

Weight Scope Alignment: A Frustratingly Easy Method for Model Merging

Yichu Xu^{ab}, Xin-Chun Li^{ab}, Le Gan^{ab} and De-Chuan Zhan^{ab,*}

^aSchool of Artificial Intelligence, Nanjing University, China

^bNational Key Laboratory for Novel Software Technology, Nanjing University, China

Abstract. Merging models becomes a fundamental procedure in some applications that consider model efficiency and robustness. The training randomness or Non-I.I.D. data poses a huge challenge for averaging-based model fusion. Previous research efforts focus on element-wise regularization or neural permutations to enhance model averaging while overlooking weight scope variations among models, which can significantly affect merging effectiveness. In this paper, we reveal variations in weight scope under different training conditions, shedding light on its influence on model merging. Fortunately, the parameters in each layer basically follow the Gaussian distribution, which inspires a novel and simple regularization approach named **Weight Scope Alignment (WSA)**. It contains two key components: 1) leveraging a target weight scope to guide the model training process for ensuring weight scope matching in the subsequent model merging. 2) fusing the weight scope of two or more models into a unified one for multi-stage model fusion. We extend the WSA regularization to two different scenarios, including Mode Connectivity and Federated Learning. Abundant experimental studies validate the effectiveness of our approach.

1 Introduction

As a technique for combining multiple deep models into a single model, model fusion [36, 26] [36] has gained widespread applications across various domains, including Mode Connectivity [4, 9, 10] and Federated Learning [1, 40, 55]. First, the model interpolation could shed light on the properties of the mode connectivity in neural networks [16, 11, 14]. Then, due to data privacy protection, transmitting intermediate models across edge nodes and fusing them on the server has been the common procedure in federated learning [50, 35, 39]. To be brief, model fusion matters a lot in these applications and has attracted a wide range of research interest. The primary goal of model fusion is to retain the capabilities of the original models while achieving improved generalization, efficiency, and robustness.

The coordinate-based parameter averaging is the most common approach for model fusion in deep neural networks [40, 36, 52]. The research on mode connectivity involves a linear or piece-wise interpolation between models [16, 48], while federated learning takes the averaging of local models from edge nodes for aggregation [40, 7]. Although parameter averaging exhibits favorable properties, it may not perform optimally in more complex training scenarios, especially when faced with various training conditions or Non-Independent

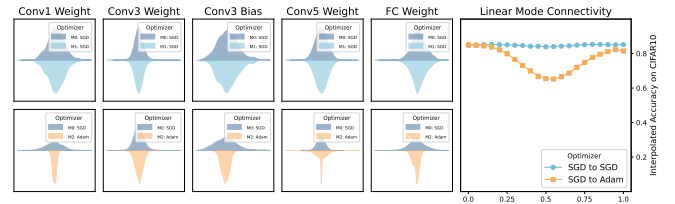


Figure 1: Left: Similar optimizers yield similar weight distributions. Conversely, different optimizers lead to distinct weight profiles. **Right:** Models trained from different training conditions (e.g., the used optimizer) tend to produce poorer model interpolations. Details are in Section 5.1.

and Identically Distributed (Non-I.I.D.) data. For example, the Non-I.I.D. data in federated learning means that the data of local nodes are naturally heterogeneous, making the model aggregation suffer from diverged update directions [21, 24]. Additionally, the property of permutation invariance that neural networks own exacerbates the challenge of model fusion because of the neuron misalignment phenomenon [10, 57, 5, 15]. Hence, solutions have been proposed from the aspect of element-wise regularization [35, 1, 7] or mitigating the permutation invariance [37, 3, 45, 43]. Few of these methods, however, have considered the impact of weight ranges across models on model fusion.

In this paper, we investigate the influence of different training conditions on model weight distributions (defined as Weight Scope) and further study model merging under various weight scopes. We first conduct several experiments under various training hyper-parameters or data quality conditions and find that the weight scopes of the converged models differ a lot, a phenomenon we define as “weight scope mismatch”. Figure 1 illustrates the model weight distributions under different training conditions, revealing noticeable differences despite all distributions being approximated by Gaussian distributions. Specifically, the top five sub-figures are parameters from models that use the same optimizer, while the bottom ones take different optimizers. In the rightmost of Figure 1, the linear interpolation results that reflect the mode connectivity property are also provided. Clearly, the mismatched weight scope leads to a worse linear interpolation, highlighting the impact of weight range inconsistency on model fusion. To intuitively explain, parameters with similar distributions can be aggregated more easily, whereas those with dissimilar distributions often present challenges in model merging.

* Corresponding Author. Email: zhandc@nju.edu.cn

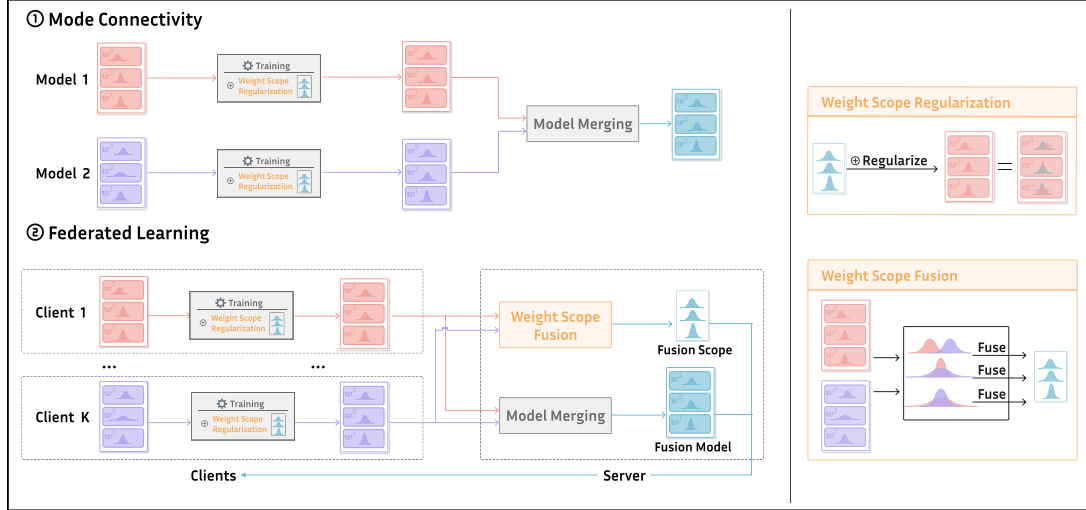


Figure 2: The Framework of Weight Scope Training Alignment and Its Applications. **Left:** This method can be adapted to various applications of model merging, such as Model Connectivity and Federated Learning. Mode Connectivity includes a single model fusion, while Federated Learning requires multi-stage fusion. **Right:** The method comprises two components: weight scope regularization and weight scope fusion.

