MLPInit: Embarrassingly Simple GNN Training Acceleration with MLP Initialization

ICLR2023

Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, Neil Shah

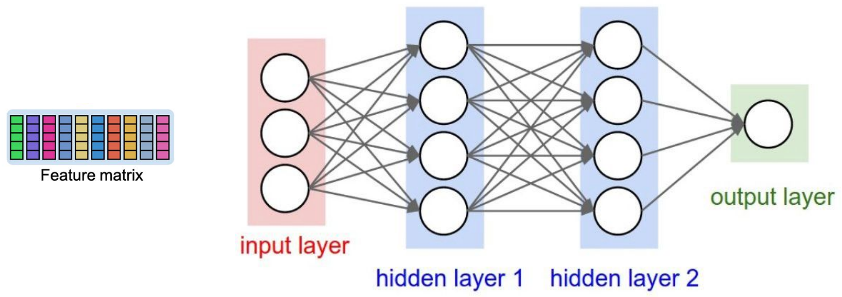
Texas A&M University, Snap Inc., Rice University

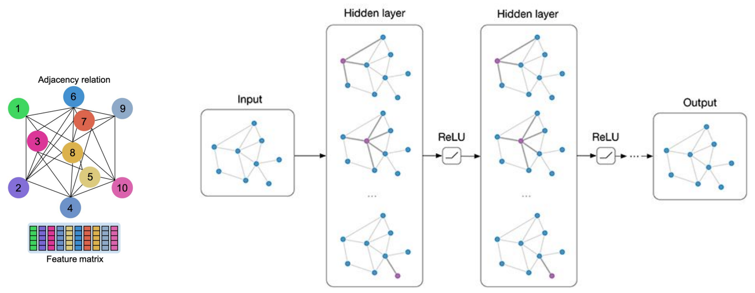
Graph Context Empower Graph Learning

- Graph Neural Network

Graph Context

$$\text{MLP:} \quad \mathbf{H}^l = \sigma(\mathbf{W}^l_{mlp}\mathbf{H}^{l-1}) \qquad \text{GNN:} \quad \mathbf{H}^l = \sigma(\mathbf{A}\mathbf{W}^l_{gnn}\mathbf{H}^{l-1})$$



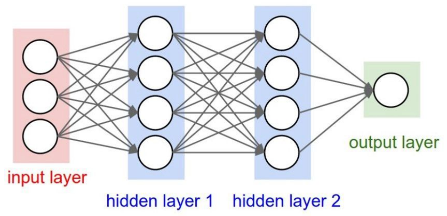


https://tkipf.github.io/graph-convolutional-networks/

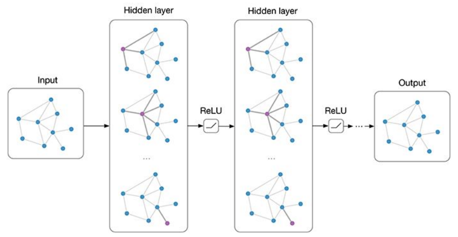
GNN empowers graph learning via message passing.

GNNs vs. MLPs

MLPs



GNNs



	MLPs	GNNs
Effectiveness (for graph)	Worse performance	Superior performance
Efficiency	Computationally efficient	Computationally cost

GNNs are powerful for graph while MLPs are computationally efficient.

Begin with an Intriguing Phenomenon

$$\text{MLP:} \quad \mathbf{H}^l = \sigma(\mathbf{W}^l_{mlp}\mathbf{H}^{l-1}) \qquad \text{GNN:} \quad \mathbf{H}^l = \sigma(\mathbf{A}\mathbf{W}^l_{gnn}\mathbf{H}^{l-1})$$

GNN and MLP have the same trainable weight.

- If the dimensions of the hidden layers are the same
- we refer to that MLP as a PeerMLP

What will happen if we directly adopt the weights of a converged PeerMLP to GNN?

