MLPINIT: EMBARRASSINGLY SIMPLE GNN TRAINING ACCELERATION WITH MLP INITIALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Training graph neural networks (GNNs) on large graphs is complex and extremely time consuming. This is attributed to overheads caused by sparse matrix multiplication, which are sidestepped when training multi-layer perceptrons (MLPs) with only node features. MLPs, by ignoring graph context, are simple and faster for graph data, however they usually sacrifice prediction accuracy, limiting their applications for graph data. We observe that for most message passing-based GNNs, we can trivially derive an analog MLP (we call this a PeerMLP) whose weights can be made identical, making us curious about ***how do GNNs using weights from a fully trained PeerMLP perform?*** Surprisingly, we find that GNNs initialized with such weights significantly outperform their PeerMLPs for graph data, motivating us to use PeerMLP training as a precursor, initialization step to GNN training. To this end, we propose an embarrassingly simple, yet hugely effective initialization method for GNN training acceleration, called MLPInit. Our extensive experiments on multiple large-scale graph datasets with diverse GNN architectures validate that MLPInit can accelerate the training of GNNs (up to 33× speedup on `OGB-products`) and often improve prediction performance (e.g., up to 7.97% improvement for GraphSAGE across 7 datasets for node classification, and up to 17.81% improvement across 4 datasets for link prediction on metric Hits@10). Most importantly, MLPInit is extremely simple to implement and can be flexibly used as a plug-and-play initialization method for message passing-based GNNs.

1 INTRODUCTION

Graph Neural Networks (GNNs) (Zhang et al., 2018; Zhou et al., 2020; Wu et al., 2020) have attracted considerable attention from both academic and industrial researchers and have shown promising results on various practical tasks, e.g., recommendation (Fan et al., 2019; Sankar et al., 2021; Ying et al., 2018; Tang et al., 2022), knowledge graph analysis (Arora, 2020; Park et al., 2019; Wang et al., 2021), forecasting (Jiang & Luo, 2022; Tang et al., 2020) and chemistry analysis (Li et al., 2018b; You et al., 2018; De Cao & Kipf, 2018; Liu et al., 2022) However, training GNN on large-scale graphs is extremely time-consuming and costly in practice, thus spurring considerable work dedicated to scaling up the training of GNNs, even necessitating new massive graph learning libraries (Zhang et al., 2020; Ferludin et al., 2022) for large-scale graphs.

Recently, several approaches for more efficient GNNs training have been proposed, including novel architecture design (Wu et al., 2019; You et al., 2020d; Li et al., 2021), data reuse and partitioning paradigms (Wan et al., 2022; Fey et al., 2021; Yu et al., 2022) and graph sparsification (Cai et al., 2020; Jin et al., 2021b). However, these kinds of methods often sacrifice prediction accuracy and increase modeling complexity, while meriting sometimes significant additional engineering efforts.

MLPs are used to accelerate GNNs (Zhang et al., 2021; Frasca et al., 2020; Hu et al., 2021) by decoupling GNNs to node features learning and graph structure learning. Our work also leverages MLPs, but adopts a distinct perspective. Notably, we observe that the weight space of MLPs and GNNs can be identical, which enable us to transfer weights between MLP and GNN models. With that MLPs train faster than GNNs in mind, this observation inspired us to raise the following question:

Can we train GNNs more efficiently by leveraging the weights of converged MLPs?

To answer this question, we first pioneer a thorough investigation to reveal the relationship between the MLPs and GNNs in terms of trainable weight space. For ease of presentation, we define the

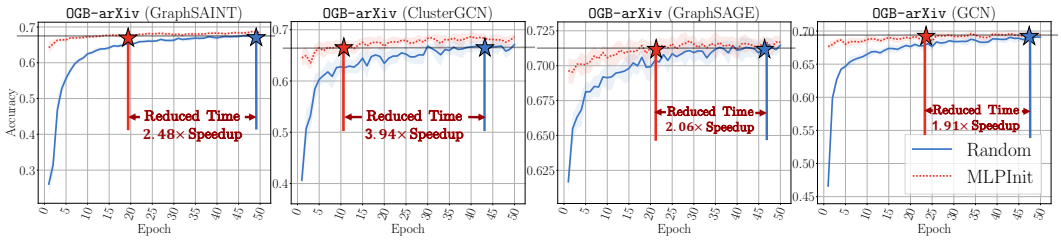


Figure 1: The training speed comparison of the GNNs with Random initialization and MLPInit. ★ indicates the best performance that GNNs with random initialization can achieve. ★ indicates the comparable performance of the GNN with MLPInit. Reduced Time indicates the training time reduced by our proposed MLPInit compared to random initialization. This experimental result shows that MLPInit is able to accelerate the training of GNNs significantly.

PeerMLP of a GNN[1] so that GNN and its PeerMLP share the same weights [2]. We find that interestingly, *GNNs can be optimized by training the weights of their PeerMLP*. Based on this observation, we adopt weights of converged PeerMLP as the weights of corresponding GNNs and find that these GNNs perform even better that converged PeerMLP on node classification tasks (results in Table 2).

Motivated by this, we propose an embarrassingly simple, yet remarkably effective method to accelerate GNNs training by initializing GNN with the weights of its converged PeerMLP. Specifically, to train a target GNN, we first train its PeerMLP and then initialize the GNN with the optimal weights of converged PeerMLP. We present the experimental results in Figure 1 to show the training speed comparison of GNNs with random initialization and with MLPInit. In Figure 1, Reduced Time shows the training time reduced by our proposed MLPInit compared to random initialized GNN, while achieving the same test performance. This experimental result shows that MLPInit is able the accelerate the training of GNNs significantly: for example, we speed up the training of commonly used GraphSAGE, GraphSAINT, ClusterGCN, GCN by $2.48\times$, $3.94\times$, $2.06\times$, $1.91\times$ on `OGB-arXiv` dataset, indicating the superiority of our method in GNNs training acceleration. And moreover, we speed up the training of GraphSAGE more than $14\times$ on `OGB-products` dataset.

We highlight **our contributions** as follows:

- We pioneer a thorough investigation to reveal the relationship between MLPs and GNNs in terms of the trainable weight space through the following observations: *(i)* GNNs and MLPs have the same weight space. *(ii)* GNNs can be optimized by training the weights of their PeerMLPs. *(iii)* GNN with weights from its converged PeerMLP surprisingly performs better than the performance of its converged PeerMLP on node classification tasks.
- Based on the above observations, we proposed a embarrassingly simple yet surprisingly effective initialization method to accelerate the GNNs training. Our method, called MLPInit, initializes the weights of GNNs with the weight of its converged PeerMLP. After initialization, we observe that GNN training takes less than half epochs to converge than those with random initialization. Thus, MLPInit is able to accelerate the training of GNNs since the training MLPs is cheaper and faster than training GNNs.
- Comprehensive experimental results on multiple large-scale graphs with diverse GNNs validate that MLPInit is able to accelerate the training of GNNs (up to $33\times$ speedup on `OGB-products`) while often improving the model performance [3] (e.g., 7.97% improvement for node classification on GraphSAGE and 17.81% improvement for link prediction on Hits@10).
- MLPInit is extremely easy to implement and has virtually negligible computational overhead compared to the conventional GNN training schemes. In addition, it is orthogonal to other GNN acceleration methods, such as weight quantization and graph coarsening, further increasing headroom for GNN training acceleration in practice.

[1] The formal definition of PeerMLP is in Section 3.

[2] By *share the same weight*, we mean that the trainable weights of GNN and its PeerMLP are the same in terms of size, dimension and values.

[3] By *performance*, we refer to the model prediction quality metric of the downstream task on the corresponding test data throughout the discussion.

2 PRELIMINARIES

Notations. We denote an attributed graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$, where $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \in \mathbb{R}^{N \times D}$ is the node feature matrix and $\mathbf{A} = \{0,1\}^{N \times N}$ is the binary adjacency matrix. N is the number of nodes and D is the dimension of node feature. For the node classification task, we denote the prediction targets denoted by $\mathbf{Y} \in \{0, 1, \ldots, C-1\}^N$, where C is the number of classes. We denote a GNN model as $f_{gnn}(\mathbf{X}, \mathbf{A}; w_{gnn})$, and an MLP as $f_{mlp}(\mathbf{X}; w_{mlp})$, where w_{gnn} and w_{mlp} denote the trainable weights in the GNN and MLP, respectively. Moreover, w^*_{gnn} and w^*_{mlp} denote the fixed weights of optimal (or converged) GNN and MLP, respectively.