Fortunately, the parameters in each layer follow a very simple distribution, i.e., Gaussian distribution. The simple distribution inspires us a novel and easy way of aligning parameters. We leverage a target weight scope to guide the model training process, ensuring weight alignment and scope matching in subsequent model merging. For more complex multi-stage fusion, we calculate the mean and variance of parameter weights in the to-be-merged models, then aggregate these statistics into unified one as the target weight scope. The proposed method is named **Weight Scope Alignment (WSA)**, and the above steps are named weight scope regularization and weight scope fusion, respectively. The whole procedure is illustrated in Figure 2. We apply WSA to scenarios of mode connectivity and federated learning for exploring the advantages of WSA compared with corresponding solutions in these areas. Our work aims to align the weights as closely as possible with a given distribution during training, thereby enhancing the match in weight ranges and facilitating model merging. Compared with other similar regularization methods, e.g., the weight decay and the proximal term, the proposed WSA seeks to balance specificity and generality, addressing the shortcomings of existing methods while optimizing for effective model fusion. Our contributions are as follows: 1) *as far as we know, our work is the first to formally study the impact of weight scope on model fusion*; 2) *the proposed WSA is simple yet effective, which is verified on two applications via abundant experimental studies*.

2 Related Works

Model fusion is a fundamental technique in several applications. The related scenarios and solutions are introduced as follows.

Model Merging in Mode Connectivity. Visualizing loss landscape is an intuitive way to understand the mode connectivity [33, 17, 38], where the landscape is shown in a 2-dim or 3-dim space via model interpolations. The model interpolations between the initialization and the converged model could reflect the monotonic linear interpolation phenomenon [16, 11, 51, 46]. [13] bridges the model connectivity and lottery ticket hypothesis [12], proposing a method

for model pruning. Mode connectivity is also related to the model optimization and generalization [29, 8].

The work [16] also points out that two independent minima suffer a barrier in their linear interpolation, which attracts several solutions to decrease the barrier. [9, 14] make a notable discovery that the independent minima are possible to be connected via a simple piece-wise or quadratic curve. [42, 52] find that the minima fine-tuned from the same pre-trained model could mitigate the barrier in linear interpolation. [10] guesses that the independent minima are located in the same basin with the consideration of permutation invariance, conjecturing that the minima matched via the simulated annealing algorithm encounter no barrier. [45] propose the weight-based and activation-based matching method via the optimal transport. [47] decreases the barrier via both the permutation alignment and the quadratic curve. [3] employs three distinct neuron-matching methods to corroborate the low-barrier hypothesis. Although the permutation invariance is considered in these works, the mismatch of weight scope in neural networks is also fundamental to model fusion. Our proposed method could further decrease the barrier on the basis of these works.

Model Merging in Federated Learning. Federated learning (FL) aims to break the limitation of data privacy, utilizing a server that collaborates with client devices to train a model [55]. As the most standard algorithm in FL, FedAvg [40] takes a simple coordinate-based parameter averaging on the server to accomplish the model fusion process. A huge challenge that FL faces is the Non-I.I.D. data [21, 28], where the inherent data heterogeneity leads to weight divergence during local training [24]. Applying coordinate-wise regularization on local models is a popular solution to solve the Non-I.I.D. challenge in FL. [34] claims that weight decay [32] can lead to divergent optimization objectives among Non-I.I.D. clients in FL. [60, 35] introduce a proximal term in local optimization. This term helps align local optimization more closely with given model parameters, facilitating model aggregation. Additionally, [28] implements constraints on the local gradient directions of each client, nudging them closer to a global direction. However, the coordinate-wise regu-

larization tends to be overly specific, which may induce perturbations to model training.

Additionally, the permutation invariance could also make the local models misaligned in FL, which is harder for aggregation. [57, 50] consider the permutation invariance of neural networks, rearranging the order of neurons can result in multiple models with the same functionality as the original network. [56] introduces a group alignment method, while [37] designs the position-aware neurons to align parameters. These works mainly focus on the permutation invariance in FL, but they have not analyzed the influence of different model weight ranges on model fusion.

3 Proposed Methods

3.1 Preliminaries

We introduce a collection of models, denoted as \mathcal{K} , with the size of the collection $|\mathcal{K}| \geq 1$. Each model within this collection is constructed based on the same underlying model architecture. The distinctiveness among the models in \mathcal{K} arises from the variations in their weight layers, which are derived from training under various hyperparameters or different data.

To encapsulate the statistical characteristics of the weight layers across different models, we denote the weight matrix of the ℓ^{th} layer in the k^{th} model as \mathbf{w}_k^ℓ . Our assumption is that the elements within this matrix follow a Gaussian distribution which is characterized by its mean μ_k^ℓ and standard deviation σ_k^ℓ . The assumption is rational and we will empirically present the weight distributions of the converged model in Section 5.1. Formally, the distribution of the weight matrix \mathbf{w}_k^ℓ is as follows:

$$p(\mathbf{w}_k^\ell) = \mathcal{N}(\mu_k^\ell, (\sigma_k^\ell)^2), \quad (1)$$

$$\mu_k^\ell = \frac{1}{|\mathbf{w}_k^\ell|} \sum_{w \in \mathbf{w}_k^\ell} w, \quad \sigma_k^\ell = \sqrt{\frac{1}{|\mathbf{w}_k^\ell|} \sum_{w \in \mathbf{w}_k^\ell} (w - \mu_k^\ell)^2}. \quad (2)$$

In Equation 2, we apply Maximum Likelihood Estimation to derive the standard deviation σ_k^ℓ and mean μ_k^ℓ of the weight \mathbf{w}_k^ℓ .

