On the Equivalence of Graph Convolution and Mixup

Xiaotian Han[1*] **Hanqing Zeng**[2] **Yu Chen**[3] **Shaoliang Nie**[2] **Jingzhou Liu**[2] **Kanika Narang**[2] **Zahra Shakeri**[2] **Karthik Abinav Sankararaman**[2] **Song Jiang**[4] **Madian Khabsa**[2] **Qifan Wang**[2] **Xia Hu**[5]

[1]*Case Western Reserve University* [2]*Meta AI* [3]*Anytime AI* [4]*UCLA* [5]*Rice University*

Reviewed on OpenReview: *https://openreview.net/forum?id=koC6zyaj73*

Abstract

This paper investigates the relationship between graph convolution and Mixup techniques. Graph convolution in a graph neural network involves aggregating features from neighboring samples to learn representative features for a specific node or sample. On the other hand, Mixup is a data augmentation technique that generates new examples by averaging features and one-hot labels from multiple samples. One commonality between these techniques is their utilization of information from multiple samples to derive feature representation. This study aims to explore whether a connection exists between the two. Our investigation reveals that, under two mild modifications, graph convolution can be viewed as a specialized form of Mixup that is applied during both the training and testing phases. The two modifications are 1) *Homophily Relabel* - assigning the target node's label to all its neighbors, and 2) *Test-Time Mixup* - Mixup the feature during the test time. We establish this equivalence mathematically by demonstrating that graph convolution networks and simplified graph convolution can be expressed as a form of Mixup. We also empirically verify the equivalence by training an MLP using the two modifications to achieve comparable performance.

1 Introduction

Graph Neural Networks (GNNs) (Wu et al., 2020; Zhou et al., 2020) have recently been recognized as the *de facto* state-of-the-art algorithm for graph learning. The core idea behind GNNs is neighbor aggregation, which involves combining the features of a node's neighbors. Specifically, for a target node with feature $\mathbf{x}_i$, one-hot label $\mathbf{y}_i$, and neighbor set $\mathcal{N}_i$, the graph convolution operation in GCN is essentially as follows:

$$(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} \mathbf{x}_k, \quad \mathbf{y}_i \right), \tag{1}$$

In parallel, Mixup (Zhang et al., 2018) is proposed to train deep neural networks effectively, which also essentially generates a new sample by averaging the features and labels of multiple samples:

$$(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\sum_{i=1}^{N} \lambda_i \mathbf{x}_i, \quad \sum_{i=1}^{N} \lambda_i \mathbf{y}_i \right), \quad \text{s.t.} \quad \sum_{i=1}^{N} \lambda_i = 1, \tag{2}$$

where $\mathbf{x}_i/\mathbf{y}_i$ are the feature/label of sample i. Mixup typically takes two data samples ($N = 2$).

Equation (1) and Equation (2) highlight a remarkable similarity between graph convolution and Mixup, i.e., *the manipulation of data samples through averaging the features*. This similarity prompts us to investigate the relationship between these two techniques as follows:

Is there a connection between graph convolution and Mixup?

*Corresponds to xhan@case.edu. The code is at `https://github.com/ahxt/GraphConv_is_Mixup`.

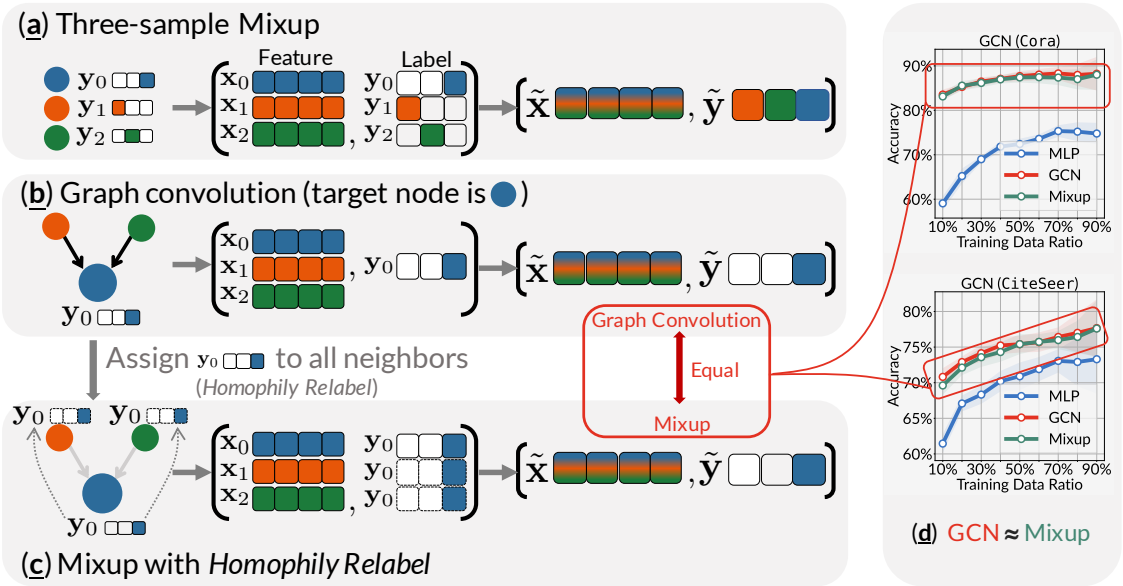

Figure 1: Graph convolution is Mixup. **(a)** illustrates the basic idea of Mixup: averaging the features and one-hot labels of multiple samples (●, ●, ●). **(b)** shows the graph convolution operation where the feature of the target node (●) is the weighted average of the features of all its neighbors. **(b)** → **(c)** shows that graph convolution is Mixup if we assign the label (▭▭■) of the target node (●) to all of its neighbors (●, ●). **(d)** shows that Mixup is empirically equivalent to GCN.

In this paper, we answer this question by establishing the connection between graph convolutions and Mixup, and further understanding the graph neural networks through the lens of Mixup. We show that graph convolutions are intrinsically equivalent to Mixup by rewriting Equation (1) as follows:

$$(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} \mathbf{x}_k, \mathbf{y}_i \right) = \left(\sum_{k \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i|} \mathbf{x}_k, \sum_{k \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i|} \mathbf{y}_i \right) \overset{\lambda_i = \frac{1}{|\mathcal{N}_i|}}{=} \left(\sum_{k \in \mathcal{N}_i} \lambda_i \mathbf{x}_k, \sum_{k \in \mathcal{N}_i} \lambda_i \mathbf{y}_i \right),$$

where $\mathbf{x}_i$ and $\mathbf{y}_i$ are the feature and label of the target node n_i. This equation states that graph convolution is equivalent to Mixup if we assign the $\mathbf{y}_i$ to all the neighbors of node n_i in set $\mathcal{N}_i$

To demonstrate the equivalence between graph convolutions and Mixup, we begin by illustrating that a one-layer graph convolutional network (GCN) (Kipf & Welling, 2016b) can be transformed into an input Mixup. A two-layer GCN can be expressed as a hybrid of input and manifold Mixup (Verma et al., 2019). Similarly, simplifing graph convolution (SGC) (Wu et al., 2019) can be reformulated as an input Mixup. We thus establish the mathematical equivalence between graph convolutions and Mixup, under two mild and reasonable modifications: 1) assign the target node's label to neighbors in the training time (referred to as *Homophily Relabel*); 2) perform feature mixup in the test time (referred to as *Test-Time Mixup*).

We further investigate the modifications required for the equivalence between graph convolution and Mixup, focusing on the effect of *Homophily Relabel* and *Test-Time Mixup*. To explore *Homophily Relabel*, we demonstrate that training an MLP with *Homophily Relabel* (called HMLP) is equivalent to training GCN in terms of prediction accuracy. This finding provides a novel perspective for understanding the relationship between graph convolution and Mixup and suggests a new direction for designing efficient GNNs. To investigate Test-Time Mixup, we train GNNs without connection information, perform neighbor aggregation during inference, and find that this approach can achieve performance comparable to traditional GNNs. This result reveals that Test-Time Mixup can be a powerful alternative to traditional GNNs in some scenarios, and suggests that Mixup may have a broader range of applications beyond traditional data augmentation. Our investigation on *Homophily Relabeling* and *Test-Time Mixup* provides valuable insights into the theoretical properties of GNNs and their potential applications in practice. We highlight our **contributions** as follows:

- We establish for the first time connection between graph convolution and Mixup, showing that graph convolutions are mathematically and empirically equivalent to Mixup. This simple yet novel finding potentially opens the door toward a deeper understanding of GNNs.

- Given that both graph convolution and Mixup perform a weighted average on the features of samples, we reveal that graph convolution is conditionally equivalent to Mixup by assigning the label of a target node to all its neighbors during training time (*Homophily Relabel*). Also, we also reveal that in the test time, graph convolution is also Mixup (*Test-Time Mixup*).
- Based on *Homophily Relabel* and *Test-Time Mixup*, we propose two variants of MLPs based on the Mixup strategy, namely HMLP and TMLP, that can match the performance of GNNs.

Related Work. Graph neural networks are widely adopted in various graph applications, including social network analysis (Fan et al., 2019), recommendation (Wei et al., 2022; Deng et al., 2022; Cai et al., 2023; Tang et al., 2022), knowledge graph (Cao et al., 2023; Zhao et al., 2023), molecular analysis (Sun et al., 2022; Zhu et al., 2023a; ZHANG et al., 2023; Wang et al., 2023b; Corso et al., 2023; Liao & Smidt, 2023; Xia et al., 2023; Hladiš et al., 2023), drug discovery (Sun et al., 2022; Zhang et al., 2023c), link prediction (Chamberlain et al., 2023; Cong et al., 2023) and others (Chen et al., 2023a). Understanding the generalization and working mechanism of graph neural networks is still in its infancy (Garg et al., 2020; Zhang et al., 2023a; Yang et al., 2023; Baranwal et al., 2023). The previous work attempts to understand graph neural networks from different perspectives, such as signal processing (Nt & Maehara, 2019; Bo et al., 2021; Bianchi et al., 2021), gradient flow (Di Giovanni et al., 2022), dynamic programming(Dudzik & Veličković, 2022), neural tangent kernels (Yang et al., 2023; Du et al., 2019; Sabanayagam et al., 2022) and influence function (Chen et al., 2023b). There is also a line of works that analyzes the connection between GNNs and MLPs (Baranwal et al., 2023; Han et al., 2022b; Yang et al., 2023; Tian et al., 2023), which is similar to the proposed method in Section 3. In this work, we understand graph neural networks through a fresh perspective, Mixup. We believe that this work will inspire further research and lead to the development of new techniques to improve the performance and interpretability of GNNs. In parallel, Mixup (Zhang et al., 2018) and its variants (Verma et al., 2019; Yun et al., 2019; Kim et al., 2020; 2021) have emerged as a popular data augmentation technique that improves the generalization performance of deep neural networks. More specifically, deep neural networks trained with Mixup achieve better generalization (Chun et al., 2020; Zhang et al., 2020; Chidambaram et al., 2022), calibration (Thulasidasan et al., 2019; Zhang et al., 2022a), adversarial robustness (Pang et al., 2019; Archambault et al., 2019; Zhang et al., 2020; Lamb et al., 2019), and explain ensemble (Lopez-Paz et al., 2023). More related work in Appendix A.