Begin with an Intriguing Phenomenon

Only trained here

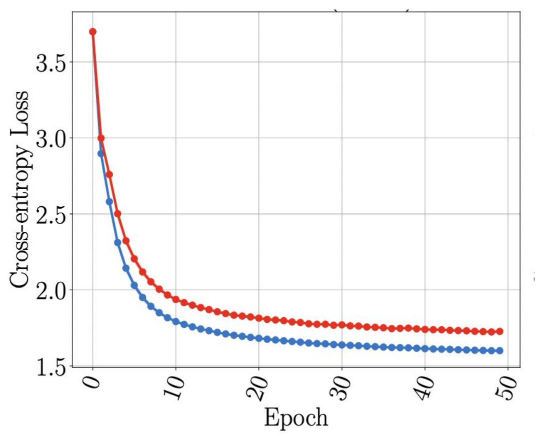
$$\text{MLP:} \quad \mathbf{H}^l = \sigma(\mathbf{W}^l_{mlp}\mathbf{H}^{l-1}) \qquad \text{GCN:} \quad \mathbf{H}^l = \sigma(\mathbf{A}\mathbf{W}^l_{mlp}\mathbf{H}^{l-1})$$

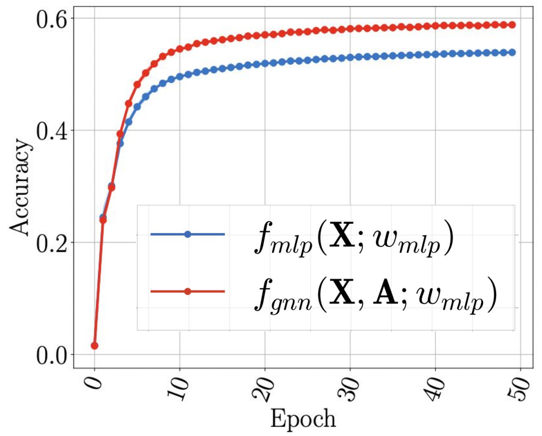
	PeerMLP	GCN w/ w_{peermlp}	Improv.	GCN
Cora	58.50	77.60	↑ 32.64%	82.60
CiteSeer	60.50	69.70	↑ 15.20%	71.60
PubMed	73.60	78.10	↑ 6.11%	79.80

GNN using the weights from a fully-trained PeerMLP performs better than itself.

Further Investigation

- PeerMLP $f_{mlp}(\mathbf{X}; w_{mlp})$; GNN $f_{gnn}(\mathbf{X}, \mathbf{A}; w_{mlp})$
- w_{mlp} is only trained by PeerMLP





The loss curve decreases while the accuracy curve are increas.

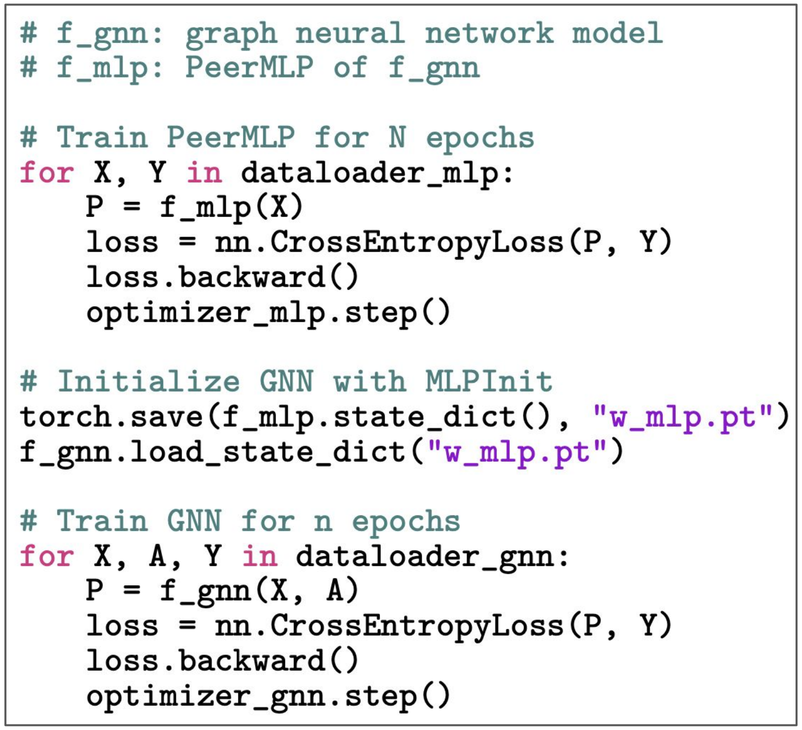
The GNN can be optimized by updating its PeerMLP.