3.2 Weight Scope Alignment

Firstly, we introduce Weight Scope Regularization. Then, for more complex multi-stage fusion, we propose Weight Scope Fusion for better target alignment.

Weight Scope Regularization. Given a weight distribution $\mathcal{N}(\mu, \sigma^2)$ and a target weight distribution $\mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2)$, our method endeavors to ensure consistency between them. This consistency is crucial for guaranteeing scope-matched distribution among the new models in the subsequent model fusion. To achieve this, we employ a divergence measure known as the Kullback-Leibler (KL) divergence, specifically focusing on calculating the divergence between the two univariate Gaussian distributions, which is given by:

$$D_{\text{KL}} = \log \left(\frac{\tilde{\sigma}}{\sigma} \right) + \frac{\sigma^2 + (\mu - \tilde{\mu})^2}{2\tilde{\sigma}^2} - \frac{1}{2}, \quad (3)$$

where $\tilde{\mu}, \tilde{\sigma}$ are hyperparameters. By minimizing the KL divergence between the training models' weight distribution and their goal, it ensures that the weight distributions are closely aligned, facilitating more effective and harmonious model fusion.

Weight Scope Fusion. In some complex scenarios with large amounts of models and multiply stages of model merging, a given pre-defined weight distribution is not adaptable. Therefore, we propose a method named Weight Scope Fusion to enhance the applicability of Weight Scope Regularization. Focusing on the weights of the ℓ^{th} layer, we assume that each weight \mathbf{w}_k^ℓ , $k \in \mathcal{K}$, follows its own Gaussian distribution, and they are independent of each other. We can get the fused Gaussian distribution $\mathcal{N}(\tilde{\mu}^\ell, (\tilde{\sigma}^\ell)^2)$ by:

$$\tilde{\mu}^\ell = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \mu_k^\ell, \quad \tilde{\sigma}^\ell = \frac{1}{\sqrt{|\mathcal{K}|}} \sqrt{\sum_{k \in \mathcal{K}} (\sigma_k^\ell)^2}. \quad (4)$$

3.3 Analysis and Comparisons with Other Methods

We next provide some analysis of the proposed simple method, especially the comparisons with existing works.

Comparison with Weight Decay. Weight decay [32] stands as a cornerstone in model regularization, advocating for weight constraints that push the weights towards zero and promote uniformity. Considering a weight vector, denoted as $\mathbf{w} \in \mathbb{R}^n$, with elements $[\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n]$. For ease of analysis, we define the mean, standard deviation, and L2 norm of this vector as follows: $\mu = \frac{1}{n} \sum_{i=1}^n \mathbf{w}_i$, $\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (\mathbf{w}_i - \mu)^2}$, $\|\mathbf{w}\|_2 = \sqrt{\sum_{i=1}^n \mathbf{w}_i^2}$. The weight decay term is formulated as $\frac{\lambda}{2} \|\mathbf{w}\|_2^2$. We can express weight decay in terms of μ and σ as follows:

$$\frac{\lambda}{2} \|\mathbf{w}\|_2^2 = \frac{\lambda n}{2} (\sigma^2 + \mu^2). \quad (5)$$

The weight decay term brings the mean and variance of model weights close to zero, and several research [1, 28] has designed FL algorithms with adjustment of weight decay term in local training. However, the impact of weight decay on models can not determine the shape of weight distributions and could not align parameter distributions under different conditions. Hence, it does not necessarily lead to a harmonization of the model scopes before fusion.

Comparison with Proximal Term. The proximal term is used to keep the model weight \mathbf{w} and another one $\tilde{\mathbf{w}} \in \mathbb{R}^n$ as close as possible during the update of \mathbf{w} , which is represented as $\|\mathbf{w} - \tilde{\mathbf{w}}\|_2^2$. In FedProx [35], the proximal term constrains the local models to stay close to the global model, thereby ensuring stability during model fusion. For clarity, we also define $\tilde{\mu}$ and $\tilde{\sigma}$ as the mean and standard deviation of vectors $\tilde{\mathbf{w}}$. The proximal term can be decomposed as follows:

$$\|\mathbf{w} - \tilde{\mathbf{w}}\|_2^2 = n\sigma^2 + n\mu^2 + n\tilde{\sigma}^2 + n\tilde{\mu}^2 - 2 \sum_{i=1}^n \mathbf{w}_i \tilde{\mathbf{w}}_i. \quad (6)$$

Because the part of $n\tilde{\sigma}^2 + n\tilde{\mu}^2$ can be seen as a constant term, the proximal term differs from weight decay by only an additional term of $-2 \sum_{i=1}^n \mathbf{w}_i \tilde{\mathbf{w}}_i$, which encourage the weights in \mathbf{w} to align with the direction of $\tilde{\mathbf{w}}$ as closely as possible. However, the restriction is too strict because it requires the direction alignment for each specific element. This may limit the normal training process and perturb the effects of other loss functions.

In the context of model fusion, weight decay represents a more flexible approach, while the proximal term seems to introduce directional constraints additionally. Both of them have overlooked the influence of weight scope on model fusion and are unable to align weight scopes, as demonstrated in Figure 11. Our proposed method aims to achieve consistency in model weight ranges for improved model fusion results.

Comparison with Network Invariance. Some studies have investigated the impact of permutation invariance on mode connectivity [10, 45, 3, 47, 4, 15, 5] and model aggregation in federated learning [57, 37, 50]. A more detailed introduction of these studies has been presented in Section 2. However, the weight scope mismatch could make the alignment algorithm inaccurate, especially the weight-based ones [45]. Additionally, even if the neurons are aligned in order, their weight scopes are still mismatched, which could also lead to performance degradation. Several works also study the scale invariance, e.g., [43] normalizes the model parameters layer-wisely and then searches the proper neural permutations, and [27] studies the relative scale of weight and bias in monotonic linear interpolation. None of these works formally studies the impact of weight scope on model fusion.

4 Applications of WSA

This section presents the applications of Weight Scope Alignment to mode connectivity and federated learning. They can be seen as one-stage and multi-stage model fusion scenarios, respectively. While the specific processes or the model set to be fused may differ across them, they all share the common thread of leveraging the proposed WSA as a constraint during training.