The Scope of Our Work. [1] This work not only provides a fresh perspective for comprehending Graph Convolution through Mixup, but also makes valuable contributions to the practical and theoretical aspects of graph neural networks, facilitating efficient training and inference for GNNs when dealing with large-scale graph data. Practical Potentials: Both HMLP and TMLP hold potential value for practical usage. TMLP is training-efficient, as its backbone and training process are both MLP-based; however, the inference requires *Test-Time Mixup* (neighbor aggregation), which can be time-consuming. This limitation also exists in previous works (Han et al., 2022b; Yang et al., 2023). HMLP, on the other hand, is both training and test efficient, as both processes are based on MLP. This suggests the potential of HMLP for practical usage on large-scale graphs. Theoretical Potentials: HMLP, derived from the Mixup, have the theoretical potential to understand the expressiveness of GNNs from the Mixup perspective. With the connection between Mixup and graph convolution, HMLP goes beyond traditional methods to understand the learning capability of GNNs.

2 Graph Convolution is Mixup

In this section, we reveal that the graph convolution is essentially equivalent to Mixup. We first present the notation used in this paper. Then we present the original graph convolution network (GCN) (Kingma & Ba, 2015) and simplifying graph convolutional networks (SGC) (Wu et al., 2019) can be expressed mathematically as a form of Mixup. Last, we present the main claim, i.e., graph convolution can be viewed as a special form of Mixup under mild and reasonable modifications.

Notations. We denote a graph as $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the node set and $\mathcal{E}$ is the edge set. The number of nodes is $N = |\mathcal{V}|$ and the number of edges is $M = |\mathcal{E}|$. We denote the node feature matrix as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \in \mathbb{R}^{N \times d}$, where $\mathbf{x}_i$ is the node feature for node n_i and d is the dimension of features. We

[1] Note that our work only focuses on the node-wise predictions made by GCNs.

denote the binary adjacency matrix as $\mathbf{A} \in \{0,1\}^{N \times N}$, where $\mathbf{A}_{ij} = 1$ if there is an edge between node n_i and node n_j in edge set $\mathcal{E}$, and $\mathbf{A}_{ij} = 0$ otherwise. We denote the neighbor set of node n_i as $\mathcal{N}_i$, and its 2-hop neighbor set as $\mathcal{N}_i^2$. For the node classification task, we denote the prediction targets of nodes as one-hot labels $\mathbf{Y} \in \{0,1\}^{N \times (C-1)}$, where C is the number of classes. For graph convolution network, we use $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$ as the normalized adjacency matrix[2], where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is diagonal degree matrix, and $\mathbf{D}_{ii}$ is the degree of node n_i. We use $\tilde{\mathbf{a}}_i \in \mathbb{R}^{1 \times N}$, i-th row of $\tilde{\mathbf{A}}$, as the normalized adjacency vector of node n_i.

2.1 Preliminaries

Graph Convolution. Graph Convolution Network (GCN) (Kipf & Welling, 2016a), as the pioneering work of GNNs, proposes $\hat{\mathbf{Y}} = \text{softmax}(\tilde{\mathbf{A}} \cdot \sigma(\tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W}_1) \cdot \mathbf{W}_2)$, where $\mathbf{W}_1$ and $\mathbf{W}_2$ are the trainable weights of the layer one and two, respectively. Simplifying Graph Convolutional Networks (SGC) (Wu et al., 2019) is proposed as $\hat{\mathbf{Y}} = \text{softmax}(\tilde{\mathbf{A}} \cdot \tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W})$. In this work, we take these two widely used GNNs to show that graph convolution is essentially Mixup.

Mixup. The Mixup technique, introduced by Zhang et al. (2018), is a simple yet effective data augmentation method to improve the generalization of deep learning models. The basic idea behind Mixup is to blend the features and one-hot labels of a random pair of samples to generate synthetic samples. The mathematical expression of the two-sample Mixup is as follows:

$$(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = (\lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j, \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j) = (\lambda_1 \mathbf{x}_i + \lambda_2 \mathbf{x}_j, \lambda_1 \mathbf{y}_i + \lambda_2 \mathbf{y}_j), \tag{3}$$

where $\lambda_i + \lambda_j = 1$. Based on the two-sample Mixup, the multiple-sample Mixup is presented in Equation (2). The mathematical expression presented above demonstrates that Mixup computes a weighted average of the features from multiple original samples to generate synthetic samples.

2.2 Connecting Graph Convolution and Mixup

We demonstrate that graph convolution, using GCN and SGC as examples, is conditionally Mixup. To do this, we mathematically reformulate the expressions of GCN and SGC to a Mixup form, thereby illustrating that graph convolutions are indeed Mixup.

One-layer GCN is Mixup We begin our analysis by examining a simple graph convolution neural network (Kipf & Welling, 2016a) with one layer, referred to as the one-layer GCN. We demonstrate that the one-layer GCN can be mathematically understood as an implementation of the input Mixup technique. The expression of the one-layer GCN is given by: $\hat{\mathbf{Y}} = \text{softmax}(\tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W})$ where $\mathbf{W} \in \mathbb{R}^{D \times C}$ is the trainable weight matrix. Let us focus on a single node n_i in the graph. The predicted one-hot label for this node is given by $\hat{\mathbf{y}}_i = \text{softmax}(\tilde{\mathbf{a}}_i \cdot \mathbf{X} \cdot \mathbf{W})$, where $\tilde{\mathbf{a}}_i \in \mathbb{R}^{1 \times N}$, i-th row of $\tilde{\mathbf{A}}$, is the normalized adjacency vector of node n_i. $\mathcal{N}_i$ is the neighbors set of node n_i. We make the following observations:

- $\tilde{\mathbf{a}}_i \cdot \mathbf{X}$ results in a weighted sum of the features of the neighbors of node n_i, which is the multiple-sample Mixup. Explicitly, we can rewrite $\tilde{\mathbf{a}}_i \cdot \mathbf{X}$ to $\tilde{\mathbf{x}} = \frac{1}{|\mathcal{N}i|} \sum_{k \in \mathcal{N}_i} \mathbf{x}_k$.
- If we assign node n_i's label to all its neighbors, the label $\mathbf{y}_i$ of node n_i can be interpreted as a weighted average of the labels of its neighboring nodes, which is equivalent to performing Mixup on the labels. Thus we have $\tilde{\mathbf{y}} = \mathbf{y}_i = \frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} \mathbf{y}_k$.

From the previous observations, we can see that one-layer GCN synthesizes a new Mixup sample for node n_i by mixing up its neighbors' features and one-hot labels, which is defined as follows:

$$(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} \mathbf{x}_k, \frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} \mathbf{y}_k \right). \tag{4}$$

Therefore, we conclude that *a one-layer GCN is a Mixup machine, which essentially synthesizes a new sample by averaging its neighbors' features and one-hot labels.*

[2]Our following analysis can be easily generalized to other normalization methods, such as $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$.

Two-layer GCN is Mixup We extend our analysis to consider a two-layer GCN. The expression for two-layer GCN is given by $\hat{\mathbf{Y}} = \text{softmax}(\tilde{\mathbf{A}} \cdot \sigma(\tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W}_1) \cdot \mathbf{W}_2)$, Let us focus on node n_i in the graph. For the first layer of the two-layer GCN, we have $\mathbf{h}_i = \text{Relu}(\tilde{\mathbf{a}}_i \cdot \mathbf{X} \cdot \mathbf{W}_1) \in \mathbb{R}^{1\times d}$, where $\tilde{\mathbf{a}}_i \in \mathbb{R}^{1\times N}$ is the normalized adjacency vector of node n_i. The first layer is the same as the one-layer GCN as discussed above, which can be regarded as a multiple-sample input Mixup.

For the second layer of the two-layer GCN, we have $\hat{\mathbf{y}}_i = \text{Softmax}(\tilde{\mathbf{a}}_i \mathbf{H} \mathbf{W}_2) \in \mathbb{R}^{1\times C}$, $\mathbf{H} \in \mathbb{R}^{N\times d}$ is the hidden representation of all nodes obtained from the first layer, and $\mathbf{W}_2 \in \mathbb{R}^{d\times C}$ is the weight matrix. The predicted one-hot label for node n_i, $\hat{\mathbf{y}}_i$, is obtained through a softmax activation function. The 2-nd layer can be regarded as multiple-sample manifold Mixup (Verma et al., 2019) in the following:

- $\tilde{\mathbf{a}}_i \cdot \mathbf{H}$ is the multiple-sample Mixup of the hidden representation of the neighbors of node n_i. We rewrite the $\tilde{\mathbf{a}}_i \cdot \mathbf{H}$ to $\tilde{\mathbf{x}} = \tilde{\mathbf{a}}_i \cdot \mathbf{H} = \frac{1}{|\mathcal{N}_i|} \sum_{k\in\mathcal{N}_i} \mathbf{h}_k$.