Graph Neural Networks. Although various forms of graph neural networks (GNNs) exist, our work refers to the conventional message passing flavor (Gilmer et al., 2017). These models work by learning a node's representation by aggregating information from the node's neighbors recursively. One simple yet widely popular GNN instantiation is the graph convolutional network (GCN), whose multi-layer from can be written concisely: the representation vectors of all nodes at the l-th layer is $\mathbf{H}^l = \sigma(\mathbf{A}\mathbf{W}^l\mathbf{H}^{l-1})$, where $\sigma(\cdot)$ denotes activation function, $\mathbf{W}^l$ is the trainable weights of the l-th layer, and $\mathbf{H}^{l-1}$ is the node representations output by the previous layer. Denoting the output of the last layer of GNN by $\mathbf{H}$, for a node classification task, the prediction of node label is $\hat{\mathbf{Y}} = \mathrm{softmax}(\mathbf{H})$. For a link prediction task, one can predict the edge probabilities with any suitable decoder, e.g., commonly used inner-product decoder as $\hat{\mathbf{A}} = \mathrm{sigmoid}(\mathbf{H} \cdot \mathbf{H}^T)$ (Kipf & Welling, 2016b).

3 MOTIVATING ANALYSES

In this section, we reveal that MLPs and GNNs share the same weight space, which facilitates the transferability of weights between the two architectures. Through this section, we use GCN (Kipf & Welling, 2016a) as a prototypical example for GNNs for notational simplicity, but we note that our discussion is generalizable to other message passing GNN architectures.

Motivation 1: GNNs share the same weight space with MLPs. To show the weight space of GNNs and MLPs, we present the mathematical expression of one layer of MLP and GCN (Kipf & Welling, 2016a) as follows:

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

where $\mathbf{W}^l_{gnn}$ and $\mathbf{W}^l_{mlp}$ are the trainable weights of l-th layer of MLP and GCN, respectively. If we set the hidden layer dimensions of GNNs and MLPs to be the same, then $\mathbf{W}^l_{mlp}$ and $\mathbf{W}^l_{gnn}$ will naturally have the same size. Thus, although the GNN and MLP are different models, their weight spaces can be identical. Moreover, for any GNN model, we can trivially derive a corresponding MLP whose weight space can be made identical. For brevity, and when context of a GNN model is made clear, we can write such an MLP which shares the same weight space as a PeerMLP, i.e., their trainable weights can be transferred to each other.

Motivation 2: MLPs train faster than GNNs. GNNs train slower than MLPs, owing to their non-trivial relational data dependency. We empirically validate that training MLPs is much faster than training GNNs in Table 1. Specifically, this is because MLPs do not involve the sparse matrix multiplication for neighbor aggregation. A GNN layer (here we consider a simple GCN layer, as defined in Equation (1)) can be broken down into two operations: feature transformation ($\mathbf{Z} = \mathbf{W}^l\mathbf{H}^{l-1}$) and neighbor aggregation ($\mathbf{H}^l = \mathbf{A}\mathbf{Z}$) (Ma et al., 2021). The neighbor aggregation and feature transformation are typically a sparse and dense matrix multiplications, respectively. Table 1 shows the time usage for these different operations on several real-world graphs. As expected, neighbor aggregation in GNNs consumes the large majority of computation time. For example, on the `Yelp` dataset, the neighbor aggregation operation induces a 3199× time overhead.

Given that the weights of GNNs and their PeerMLP can be transferred to each other, but the PeerMLP can be trained much faster, we raise the following questions:

1. ***What will happen if we directly adopt the weights of a converged PeerMLP to GNN?***
2. ***To what extent can PeerMLP speed up GNN training and improve GNN performance?***

In this paper, we try to answer these questions with comprehensive empirical analysis.

Table 1: Comparison of the running time of different operations in GNNs. We compute the time used for forward and backward of two operation. The time unit is milliseconds (*ms*).

Operation	OGB-arXiv			Flickr			Yelp		
#Nodes	169343			89250			716847		
#Edges	1166243			899756			13954819		
	Forward	Backward	Total	Forward	Backward	Total	Forward	Backward	Total
$Z = WX$	0.32	1.09	1.42	0.28	0.97	1.26	1.58	4.41	5.99
$H = AZ$	1.09	1028.08	1029.17	1.01	836.95	837.97	9.74	19157.17	19166.90
			724×			665×			3199×

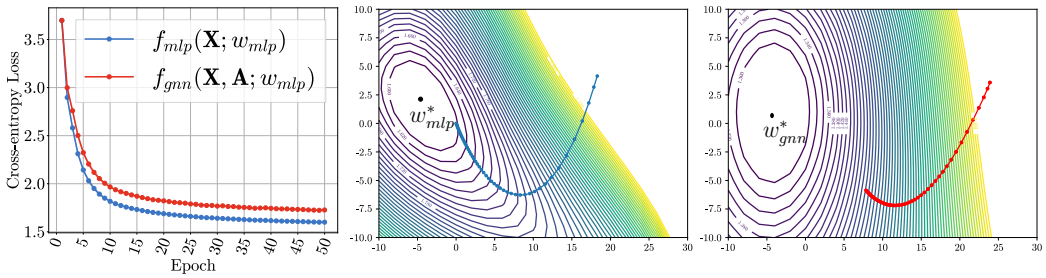


Figure 2: The relation of GNN and MLP during the training of PeerMLP. **Left**: Cross-Entropy Loss of $f_{gnn}(\mathbf{X}, \mathbf{A}; w_{mlp})$ (GNN) and $f_{mlp}(\mathbf{X}; w_{mlp})$ (PeerMLP) over training epochs of PeerMLP. In this experiment, GNN and PeerMLP share the same weight w_{mlp}, which are trained by the PeerMLP. **Middle**: training trajectory of PeerMLP on its own loss landscape. **Right**: training trajectory of GNN with weights from PeerMLP on GNN's loss landcape. The figures show that training loss of GNN with weights trained from MLP will decrease. The details are presented in Appendix C.2.

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

To answer this question, we conducted comprehensive preliminary experiments to investigate weight transferability between MLPs and GNNs. We made the following interesting and inspiring findings:

Observation 1: The training loss of GNN will decrease by optimizing the weights of its PeerMLP. We conducted a verification experiment to investigate the loss changes of the GNNs with the weights trained from its PeerMLP and present the results in Figure 2. In this experiment, we have two models, a GNN and its corresponding PeerMLP, who share the same weights w_{mlp} That is, the PeerMLP is $f_{mlp}(\mathbf{X}; w_{mlp})$ and the GNN is $f_{gnn}(\mathbf{X}, \mathbf{A}; w_{mlp})$. We optimize the weights w_{mlp} by training the PeerMLP, and the loss curve of $f_{mlp}(\mathbf{X}; w_{mlp})$ is the blue line in the left figure in Figure 2. We also compute the loss of GNN $f_{gnn}(\mathbf{X}, \mathbf{A}; w_{mlp})$ with the weights from PeerMLP. The loss curve of $f_{gnn}(\mathbf{X}, \mathbf{A}; w_{mlp})$ is shown in the red line. Figure 2 shows the surprising phenomenon that the training loss of GNN with weights trained from PeerMLP decreases consistently. Impressively, these weights (w_{mlp}) were derived without employing neighbor aggregation in training.

Observation 2: Converged weights from PeerMLP provide a good GNN initialization. As PeerMLP and GNN have the same weight spaces, a natural follow-up question is whether GNN can directly adopt the weights of the converged PeerMLP and perform well.We next aim to understand this question empirically. Specifically, we first trained a PeerMLP for a target GNN and obtain the optimal weights w^*_{mlp}. Next, we run inference on test data using a GNN with w^*_{mlp} of PeerMLP, i.e., applying $f_{gnn}(\mathbf{X}, \mathbf{A}; w^*_{mlp})$. Table 2 shows the results of $f_{mlp}(\mathbf{X}; w^*_{mlp})$ and $f_{gnn}(\mathbf{X}, \mathbf{A}; w^*_{mlp})$. We can observe that the GNNs with the optimal weights of PeerMLP consistently outperform PeerMLP, indicating that the weights from converged PeerMLP can serve as good enough initialization of the weights of GNNs.

Table 2: The performance of GNNs and its PeerMLP with the weights of a converged PeerMLP on test data.

	Methods	PeerMLP	GNN	Improv.
OGB-arXiv	GraphSAGE	56.04±0.27	62.87±0.95	↑ 12.18%
	GraphSAINT	53.88±0.41	63.26±0.71	↑ 17.41%
	ClusterGCN	54.47±0.41	60.81±1.30	↑ 11.63%
	GCN	56.31±0.21	56.28±0.89	↓ 0.04%
OGB-products	GraphSAGE	63.43±0.14	74.32±1.04	↑ 17.16%
	GraphSAINT	57.29±0.32	69.00±1.54	↑ 20.44%
	ClusterGCN	59.53±0.46	71.74±0.70	↑ 20.51%
	GCN	62.63±0.15	71.11±0.10	↑ 13.55%

4.1 THE PROPOSED METHOD: MLPINIT

The above findings show that MLPs can help the training of GNNs. In this section, we formally present our method MLPInit, which is an embarrassingly simple, yet extremely effective approach to accelerate GNN training.