4.1 Mode Connectivity

Given two well-trained models, \mathbf{w}_1 and \mathbf{w}_2 , the loss barrier along their linear interpolation path is:

$$\max_{\alpha \in [0,1]} \mathcal{L}((1-\alpha)\mathbf{w}_1 + \alpha\mathbf{w}_2) - [(1-\alpha)\mathcal{L}(\mathbf{w}_1) + \alpha\mathcal{L}(\mathbf{w}_2)], \quad (7)$$

where α is the interpolation coefficient, and the loss barrier represents the point of maximum loss increase along the linear interpolation path. A higher loss barrier suggests that the two models may not be in the same basin within the loss landscape, while a lower loss barrier indicates linear mode connectivity.

Both OTFusion [45] and Git Re-basin [3] have improved the way they perform model interpolation by considering neuron matching. They calculate matching relationships Π between the weights of each layer in the two models, using a permutation matrix or optimal transport matrix, leading to the fusion formula:

$$(1-\alpha)\mathbf{w}_1 + \alpha\Pi\mathbf{w}_2. \quad (8)$$

Our approach can easily be integrated into the aforementioned model interpolation methods to achieve improved mode connectivity. For separately trained models, they are typically randomly initialized from the same distribution, and their weight scopes are similar. The mean $\tilde{\mu}$ and standard deviation $\tilde{\sigma}$ of the target weight distribution are the hyperparameters and keep invariant during each model’s training. We incorporate a Weight Scope Regularization term (i.e., Equation 3) to ensure that the weight range remains close to the target weight scope. With our method, the weight scope of the converged models are matched, which is beneficial for searching the matching matrix.

4.2 Federated Learning

Model fusion is a fundamental procedure in Federated Learning (FL), i.e., improving the performance of joint training by merging models trained on different data sources. Its challenge lies in the fact

Algorithm 1 FedAvg with WSA

```

1: Input: model  $\mathbf{w}$ , number of rounds  $T$ , local iteration steps  $\tau$ ,
   parameters  $\eta, \varepsilon$ ,
2: for  $t = 0, \dots, T - 1$  communication rounds do
3:   Global server:
4:     Send  $\mathbf{w}, \tilde{\mathbf{p}} = \{\tilde{p}^\ell = (\tilde{\mu}^\ell, \tilde{\sigma}^\ell)\}_{\ell \in [L]}$  to all clients
5:   Client  $k \in \mathcal{K}$  in parallel do:
6:     Set  $\mathbf{w}_k \leftarrow \mathbf{w}$ 
7:     for  $s = 0, \dots, \tau - 1$  local iterations do
8:       Update  $\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta \nabla \mathcal{L}(\mathbf{w}_k; \mathcal{D}_k, \tilde{\mathbf{p}})$  using loss in
   Equation 10
9:     Send  $\mathbf{w}_k, \{\mu_k^\ell, \sigma_k^\ell\}_{\ell \in [L]}$  to the server
10:  Global server:
11:    Update global model  $\mathbf{w} \leftarrow \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \mathbf{w}_k$ 
12:    Update  $\tilde{\mu}^\ell, \tilde{\sigma}^\ell$  using Equation 4

```

that real-world data distributions across different clients are heterogeneous, leading to a phenomenon known as “client drift” during local client training, which impacts the overall performance of FL. Some recent research has addressed this issue from a Non-I.I.D. perspective, proposing various improved methods.

We use the previously defined \mathcal{K} to represent the set of clients, with each client associated with the data distribution \mathcal{D}_k . The objective of FL is:

$$\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}), \quad f(\mathbf{w}) \triangleq \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \mathcal{L}(\mathbf{w}; \mathcal{D}_k), \quad (9)$$

where $\mathcal{L}(\cdot)$ evaluates the loss for each data sample on model \mathbf{w} . Cross-entropy is commonly used as the loss function. To apply our method in FL, we additionally upload the weight scope (i.e., Equation 2) to the server after each local training procedure. The server then performs the fusion of these weight scopes (i.e., Equation 4) and sends the merged weight scopes $\tilde{\mathbf{p}}$ back to the clients. During local client training, the optimization proceeds as follows:

$$\mathcal{L}_{\text{local}}(\mathbf{w}; \mathcal{D}_k, \tilde{\mathbf{p}}) = \mathcal{L}(\mathbf{w}; \mathcal{D}_k) + \lambda \sum_{\ell=1}^L D_{\text{KL}}(p(\mathbf{w}^\ell), \tilde{p}^\ell), \quad (10)$$

where λ is used to control the strength of weight scope alignment, and \tilde{p}^ℓ is the fused weight scope of the ℓ^{th} layer. The full framework of FedAvg [40] with WSA is outlined in Algorithm 1.

5 Experiments

In Section 5.1, we verify that different training conditions lead to variations in weight scope, and the differences between weight scope can affect the performance of model merging. In Section 5.2, we explore the effectiveness of the WSA method in the mode connectivity scenario. In Section 5.3, we demonstrate the performance improvements achieved by WSA in various federated learning scenarios and analyze the impact of the method.

5.1 Basic Experiments

Observations. Our basic assumption is that the weight scope could be formulated as the Gaussian distribution. Hence, we first study the weight distributions of the models. The first observation is that the converged weight scope is irrelevant to the way of weight initialization, e.g., the Kaiming uniform or the Kaiming normal initialization method [18]. Figure 3 shows the weight distributions of several layers in VGG16 [44] with BatchNorm [23], where we train the network on CIFAR-100 [31] for 200 epochs. In the Appendix, we will present

Table 1: The impact of training conditions on LMC.

Factor	KL D. ($\times 10^{-4}$)	Ba(=) (%)	Ba(\neq) (%)	Diff (%)
Label Imb.	9.60	4.60	22.05	17.45
Optimizer	8.95	1.13	17.84	16.71
Batch Size	5.96	44.98	56.07	11.09
Label Noise	1.49	0.58	0.79	0.21
Learning Rate	0.67	0.38	6.77	6.39
Feature Noise	0.60	0.70	9.60	8.90
Data Size	0.37	0.00	0.75	0.75
Weight Decay	0.31	1.59	2.31	0.72

the weight scopes in pre-trained models, which also follow the Gaussian distributions.

