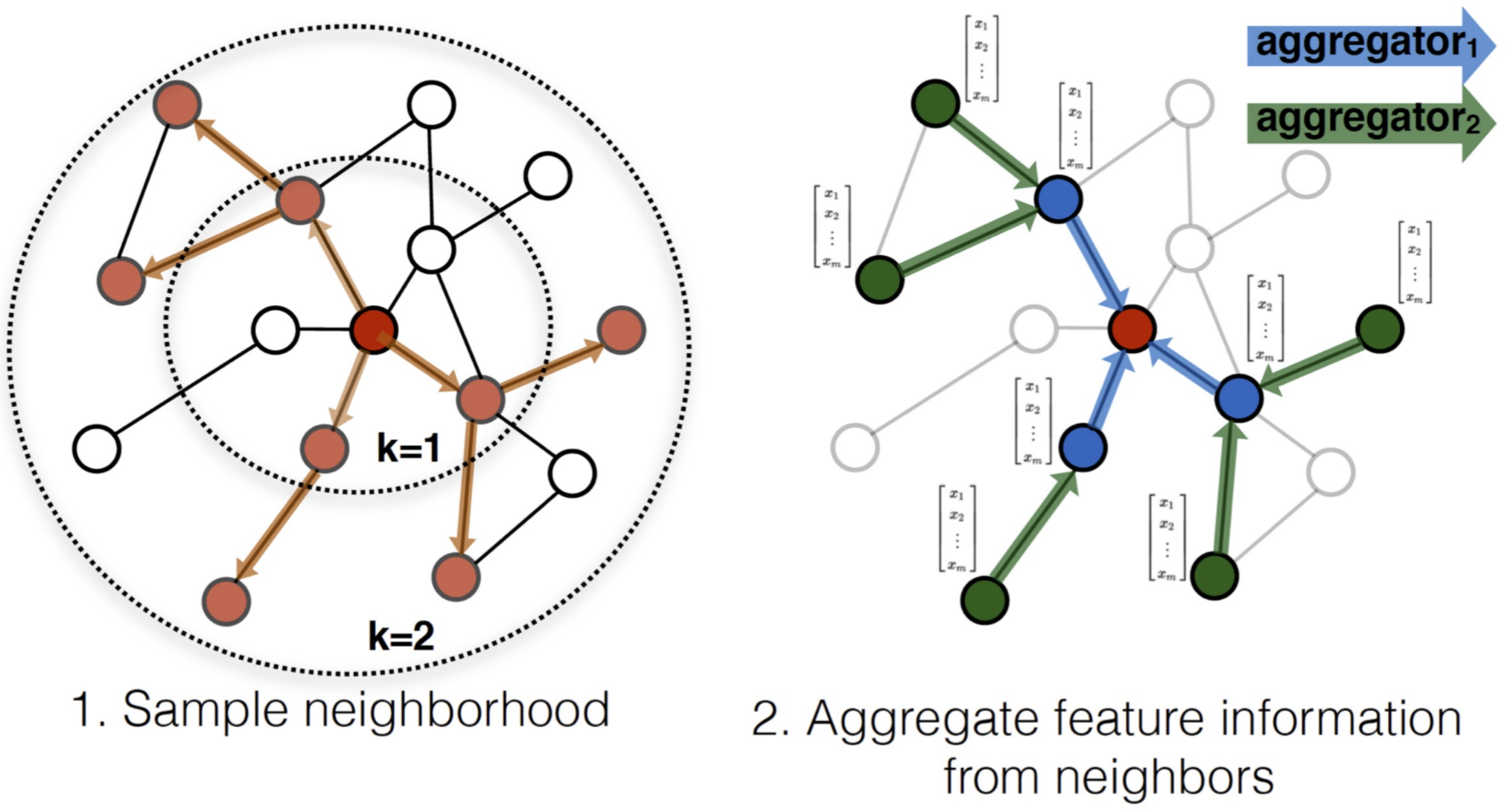
	Methods	Flickr	Yelp	Reddit	Reddit2	A-products	OGB-arXiv	OGB-products	Avg.
SAGE	Random	53.72±0.16	63.03±0.20	96.50±0.03	51.76±2.53	77.58±0.05	72.00±0.16	80.05±0.35	70.66
	MLPInit	53.82±0.13	63.93±0.23	96.66±0.04	89.60±1.60	77.74±0.06	72.25±0.30	80.04±0.62	76.29
	Improv.	↑ 0.19%	↑ 1.43%	↑ 0.16%	↑ 73.09%	↑ 0.21%	↑ 0.36%	↓ 0.01%	↑ 7.97%