Therefore, we conclude that *a multi-layer GCN is a hybrid of input Mixup (first layer) and manifold Mixup (second layer).*

SGC is Mixup The 2-layer SGC architecture is represented by the following expression, $\hat{\mathbf{Y}} = \text{softmax}(\tilde{\mathbf{A}} \cdot \tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W})$. Similar to one-layer GCN, for the node n_i in a 2-layer SGC, we have

$$\hat{\mathbf{y}}_i = \text{softmax}(\tilde{\mathbf{a}}_i^2 \cdot \mathbf{X} \cdot \mathbf{W}), \mathcal{L} = \text{Cross-Entropy}(\hat{\mathbf{y}}_i, \mathbf{y}_i), \tag{5}$$

where $\tilde{\mathbf{a}}_i^2 \in \mathbb{R}^{1\times N}$ is the adjacency vector within 2-hop neighbors of node n_i. The 2-hop neighbor set of node n_i is represented by $\mathcal{N}_i^2$, Hereby we have

- $\tilde{\mathbf{a}}_i^2 \cdot \mathbf{X}$ is the multiple Mixup of the features of the neighbors of node n_i. We rewrite the $\tilde{\mathbf{a}}i \cdot \mathbf{X}$ to $\tilde{\mathbf{x}} = \tilde{\mathbf{a}}_i^2 \cdot \mathbf{X} = \frac{1}{|\mathcal{N}_i^2|} \sum k \in \mathcal{N}_i^2 \mathbf{x}_k$.

From the above, we have a Mixup of samples for node n_i is

$$(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\frac{1}{|\mathcal{N}_i^2|} \sum_{k\in\mathcal{N}_i^2} \mathbf{x}_k, \quad \frac{1}{|\mathcal{N}_i^2|} \sum_{k\in\mathcal{N}_i^2} \mathbf{y}_k \right). \tag{6}$$

Thus we conclude that *an SGC is an input Mixup machine.*

2.3 Graph Convolution is (*Conditionally*) Mixup

It is straightforward to derive the Mixup form of 1-layer, 2-layer GCN, and SGC as discussed above. This leads to the conclusion that 1-layer, 2-layer GCN, and SGC can all be reformulated in the form of Mixup. This establishes a mathematical connection between graph neural networks and the Mixup method. Building on these findings, in this section, we introduce our main contribution as follows:

Main Results

Graph convolution is Train- and Test-Time Mixup, under two mild and reasonable modifications: *Homophily Relabel* and *Test-time Mixup*.

The two primary differences between GNNs and Mixup are as follows:

- ***Homophily Relabel*** At training time, if we assign the label of the target node to all its neighbors, the Graph Neural Network can be naturally rewritten as a form of Mixup.
- ***Test-Time Mixup*** At test time, the GNNs perform feature mixing on the nodes and then use the mixed feature for the inference.

Both of the above modifications are mild and reasonable and have practical implications for GNN design and analysis. The *Homophily Relabel* operation can be understood as imposing the same label on the target node and all its neighbors, which corresponds to the homophily assumption for graph neural networks. The homophily assumption posits that nodes with similar features should have the same label, which is a common

assumption in many real-world graph applications. On the other hand, *Test-time Mixup* can be understood as Mixup at the test time by mixing the features from neighbors for prediction. We examine the above differences in depth in Section 3 and Section 4.

2.4 Discussion

Comparison to Other Work. Hereby, we compare our work and previous work. Yang et al. (2023); Han et al. (2022b) propose to train an MLP and transfer the converged weight from MLP to GNN, which can achieve comparable performance to GCN. In this work, these two methods are all included in part of our work, *Test-Time Mixup*. Different from previous work, we provide a fresh understanding of this phenomena by connecting graph convolution to Mixup and also derive a TMLP Appendix C.2 to implement *Test-Time Mixup*. Baranwal et al. (2022) understand the effect of graph convolution by providing a rigorous theoretical analysis based on a synthetic graph. Different from this work, our work understands graph convolution with a well-studied technique, Mixup.

Why MLP is more efficient to GNN: GNN is $\hat{\mathbf{Y}} = \text{softmax}(\tilde{\mathbf{A}} \cdot \mathbf{X} \cdot \mathbf{W})$, where $\mathbf{A}$ is a large sparse matrix, resulting in significant computation time. This is well studied in previous work Zhang et al. (2021); Yang et al. (2023). We also conducted experiments to measure the actual runtime of MLP and GNN. The results demonstrate the efficiency of MLP compared to GNN. With mixup, the number of data points will double. The running time of MLP will at most double, resulting in a running time ratio of MLP to GNN = (1.8396 * 2) : 174.1459 = 1 : 47.

Table 1: The running time of one epoch comparison of original MLP vs. GNN.

	MLP	**GNN**
Running time	1.8396±0.0264	174.1459±7.0611

3 Is *Homophily Relabel* Equivalent to GCNs Training ?

In this section, we conduct experiments to examine the *Homophily Relabel* proposed in Section 2.3. The empirical evidence substantiates our claims, emphasizing the significant effect of Homophily Relabel on GNNs. This understanding can facilitate the design of an MLP equivalent that competes with the performance of GNNs. Note that we explore the transductive setting in our experiments.

3.1 *Homophily Relabel* Yields a Training-Equivalent MLP to GCN

We utilize the simple example graph to demonstrate the calculation of the loss function in a GCN in the transductive setting. The example graph, in Figure 2, comprises three nodes. The blue and red nodes belong to the training set, while the gray node belongs to the test set. In the transductive scenario, the loss function is computed for the entire graph during the training phase, incorporating both the training and test nodes. In our simple example graph, the loss function will be calculated on the blue and red nodes with their actual labels. Notably, the prediction for the gray node, despite it being part of the test set, will also contribute to the overall loss calculation in the transductive setting. For node n_i, the prediction (p_0, p_1), the loss of the example graph will be as follows

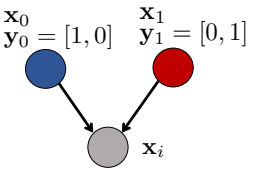

Figure 2: The example graphs. $\mathbf{x}_i$ is the target node, The loss of $\mathbf{x}_i$ that connected two nodes with different labels.

$$
\begin{aligned}
L_{\mathcal{G}} &= \text{cross-entropy}(\mathbf{Y}, \hat{\mathbf{Y}}) = \text{cross-entropy}(\mathbf{y}_0, \hat{\mathbf{y}_0}) + \text{cross-entropy}(\mathbf{y}_1, \hat{\mathbf{y}_1}) \\
&= -y_{00}\log(p_0) - y_{01}\log(p_1) - y_{10}\log(p_0) - y_{11}\log(p_1) \\
&= -1 * \log(p_0) - 0 * \log(p_1) - 0 * \log(p_0) - 1 * \log(p_1) \\
&= -1 * \log(p_0) - 1 * \log(p_1) = -2 * (0.5 * \log(p_0) + 0.5 * \log(p_1)).
\end{aligned} \tag{7}
$$

where y_{00}/y_{01} is the 0th/1st elements in the one-hot label $\mathbf{y}_0$ for node n_0. The above analysis shows that the actual training label for target node n_i is $(0.5, 0.5)$. From the Mixup perspective, in an ideally well-trained model utilizing a two-sample Mixup, as shown in previous work (Guo et al., 2019; Pang et al., 2019), the following approximation will hold $f(\lambda_k x + (1-\lambda_k)x_k) \approx \lambda_k f(x) + (1-\lambda_k) f(x_k)$. Intuitively, this approximation

holds because the model is trained with mixup examples $(x_{new}, y_{new}) = (\lambda x_i + (1-\lambda)x_j, \lambda y_i + (1-\lambda)y_j)$. The linear correlation between x_{new} and y_{new} leads to a model that can be approximated by a linear function. If we regard the neighbor aggregation as Mixup, we can easily derive that

$$\hat{\mathbf{y}}_0 = f\left(\frac{\mathbf{x}_0 + \mathbf{x}_i}{2}\right) = \frac{1}{2}\left(f(\mathbf{x}_0) + f(\mathbf{x}_i)\right), \hat{\mathbf{y}}_1 = f\left(\frac{\mathbf{x}_1 + \mathbf{x}_i}{2}\right) = \frac{1}{2}\left(f(\mathbf{x}_1) + f(\mathbf{x}_i)\right). \tag{8}$$

From the above two equations, we can see that $f(x_i)$ are trained with the labels of all its neighbors $\mathbf{x}_0$ and $\mathbf{x}_1$. Thus in the next, *we propose explicitly training the nodes in the test set with the label of their neighbors if they are in the training set.* In our example graph, we explicitly train the gray node with label $(0.5, 0.5)$

HMLP Based on the above analysis, we proposed Homophily Relabel MLP (HMLP), which achieves the comparable performance of GCN via training an MLP as a backbone with *Homophily Relabel*. In detail, the proposed method HMLP has two major steps: 1) relabel all the nodes in the graph with the mixed label $\hat{\mathbf{Y}} = \mathbf{A} \cdot \mathbf{Y}$. 2) train an MLP on all the nodes with $\hat{\mathbf{Y}}$ on relabeled graph, only using node features. We illustrate our proposed HMLP with the example graph in Figure 2. 1) we relabel all the nodes using *Homophily Relabel*, the node with new label will be $(\mathbf{x}_0, (1, 0)), (\mathbf{x}_1, (0, 1)), (\mathbf{x}_i, (0.5, 0.5))$ 2) then train the HMLP with these data samples. The test sample will be $(\mathbf{x}_i, ?)$ during the test time.

3.2 Can HMLP Achieve Comparable Performance to GCN?

To verify the proposed HMLP can achieve comparable performance to the original GCN, we train the one-layer GCN (GCN-1), two-layer GCN (GCN-2), SGC, and HMLP on the training set and report the test accuracy based on the highest accuracy achieved on the validation set. We experimented with different data splits of train/validation/test (the training data ratio span from $10\% - 90\%$), and we also conducted experiments with the public split on `Cora`, `CiteSeer`, and `PubMed` datasets. We present the results in Table 2 and Figure 4. We also present the training and test curves of the MLP. GCN and HMLP in Figure 3.

Table 2: Performance comparison of GCN and HMLP.

Datasets	MLP	GCN	HMLP
`Cora`	$73.57_{\pm 0.98}$	$88.04_{\pm 1.26}$	$86.42_{\pm 1.78}$
`CiteSeer`	$71.90_{\pm 1.69}$	$75.69_{\pm 1.36}$	$75.94_{\pm 1.54}$
`PubMed`	$86.90_{\pm 0.74}$	$87.94_{\pm 0.64}$	$88.34_{\pm 0.48}$
Average	77.45	**83.89**	**83.57**

From the results in Table 2 and Figure 4, ❶ **HMLP achieves comparable performance to GCN when the ratio of training to test data is large.** For the Figure 4, the performance of HMLP is significantly better than MLP, and on par with GCN, especially when the training data ratio is large. The results also show that when the training data ratio is small, HMLP performs worse than GCN. The training and test curves show that HMLP achieve the similar test curves (red and green lines in the right subfigure) even though the training curve are not the same. This may be because the labels of nodes are changed.