The basic idea of MLPInit is straightforward: we adopt the weights of a converged PeerMLP to initialize GNN, and subsequently fine-tune the GNN. Specifically, for a target GNN ($f_{gnn}(\mathbf{X}, \mathbf{A}; w_{gnn})$), we first construct a PeerMLP ($f_{mlp}(\mathbf{X}, \mathbf{A}; w_{mlp})$), with matching target weights. Next, we optimize the weight of the PeerMLP model by training the PeerMLP solely with the node features $\mathbf{X}$ for m epochs. Upon training the PeerMLP to convergence and obtaining the optimal weights (w^*_{mlp}), we initialize the GNN with w^*_{mlp} and then fine-tune the GNN with n epochs. We present PyTorch-style pseudo-code of MLPInit in node classification setting in Algorithm 1.

Training Acceleration. Since training of the PeerMLP is comparatively cheap, and the weights of the converged PeerMLP can provide a good initialization for the corresponding GNN, the end result is that we can significantly reduce the training time of the GNN. Assuming that the training of GNN from a random initialization needs N epochs to converge, and $N >> n$, the total training time can be largely reduced given that MLP training time is negligible compared to GNN training time. The experimental results in Table 3 shows that N is generally much larger than n.

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()
```

Ease of Implementation. MLPInit is extremely easy to implement as shown in Algorithm 1. First, we construct a MLP (PeerMLP), which has the same weights with the target GNN. Next, we use the node features $\mathbf{X}$ and node labels $\mathbf{Y}$ to train the PeerMLP to converge. Then, we adopt the weights of converged PeerMLP to the GNN, and fine-tune the GNN while additionally leveraging the adjacency $\mathbf{A}$. In addition, our method can also directly serve as the final, or deployed GNN model, in resource constrained settings: assuming $n = 0$, we can simply train the PeerMLP and adopt w^*_{mlp} directly. This reduces training cost further, while enabling us to serve a likely higher performance model in deployment or test settings, as Table 2 shows.

4.2 DISCUSSION

In the section, we discuss the relation between MLPInit and existing methods. Since we position MLPInit as an acceleration method involve MLP, we first compare it with MLP-based GNN Acceleration Methods. Additionally, we also compare it with GNN Pre-training methods.

Comparison to MLP-based GNN Acceleration Methods. Recently, several works aim simplify the GNN to MLP-based constructs during training or inference (Zhang et al., 2022; Wu et al., 2019; Frasca et al., 2020; Sun et al., 2021; Huang et al., 2020; Hu et al., 2021). Our method is proposed to accelerate the message passing based GNN for large-scale graphs. Thus, MLP-based GNN acceleration is a completely different line of work compared to ours since it removes the message passing in the GNNs and use MLP to model graph structure instead. Thus, MLP-based GNN acceleration methods are out of scope of the discussion in this work.

Comparison to GNN Pre-training Methods. Our proposed MLPInit are orthogonal to the GNN pre-training methods(You et al., 2020b; Zhu et al., 2020b; Veličković et al., 2018b; You et al., 2021; Qiu et al., 2020; Zhu et al., 2021; Hu et al., 2019). GNN pretraining typically leverages graph augmentation to pretrain weights of GNNs or obtain the node representation downstream task usage. Comparing with the pre-training methods, MLPInit has two main differences (or advantages) that significantly contribute to the speed up: (i) the training of PeerMLP does not involve using the graph structure data, while pre-training methods rely on it. (ii) Pre-training methods usually involves graph data augmentation (Zhao et al., 2022; Qiu et al., 2020), which requires additional training time.

Table 3: Speed improvement when MLPInit achieves comparable performance with random initialized GNN. The number reported is the training epochs needed. (—) means our method can not reach comparable performance. The epoch used by Random/MLPInit is denoted as ★/★ in Figure 1 and N/n in Section 4.1. The detailed speedup computation method are presented in Appendix C.3.

	Methods	Flickr	Yelp	Reddit	Reddit2	A-products	OGB-arXiv	OGB-products	Avg.
SAGE	Random(★, N)	45.6	44.7	36.0	48.0	48.9	46.7	43.0	44.7
	MLPInit (★, n)	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	39.8
	MLPInit	14.1	0.0	21.8	6.1	9.1	19.5	16.9	12.5
	Improv.	2.20×	—	1.86×	4.64×	5.49×	2.48×	2.66×	3.18×
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	39.1
	MLPInit	30.5	23.3	0.0	0.0	0.0	24.5	1.3	11.3
	Improv.	1.52×	1.91×	—	—	—	1.91×	33.69×	3.46×

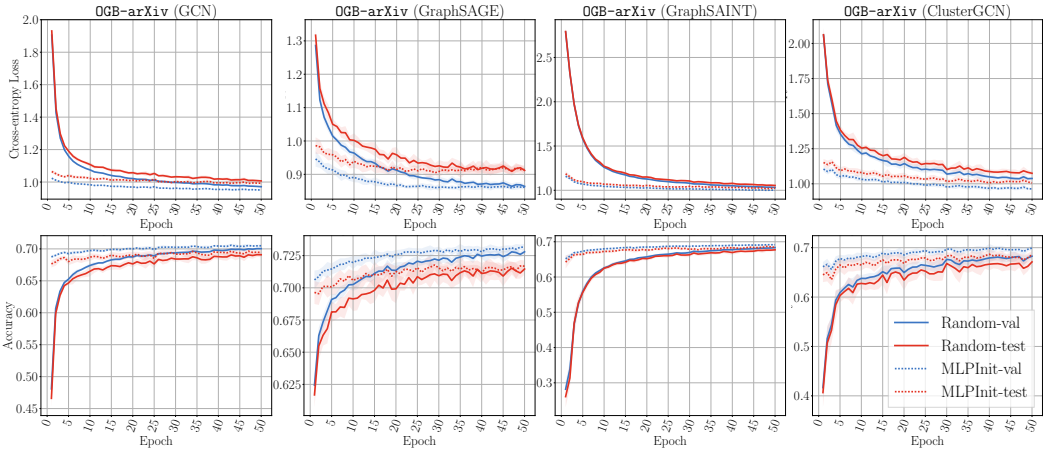


Figure 3: The training curves of different GNNs on OGB-arXiv. GNN with MLPInit generally obtain lower loss and higher accuracy that those with random initialization, and converge faster. The training curves are depicted based on ten runs. More experiment results are presented in Appendix A.

5 Experiments

In the next subsections, we conduct and discuss experimental results to understand MLPInit from the following aspects: **(i)** training speedup, **(ii)** performance improvements, **(iii)** hyperparameter sensitivity, **(iv)** robustness and loss landscape. For node classification, we consider Flickr, Yelp, Reddit, Reddit2, A-products, and two OGB datasets (Hu et al., 2020), OGB-arXiv and OGB-products as benchmark dataset. We adopt GCN (w/ mini-batch) (Kipf & Welling, 2016a), GraphSAGE (Hamilton et al., 2017), GraphSAINT(Zeng et al., 2019) and ClusterGCN (Chiang et al., 2019). The details of datasets and baselines are in Appendices B.1 and B.2, respectively. For link prediction task, we consider Cora, CiteSeer, PubMed, CoraFull, CS, Physics, A-Photo and A-Computers as our datasets. Our link prediction setup is using as GCN as encode which transform a graph to node representation $\mathbf{H}$ and then use an inner-product decoder $\hat{\mathbf{A}} = \mathrm{sigmoid}(\mathbf{H} \cdot \mathbf{H}^T)$ to predict the probability of the link existence, which is discussed in Section 2.

5.1 How much can MLPInit accelerate GNN training?

In this section, we compare the training of GNNs with random initialization and MLPInit. We compute training epochs needed by GNNs with random initialization to achieve the best test performance. We also compute the running epochs needed by GNNs with MLPInit to achieve comparable test performance. We present the comparison in Table 3. We also plotted the loss and accuracy curves of different GNNs on OGB-arXiv in Figure 3. We made the following major observations:

Table 4: Performance improvement when GNN with random initialization and with MLPInit achieve best test performance, respectively. Mean and standard deviation are calculated based on ten runs. The best test performance for two methods are independently selected based on validation data.