MLPInit proposes to train an MLP on the features, and use the weights to initialize the GNN (and go further training)

Final notes on sampling, sparsification, and decoupling methods

An overview of different sampling methods

Node-wise sampling: GraphSAGE [1]

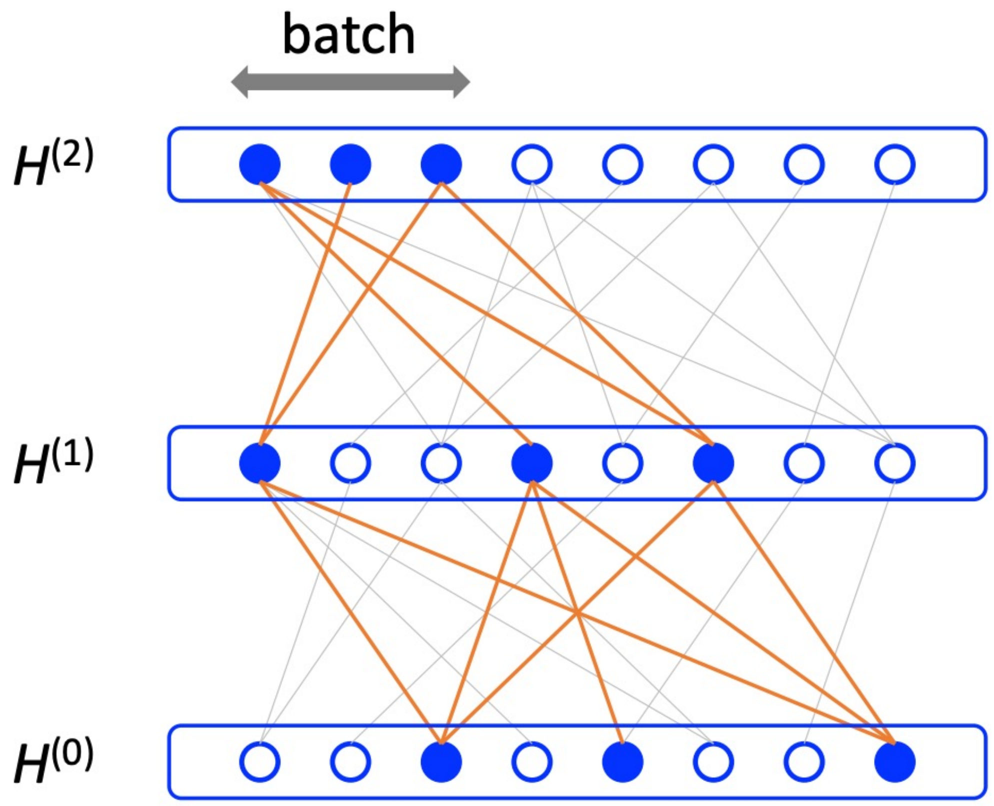


Strict limit on the maximum number of nodes to aggregate from for all layers

[1] Hamilton et al., Inductive Representation Learning on Large Graphs, NeurIPS 2017

An overview of different sampling methods

Layer-wise sampling: FastGCN [1], LADIES [2]