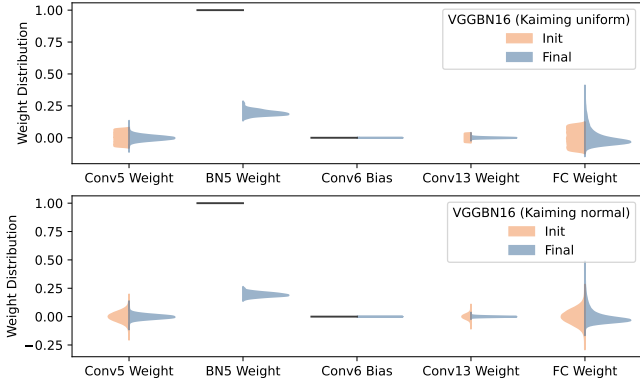


Figure 3: The Gaussian distribution of parameters in VGGBN16 trained on CIFAR-100. Both uniform and normal initialization lead to Gaussian parameter distributions.

The Influence of Weight Scope on Model Fusion. We then investigate the difference of weight scope under different conditions, which include: 1) hyperparameters, e.g., optimizer, batch-size, learning rate, and weight decay; 2) data quality, e.g., label imbalance, label noise, feature noise, and data size. For each condition, we select two specific settings, and then train models under the same setting or not. For example, the optimizer could be SGD or Adam [30], and we train three models with each using SGD, SGD, and Adam as the optimizer, respectively. Then, we plot the weight scopes of these models and investigate the linear interpolation accuracy. Figure 1 shows that models trained using different optimizers own varying weight scopes and the interpolation meets an obvious barrier. The details of training conditions and the corresponding illustration results can be found in the Appendix.

From Figure 1, we can observe that the weight scopes under the same training condition are near the same. In fact, we calculate their KL divergence, and the results are near zero. Hence, we calculate the average KL divergence of weight scopes under different conditions, e.g., the average divergence of Gaussian distributions in the bottom five sub-figures (the “KL D.” column in Table 1). Additionally, we calculate the barriers of the two interpolated curves, and their difference, which are listed in columns of “Ba(=)”, “Ba(\neq)”, and “Diff” in Table 1. In the table, the “Ba(=)” column is commonly smaller than “Ba(\neq)”, which verifies that models under the same condition are indeed similar in weight scopes. Empirically, for models under different conditions, a larger KL divergence corresponds to a larger

barrier. This shows that the weight scope mismatch is strongly correlated with the model fusion performance.

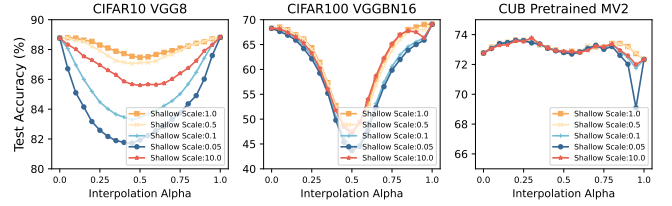


Figure 4: The linear interpolation curves between a model with another model (Scale=1.0) and its scaled versions (Scale \neq 1.0).

Model Fusion under Manual Scaling. The above experiments could not support that the weight scope is the cause of the barrier in model fusion. Thanks to the scale invariance of neural networks, we could manually scale the networks and create the weight scope mismatch phenomenon. Specifically, we train two networks under the same condition with different random initialization. To avoid overuse of symbols, we denote them as w_1 and w_2 , respectively. To create the scope mismatch, we select one layer in w_2 and multiply its weight by α , and divide its following layer’s weight by α . The obtained model is denoted as $w_2 \cdot \alpha$, which performs the same as w_2 . However, when taking interpolations between w_1 and $w_2 \cdot \alpha$, the interpolation curves differ a lot. The results are shown in Figure 4, where the “Scale” in the legend denotes α . Using $\alpha = 1.0$ means no scaling. Clearly, scaling the layers could make the barrier more obvious, especially in VGG8. Experimental details and more analysis can be found in the Appendix.

5.2 Performance in Mode Connectivity

To investigate the effectiveness of our approach on mode connectivity, we apply the proposed WSA to OTFusion [45] and Git Re-Basin [3]. Specifically, we first initialize and train two models, and plot the vanilla interpolation curve. Then, we use the activation-based OTFusion method to search for an alignment and plot the interpolated curve after matching. Finally, we replace the models with another two models trained using the WSA as introduced in Section 4.1 and also use the OTFusion to search the alignment matrix. The results are shown in Figure 5. OTFusion could decrease the barrier of vanilla interpolation, and our proposed WSA could improve its performance further.

Then, we combine our WSA with Git Re-Basin, which searches the permutation matrix instead of an optimal transport matrix compared with OTFusion. Similar to OTFusion, we also train models

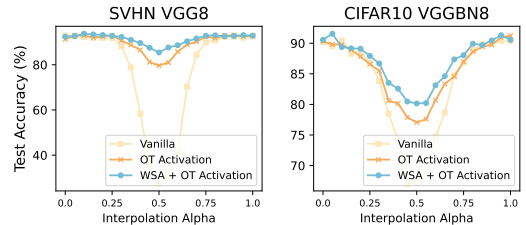


Figure 5: Model interpolation on SVHN and CIFAR-10. WSA enhances OTFusion [45] performance.

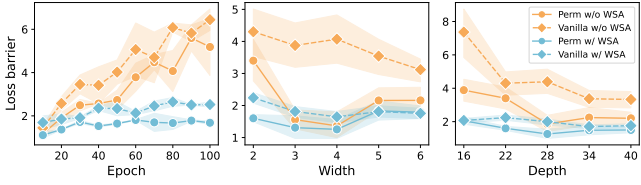


Figure 6: WSA facilitates Git Re-Basin [3] under different epochs, widths and depths, ensuring a smaller loss barrier.