MLPInit

For a target GNN,

1. Construct its PeerMLP
2. Train PeerMLP to converge

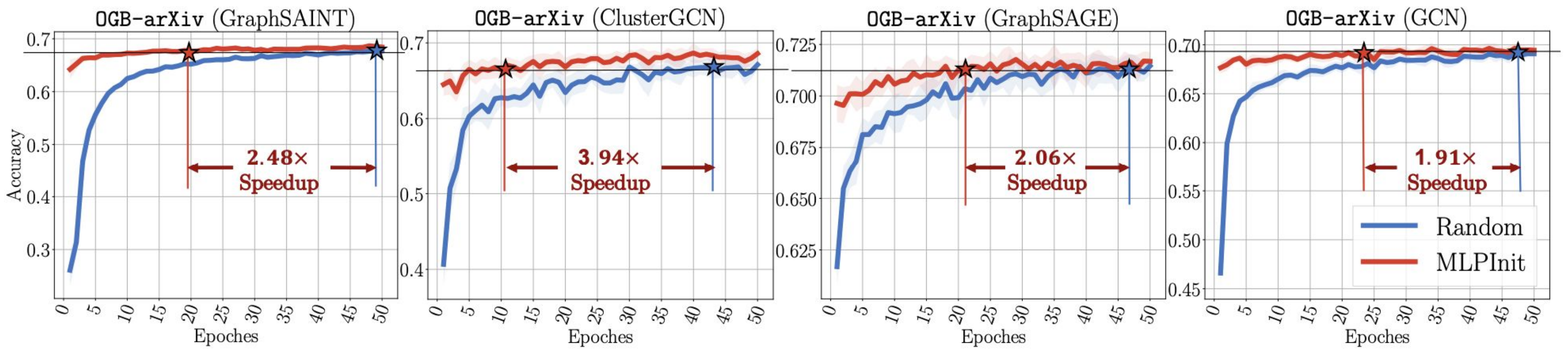
   converge $\longrightarrow w^{*}_{mlp}$
3. Initialize GNN with w^{*}_{mlp}
4. Fine tune the GNN



```
# 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()
```