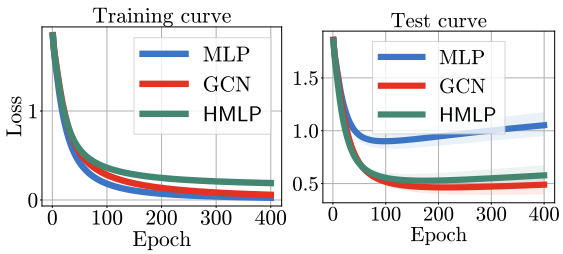

Figure 3: The training and test curves of GCN and HMLP.

4 How Does *Test-time Mixup* Affect GCN Inference?

In this section, we aim to investigate how *Test-time Mixup* affects the GCN inference process. We conduct an empirical study to understand the functionality of *Test-time Mixup* (neighbor aggregation during the test time). To this end, we propose an MLP involving *Test-time Mixup* only (TMLP).

TMLP The proposed method TMLP follows the steps: 1) we train an MLP using only the node features of the training set. 2) we employ the well-trained MLP for inference with neighbor aggregation during testing. This paradigm has also been investigated in previous works (Han et al., 2022b; Yang et al., 2023). We call this method TMLP. We illustrate the TMLP with the example graph in Figure 2. Specifically, we train the model

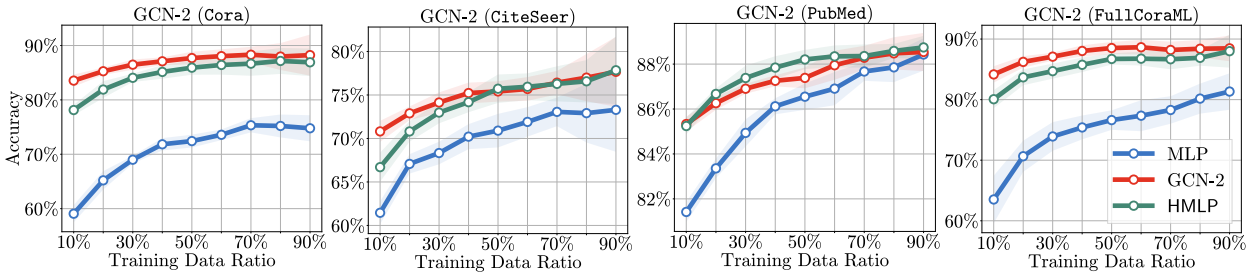

Figure 4: The performance comparison of the GCN, MLP and HMLP (Ours). The x-axis represents the ratio of training data, and the y-axis represents the accuracy of classification. The results show that our proposed method (HMLP) achieves comparable performance to GCN. Note that the architecture of our method in train and test time are both MLP. More experimental results on other datasets and GNNs are presented in Appendix C.1.

using the following training examples, $(\mathbf{x}_0, (1, 0)), (\mathbf{x}_1, (0, 1))$. Note that the backbone of TMLP is an MLP. In the following, we explore the performance of TMLP.

4.1 How Does TMLP Perform Compared to GCN?

In the section, we conducted experiments to verify the effect of *Test-Time Mixup*. In the experiments, during the training, we only use the node feature to train an MLP, and then during the test time, we use the well-trained MLP with the neighbor aggregation to perform the inference. We present the results of varying training data ratio in Figure 6, and we also present the accuracy in Table 3 when the training data ratio is 0.6. Additionally, we compare the learned representation of GNNs and TMLP using t-SNE Van der Maaten & Hinton (2008). From these results, we make the following observations:

From these results, we make the following observations: ❷ **TMLP almost achieves comparable performance to GNNs.** In the Figure 6, the performance of TMLP is significantly better than MLP and on par with GCN, especially when the training data ratio is large. In some datasets (i.e., Cora, FullCoraML), the performance of TMLP is slightly worse than GCN but still significantly better than MLP. It is worth noting that our proposed TMLP is only trained with the node features, ignoring the connection information. Table 3 show the average accuracy (84.06) of TMLP is comparable to that of GCN (83.89).

Table 3: Performance comparison of GCN and HMLP.

Datasets	MLP	GCN	TMLP
Cora	73.57±0.98	88.04±1.26	88.26±1.60
CiteSeer	71.90±1.69	75.69±1.36	76.35±1.13
PubMed	86.90±0.74	87.94±0.64	87.58±0.44
Average	77.45	**83.89**	**84.06**

Based on the node representation, ❸ ***Test-Time Mixup* make the node representation more discriminative than MLP.** The visualization clearly illustrates that the node representations within the same class, as learned by TMLP, are clustered together. This clustering pattern demonstrates the advanced predictive power of TMLP, as it appears to have a keen ability to distinguish similar group classes effectively. Consequently, this capacity allows TMLP to deliver superior classification performance. The node representation clearly shows what TMLP does from the representation perspective.

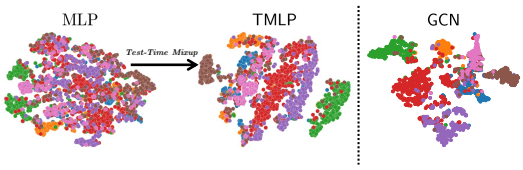

Figure 5: Visualization of node representations learned by MLP and TMLP. Node representations from the same class, after *Test-Time Mixup*, become clustered together.

4.2 How Does *Test-time Mixup* Benefit GCN? Investigating Decision Boundary

In this section, we evaluate the effect of *Test-Time Mixup* on the decision boundary change from MLP to TMLP. To do so, we compare the distance between node features and the decision boundary of MLP before

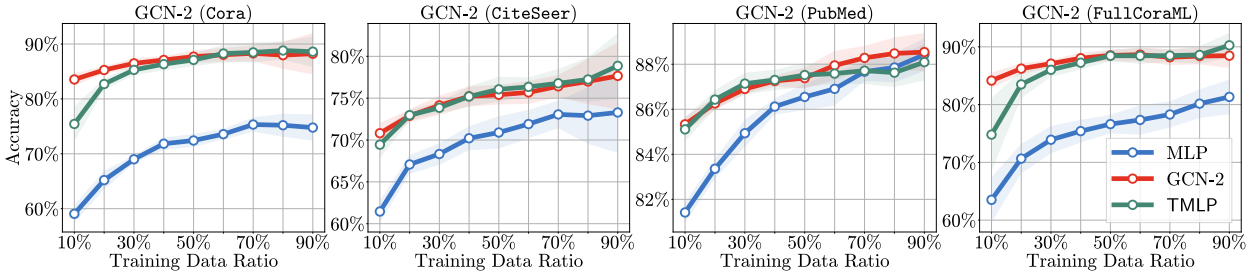

Figure 6: The performance comparison of the GCNs, MLP and TMLP (Ours). GCN-1 is a one-layer GCN, and GCN-2 is a two-layer GCN. The results show that TMLP achieves a comparable performance to GCN and SGC. Note that the architecture is essentially MLP in training time and GCN in the test time. More results on other datasets and GNNs are presented in Appendix C.2.

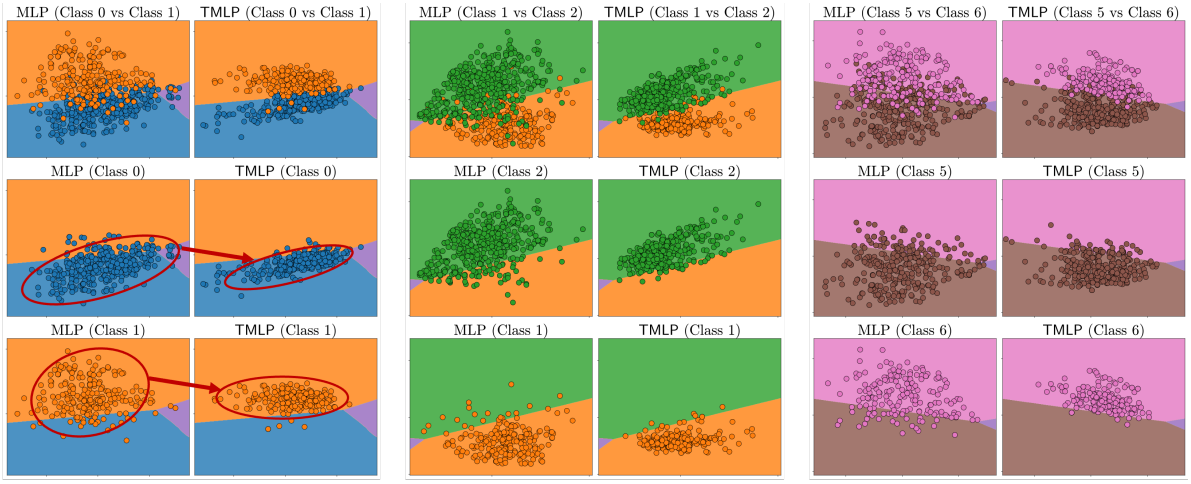

Figure 7: Decision Boundary of MLP and TMLP. TMLP can cause node features to move away from the learned boundary, as shown with . More experimental details are in Appendix C.3

and after applying *Test-Time Mixup*. After training a model with training-time data mixup, we evaluate its decision boundary, which separates different classes in the feature space. We then apply test-time mixup techniques to node features and reassess the decision boundary. Our findings indicate that ❹ ***Test-Time Mixup* can cause node features to move away from the learned boundary**, enhancing model robustness to unseen data. However, it's crucial to strike a balance between *Test-Time Mixup* and model performance, as overdoing Mixup may reduce accuracy.