	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%

Table 5: The performance of link prediction task. The results are based on ten runs. The experiments on other datasets are presented in Table 6. More experiments are presented in Appendix A.1.

	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%

Observation 3: MLPInit can significantly reduce the training time of GNNs. In this experiment, we summarize the epochs needed by GNN with random initialization to obtain the best performance, and then we calculate the epochs needed by GNN with MLPInit to reach the comparable performance on par with the random initialized GNN. We present the time speedup of MLPInit in Table 3. Table 3 shows MLPInit speed up the training of GNNs by $2-5$ times generally and in some cases even more than 30 times. The consistent reduction of training epochs on different datasets demonstrates that MLPInit can generally speed up GNN training quite significantly.

5.2 How well does MLPInit perform on node classification and link prediction tasks?

In this section, we conducted experiments to show the superiority of the proposed method in terms of final, converged GNN model performance on node classification and link prediction tasks. The reported test performance of both random initialization and MLPInit are selected based on the validation data. We present the performance improvement of MLPInit compared to random initialization in Tables 4 and 5 for node classification and link prediction, respectively.

Observation 4: MLPInit improves the prediction performance for both node classification and link prediction task in most cases. Table 4 shows our proposed method gains 7.97%, 7.00%, 6.61%

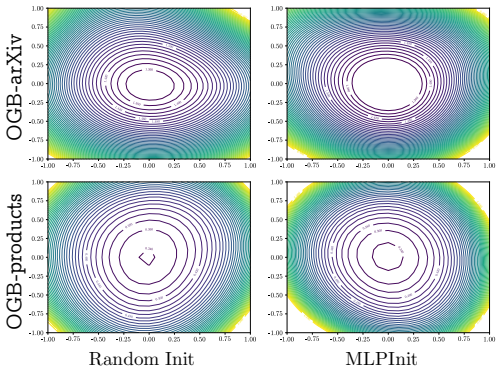


Figure 5: The loss landscape of GNN trained with random initialization (left) and MLPInit (right). The basin with converged minimum of MLPInit has larger area than that of random.

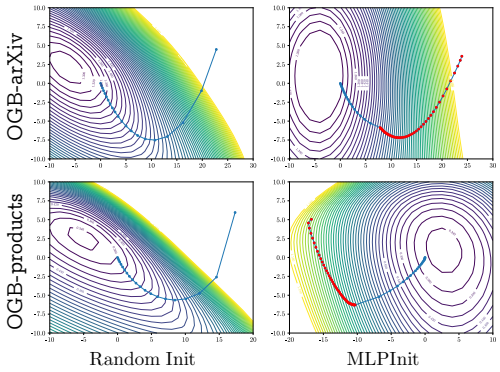


Figure 6: The training trajectory of the GNN with random initialization (left) and MLPInit (right). The first-phase training of GNNs can be taken over by lightweight MLPs.

and 14.00% improvements for GraphSAGE, GraphSAINT, ClusterGCN and GCN on average cross all the datasets for node classification task. The results in Table 5 and Table 6 show our proposed method gains 1.05%, 1.10%, 17.81%, 20.97%, 14.88%,10.46% on average cross various metrics for link prediction task.

5.3 Is MLPInit robust under different hyperparameters?

In practice, one of the most time-consuming parts of training large-scale GNNs is hyperparameter tuning (You et al., 2020a). Here, we perform experiments to investigate the sensitivity of MLPInit to various hyperparameters, including the architecture hyperparameters and training hyperparameters.

Observation 5: MLPInit makes GNNs less sensitive to hyperparameters and improves the overall performance across various hyperparameters. In this experiment, we trained PeerMLP and GNN with different "Training HP" (Learning rate, weight decay and batch size) and "Architecture HP" (i.e., layers, number of hidden neurons,) and we presented the learning curves of GNN with different hyperparameters in Figure 4. One can see from the results that GNNs trained with MLPInit have much smaller standard deviation than that with random initialization. Moreover, MLPInit consistently outperforms random initialization in task performance. This advantage allows our approach to save time in searching for architectural hyperparameters. In practice, different datasets require different model parameters. Using our proposed method, we can generally choose random hyperparameters and obtain reasonable and relatively stable performance owing to the PeerMLP's lower sensitivity to hyperparameters.

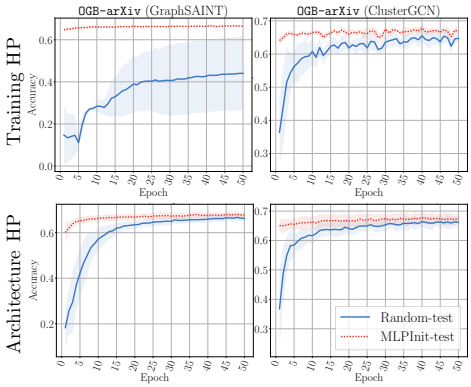


Figure 4: Training curves of GNNs with various hyperparameters. GNNs with MLPInit consistently outperforms random initialization and have smaller standard deviation.

5.4 Will MLPInit facilitate better convergence for GNNs?

In this experiment, we performed experiments to analyze the convergence of a fine-tuned GNN model. In other words, does this pre-training actually help find better local minima?

Observation 6: MLPInit actually helps find better local minima for GNNs. The geometry of loss surfaces (e.g., flatness) reflects the properties of the learned model, which provides various insights to assess the generalization ability (Fort et al., 2019; Huang et al., 2019; Wen et al., 2018) and robustness (Fort et al., 2019; Liu et al., 2020; Engstrom et al., 2019) of a neural network. In this experiment, we plot the loss landscape of GNN (GraphSAGE) with random weight initialization and MLPInit using the visualization tool introduced in Li et al. (2018a). The loss landscape are plotted

based on the training loss and with OGB-arXiv and OGB-products datasets. The loss landscapes are shown in Figure 5. For the fair comparison, two random direction of the loss landscape are the same and the lowest values of losses in the landscape are the same within one dataset, 1.360 for OGB-arXiv and 0.340 for OGB-products. From the loss landscape, we can see that the basin with converged minimum has larger area in terms of the same level of loss in each dataset's (row's) plots, indicating the loss landscape of the model trained with MLPInit are more flat than the model with random initialization. In summary, MLPInit helps find better local minima for GNNs.

Observation 7: MLPInit actually helps find better local minima for graph neural networks. To further understand the training procedure of MLPInit, we visualize the training trajectories along with loss landscapes through tools (Li et al., 2018a) in Figure 6. In this experiment, we use GraphSAGE on OGB-arXiv and OGB-products datasets, we first train a PeerMLP and then use the weights of the converged PeerMLP to initialize the GraphSAGE and fine-tune it. The we plot the training trajectories of PeerMLP and GraphSAGE on the loss landscape of GraphSAGE. The red line indicate the training trajectories of the training of PeerMLP and the blue line indicates the fine-tuning of GNNs. We can see that the end point of the training of MLP (red line) is close to the minima area in the loss landscape. The training trajectory clearly shows the reason why MLPInit works, i.e., the first-phase training of GNNs can be taken over by lightweight MLPs.

6 Related Work

In this section, we present several lines of related work.