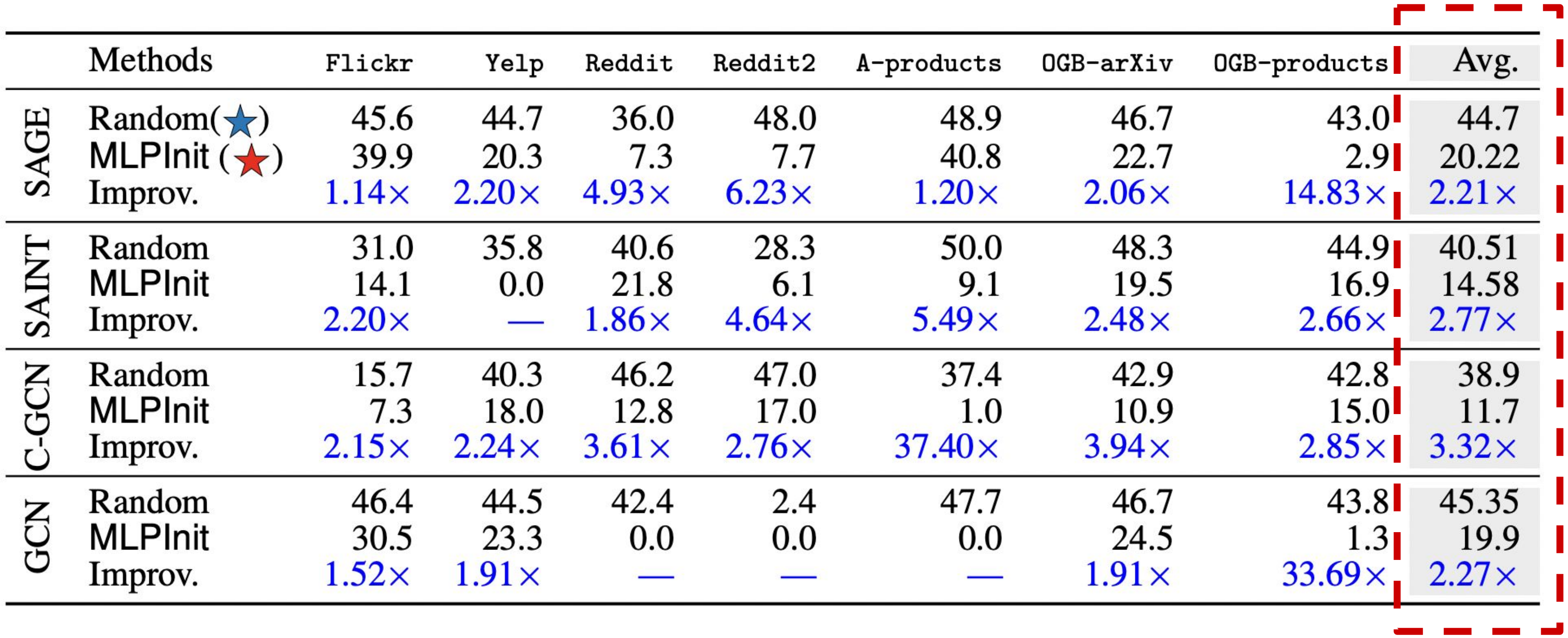
Why "Embarrassingly Simple"? Construct a PeerMLP and train it.

At a Glance: Faster and Better



1. MLPInit can accelerate GNN training by providing a better initialization of GNN.
2. MLPInit obtain better accuracy, gain performance improvement.

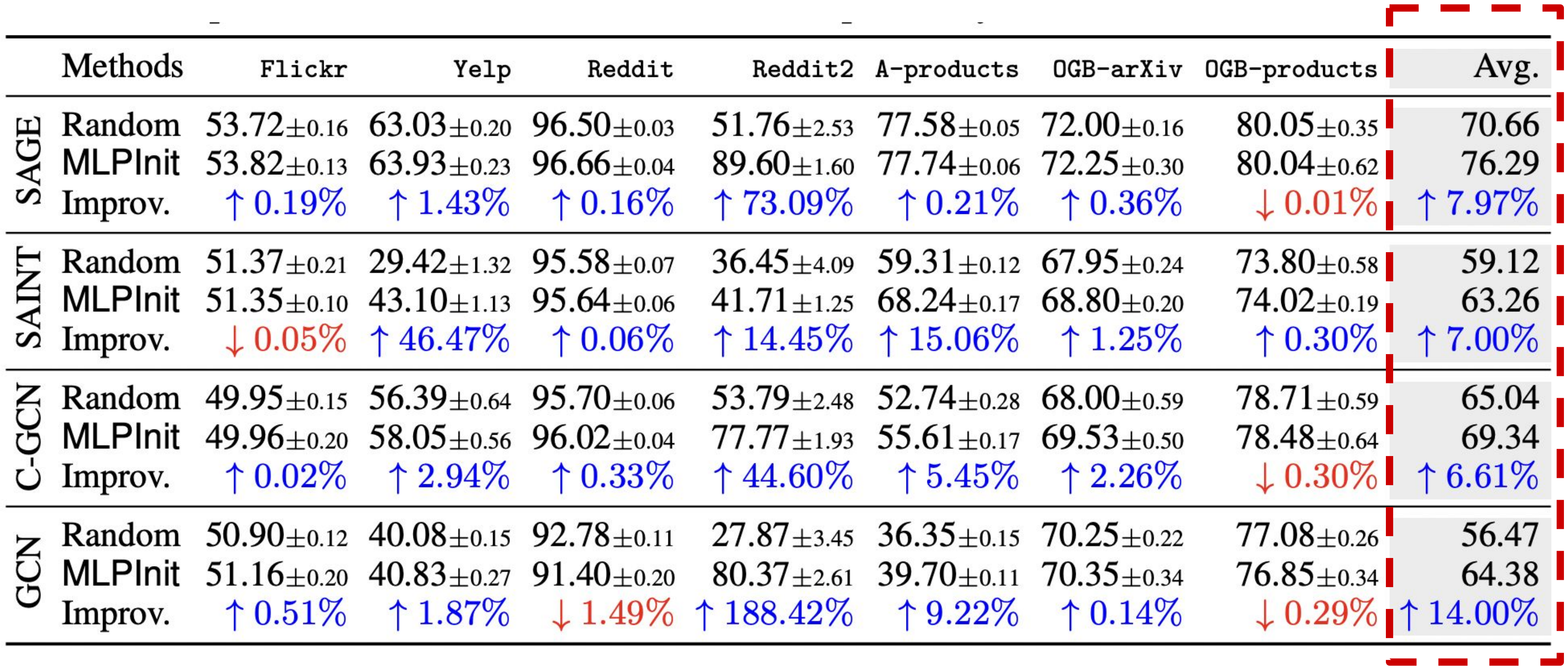
How Fast MLPInit Accelerate GNN?