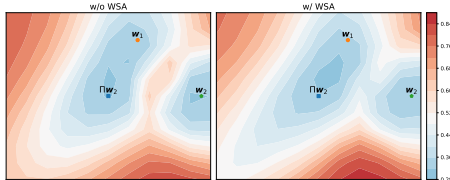


Figure 7: Loss landscape w/o WSA and w/ WSA.

with or without our proposed WSA. In Figure 6, we conduct experiments on a ResNet-22x2 [19, 3] model on CIFAR-10 [31] to compare the loss barrier at different epochs. The solid lines represent the vanilla model interpolation, while the dashed lines represent the interpolation after using Git Re-basin. On the one hand, applying Git Re-Basin could indeed decrease the barrier of vanilla interpolation. However, as the number of epochs increases, the results of w/o WSA continue to grow, even when permutation is considered. In contrast, the lines of w/ WSA can consistently maintain lower loss barriers. In the middle and right figures, we compare loss barriers under different model widths and depths. Specifically, we compare ResNet-22 models with a widening factor of 2, 4, and 6, and ResNet models with a widening factor of 2 and depths of 16, 22, and 28. The results show that our method consistently improves model fusion across models with varying depths and widths. Furthermore, permutation interpolation further improves the loss barrier, demonstrating that considering both weight permutation and scope can effectively reduce the loss barrier. This indicates that WSA helps align the weight ranges of models, contributing to enhanced mode connectivity.

In Figure 7, we illustrate the loss landscape of model w_1 , w_2 and Πw_2 , where the permutation matrix Π is computed by Git Re-basin. We train the two models with and without our proposed WSA, separately. First, there is an obvious peak between w_1 and w_2 on the left; meanwhile, on the right, the loss between w_1 and w_2 is alleviated with WSA. Secondly, through Git Re-basin, the valley between Πw_2 and w_1 is a larger flat area which leads to better model merging.

Pretrained Model. We conduct experiments using the Vision Transformer (ViT) pre-trained on ImageNet-21k on mode connectivity. Figure 8 presents the results of finetuning the pre-trained ViT on ImageNet-100. Specifically, We finetune the ViT using two distinct random seeds and then conduct model interpolation between the resultant models. “baseline” refers to training without intervention, while “w/ WSA” indicates training with Weight Scope Alignment. Because of permutation-based weight averaging methods are not designed for transformers, we employ the naive model interpolation $(1 - \alpha)w_1 + \alpha w_2$. The results demonstrate that WSA can improve the performance of interpolated models.

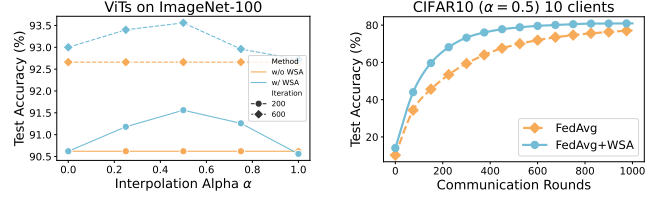


Figure 8: Mode Connectivity of ViTs finetuned on ImageNet-100.

Figure 9: Test accuracy curve of VGG8 trained on CIFAR-10 in FL.

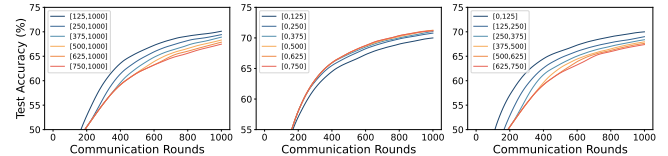


Figure 10: The Impact of using WSA in different rounds in FL.

5.3 Performance in Federated Learning

To simulate real-world federated learning (FL) scenarios, we initially consider image classification tasks across three datasets: CIFAR-10 [31], CIFAR-100 [31], and CINIC-10 [6]. We explore two different client scenarios: “cross-device” with 100 small-data clients where only 10 clients participate in each training round, and “cross-silo” with 10 clients, each having larger datasets and engaging each training round. We use the same CNN architecture used in [28, 25].

To ensure Non-I.I.D. data distribution in data partitioning, we employ the Dirichlet distribution for creating heterogeneous data on each client [56, 37]. We compare with several state-of-the-art (SOTA) methods, including FedAvg [40], FedProx [35], SCAF-FOLD [28], and FedExp [25]. Throughout all experiments, unless otherwise specified, we use a default batch size of 50, 20 local update steps, and 1500 communication rounds. Additional details and results are provided in the appendix.

Table 2 presents the performance comparison of CNN model trained in various FL settings with 10 clients where FedAvg+WSA outperforms FedAvg in performance. Moreover, as a plug-in, we can easily adapt it to existing FL methods, and the experiments indicate that incorporating WSA leads to significant improvements in each method. This underscores the effectiveness of our approach in model fusion of FL. Table 3 shows that similar results are observed with 100 clients. Furthermore, Figure 9 illustrates the adaptability of WSA to deeper models on CIFAR-10.

To further understand the impact of WSA on model merging, we conduct experiments on the CIFAR-10 dataset over various durations. We consider three different intervention times during training: from the start of training until iteration $[0, t_1]$, from iteration $[t_1, T]$ until convergence, and during the iterations within the interval $[t_1, t_2]$. Figure 10 displays the convergence curves for these three scenarios. In the first scenario, we observe that the longer the intervention time, the better our method enhances the convergence performance. In the second one, we find that employing our method earlier results in greater performance improvements and faster convergence. Lastly, even if our method only intervenes during the initial $[0, 125]$ rounds, it still significantly boosts performance. Drawing inspiration from the concept of a “critical learning period” [54, 2], we conclude

Table 2: Comparison of Test Accuracy (%) across various FL settings on three datasets, involving 10 clients: “Baseline” represents the original method, while “+WSA” indicates the incorporation of WSA into “Baseline”. Alpha ↓ denotes client heterogeneity ↑. **Bold** fonts highlight the optimal results between “Baseline” and “+WSA”. The results demonstrate that +WSA enhances performance in all FL settings.