Massage passing-based GNNs. Graph neural networks typically follows the massage passing mechanism, which aggregates the information form node's neighbors and learns node representation for downstream tasks. Following with the pioneer work GCN (Kipf & Welling, 2016a), lots of work (Veličković et al., 2018a; Xu et al., 2018; Balcilar et al., 2021; Thorpe et al., 2021; Brody et al., 2021; Tailor et al., 2021) are proposed to improve or understand massage passing-based GNNs.

Efficient GNNs. In order to scale GNNs to large graphs, the efficiency of GNNs attracted lots of attentions recently. subgraph sampling techniques (Hamilton et al., 2017; Chiang et al., 2019; Zeng et al., 2019) have been proposed for efficient mini-batch training for large-scale graphs. All of this kind of methods follow the message passing mechanism for efficient GNN training. There is another line of work on scalable GNNs is to use MLP to simplify GNNs (Zhang et al., 2022; Wu et al., 2019; Frasca et al., 2020; Sun et al., 2021; Huang et al., 2020; Hu et al., 2021), which decouples the feature aggregation and transformation operations to avoid feature aggregation operation. We compared our work with this line of work in Section 4.2. There are also other works such as weights quantization and graph sa and graph sparsification (Cai et al., 2020). However, these kinds of methods often sacrifice prediction accuracy and increase modeling complexity, while meriting sometimes significant additional engineering efforts.

Pre-training GNNs. The recent GNN pretraining methods mainly adopt contrastive learning (Hassani & Khasahmadi, 2020; Qiu et al., 2020; Zhu et al., 2020b; 2021; You et al., 2021; 2020c; Jin et al., 2021a; Hu et al., 2019; Zhu et al., 2020a). GNN pretraining typically leverages graph augmentation to pretrain weights of GNNs or obtain the node representation downstream task usage or generative objectives. For example, Grace Zhu et al. 2020b maximizes the agreement between two view of one graph. This line of work usually involves graph data augmentation, which requires additional training time and engineering effort.

7 Conclusion

This work presents a simple yet effective initialization method, MLPInit, to accelerate the training of GNNs, which adopts the weights form their converged PeerMLP initialize GNN and then fine-tune GNNs. With comprehensive experimental evidence, we demonstrate that superiority of our proposed method on training speedup (up to 33× speedup on OGB-products), downstream task performance improvements(up to 7.97% performance improvement for GraphSAGE) and robustness improvements (larger minimal area in loss landscape) on the resulting GNNs. Notably, our proposed method is easy to implement and employ in real applications to speed up the training of GNNs.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our experiments and benefit the research community, we provide the sample source code for MLPInit on one dataset along with the submission in the supplementary materials. We will also open-source our implementation after we clean up the codes. The hyper-parameters and other variables required to reproduce our experiments are described in Appendix C.

REFERENCES

Siddhant Arora. A survey on graph neural networks for knowledge graph completion. *arXiv preprint arXiv:2007.12374*, 2020.

Muhammet Balcilar, Renton Guillaume, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *ICLR*, 2018.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2021.

Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *International Conference on Learning Representations*, 2020.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. In *International conference on machine learning*, pp. 1802–1811. PMLR, 2019.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pp. 417–426, 2019.

Oleksandr Ferludin, Arno Eigenwillig, Martin Blais, Dustin Zelle, Jan Pfeifer, Alvaro Sanchez-Gonzalez, Sibon Li, Sami Abu-El-Haija, Peter Battaglia, Neslihan Bulut, et al. Tf-gnn: Graph neural networks in tensorflow. *arXiv preprint arXiv:2207.03522*, 2022.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning*, pp. 3294–3304. PMLR, 2021.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pp. 1024–1034, 2017.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pp. 4116–4126. PMLR, 2020.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Yang Hu, Haoxuan You, Zhecan Wang, Zhicheng Wang, Erjin Zhou, and Yue Gao. Graph-mlp: node classification without message passing in graph. *arXiv preprint arXiv:2106.04051*, 2021.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

W Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K Terry, Furong Huang, and Tom Goldstein. Understanding generalization through visualizations. *arXiv preprint arXiv:1906.03291*, 2019.

Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, pp. 117921, 2022.

Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. In *WWW Workshop*, 2021a.

Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. In *International Conference on Learning Representations*, 2021b.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In *International conference on machine learning*, pp. 6437–6449. PMLR, 2021.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018a.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018b.

Chen Liu, Mathieu Salzmann, Tao Lin, Ryota Tomioka, and Sabine Süsstrunk. On the loss landscape of adversarial training: Identifying challenges and how to overcome them. *Advances in Neural Information Processing Systems*, 33:21476–21487, 2020.

Gang Liu, Tong Zhao, Jiaxin Xu, Tengfei Luo, and Meng Jiang. Graph rationalization with environment-based augmentations. In *Proceedings of the 28th ACM SIGKDD international conference on knowledge discovery & data mining*, 2022.

Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 1202–1211, 2021.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52, 2015.

Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 596–606, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1150–1160, 2020.

Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*, pp. 2535–2546, 2021.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Chuxiong Sun, Hongming Gu, and Jie Hu. Scalable and adaptive graph neural networks with self-label-enhanced training. *arXiv preprint arXiv:2104.09376*, 2021.

Shyam A Tailor, Felix Opolka, Pietro Lio, and Nicholas Donald Lane. Do we need anisotropic graph neural networks? In *International Conference on Learning Representations*, 2021.

Xianfeng Tang, Yozen Liu, Neil Shah, Xiaolin Shi, Prasenjit Mitra, and Suhang Wang. Knowing your fate: Friendship, action and temporal explanations for user engagement prediction on social apps. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2269–2279, 2020.