5 Unifying HMLP and TMLP: Verifying the Main Results

In this section, we unify HMLP and TMLP into a single MLP, which integrates both *Homophily Relabel* and *Test-time Mixup*. They are supposed to be equivalent to graph convolution. To unify these two MLPs, we 1) relabel all the nodes in the graph with the mixed label $\hat{\mathbf{Y}} = \mathbf{A} \cdot \mathbf{Y}$. 2) train an MLP on all the nodes with $\hat{\mathbf{Y}}$ on relabeled graph, only using node features. 3) employ the well-trained MLP for inference with neighbor aggregation during testing. To test the efficacy of this combined approach, we perform experiments comparing the performance of the unified MLP with that of individual HMLP and TMLP, as well as other GCN. The experimental setup remains consistent with the previous setting with HMLP and TMLP. We present the results in Figure 8. ❺ **Unifying HMLP and TMLP can achieve comparable performance to original GNNs.**

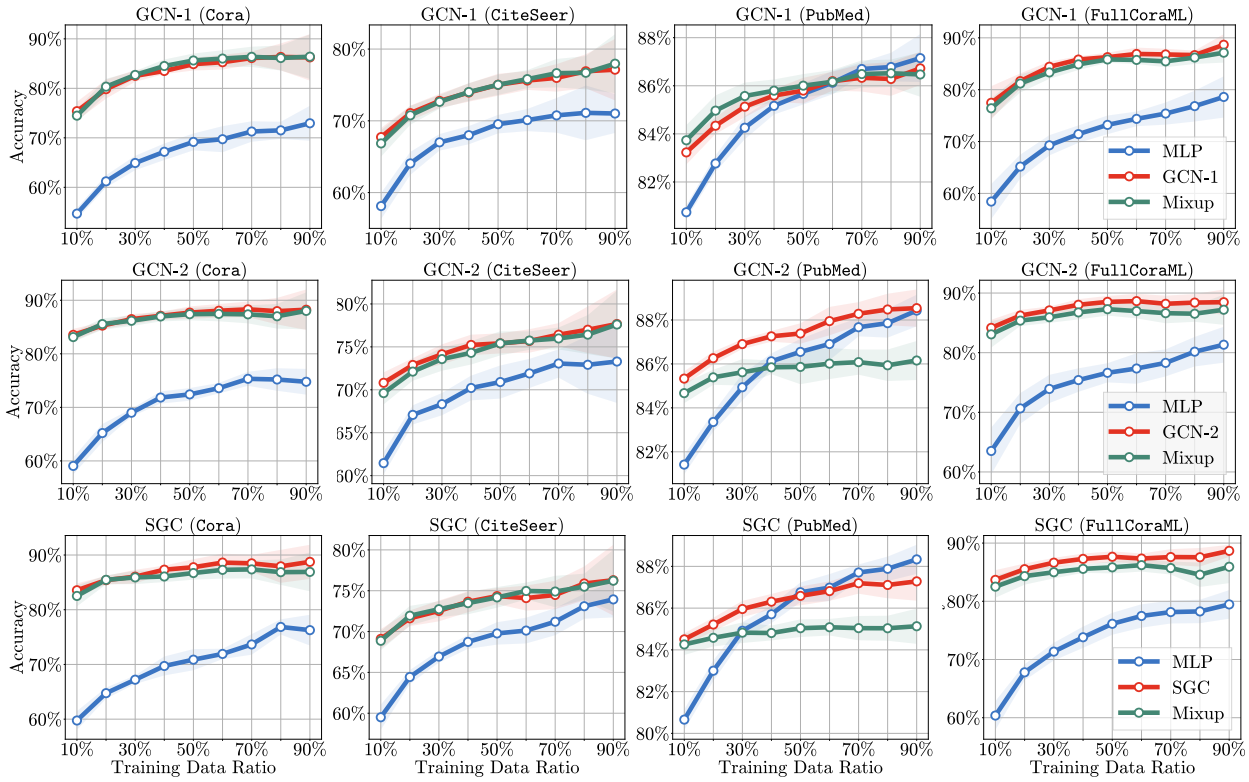

Figure 8: Unifying HMLP and TMLP. these methods together achieve performance comparable to GNNs (excluding the PubMed dataset, as the MLPs also exhibit better performance than GNNs.).

6 Discussion and Future Work

In this paper, we reveal the relation between graph neural networks and Mixup that graph neural networks are essentially Mixup, in the sense that they can be seen as a continuous interpolation between nodes in the graph. This equivalence allows for a new interpretation of GNNs as a form of regularization, and provides a new perspective on the design of GNNs.

- **Efficient Alternative Training and Inference Method for GNNs on Large Graph Data.** HMLP presents an efficient solution for both training and testing on large-scale graphs, as it completely eliminates the need for connection information during both stages. TMLP, though also MLP-based, emphasizes training efficiency. The improved training efficiency of TMLP can still be advantageous for reducing the training cost on large graph.
- **Opening a New Door for Understanding Graph Convolution.** The introduction of mixup, a simple node-level operation, offers a novel perspective on graph convolution by representing it in a mixup form. Traditional graph convolution employs the adjacency matrix of the entire graph, a concept that can be complex to comprehend or articulate. By approaching the understanding of graph neural networks from the entire graph level, we can demystify these complexities. This presents an essential avenue for exploration in future work related to our paper.

Future Work. Based on the above practical and theoretical potentials of the proposed methods in this paper. Here are some potential future works: 1) improve HMLP to make it more practical for large-scale graphs and the performance when the training data ratio is small. 2) The well-studied Mixup strategy would be helpful for understanding the expressiveness of graph neural networks.

Acknowledgements

This work was done while Xiaotian Han was an intern at Meta. We are grateful for the detailed reviews from the reviewers. Their thoughtful comments and suggestions were invaluable in strengthening this paper. Portions of this research were conducted with high performance research computing resources provided by Texas A&M University (https://hprc.tamu.edu). This work was supported in part by the National Science Foundation (NSF) IIS-2224843 and by the US Department of Transportation (USDOT) Tier1 University Transportation Center (UTC) Transportation Cybersecurity Center for Advanced Research and Education (CYBER-CARE) grant #69A3552348332.

References

Guillaume P Archambault, Yongyi Mao, Hongyu Guo, and Richong Zhang. Mixup as directional adversarial training. *arXiv preprint arXiv:1906.06875*, 2019.

Aseem Baranwal, Kimon Fountoulakis, and Aukosh Jagannath. Effects of graph convolutions in deep networks. *arXiv preprint arXiv:2204.09297*, 2022.

Aseem Baranwal, Kimon Fountoulakis, and Aukosh Jagannath. Effects of graph convolutions in multi-layer networks. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=P-73JPgRs0R.

Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3496–3507, 2021.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 3950–3957, 2021.

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

Xuheng Cai, Chao Huang, Lianghao Xia, and Xubin Ren. LightGCL: Simple yet effective graph contrastive learning for recommendation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=FKXVK9dyMM.

Kaidi Cao, Jiaxuan You, Jiaju Liu, and Jure Leskovec. Autotransfer: AutoML with knowledge transfer - an application to graph neural networks. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=y81ppNf_vg.

Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Yannick Hammerla, Michael M. Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=m1oqEOAozQU.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pp. 1725–1735. PMLR, 2020.

Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. On representing linear programs by graph neural networks. In *International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=cP2QVK-uygd.

Zizhang Chen, Peizhao Li, Hongfu Liu, and Pengyu Hong. Characterizing the influence of graph elements. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=51GXyzOKOp.

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.

Muthu Chidambaram, Xiang Wang, Yuzheng Hu, Chenwei Wu, and Rong Ge. Towards understanding the data dependency of mixup-style training. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ieNJYujcGDO.

Sanghyuk Chun, Seong Joon Oh, Sangdoo Yun, Dongyoon Han, Junsuk Choe, and Youngjoon Yoo. An empirical evaluation on robustness and uncertainty of regularization methods. *arXiv preprint arXiv:2003.03879*, 2020.

Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks? In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ayPPc0SyLv1.

Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi S. Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=kKF8_K-mBbS.

Leyan Deng, Defu Lian, Chenwang Wu, and Enhong Chen. Graph convolution network based recommender systems: Learning guarantee and item mixture powered strategy. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=aUoCgjJfmY9.

Francesco Di Giovanni, James Rowbottom, Benjamin P Chamberlain, Thomas Markovich, and Michael M Bronstein. Graph neural networks as gradient flows. *arXiv preprint arXiv:2206.10991*, 2022.

Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32, 2019.

Andrew Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. *arXiv preprint arXiv:2203.15544*, 2022.

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.

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.

Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pp. 3419–3430. PMLR, 2020.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2018.

Hongyu Guo, Yongyi Mao, and Richong Zhang. Mixup as locally linear out-of-manifold regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3714–3722, 2019.

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

Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. In *International Conference on Machine Learning*, pp. 8230–8248. PMLR, 2022a.

Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. Mlpinit: Embarrassingly simple gnn training acceleration with mlp initialization. *arXiv preprint arXiv:2210.00102*, 2022b.

Matej Hladiš, Maxence Lalis, Sebastien Fiorucci, and Jérémie Topin. Matching receptor to odorant with protein language and graph neural networks. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=q9VherQJd8_.

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.

Bo Hui, Da Yan, Xiaolong Ma, and Wei-Shinn Ku. Rethinking graph lottery tickets: Graph sparsity matters. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=fjh7UGQgOB.

Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *International Conference on Machine Learning*, pp. 5275–5285. PMLR, 2020.

Jang-Hyun Kim, Wonho Choo, Hosan Jeong, and Hyun Oh Song. Co-mixup: Saliency guided joint mixup with supermodular diversity. *arXiv preprint arXiv:2102.03065*, 2021.

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.

Alex Lamb, Vikas Verma, Juho Kannala, and Yoshua Bengio. Interpolated adversarial training: Achieving robust neural networks without sacrificing too much accuracy. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pp. 95–103, 2019.

Yi-Lun Liao and Tess Smidt. Equiformer: Equivariant graph attention transformer for 3d atomistic graphs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=KwmPfARgOTD.

Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. EXACT: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vkaMaq95_rX.

David Lopez-Paz, Ishmael Belghazi, Diane Bouchacourt, Elvis Dohmatob, Badr Youbi Idrissi, Levent Sagun, and Andrew M Saxe. An ensemble view on mixup, 2023. URL https://openreview.net/forum?id=k_iNqflnekU.

Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Tianyu Pang, Kun Xu, and Jun Zhu. Mixup inference: Better exploiting mixup to defend adversarial attacks. *arXiv preprint arXiv:1909.11515*, 2019.

Mahalakshmi Sabanayagam, Pascal Esser, and Debarghya Ghoshdastidar. Representation power of graph convolutions: Neural tangent kernel analysis. *arXiv preprint arXiv:2210.09809*, 2022.

Zhihao Shi, Xize Liang, and Jie Wang. LMC: Fast training of GNNs via subgraph sampling with provable convergence. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=5VBBA91N6n.