	Methods	Flickr	Yelp	Reddit	Reddit2	A-products	OGB-arXiv	OGB-products	Avg.
SAGE	Random(★)	45.6	44.7	36.0	48.0	48.9	46.7	43.0	44.7
	MLPInit (★)	39.9	20.3	7.3	7.7	40.8	22.7	2.9	20.22
	Improv.	1.14×	2.20×	4.93×	6.23×	1.20×	2.06×	14.83×	2.21×
SAINT	Random	31.0	35.8	40.6	28.3	50.0	48.3	44.9	40.51
	MLPInit	14.1	0.0	21.8	6.1	9.1	19.5	16.9	14.58
	Improv.	2.20×	—	1.86×	4.64×	5.49×	2.48×	2.66×	2.77×
C-GCN	Random	15.7	40.3	46.2	47.0	37.4	42.9	42.8	38.9
	MLPInit	7.3	18.0	12.8	17.0	1.0	10.9	15.0	11.7
	Improv.	2.15×	2.24×	3.61×	2.76×	37.40×	3.94×	2.85×	3.32×
GCN	Random	46.4	44.5	42.4	2.4	47.7	46.7	43.8	45.35
	MLPInit	30.5	23.3	0.0	0.0	0.0	24.5	1.3	19.9
	Improv.	1.52×	1.91×	—	—	—	1.91×	33.69×	2.27×

MLPInit can significantly reduce the training time of GNNs.