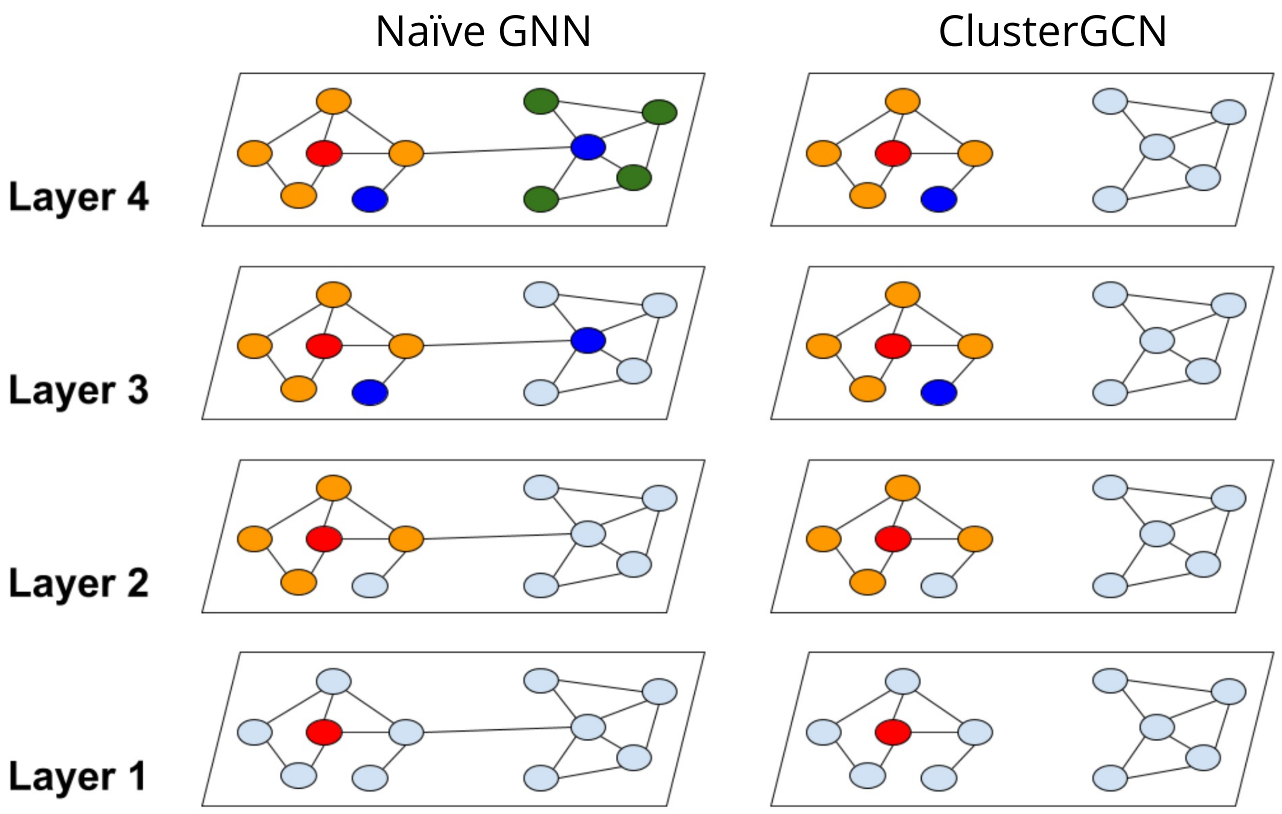


Sample nodes *per* layer to avoid redundancy via importance sampling.

[1] Chen et al., FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling, ICLR 2018
[2] Zou et al., Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks, NeurIPS 2019
Figure from FastGCN.

An overview of different sampling methods

Subgraph sampling: ClusterGCN [1], GraphSAINT [2]



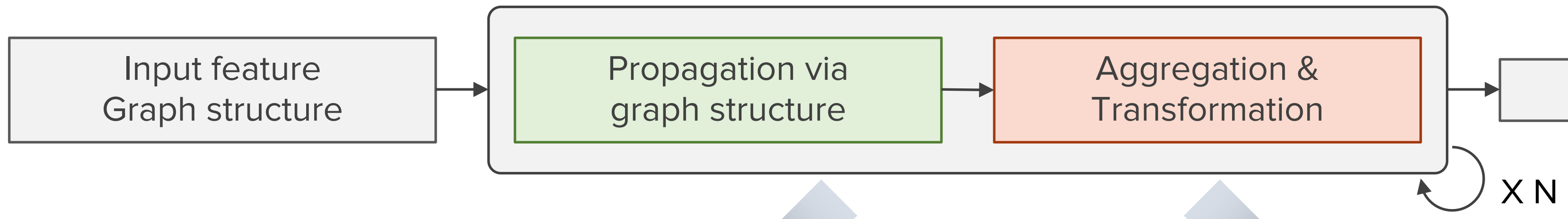
Let’s extract/partition smaller subgraphs and run full GNNs instead on the full graph.