Xianfeng Tang, Yozen Liu, Xinran He, Suhang Wang, and Neil Shah. Friend story ranking with edge-contextual local graph convolutions. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 1007–1015, 2022.

Matthew Thorpe, Tan Minh Nguyen, Hedi Xia, Thomas Strohmer, Andrea Bertozzi, Stanley Osher, and Bao Wang. Grand++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2021.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018a.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2018b.

Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. *Proceedings of Machine Learning and Systems*, 4:673–693, 2022.

Shen Wang, Xiaokai Wei, Cicero Nogueira Nogueira dos Santos, Zhiguo Wang, Ramesh Nallapati, Andrew Arnold, Bing Xiang, Philip S Yu, and Isabel F Cruz. Mixed-curvature multi-relational graph neural network for knowledge graph completion. In *Proceedings of the Web Conference 2021*, pp. 1761–1771, 2021.

Wei Wen, Yandan Wang, Feng Yan, Cong Xu, Chunpeng Wu, Yiran Chen, and Hai Li. Smoothout: Smoothing out sharp minima to improve generalization in deep learning. *arXiv preprint arXiv:1805.07898*, 2018.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pp. 6861–6871. PMLR, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd ICML*, 2016.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pp. 5708–5717. PMLR, 2018.

Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020a.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020b.

Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. When does self-supervision help graph convolutional networks? In *ICML*, pp. 10871–10880. PMLR, 2020c.

Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2127–2135, 2020d.

Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *International Conference on Machine Learning*, pp. 12121–12132. PMLR, 2021.

Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. Graphfm: Improving large-scale gnn training via feature momentum. In *International Conference on Machine Learning*, pp. 25684–25701. PMLR, 2022.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-saint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2019.

Dalong Zhang, Xin Huang, Ziqi Liu, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Zhiqiang Zhang, Lin Wang, Jun Zhou, Yang Shuang, et al. Agl: a scalable system for industrial-purpose graph machine learning. *arXiv preprint arXiv:2003.02454*, 2020.

Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. In *International Conference on Learning Representations*, 2021.

Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: Algorithms, applications and open challenges. In *International Conference on Computational Social Networks*, pp. 79–91. Springer, 2018.

Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. Graph attention multi-layer perceptron. *arXiv preprint arXiv:2206.04355*, 2022.

Tong Zhao, Gang Liu, Stephan Günnemann, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Qikui Zhu, Bo Du, and Pingkun Yan. Self-supervised training of graph convolutional networks. *arXiv:2006.02380*, 2020a.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020b.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pp. 2069–2080, 2021.

A ADDITIONAL EXPERIMENTS

In this appendix, we present additional experiments to show the superiority of our proposed method, MLPInit, including additional result for link prediction and training curves. We also present more analysis on the weights distribution changes of MLPInit.

A.1 ADDITIONAL EXPERIMENTAL LINK PREDICTION

Here, we present the additional experiment results on link prediction task in Table 6, which is similar to our results in Table 5, but on more datasets. In general, we observe that GNN with MLPInit outperforms that with random initialization on link prediction task on these additional datasets. MLPInit does not gain better performance on `A-products` dataset. We conjecture that the reason for this is that node features may contain less task-relevant information on `A-products`.

Table 6: The performance of link prediction task. The results are based on ten runs.

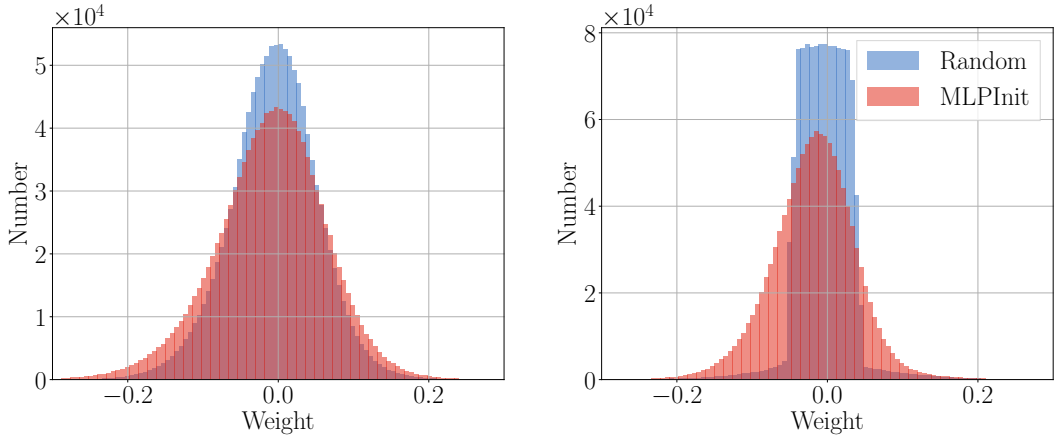
	Methods	AUC	AP	Hits@10	Hits@20	Hits@50	Hits@100
Cora	$\text{MLP}_{\text{random}}$	$91.87_{\pm 1.25}$	$92.16_{\pm 0.99}$	$46.98_{\pm 4.33}$	$58.05_{\pm 6.44}$	$76.03_{\pm 4.17}$	$86.28_{\pm 3.05}$
	$\text{GNN}_{\text{random}}$	$91.80_{\pm 1.39}$	$92.68_{\pm 1.28}$	$51.48_{\pm 7.57}$	$63.04_{\pm 6.12}$	$78.12_{\pm 4.06}$	$86.07_{\pm 2.77}$
	$\text{GNN}_{\text{mlpinit}}$	$92.93_{\pm 0.88}$	$93.36_{\pm 0.86}$	$53.72_{\pm 7.43}$	$68.65_{\pm 4.22}$	$80.30_{\pm 2.35}$	$89.37_{\pm 2.21}$
	Improvement	$\uparrow$ 1.23%	$\uparrow$ 0.74%	$\uparrow$ 4.35%	$\uparrow$ 8.91%	$\uparrow$ 2.79%	$\uparrow$ 3.84%
CiteSeer	$\text{MLP}_{\text{random}}$	$90.10_{\pm 0.99}$	$90.65_{\pm 0.98}$	$46.64_{\pm 5.11}$	$57.21_{\pm 4.84}$	$71.45_{\pm 3.15}$	$81.93_{\pm 2.24}$
	$\text{GNN}_{\text{random}}$	$89.86_{\pm 1.18}$	$90.88_{\pm 0.99}$	$49.98_{\pm 5.54}$	$58.02_{\pm 5.00}$	$71.65_{\pm 3.50}$	$82.11_{\pm 2.84}$
	$\text{GNN}_{\text{mlpinit}}$	$90.51_{\pm 1.07}$	$90.96_{\pm 0.80}$	$50.02_{\pm 3.09}$	$58.77_{\pm 4.48}$	$71.78_{\pm 4.12}$	$82.66_{\pm 2.53}$
	Improvement	$\uparrow$ 0.73%	$\uparrow$ 0.08%	$\uparrow$ 0.09%	$\uparrow$ 1.29%	$\uparrow$ 0.18%	$\uparrow$ 0.67%
CS	$\text{MLP}_{\text{random}}$	$96.29_{\pm 0.12}$	$95.79_{\pm 0.13}$	$13.36_{\pm 1.49}$	$19.67_{\pm 2.21}$	$33.46_{\pm 2.17}$	$46.82_{\pm 1.91}$
	$\text{GNN}_{\text{random}}$	$96.11_{\pm 0.08}$	$95.75_{\pm 0.10}$	$14.27_{\pm 2.77}$	$22.57_{\pm 2.52}$	$35.40_{\pm 2.01}$	$48.21_{\pm 2.00}$
	$\text{GNN}_{\text{mlpinit}}$	$96.72_{\pm 0.10}$	$96.49_{\pm 0.14}$	$16.96_{\pm 3.37}$	$25.44_{\pm 3.00}$	$40.69_{\pm 2.99}$	$53.78_{\pm 2.00}$
	Improvement	$\uparrow$ 0.63%	$\uparrow$ 0.77%	$\uparrow$ 18.81%	$\uparrow$ 12.70%	$\uparrow$ 14.96%	$\uparrow$ 11.55%
A-Computers	$\text{MLP}_{\text{random}}$	$81.85_{\pm 0.79}$	$82.41_{\pm 0.69}$	$2.10_{\pm 0.48}$	$4.13_{\pm 0.86}$	$7.83_{\pm 0.95}$	$12.18_{\pm 1.01}$
	$\text{GNN}_{\text{random}}$	$91.78_{\pm 0.48}$	$91.94_{\pm 0.42}$	$7.60_{\pm 1.47}$	$11.10_{\pm 1.74}$	$18.64_{\pm 1.94}$	$25.42_{\pm 2.15}$
	$\text{GNN}_{\text{mlpinit}}$	$90.76_{\pm 1.61}$	$91.06_{\pm 1.47}$	$6.76_{\pm 3.27}$	$11.11_{\pm 1.82}$	$17.40_{\pm 2.58}$	$24.59_{\pm 2.56}$
	Improvement	$\downarrow$ 1.11%	$\downarrow$ 0.96%	$\downarrow$ 11.04%	$\uparrow$ 0.16%	$\downarrow$ 6.65%	$\downarrow$ 3.26%
CoraFull	$\text{MLP}_{\text{random}}$	$95.72_{\pm 0.18}$	$95.55_{\pm 0.23}$	$19.38_{\pm 4.71}$	$27.83_{\pm 3.27}$	$42.98_{\pm 2.01}$	$57.20_{\pm 1.27}$
	$\text{GNN}_{\text{random}}$	$95.87_{\pm 0.36}$	$95.77_{\pm 0.42}$	$21.33_{\pm 4.77}$	$30.57_{\pm 3.49}$	$45.08_{\pm 3.46}$	$59.58_{\pm 2.53}$
	$\text{GNN}_{\text{mlpinit}}$	$96.71_{\pm 0.16}$	$96.73_{\pm 0.22}$	$25.78_{\pm 4.92}$	$36.68_{\pm 5.36}$	$53.81_{\pm 2.34}$	$66.73_{\pm 1.96}$