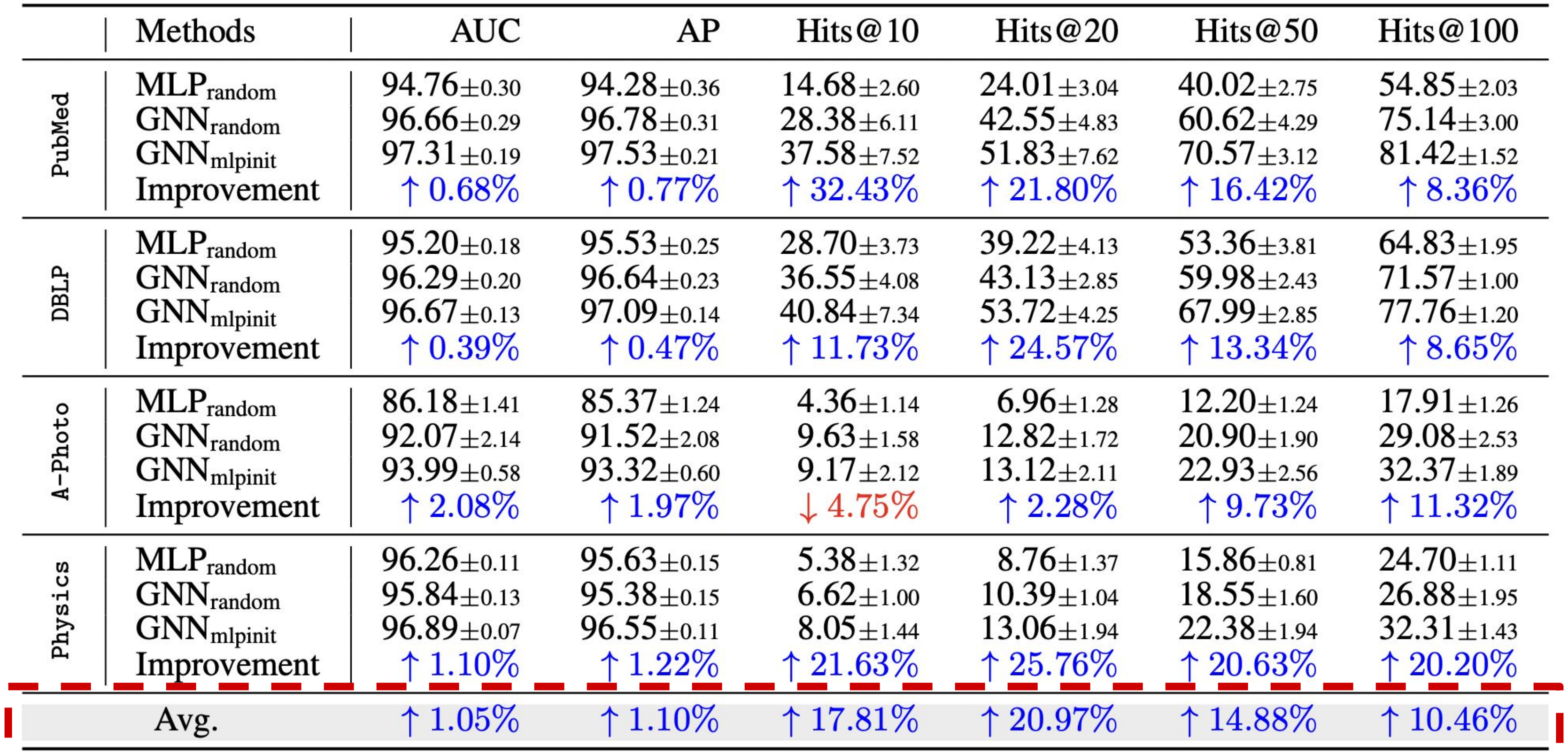
How Well does MLPInit Perform?



	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%
SAINT	Random	51.37±0.21	29.42±1.32	95.58±0.07	36.45±4.09	59.31±0.12	67.95±0.24	73.80±0.58	59.12
	MLPInit	51.35±0.10	43.10±1.13	95.64±0.06	41.71±1.25	68.24±0.17	68.80±0.20	74.02±0.19	63.26
	Improv.	↓ 0.05%	↑ 46.47%	↑ 0.06%	↑ 14.45%	↑ 15.06%	↑ 1.25%	↑ 0.30%	↑ 7.00%
C-GCN	Random	49.95±0.15	56.39±0.64	95.70±0.06	53.79±2.48	52.74±0.28	68.00±0.59	78.71±0.59	65.04
	MLPInit	49.96±0.20	58.05±0.56	96.02±0.04	77.77±1.93	55.61±0.17	69.53±0.50	78.48±0.64	69.34
	Improv.	↑ 0.02%	↑ 2.94%	↑ 0.33%	↑ 44.60%	↑ 5.45%	↑ 2.26%	↓ 0.30%	↑ 6.61%
GCN	Random	50.90±0.12	40.08±0.15	92.78±0.11	27.87±3.45	36.35±0.15	70.25±0.22	77.08±0.26	56.47
	MLPInit	51.16±0.20	40.83±0.27	91.40±0.20	80.37±2.61	39.70±0.11	70.35±0.34	76.85±0.34	64.38
	Improv.	↑ 0.51%	↑ 1.87%	↓ 1.49%	↑ 188.42%	↑ 9.22%	↑ 0.14%	↓ 0.29%	↑ 14.00%

MLPInit improves the prediction performance for node classification .