[1] Chiang et al., Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks, KDD2018
[2] Zeng et al., GraphSAINT: Graph Sampling Based Inductive Learning Method, ICLR 2020
Figure from Cluster-GCN.

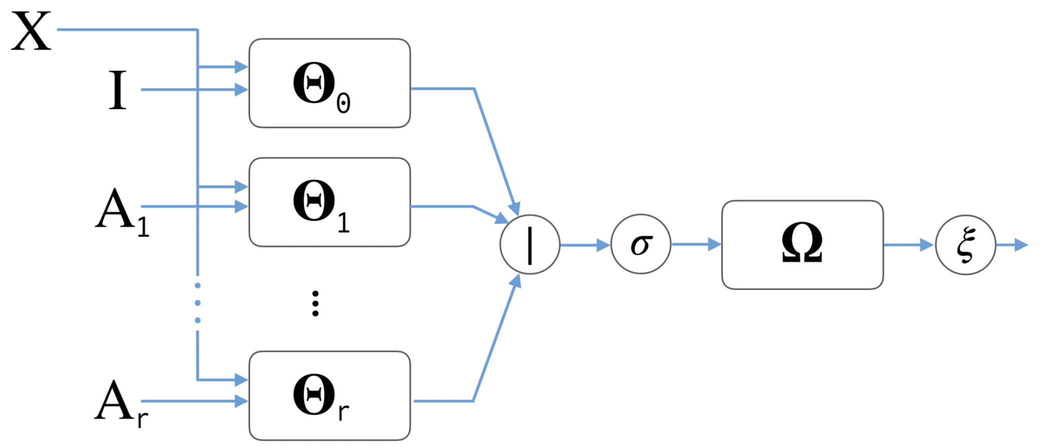
An overview of decoupling methods

Decoupling-based methods **perform message-passing separately feature transformation**, and it is performed once in the CPU to exploit the large memory capacity.

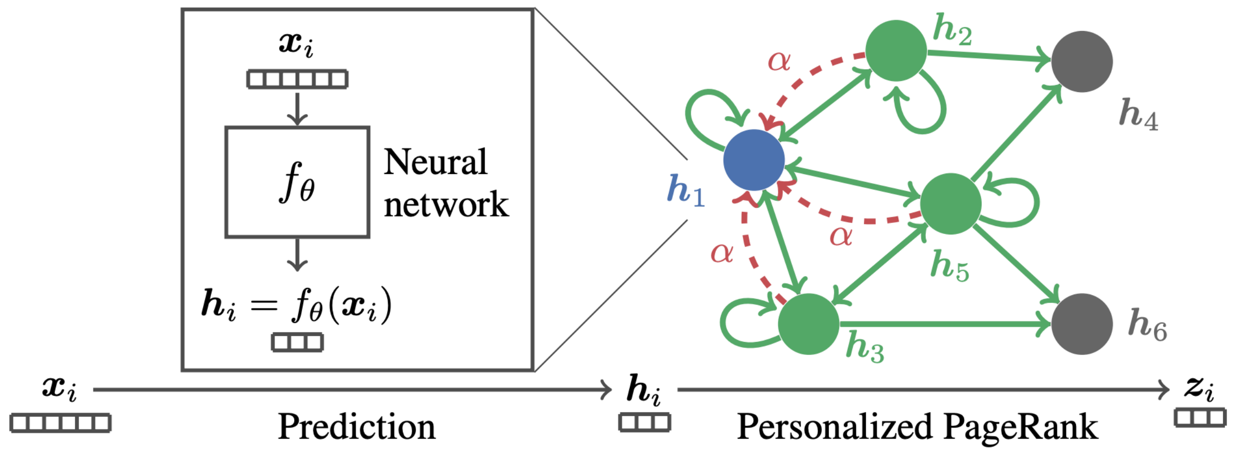
Typical GNN architecture:



Propagation as ***pre***-processing (SIGN [1])



Propagation as ***post***-processing (APPNP [2])



[1] Fransca et al., SIGN: Scalable Inception Graph Neural Networks, arXiv (2020)
[2] Gasteiger et al., Predict then Propagate: Graph Neural Networks meet Personalized PageRank, ICLR 2019