Dataset	CIFAR-10				CIFAR-100				CINIC-10			
	0.5		1		0.5		1		0.5		1	
Algorithm	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA
FedAvg [40]	68.42	72.39	69.06	73.09	30.66	35.23	30.86	36.16	53.05	57.88	53.94	58.53
FedProx [35]	68.40	72.49	69.11	73.18	30.47	35.39	31.20	36.24	52.57	57.54	53.34	58.26
SCAFFOLD [28]	66.69	71.61	68.14	72.65	29.58	34.38	31.12	36.76	50.88	55.87	52.13	57.52
FedExp [25]	73.24	75.55	72.73	76.10	38.93	41.71	40.48	43.16	56.62	60.77	56.30	59.93

Table 3: Comparison of Test Accuracy (%) across various FL settings on three datasets, involving 100 clients where 10% participate in each round: “Baseline” represents the original method, while “+WSA” indicates the incorporation of WSA into “Baseline”. Alpha ↓ denotes client heterogeneity ↑. **Bold** fonts highlight the optimal results between “Baseline” and “+WSA”. The results demonstrate that +WSA enhances performance in all FL settings.

Dataset	CIFAR-10				CIFAR-100				CINIC-10			
	0.5		1		0.5		1		0.5		1	
Algorithm	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA	Baseline	+WSA
FedAvg	66.77	71.11	69.06	72.80	29.57	34.47	30.45	35.31	50.85	56.68	52.64	57.82
FedProx	66.82	71.14	69.06	72.73	29.72	34.41	30.35	35.17	50.14	56.31	52.13	57.55
SCAFFOLD	65.74	70.36	69.47	73.25	29.37	34.85	31.81	37.31	48.39	54.49	52.84	57.07
FedExp	71.79	74.33	71.82	74.80	37.98	41.34	38.46	41.58	50.22	56.99	55.84	60.55

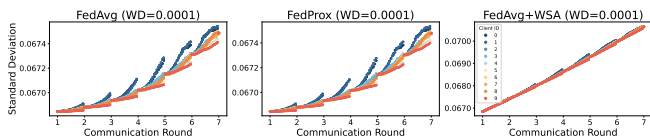


Figure 11: The standard deviation of conv2 weights for each client in FL training with 10 clients, varying across rounds.

that assisting weight alignment in the early stages of learning provides more substantial benefits for the model fusion scenario.

Analysis of weight distribution. In Figure 11, we illustrate the alignment effect introduced by WSA on parameter ranges during the training process of FedAvg. The figure displays how the weight distributions of 10 clients evolve during training. To enhance clarity, we represent the distributions separately using mean and variance, showing the results for the original FedAvg, FedProx, and FedAvg+WSA. Different colors represent different clients, and aggregation occurs when the server averages the model parameters across all clients after each communication round, making the model distributions identical at that point. In the variance plots, we observe aggregation and divergence in FedAvg even with a weight decay of 0.0001, particularly in the early stages, indicating that the mismatch of weight ranges may contribute to slow convergence in the early phases of FL. In contrast, our constraint promotes a more cohesive model variance.

Comparison with pre-defined distributions. To demonstrate the effectiveness of weight scope fusion, we try to design some pre-defined distributions to replace it. Given that Gaussian distributions match the frequently noted attributes of model weights, we devise various pre-defined Gaussian distributions $N(\mu, \sigma^2)$, by altering the mean μ and standard deviation σ . Each of these distributions is utilized to substitute the fusion distributions in WSA, and are set across all layers. As shown in Figure 12, our method “w/ WSA” outper-

forms all other comparative approaches. The pre-defined distribution $w/ N(0, 0.1^2)$ is the second-best result, while other pre-defined ones led to a degradation in performance. We believe that a well-designed fix distribution can be beneficial during the initial training stage. Yet, weight distribution variation significantly shifts throughout the training, as seen in Figure 11, necessitating a dynamic and adaptable fusion distribution. Hence, our suggested weight scope fusion becomes crucial for broad model fusion applications.

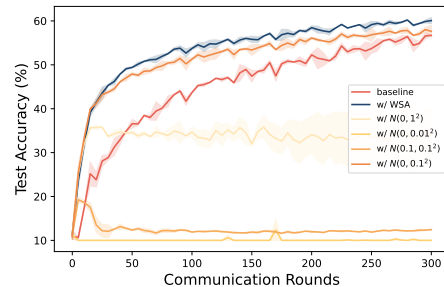


Figure 12: Comparison with pre-defined distributions in FL.

6 Conclusion

In this study, we investigate the impact of different weight ranges of model parameters on model merging. We observe that models trained under various conditions exhibit inconsistencies in their weight ranges, leading to a decrease in fusion performance. To address this issue, we propose Weight Scope Alignment method, which utilize Weight Scope Regularization to constrain the alignment of weight scopes during training, ensuring that the model’s weights closely match the specified scope. Moreover, for multi-stage fusion scenarios, we design Weight Scope Fusion to fuse the weight scopes

of multiple models and regards the fused one as the target weight scope for Weight Scope Regularization. This alignment enhances performance during model merging. We validate the effectiveness of our approach in mode connectivity and federated learning.

Acknowledgements

This research was supported by National Science and Technology Major Project (2022ZD0114805), NSFC (61773198, 62376118,61921006), Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