How Well does MLPInit Perform?

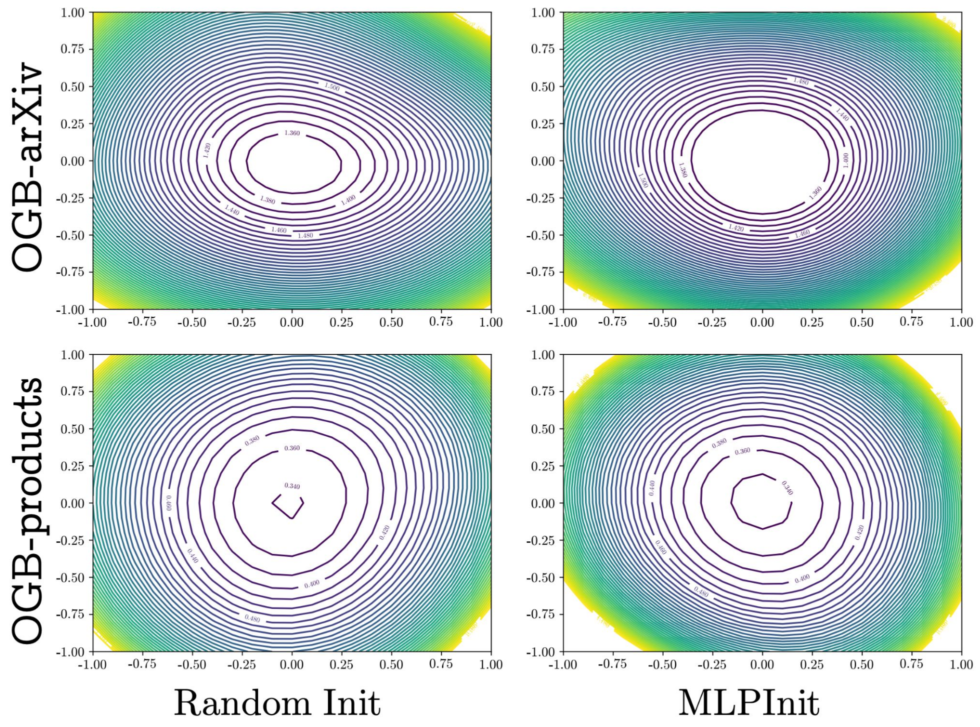


	Methods	AUC	AP	Hits@10	Hits@20	Hits@50	Hits@100
PubMed	MLP$_{random}$	94.76±0.30	94.28±0.36	14.68±2.60	24.01±3.04	40.02±2.75	54.85±2.03
	GNN$_{random}$	96.66±0.29	96.78±0.31	28.38±6.11	42.55±4.83	60.62±4.29	75.14±3.00
	GNN$_{mlpinit}$	97.31±0.19	97.53±0.21	37.58±7.52	51.83±7.62	70.57±3.12	81.42±1.52
	Improvement	↑ 0.68%	↑ 0.77%	↑ 32.43%	↑ 21.80%	↑ 16.42%	↑ 8.36%
DBLP	MLP$_{random}$	95.20±0.18	95.53±0.25	28.70±3.73	39.22±4.13	53.36±3.81	64.83±1.95
	GNN$_{random}$	96.29±0.20	96.64±0.23	36.55±4.08	43.13±2.85	59.98±2.43	71.57±1.00
	GNN$_{mlpinit}$	96.67±0.13	97.09±0.14	40.84±7.34	53.72±4.25	67.99±2.85	77.76±1.20
	Improvement	↑ 0.39%	↑ 0.47%	↑ 11.73%	↑ 24.57%	↑ 13.34%	↑ 8.65%
A-Photo	MLP$_{random}$	86.18±1.41	85.37±1.24	4.36±1.14	6.96±1.28	12.20±1.24	17.91±1.26
	GNN$_{random}$	92.07±2.14	91.52±2.08	9.63±1.58	12.82±1.72	20.90±1.90	29.08±2.53
	GNN$_{mlpinit}$	93.99±0.58	93.32±0.60	9.17±2.12	13.12±2.11	22.93±2.56	32.37±1.89
	Improvement	↑ 2.08%	↑ 1.97%	↓ 4.75%	↑ 2.28%	↑ 9.73%	↑ 11.32%
Physics	MLP$_{random}$	96.26±0.11	95.63±0.15	5.38±1.32	8.76±1.37	15.86±0.81	24.70±1.11
	GNN$_{random}$	95.84±0.13	95.38±0.15	6.62±1.00	10.39±1.04	18.55±1.60	26.88±1.95
	GNN$_{mlpinit}$	96.89±0.07	96.55±0.11	8.05±1.44	13.06±1.94	22.38±1.94	32.31±1.43
	Improvement	↑ 1.10%	↑ 1.22%	↑ 21.63%	↑ 25.76%	↑ 20.63%	↑ 20.20%
	Avg.	↑ 1.05%	↑ 1.10%	↑ 17.81%	↑ 20.97%	↑ 14.88%	↑ 10.46%