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

Ruoxi Sun, Hanjun Dai, and Adams Wei Yu. Does GNN pretraining help molecular representation? In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=uytgM9N0vlR.

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.

Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh Chawla. Learning MLPs on graphs: A unified view of effectiveness, robustness, and efficiency. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Cs3r5KLdoj.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

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

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pp. 6438–6447. PMLR, 2019.

Kun Wang, Yuxuan Liang, Pengkun Wang, Xu Wang, Pengfei Gu, Junfeng Fang, and Yang Wang. Searching lottery tickets in graph neural networks: A dual perspective. In *The Eleventh International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=Dvs-a3aymPe.

Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*, pp. 3663–3674, 2021.

Zichao Wang, Weili Nie, Zhuoran Qiao, Chaowei Xiao, Richard Baraniuk, and Anima Anandkumar. Retrieval-based controllable molecule generation. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=vDFA1tpuLvk.

Chunyu Wei, Jian Liang, Di Liu, and Fei Wang. Contrastive graph structure learning via information bottleneck for recommendation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=lhl_rYNdiH6.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, 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*, 32(1):4–24, 2020.

Jun Xia, Chengshuai Zhao, Bozhen Hu, Zhangyang Gao, Cheng Tan, Yue Liu, Siyuan Li, and Stan Z. Li. Mole-BERT: Rethinking pre-training graph neural networks for molecules. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=jevY-DtiZTR.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pp. 5453–5462. PMLR, 2018.

Chenxiao Yang, Qitian Wu, Jiahua Wang, and Junchi Yan. Graph neural networks are inherently good generalizers: Insights by bridging gnns and mlps. *International Conference on Learning Representations*, 2023.

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

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

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

Bohang Zhang, Shengjie Luo, Liwei Wang, and Di He. Rethinking the expressive power of GNNs via graph biconnectivity. In *International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=r9hNv76KoT3.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Zou. How does mixup help with robustness and generalization? In *International Conference on Learning Representations*, 2020.

Linjun Zhang, Zhun Deng, Kenji Kawaguchi, and James Zou. When and how mixup improves calibration. In *International Conference on Machine Learning*, pp. 26135–26160. PMLR, 2022a.

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.

Shuai Zhang, Meng Wang, Pin-Yu Chen, Sijia Liu, Songtao Lu, and Miao Liu. Joint edge-model sparse learning is provably efficient for graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=4UldFtZ_CVF.

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*, 2022b.

Yangtian Zhang, Huiyu Cai, Chence Shi, and Jian Tang. E3bind: An end-to-end equivariant network for protein-ligand docking. In *International Conference on Learning Representations*, 2023c. URL https://openreview.net/forum?id=sO1QiAftQFv.

ZAIXI ZHANG, Qi Liu, Shuxin Zheng, and Yaosen Min. Molecule generation for target protein binding with structural motifs. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Rq13idF0F73.

Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. Learning on large-scale text-attributed graphs via variational inference. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=q0nmYciuuZN.

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.

Jinhua Zhu, Kehan Wu, Bohan Wang, Yingce Xia, Shufang Xie, Qi Meng, Lijun Wu, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. $\mathcal{O}$-GNN: incorporating ring priors into molecular modeling. In *International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=5cFfz6yMVPU.

Zeyu Zhu, Fanrong Li, Zitao Mo, Qinghao Hu, Gang Li, Zejian Liu, Xiaoyao Liang, and Jian Cheng. $\rm a^2q$: Aggregation-aware quantization for graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=7L2mgi0TNEP.

Appendix

Table of Contents

A Related Works **16**
A.1 Graph Neural Networks . . . 16
A.2 Mixup and Its Variants . . . 17
A.3 Efficient Graph Neural Networks . . . 17

B Connection between Mixup and More Graph Convolutions **17**
B.1 Discussion on More Graph Convolutions . . . 17
B.2 Experiments on APPNP . . . 18
B.3 GNN with trainable mixing parameter λ . . . 18

C Additional Experiments **18**
C.1 More Experiments for HMLP . . . 18
C.2 More Experiments for TMLP . . . 20
C.3 More Experiments on Decision Boundary . . . 20
C.4 More Experiments on Deeper GCNs . . . 20
C.5 Experiments on More Datasets . . . 22

D Experiment Setting **22**
D.1 Running Environments . . . 23
D.2 Experiment Setup for Figure 4 . . . 23
D.3 Experiment Setup for Figure 6 . . . 23
D.4 Experiment Setup for Figure 8 . . . 23

E Datasets **23**

A Related Works

In this appendix, we discuss more related work to our paper.

A.1 Graph Neural Networks

Starting from graph convolutional network (Kipf & Welling, 2016a), graph neural networks have been applied to various graph learning tasks and show great power for graph learning tasks. Despite the practical power of graph neural network, understanding the generalization and working mechanism of the graph neural networks are still in its infancy (Garg et al., 2020; Zhang et al., 2023a; Yang et al., 2023; Baranwal et al., 2023). The previous works try to understand graph neural networks from different perspectives, such as signal processing (Nt & Maehara, 2019; Bo et al., 2021; Bianchi et al., 2021), gradient flow (Di Giovanni et al., 2022), dynamic programming(Dudzik & Veličković, 2022), neural tangent kernels (Yang et al., 2023; Du et al., 2019; Sabanayagam et al., 2022) and influence function (Chen et al., 2023b). There is also a line of works try to understand graph neural networks by analyzing the connection between GNNs and MLPs (Baranwal et al., 2023; Han et al., 2022b; Yang et al., 2023; Tian et al., 2023).

In our work, we understand graph neural networks through a fresh perspective, Mixup. We believe this work will inspire further research and lead to the development of new techniques for improving the performance and interpretability of GNNs.

A.2 Mixup and Its Variants

Mixup (Zhang et al., 2018) and its variants (Verma et al., 2019; Yun et al., 2019; Kim et al., 2020; 2021) are important data augmentation methods that are effective in improving the generalization performance of deep neural networks. Mixup is used to understand or improve many machine learning techniques. More specifically, deep neural networks trained with Mixup achieve better generalization (Chun et al., 2020; Zhang et al., 2020; Chidambaram et al., 2022), calibration (Thulasidasan et al., 2019; Zhang et al., 2022a), and adversarial robustness (Pang et al., 2019; Archambault et al., 2019; Zhang et al., 2020; Lamb et al., 2019). As a studies data augmentation method, Mixup can be understood as a regularization term to improve the effectiveness and robustness of neural network (Pang et al., 2019; Archambault et al., 2019; Zhang et al., 2020; Lamb et al., 2019). Mixup are also used to explain some other techniques, such as ensmeble (Lopez-Paz et al., 2023). Additionally, Mixup technique is also used to augment graph data for better performance of GNNs Han et al. (2022a); Wang et al. (2021).

In this work, we use Mixup to explain and understand the graph neural network, and based on the Mixup, we propose two kinds of MLP that achieve almost equivalent performance to graph neural networks. Also, our work is different form the works that chance the GNN training with Mixup, while we connect the graph convolution and Mixup.

A.3 Efficient Graph Neural Networks

Although the graph neural networks are powerful for graph learning tasks, they suffer slow computational problems since they involve either sparse matrix multiplication or neighbors fetching during the inference, resulting in active research on the efficiency of graph neural networks. The recent advances in efficient graph neural networks are as follows. Efficient graph neural network (Wang et al., 2023a; Shi et al., 2023; Hui et al., 2023; Zhang et al., 2023b; Zhu et al., 2023b) includes the sampling-based method (Hamilton et al., 2017; Chiang et al., 2019; Zeng et al., 2019), quantization (Liu et al., 2022), knowledge distillation (Zhang et al., 2021; Tian et al., 2023), and graph sparsification (Cai et al., 2020). There is a line of work to try to use MLP to accelerate the GNN training and inference (Zhang et al., 2022b; Wu et al., 2019; Frasca et al., 2020; Sun et al., 2021; Huang et al., 2020; Hu et al., 2021). Simplifying graph convolutional (SGC) (Wu et al., 2019) "simplifying" graph convolutional network by decoupling the neighbor aggregation and feature transformation, the following work such as You et al. (2020) employ the similar basic idea. Zhang et al. (2021) leverage knowledge distillation to transfer the knowledge from GNN to MLP. Han et al. (2022b) and Yang et al. (2023) transfer the weight from MLP to GNN. These kinds of methods use MLP for efficient graph neural networks.

Our proposed methods HMLP and TMLP are efficient for training and/or inference, which can be categorized as an efficient graph neural network. Besides, our proposed method provides a new perspective to understand graph neural networks.

B Connection between Mixup and More Graph Convolutions

B.1 Discussion on More Graph Convolutions

We examine the connection between complex graph convolutions used in various graph neural networks and Mixup. We aim to approximately rewrite complex graph convolutions (like those in graph attention networks) into Mixup forms. We present a general Mixup form of graph convolution for node n_i in the graph as follows

$$(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \left(\sum_{j \in \mathcal{N}_i} \lambda_j \mathbf{H}_j, \sum_{j \in \mathcal{N}_i} \lambda_j \mathbf{y}_i \right). \tag{9}$$

Equation (9) shows that graph neural networks architectures vary depending on: 1) how to select neighbors set ($\mathcal{N}_i$), 2) how to select aggregation coefficient (λ_j), and 3) what is mixed up ($\mathbf{H}_j$). For example, if we set $\mathcal{N} = \mathcal{N}_i^2$, $\lambda = \frac{1}{|\mathcal{N}_i^2|}$, the above equation will be SGC. However, there are a lot of advanced graph neural networks, such as GAT (Veličković et al., 2018)) (attention-based GNN), and PPNP (Gasteiger et al., 2018)

Table 4: Examples of the relation between graph convolutions to Mixup at k-layer.

	$(\mathcal{N}_i)$	(λ_j)	$(\mathbf{H}_j)$
GCN	$\mathcal{N}_i$	$\mathbf{a}_i$	$\mathbf{H}^k$
SGC	$\mathcal{N}_i^k$	$\mathbf{a}_i^k$	$\mathbf{X}$
GAT	$\mathcal{N}_i$	Attention Score	$\mathbf{H}^k$
PPNP	$\mathcal{N}_i$	$\mathbf{a}_i$	Logit

(MLP as the backbone). Table 4 shows the relationship between those graph convolutions to Mixup. To include residual connections in some graph convolutions (Chen et al., 2020; Xu et al., 2018), we can expand the neighbors set $\mathcal{N}_i$ of node n_i to include its node representation of the previous time step.