DSpar: Graph sparsification strategy

Liu et al., DSpar: An Embarrisingly Simple Strategy for Efficient GNN Training and Inference via Degree-based Sparsification, TMLR (2023)

Sparse matrix multiplication takes most of the computation time in GNNs

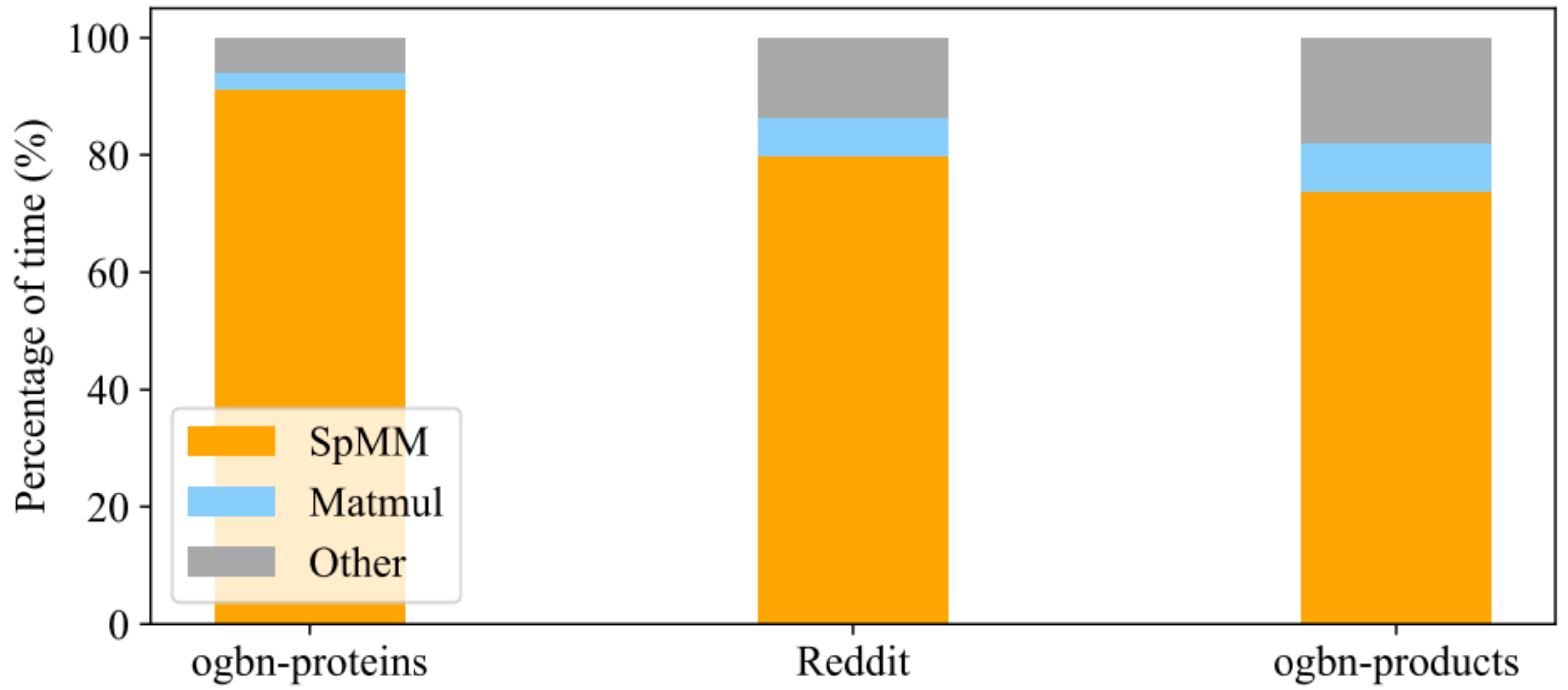
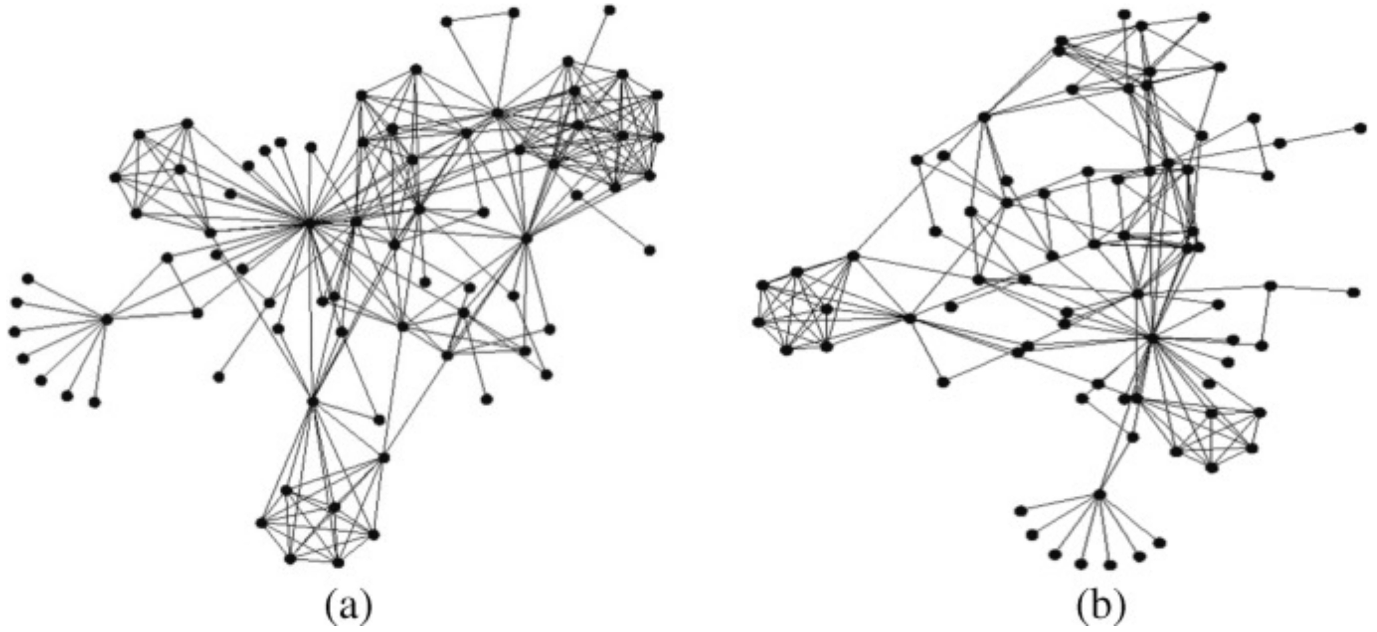