	Improvement	$\uparrow$ 0.87%	$\uparrow$ 1.01%	$\uparrow$ 20.87%	$\uparrow$ 19.98%	$\uparrow$ 19.37%	$\uparrow$ 12.01%

A.2 WEIGHT DIFFERENCE OF GNNS WITH RANDOM INITIALIZATION AND MLPINIT

Prior work (Li et al., 2018a) suggests that "small weights still appear more sensitive to perturbations, and produce sharper looking minimizers." To this end, we explore the distribution of weights of GNNs with both random initialization and MLPInit, and present the results in Figure 7. We can observe that with the same number of training epochs, the weights of GraphSAGE with MLPInit produce more high-magnitude (both positive and negative) weights, indicating the MLPInit can help the optimization of GNN. This difference stems from a straightforward reason: MLPInit provides a good initialization for GNNs since the weight are trained by the PeerMLP before (also aligning with our observations in Section 5.4).

A.3 ADDITIONAL HYPERPARAMETER SENSITIVITY

In this appendix, we present the additional results to explore more on the sensitivity to the various hyperparameters. The results are the full version of Figure 8.



(a) OGB-arXiv (b) OGB-products

Figure 7: Weight histograms of GraphSAGE model weights with random initialization (blue) and MLPInit (red) on OGB-arXiv (left) and OGB-products (right). The results are plotted based on 20 epochs after training GraphSAGE with zero weight decay. Note that MLPInit produces more weights of higher magnitude.

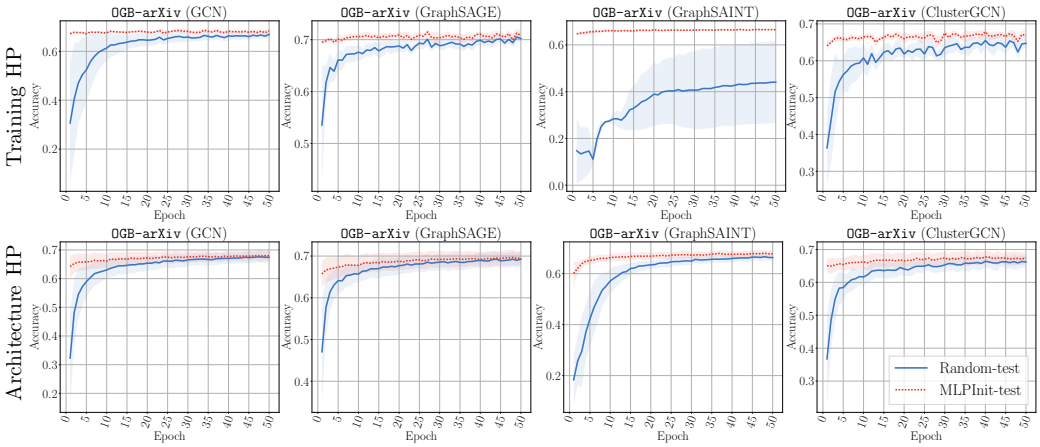


Figure 8: The training curves of the GNNs with different hyperparameters on OGB-arXiv dataset. The training curves of GNN with MLPInit generally obtain lower loss and higher accuracy that those with random initialization, and converge faster. The training curves are depicted based on ten runs.

A.4 Additional Training Curves

In this appendix, we present the additional training curves of other dataset in Figure 9 and Figure 10, which are additional experimental results of Figure 3. The results comprehensively show the training curves on various datasets. As we can see from Figure 9 and Figure 10 that MLPInit consistently outperform the random initialization and is able to accelerate the training of GNNs..

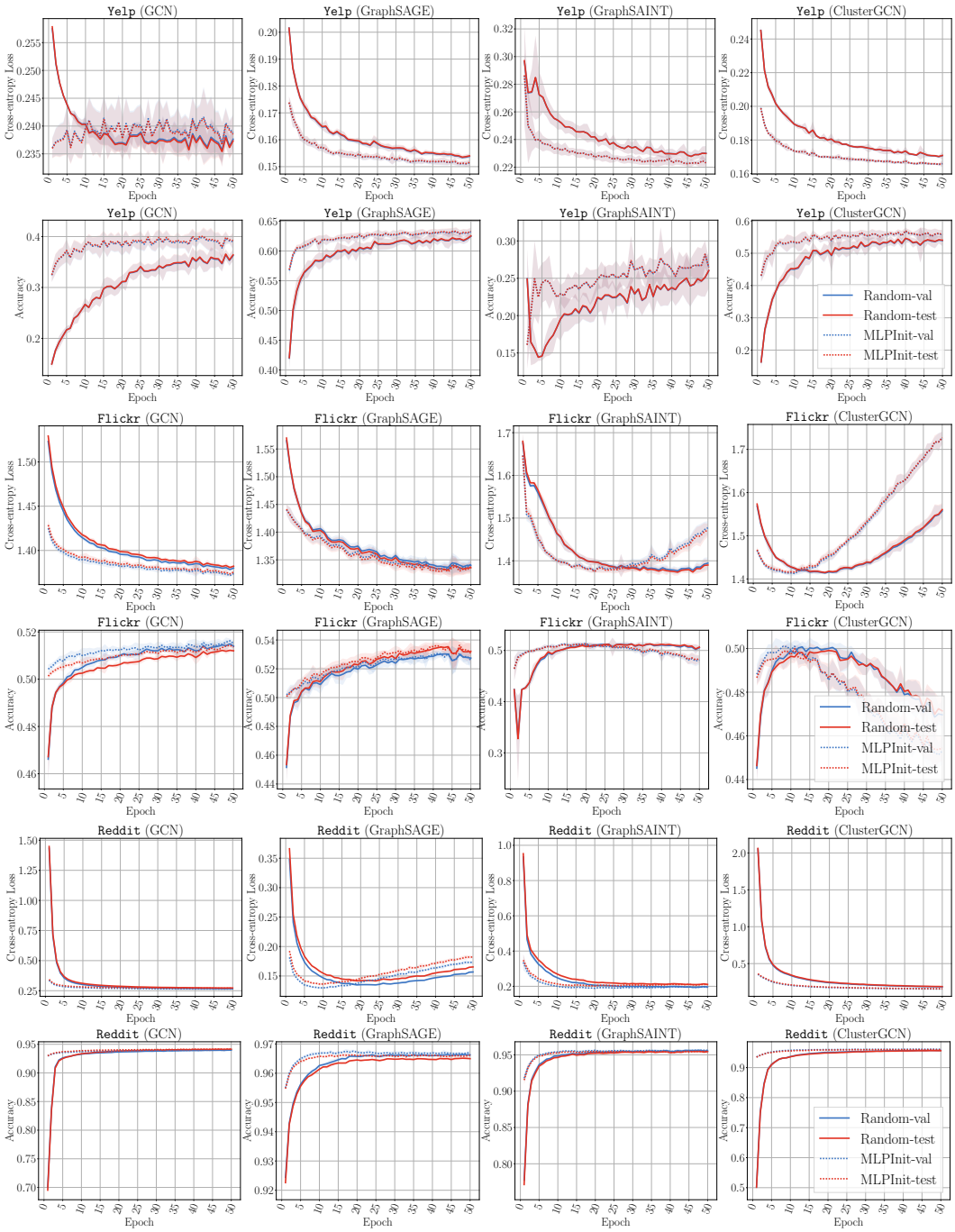


Figure 9: The training curves of GNN with random initialization and MLPInit on `Yelp`, `Flickr`, `Reddit` datasets.

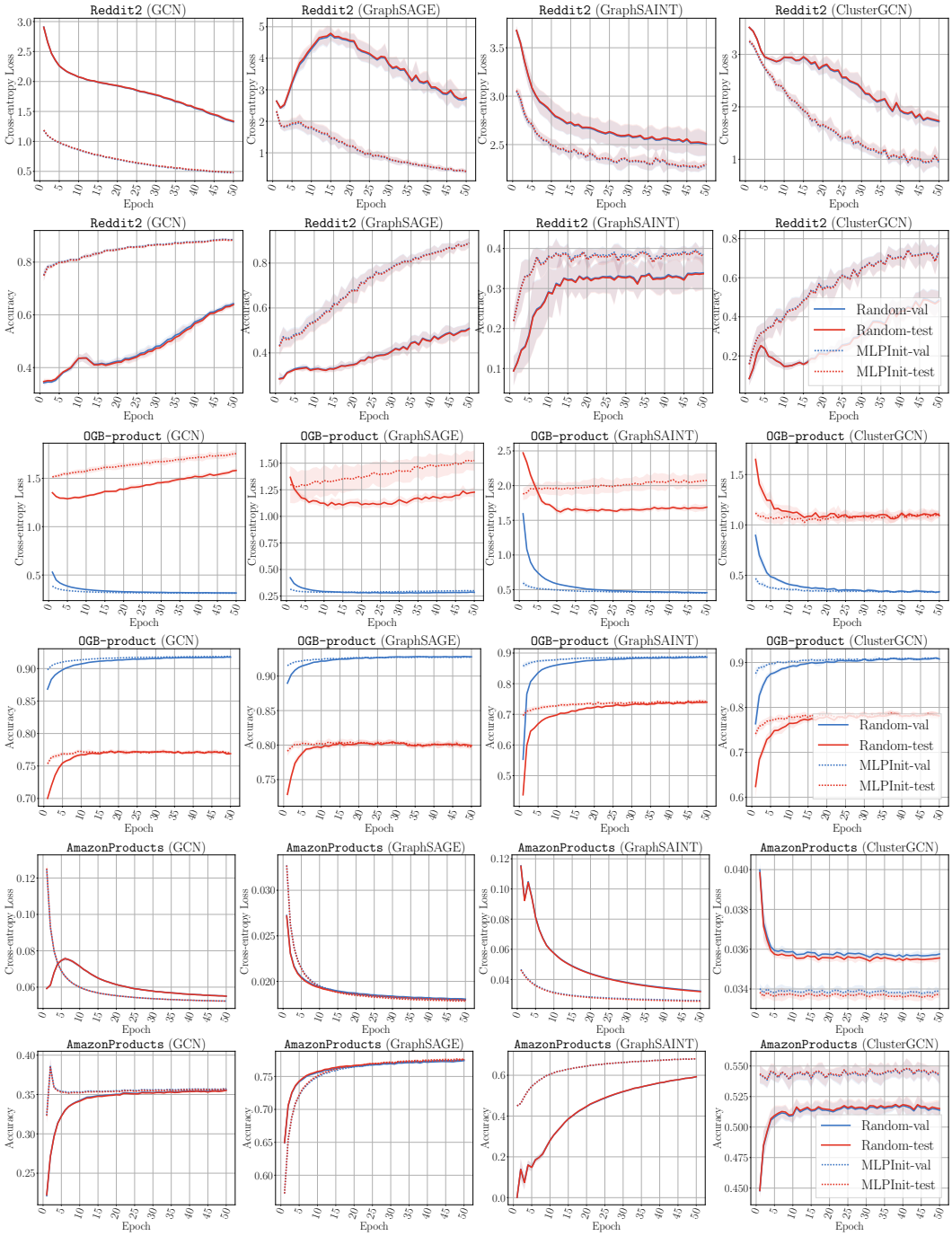


Figure 10: The training curves of GNN with random initialization and with MLPInit on Reddit2, OGB-products, A-products datasets.

B DATASETS AND BASELINES

In the appendix, we present the details of datasets and baselines for node classification and link prediction tasks.

B.1 DATASETS FOR NODE CLASSIFICATION

The details of datasets used for node classification are listed as follows:

- `Yelp` (Zeng et al., 2019) contains customer reviewers as nodes and their friendship as edges. The node feature are the low-dimensional review representation for their reviews.
- `Flickr` (Zeng et al., 2019) contains customer reviewers as nodes and their common properties as edges. The node features the 500-dimensional bag-of-word representation of the images.
- `Reddit`, `Reddit2` (Hamilton et al., 2017) is constructed by Reddit posts. The node in this dataset is post and belonging to different communities. `Reddit2` is the sparser version of `Reddit` by deleting some edges.
- `A-products` (Zeng et al., 2019) contains products and its categories.
- `OGB-arXiv` (Hu et al., 2020) is the citation network between all arXiv papers. Each node denotes an paper and each edge denotes citation between two papers. The node features are the average 128-dimensional word vector of its title and abstract.
- `OGB-products` (Hu et al., 2020; Chiang et al., 2019) is Amazon product co-purchasing network. Nodes represent products in Amazon, and edges between two products indicate that the products are purchased together. Node features in this dataset are low-dimensional representation of the product description text.

We present the statistic of datasets used for node classification task in Table 7.

Table 7: Statistics for datasets used for node classification.

Dataset	# Nodes.	# Edges	# Classes	# Feat	Density
`Flickr`	89,250	899,756	7	500	0.11‰
`Yelp`	716,847	13,954,819	100	300	0.03‰
`Reddit`	232,965	114,615,892	41	602	2.11‰
`Reddit2`	232,965	23,213,838	41	602	0.43‰
`A-products`	1,569,960	264,339,468	107	200	0.11‰
`OGB-arXiv`	169,343	1,166,243	40	128	0.04‰
`OGB-products`	2,449,029	61,859,140	47	218	0.01‰

B.2 BASELINES FOR NODE CLASSIFICATION

We present the details of GNN models as follows:

- **GCN** (Kipf & Welling, 2016a; Hamilton et al., 2017) is the original graph convolution network, which aggregates neighbor's information to obtain node representation. In our experiment, we train GCN in a mini-batch fashion by adopting a subgraph sampler from Hamilton et al. (2017).
- **GraphSAGE** (Hamilton et al., 2017) proposes a graph sampling based training strategy to scale up graph neural networks. It samples a fixed number of neighbors per node and training the GNNs in a mini-batch fashion.
- **GraphSAINT** (Zeng et al., 2019) is a graph sampling-based method to train GNNs on large-scale graphs, which proposes a set of graph sampling algorithms to partition graph data into subgraphs. This method also present normalization techniques to eliminate biases during the graph sampling.

- **ClusterGCN** (Chiang et al., 2019) is proposed to train GCN algorithm in small batches by using the graph clustering structure. This approach samples the nodes associated with the dense subgraphs identified by the graph clustering algorithm. Then the GCN is trained by the subgraphs.

B.3 DATASETS FOR LINK PREDICITON

For link prediction task, we consider `Cora`, `CiteSeer`, `PubMed`, `CoraFull`, `CS`, `Physics`, `A-Photo` and `A-Computers` as our baselines. The details of the datasets used for node classification are listed as follows:

- `Cora`, `CiteSeer`, `PubMed` (Yang et al., 2016) are representative citation network datasets. These datasets contains a number of research papers, where nodes and edges denote documents and citation relationship, respectively. Node features are low-dimension representation for papers. Labels indicate the research field of documents.
- `CoraFull` (Bojchevski & Günnemann, 2018) is a citation network that contains paper and their citation relationship. Labels are generated based on topics. This dataset is the original data of the entire network of Cora and Cora dataset in Planetoid is its subset.