MLPInit improves the prediction performance for link prediction task.

Why Perform Well?

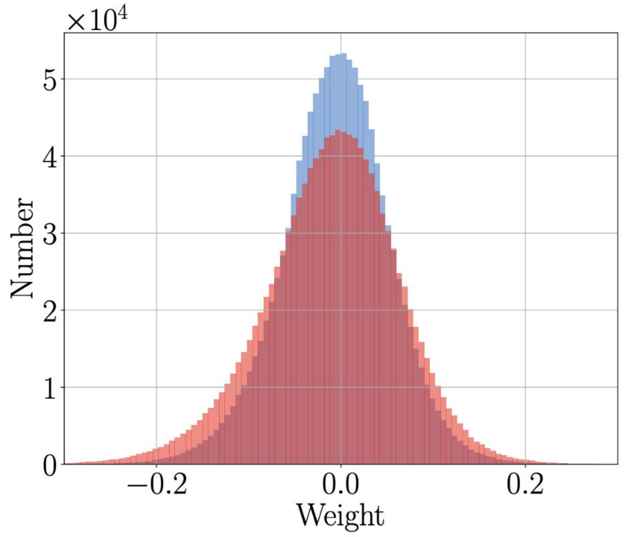
- Loss Landscape:



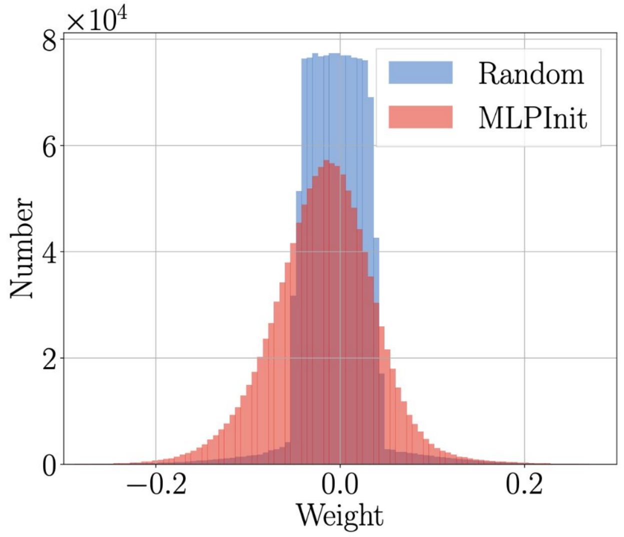
MLPInit helps find better local minima for GNNs.

Why Perform Well?

- Weight distribution



(a) `OGB-arXiv`



(b) `OGB-products`

MLPInit produces more high-magnitude weights, indicating better optimization of GNN.

Thank you!

Xiaotian Han
Texas A&M University
han@tamu.edu
https://ahxt.github.io

paper

code

MLPInit: Embarrassingly Simple GNN Training Acceleration with MLP Initialization
Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, Neil Shah
Paper: https://openreview.net/forum?id=P8YIphWNEGO
Code: https://github.com/snap-research/MLPInit-for-GNNs
Slides: https://ahxt.github.io/files/mlpinit_slides.pdf