Figure 1: The time profiling of a two-layer GCNs on different datasets. `SpMM` in the aggregation phase may take 70% ∼ 90% of the total time.

DSpar: Graph sparsification strategy

Liu et al., DSpar: An Embarrisingly Simple Strategy for Efficient GNN Training and Inference via Degree-based Sparsification, TMLR (2023)

Graph pruning is a technique that attempts to address this problem by deleting "unimportant" edges.



- ✓ **Theory-based:** Prune the edges based on theoretical properties of the graph structure (e.g., effective resistance [1])
- **Learning-based:** Directly learn edge importance from the data (e.g., Neural sparsifier, LTH).
  Disadvantage: Requires additional learning module to solve an efficiency task.

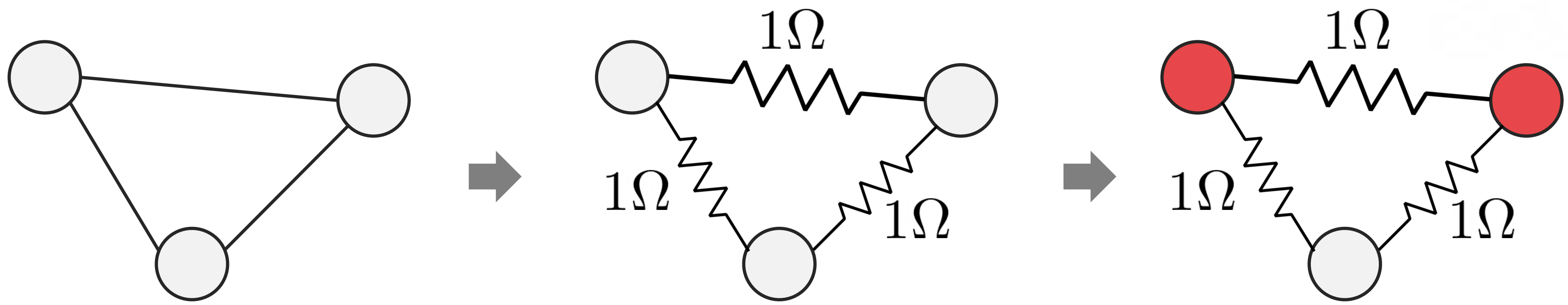
Figure from: Dolgorsuren et al., EM-FGS: Graph sparsification via faster semi-metric edges pruning. Appl. Intell. 49(10): 3731-3748 (2019)
[1] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. SIAM Journal on Computing, 40(6):1913–1926, 2011