B.2 Experiments on APPNP

APPNP essentially performs the neighbor aggregation in the last layer. Thus, TMLP and HMLP, according to APPNP, are equivalent to GCN. We report the performance in the table below. The results show that TMLP and HMLP achieve better performance than APPNP.

Table 5: Performance Metrics of Different Methods

MLP	**PPNP**	**TMLP**	**HMLP**
73.57	83.80	88.26	86.42

B.3 GNN with trainable mixing parameter λ

In this experiment, we adopted a method similar to GAT to examine the performance of our proposed method, with the mixing parameter λ being learned during the training process. The Softmax (Li et al., 2020) is a learnable aggregation operator that normalizes the features of neighbors based on a learnable temperature term. We report the performance in the table below. The results show that the TMLP performed worse than the MLP. This indicates that the learnable mixing parameter λ does not work well for our proposed method

Table 6: Performance Metrics of Different Methods

MLP	**GAT-alike**	**TMLP**
73.57	88.42	65.62

C Additional Experiments

This appendix presents more experiment results on HMLP and TMLP, including 1) comparison experiments of MLP, GCN, and HMLP/TMLP; 2) more experimental results on decision boundary.

C.1 More Experiments for HMLP

We present further experiments comparing MLP, GCN, and HMLP. These experiments complement those discussed in Figure 4. We expand on that work by evaluating these methods with more GNN architectures. The results are presented in Figure 9.

The results on more GNN architectures indicate that our proposed HMLP method not only outperforms traditional MLP, but also attains a performance that is largely on par with GNNs in the majority of instances.

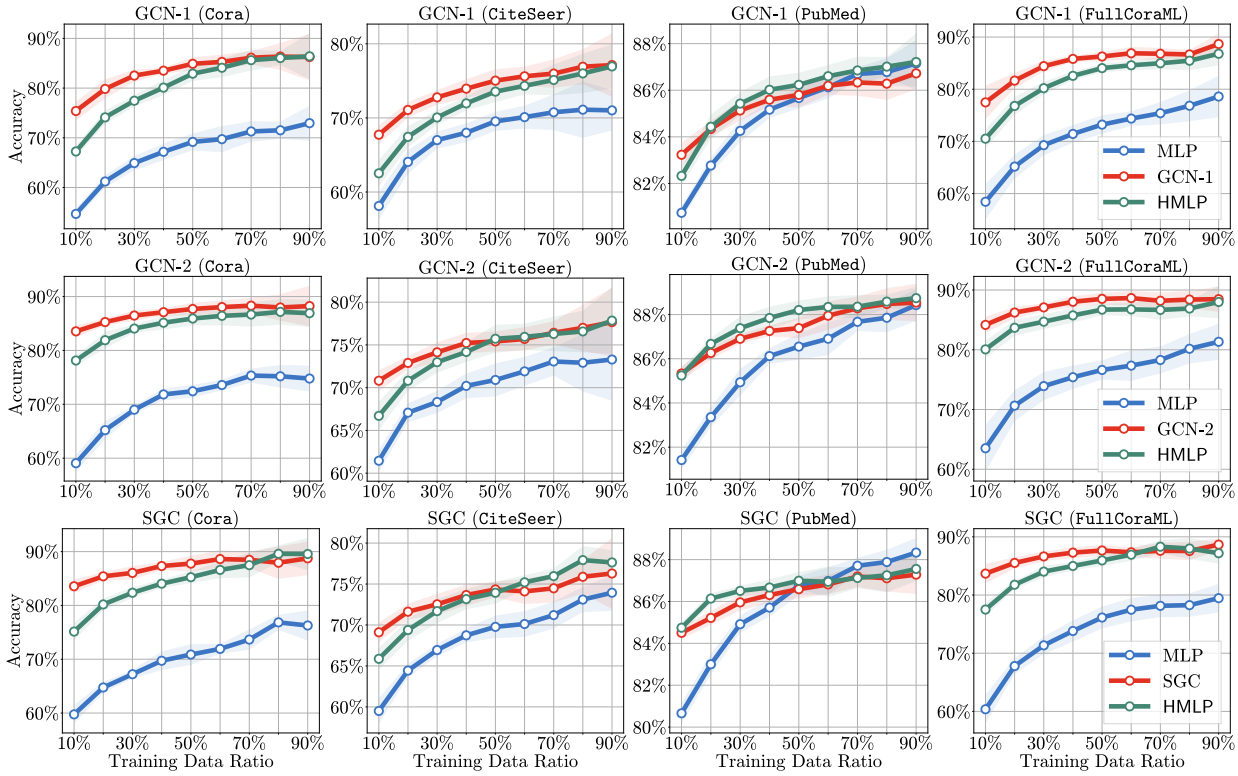

Figure 9: The performance comparison of the GCN, MLP and HMLP (Ours) with GNN architectures. The x-axis represents the ratio of training data, and the y-axis represents the classification accuracy. The results show that our proposed method (HMLP) achieves comparable performance to GCN.

C.2 More Experiments for TMLP

We present further experiments comparing MLP, GCN, and TMLP. These experiments complement those discussed in Figure 6. We expand on that work by evaluating these methods with more GNN architectures. The results are presented in Figure 10.

With the evaluation of an expanded selection of GNN architectures, the results reveal that our proposed method TMLP typically achieves performance comparable to traditional GNNs in most situations.

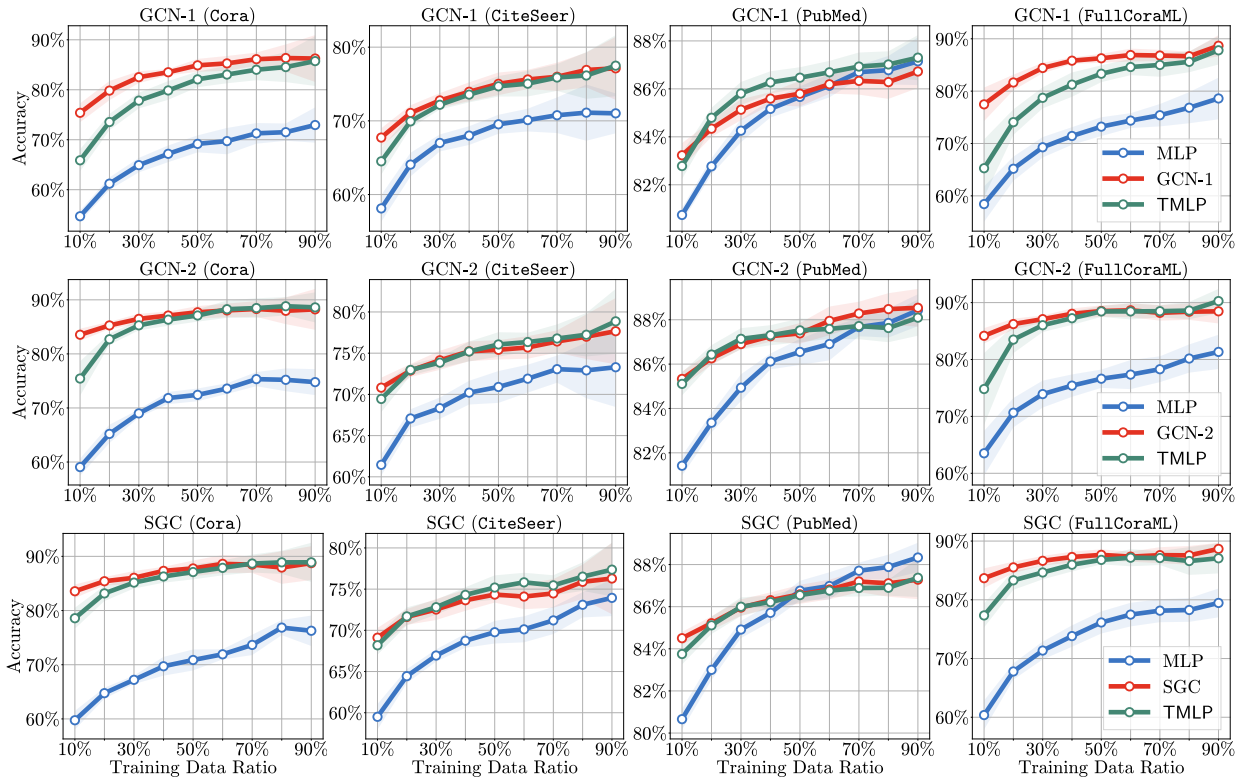

Figure 10: The performance comparison of the GCNs, MLP and TMLP (Ours). GCN-1 is a one-layer GCN, and GCN-2 is a two-layer GCN. The results show that TMLP achieves a comparable performance to GCN and SGC. Note that the architecture is essentially MLP in training time and GCN in the test time.

C.3 More Experiments on Decision Boundary

We provide additional experimental results pertaining to the decision boundary of TMLP. Our findings indicate that TMLP tends to shift the node features in the test set away from the pre-learned decision boundary by TMLP.

C.4 More Experiments on Deeper GCNs

We conducted additional experiments on multi-layer graph neural network (SGC) on the Cora dataset. We use the 20%\40%\40% for train\val\test data split. The test accuracy was determined based on the results from the validation dataset. The test accuracy is reported in the following table. We observe from the results that our proposed HMLP+TMLP method achieves performance comparable to that of SGC when multiple graph convolutions are utilized.