- `CS`, `Physics` (Shchur et al., 2018) are from Co-author dataset, which is co-authorship graph based on the Microsoft Academic Graph from the KDD Cup 2016 challenge. Nodes in this dataset are authors and edges indicate co-author relationship. Node features represent paper keywords. Labels indicate the research field of authors.
- `A-Photo`, `A-Computers` (Shchur et al., 2018) are two datasets from Amazon co-purchase dataset McAuley et al. (2015). Nodes in this dataset represent products while edges represent co-purchase relationship between two products. Node features are low-dimension representation of product reviews. Labels are categories of products.

We also present the statistic of dataset used for link prediction task in Table 8.

Table 8: Statistics for datasets used for link prediction.

Dataset	# Nodes	# Edges	# Feat	Density
`Cora`	2,708	5,278	1,433	0.72‰
`CiteSeer`	3,327	4,552	3,703	0.41‰
`PubMed`	19,717	44,324	500	0.11‰
`DBLP`	17,716	105,734	1,639	0.34‰
`CoraFull`	19,793	126,842	8,710	0.32‰
`A-Photo`	7,650	238,162	745	4.07‰
`A-Computers`	13,752	491,722	767	2.60‰
`CS`	18,333	163,788	6,805	0.49‰
`Physics`	34,493	495,924	8,415	0.14‰

B.4 BASELINES FOR LINK PREDICITON

Our link prediction setup is consistent with our discussion in Section 2, in which we use an inner-product decoder $\hat{\mathbf{A}} = \mathrm{sigmoid}(\mathbf{H} \cdot \mathbf{H}^T)$ to predict the probability of the link existence. We presented the results in Tables 5 and 6. Following standard benchmarks (Hu et al., 2020), the evaluation metrics adopted are AUC, Average Precision (AP), are hits ratio (Hit@#). The details of link predction are presented in Appendix C.5.

C IMPLEMENTATION DETAILS

In this appendix, we present the hyperparameters used for node classification task and link prediction task for all models and dataset.

Table 9: Training configuration for GNNs training in Figures 3, 9 and 10 and Tables 3 and 4.

Model	Dataset	#Layers	#Hidden	Learning rate	Batch size	Dropout	Weight decay	Epoch
GraphSAGE	Flickr	4	512	0.0001	1000	0.5	0.0001	50
	Yelp	4	512	0.0001	1000	0.2	0	50
	Reddit	4	512	0.0001	1000	0.2	0	50
	Reddit2	4	512	0.0001	1000	0.2	0	50
	A-products	4	512	0.001	1000	0.5	0	50
	OGB-arXiv	4	512	0.001	1000	0.5	0	50
	OGB-products	4	512	0.001	1000	0.5	0	50
GraphSAINT	Flickr	4	512	0.001	5000	0.2	0.0004	50
	Yelp	2	128	0.01	5000	0.7	0.0002	50
	Reddit	2	128	0.01	5000	0.7	0.0002	50
	Reddit2	2	128	0.01	5000	0.7	0.0002	50
	A-products	2	128	0.01	5000	0.2	0	50
	OGB-arXiv	2	128	0.01	5000	0.2	0	50
	OGB-products	2	128	0.01	5000	0.2	0	50
ClusterGCN	Flickr	2	256	0.001	5000	0.2	0.0002	50
	Yelp	4	256	0.0001	2000	0.5	0	50
	Reddit	4	256	0.0001	2000	0.5	0	50
	Reddit2	4	256	0.0001	2000	0.5	0	50
	A-products	4	128	0.001	2000	0.2	0.0001	50
	OGB-arXiv	4	128	0.001	2000	0.2	0.0001	50
	OGB-products	4	128	0.001	2000	0.2	0.0001	50
GCN	Flickr	2	512	0.0001	1000	0.5	0.0001	50
	Yelp	2	512	0.0001	1000	0.2	0	50
	Reddit	2	512	0.0001	1000	0.2	0	50
	Reddit2	2	512	0.0001	1000	0.2	0	50
	A-products	2	512	0.001	1000	0.5	0	50
	OGB-arXiv	2	512	0.001	1000	0.5	0	50
	OGB-products	2	512	0.001	1000	0.5	0	50

C.1 Running Environment

We run our experiments on the machine with NVIDIA Tesla T4 GPU (16GB memory) and 60GB DDR4 memory to train the models. For A-products and OGB-products datasets, we run the experiments with one NVIDIA A100 GPU (40GB memory). The code is implemented based on PyTorch 1.9.0 (Paszke et al., 2019) and PyTorch Geometric 2.0.4 (Fey & Lenssen, 2019). The optimizer is Adam (Kingma & Ba, 2015) to train all GNNs and their PeerMLPs.

C.2 Experiment Setting for Figure 2

In this experiment, we use GraphSAGE as GNN on OGB-arXiv dataset. We construct that PeerMLP($f_{mlp}(\mathbf{X}; w_{mlp})$) for GraphSAGE ($f_{gnn}(\mathbf{X}, \mathbf{A}; w_{mlp})$) and train it for 50 epochs. From the mathematical expression, GraphSAGE and its PeerMLP share the same weights w_{mlp} and the weights w_{mlp} are only trained by PeerMLP. We use the trained weights for GraphSAGE to do inference along the training procedure. For the landscape, suppose we have two optimal weight w^*_{gnn} and w^*_{mlp} for GraphSAGE and its PeerMLP, the middle one is the loss landscape based the PeerMLP with optimal with w^*_{mlp} while the the right one is the loss landscape based the GraphSAGE with optimal weights w^*_{gnn}.

C.3 Experiment Setting for Figures 3, 9 and 10 and Tables 3 and 4

In this appendix, we present the detailed experiment setting for our main result Figures 3, 9 and 10 and Tables 3 and 4. We construct the PeerMLP for each GNNs. And the we first train the PeerMLP

for 50 epochs and save the best model with the best validation performance. And then we use the weight trained by PeerMLP to initialize the GNNs, then fine-tune the GNNs. To investigate the performance of GNNs, we fine-tune the GNNs for 50 epochs. We list the hyperparameters used in our experiments. We borrow the optimal hyperparameters from paper (Duan et al., 2022). And our code is also based on the official code [4] of paper (Duan et al., 2022). For datasets not included in paper (Duan et al., 2022), we use the heuristic hyperparameter setting for them.

C.4 Experiment Setting for Table 2

The GNN used in Table 2 is GraphSAGE. We construct the PeerMLP for GraphSAGE on `OGB-arXiv` and `OGB-products` datasets. And the we first train the PeerMLP for 50 epochs and save the best model with the best validation performance. And we do the infer the test performance with PeerMLP and GraphSAGE with the weight trained by PeerMLP and we report the test performance in Table 2.

C.5 Experiment Setting for Tables 5 and 6

In this appendix, we present the detailed experiment setting for link prediction task. We adopt the default setting for the official examples [5] of PyTorch Geometric 2.0.4. The GNN used for link prediction task is a 2-layer GCN, and the decoder is the commonly used inner-product decoder as $\hat{\mathbf{A}} = \mathrm{sigmoid}(\mathbf{H} \cdot \mathbf{H}^T)$ (Kipf & Welling, 2016b).

C.6 Experiment Setting for Figure 4

We explore two kinds of hyperparameters "Training HP" (Learning rate, weight decay, batch size and dropout) and "Architecture HP" (i.e., layers, number of hidden neurons) in this experiments.

- Training Hyperparameter (Training HP), total combinations: $2 \times 2 \times 2 \times 2 = 16$
  - Learning rate: $\{0.001, 0.0001\}$
  - Weight decay: $\{1e-4, 4e-4\}$
  - Batch size: $\{500, 1000\}$
  - Dropout: $\{0.2, 0.5\}$
- Architecture Hyperparameter (Architecture HP), total combinations: $3 \times 5 = 15$
  - Number of layers: $\{2, 3, 4\}$
  - number of hidden neurons: $\{32, 64, 128, 256, 512\}$

In Figures 4 and 8, we plotted the learning curves based on the mean and standard deviation over all the hyperparameters combinations.

[4] https://github.com/VITA-Group/Large_Scale_GCN_Benchmarking
[5] https://github.com/pyg-team/pytorch_geometric/blob/2.0.4/examples/link_pred.py