DSpar: Graph sparsification strategy

Liu et al., DSpar: An Embarrisingly Simple Strategy for Efficient GNN Training and Inference via Degree-based Sparsification, TMLR (2023)

Based on an effective resistance-based graph sparsification method

$$R_{uv} = \frac{1}{1+\frac{1}{2}} = \frac{2}{3}$$



- Effective resistance: The total resistance that a current would experience in a circuit, especially when multiple resistors are connected in series or parallel.
- What is the resistance between two nodes in a 'circuitfied' graph?
- Given: The general form of calculating effective resistance is as follows.

$$(X_u - X_v)^\top \mathcal{L}^+ (X_u - X_v)$$

Pseudoinverse of the graph Laplacian

DSpar: Graph sparsification strategy

Liu et al., DSpar: An Embarrisingly Simple Strategy for Efficient GNN Training and Inference via Degree-based Sparsification, TMLR (2023)

We can replace the pseudoinverse with a degree-based heuristic

Theorem 1 (Corollary 3.3 in Lovász (1993)). *For all $e = (u, v) \in \mathcal{E}$, we have* $\frac{1}{2}(\frac{1}{d_u} + \frac{1}{d_v}) \leq R_e \leq \frac{1}{\alpha}(\frac{1}{d_u} + \frac{1}{d_v})$, *where* α $(\alpha \leq 2)$ *is the smallest non-zero eigenvalue of* $\mathcal{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$.

= The effective resistance can be effectively approximated by degree information

Algorithm 1: Sampling-based Graph Sparsification Spielman & Srivastava (2011)

```
Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, sampling probability $\{p_e\}_{e \in \mathcal{E}}$, number of samples to draw $Q$.
Output: the sparsified weighted graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ with edge weights $\{w_e\}_{e \in \mathcal{E}'}$
1  $\mathcal{E}' \leftarrow \{\}$
2  for $j = 1, \cdots, \boxed{Q}$ do
3      Sample an edge $e \sim \mathcal{E}$ with replacement according to $\boxed{p_e}$
4      if $e \notin \mathcal{E}'$ then
5          Add $e$ to $\mathcal{E}'$ with weight $w_e = \frac{A_e}{Qp_e}$
6      end
7      else
8          $w_e \leftarrow w_e + \frac{A_e}{Qp_e}$.
9      end
10 end
11 return $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ with edge weights $\{w_e\}_{e \in \mathcal{E}'}$
```

determined by the graph size and desired approximation error

proportional to the effective resistance ✓

→ replace to $p'_e \propto \frac{1}{d_u} + \frac{1}{d_v}$

Reducing the process time of Reddit dataset from **263 seconds to 0.6 seconds**

Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. SIAM Journal on Computing, 40(6):1913–1926, 2011

Takeaways

1. Exponential increase of complexity in message-passing results in practical limitations
2. Simple GNN models (GCN): Get rid of non-linearities (mostly) and compress matrix multiplications into one if possible
3. GNN-to-MLP: Use the knowledge of the GNN model to guide the student MLP model
4. MLPInit: Use the weights learned from the MLP to initialize a GNN
5. Sampling: Only use a part of the graph during feed-forward
6. Sparsification (DSpar): Pre-process the graph to reduce unnecessasry edges
7. Decoupling: Reduce the number of aggregation steps as much as possible

Thank you!

Please feel free to ask any questions :)

jordan7186.github.io