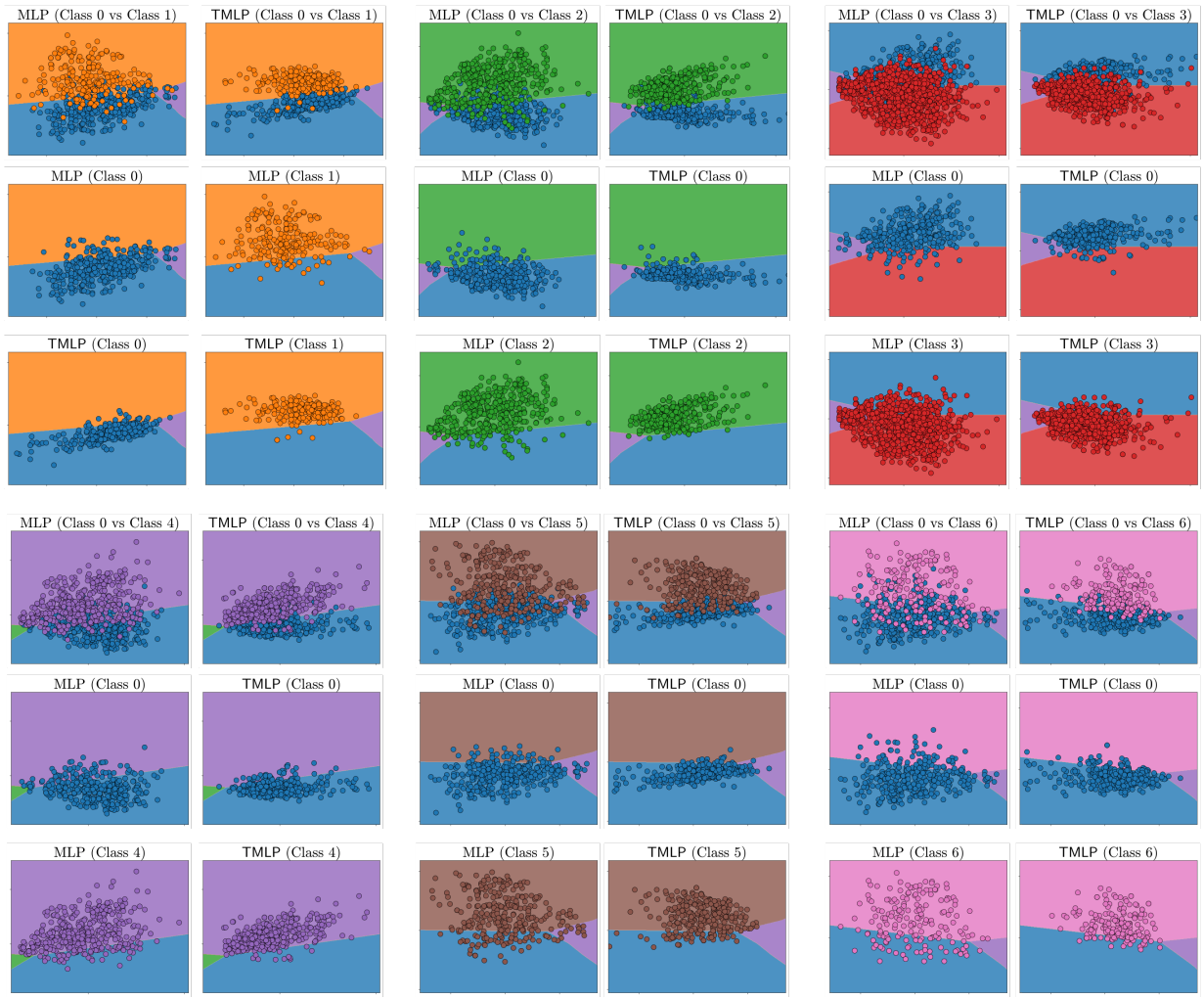

Figure 11: Decision Boundary of MLP and TMLP. TMLP can cause node features to move away from the learned boundary.

Table 7: Performance comparison of GNNs and our proposed method with different layers.

	1-layer	**2-layer**	**3-layer**	**4-layer**
MLP	69.11	68.95	68.39	69.22
SGC	82.38	84.50	84.23	83.95
HMLP	78.56	80.41	80.96	81.15
TMLP	82.10	83.76	83.49	83.67
HMLP +TMLP	83.12	83.95	84.87	84.41

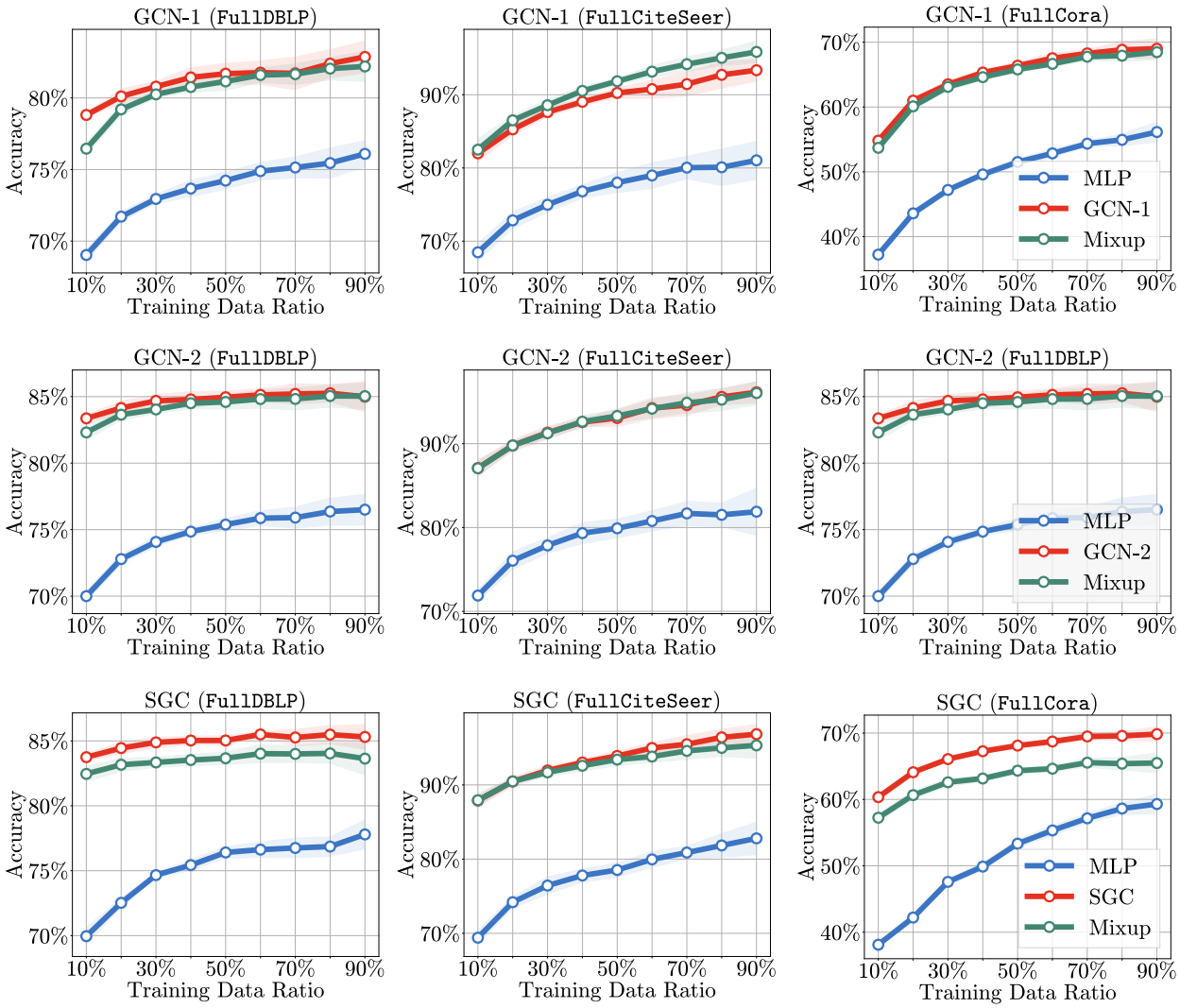

Figure 12: The additional experimental result on unifying HMLP and TMLP. These methods together achieve performance comparable to GNNs on these additional datasets. The additional results show that graph convolution is a mixup under our proposed modifications.

C.5 Experiments on More Datasets

We conducted additional experiments on more datasets, including FullCora, FullSiteSeer, FullDBLP, and the results are presented in Figure 12. The additional results provide more evidence that graph convolution is a mixup under our proposed modifications.

D Experiment Setting

In this appendix, we present the running environment for our experiment and the experimental setup for all the experiments.

D.1 Running Environments

We run all the experiments on NVIDIA RTX A100 on AWS. The code is based on PyTorch [3] and torch_geometric [4]. The datasets used in this paper are built-in datasets in torch_geometric, which will be automatically downloaded via the torch_geometric API.

D.2 Experiment Setup for Figure 4

We provide details of the experimental setup associated with Figure 4. The primary objective of this experiment was to compare the performance of our proposed HMLP model with that of the GCN model. We utilized diverse datasets to facilitate a comprehensive evaluation. We set the learning rate to 0.1 for all methods and datasets, and each was trained for 400 epochs.Moreover, we did not use weight decay for model training. The test accuracy reported is based on the best results from the validation set. The node classification accuracy served as the evaluation metric for this experiment.

D.3 Experiment Setup for Figure 6

We provide details of the experimental setup associated with Figure 6. We set the learning rate to 0.1 for all methods, and each was trained for 400 epochs. Additionally, weight decay was not utilized in this process. The test accuracy reported is based on the best results from the validation set. The node classification accuracy served as the evaluation metric for this experiment.

D.4 Experiment Setup for Figure 8

We provide details of the experimental setup associated with Figure 8. We set the learning rate to 0.1 for all methods, and each was trained for 400 epochs. Additionally, weight decay was not utilized in this process. The test accuracy reported is based on the best results from the validation set. The node classification accuracy served as the evaluation metric for this experiment.

E Datasets

In this experiment, we use widely adopted node classification benchmarks involving different types of networks: three citation networks (`Cora`, `CiteSeer` and `PubMed`). The datasets used in this experiment are widely adopted node classification benchmarks. These networks are often used as standard evaluation benchmarks in the field of graph learning.

- `Cora`: is a widely used benchmark data for graph learning, which is a citation network, where each node represents a research paper, and edges denote the citation relation. Node features are extracted from the abstract of the paper.
- `CiteSeer` is another citation network used frequently in GNN literature. It follows a similar structure to the `Cora` dataset, where each node represents a scientific document, and the edges are citation information between them. The node features are derived from the text of the documents.
- `PubMed`: PubMed is a citation network composed of scientific papers related to biomedicine. As with the other datasets, nodes represent documents, and edges indicate citation relationships. Node features correspond to term frequency-inverse document frequency (TF-IDF) representations of the documents.
- `FullCoraML` is a subset of the full `Cora` dataset. It retains the same structure as the `Cora` dataset but is restricted to documents and citations within the field of machine learning. The node features are the bag-of-words representation for the papers.

[3] https://pytorch.org/
[4] https://pyg.org/