- [1] D. A. E. Acar, Y. Zhao, R. M. Navarro, M. Mattina, P. N. Whatmough, and V. Saligrama. Federated learning based on dynamic regularization. In *ICLR*, 2021.
- [2] A. Achille, M. Rovere, and S. Soatto. Critical learning periods in deep networks. In *ICLR*, 2019.
- [3] S. K. Ainsworth, J. Hayase, and S. S. Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *ICLR*, 2023.
- [4] S. C. Ashmore and M. S. Gashler. A method for finding similarity between multi-layer perceptrons by forward bipartite alignment. In *IJCNN*, pages 1–7, 2015.
- [5] J. Brea, B. Simsek, B. Illing, and W. Gerstner. Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape. *CoRR*, abs/1907.02911, 2019.
- [6] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey. Cinic-10 is not imagenet or cifar-10. *CoRR*, abs/1810.03505, 2018.
- [7] C. T. Dinh, N. H. Tran, and T. D. Nguyen. Personalized federated learning with moreau envelopes. In *NeurIPS*, 2020.
- [8] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. In *ICML*, volume 70, pages 1019–1028, 2017.
- [9] F. Draxler, K. Veschgini, M. Salmhofer, and F. A. Hamprecht. Essentially no barriers in neural network energy landscape. In *ICML*, volume 80, pages 1308–1317, 2018.
- [10] R. Entezari, H. Sedghi, O. Saukh, and B. Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *ICLR*, 2022.
- [11] J. Frankle. Revisiting "qualitatively characterizing neural network optimization problems". *CoRR*, abs/2012.06898, 2020.
- [12] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.
- [13] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, volume 119, pages 3259–3269, 2020.
- [14] T. Garipov, P. Izmailov, D. Podoprikin, D. P. Vetrov, and A. G. Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, pages 8803–8812, 2018.
- [15] C. Godfrey, D. Brown, T. Emerson, and H. Kvinge. On the symmetries of deep learning models and their internal representations. In *NeurIPS*, 2022.
- [16] I. J. Goodfellow and O. Vinyals. Qualitatively characterizing neural network optimization problems. In *ICLR*, 2015.
- [17] Y. Hao, L. Dong, F. Wei, and K. Xu. Visualizing and understanding the effectiveness of bert. In *EMNLP-IJCNLP*, pages 4141–4150, 2019.
- [18] K. He, X. Zhang, S. Ren, and J. Sun. Diving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.
- [19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.
- [20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.
- [21] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons. The non-iid data quagmire of decentralized machine learning. In *ICML*, volume 119, pages 4387–4398, 2020.
- [22] T.-M. H. Hsu, H. Qi, and M. Brown. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, 2019.
- [23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37, pages 448–456, 2015.
- [24] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *CoRR*, abs/1811.11479, 2018.
- [25] D. Jhunjunwala, S. Wang, and G. Joshi. Fedexp: Speeding up federated averaging via extrapolation. In *ICLR*, 2023.
- [26] X. Jin, X. Ren, D. Preotiuc-Pietro, and P. Cheng. Dataless knowledge fusion by merging weights of language models. In *ICLR*, 2023.
- [27] K. Jordan, H. Sedghi, O. Saukh, R. Entezari, and B. Neyshabur. Repair: Renormalizing permuted activations for interpolation repair. In *ICLR*, 2023.
- [28] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *ICML*, volume 119, pages 5132–5143, 2020.
- [29] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*, 2017.
- [30] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
- [31] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [32] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *NeurIPS*, pages 950–957, 1991.
- [33] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, pages 6391–6401, 2018.
- [34] J. Li, A. Li, C. Tian, Q. Ho, E. P. Xing, and H. Wang. Fednar: Federated optimization with normalized annealing regularization. *CoRR*, abs/2310.03163, 2023.
- [35] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. In *MLSys*, 2020.
- [36] W. Li, Y. Peng, M. Zhang, L. Ding, H. Hu, and L. Shen. Deep model fusion: A survey. *CoRR*, abs/2309.15698, 2023.
- [37] X.-C. Li, Y. Xu, S. Song, B. Li, Y. Li, Y. Shao, and D.-C. Zhan. Federated learning with position-aware neurons. In *CVPR*, pages 10072–10081, 2022.
- [38] X.-C. Li, J. Tang, B. Zhang, L. Li, and D.-C. Zhan. Exploring and exploiting the asymmetric valley of deep neural networks. *ArXiv*, abs/2405.12489, 2024.
- [39] Z. Li, H.-Y. Chen, H. W. Shen, and W.-L. Chao. Understanding federated learning through loss landscape visualizations: A pilot study. In *NeurIPS*, 2022.
- [40] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, volume 54, pages 1273–1282, 2017.
- [41] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.
- [42] B. Neyshabur, H. Sedghi, and C. Zhang. What is being transferred in transfer learning? In *NeurIPS*, 2020.
- [43] F. Pittorino, A. Ferraro, G. Perugini, C. Feinauer, C. Baldassi, and R. Zecchina. Deep networks on toroids: Removing symmetries reveals the structure of flat regions in the landscape geometry. In *ICML*, volume 162, pages 17759–17781, 2022.
- [44] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.
- [45] S. P. Singh and M. Jaggi. Model fusion via optimal transport. In *NeurIPS*, 2020.
- [46] B. Tao, X.-C. Li, and D.-C. Zhan. MLI formula: A nearly scale-invariant solution with noise perturbation. In *ICML*, 2024.
- [47] N. J. Tatro, P.-Y. Chen, P. Das, I. Melnyk, P. Sattigeri, and R. Lai. Optimizing mode connectivity via neuron alignment. In *NeurIPS*, 2020.
- [48] T. J. Vlaar and J. Frankle. What can linear interpolation of neural network loss landscapes tell us? In *ICML*, volume 162, pages 22325–22341, 2022.
- [49] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.
- [50] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos, and Y. Khazaeni. Federated learning with matched averaging. In *ICLR*, 2020.
- [51] X. Wang, A. N. Wang, M. Zhou, and R. Ge. Plateau in monotonic linear interpolation - a "biased" view of loss landscape for deep networks. In *ICLR*, 2023.
- [52] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. G. Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, volume 162, pages 23965–23998, 2022.
- [53] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 5987–5995, 2017.

- [54] G. Yan, H. Wang, and J. Li. Critical learning periods in federated learning. *CoRR*, abs/2109.05613, 2021.
- [55] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2): 12:1–12:19, 2019.
- [56] F. Yu, W. Zhang, Z. Qin, Z. Xu, D. Wang, C. Liu, Z. Tian, and X. Chen. Fed2: Feature-aligned federated learning. In *KDD*, pages 2066–2074, 2021.
- [57] M. Yurochkin, M. Agarwal, S. Ghosh, K. H. Greenewald, T. N. Hoang, and Y. Khazaeni. Bayesian nonparametric federated learning of neural networks. In *ICML*, volume 97, pages 7252–7261, 2019.
- [58] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.
- [59] X. Zhai, J. Puigcerver, A. Kolesnikov, P. Ruysen, C. Riquelme, M. Lucic, J. Djolonga, A. S. Pinto, M. Neumann, A. Dosovitskiy, L. Beyer, O. Bachem, M. Tschannen, M. Michalski, O. Bousquet, S. Gelly, and N. Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark. *ArXiv*, 2019.
- [60] S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *NeurIPS*, pages 685–693, 2015.

7 Dataset and Network Details

7.1 Datasets

In this paper, we use SVHN [41], CIFAR-10 [31], CIFAR-100 [31], CINIC-10 [6] and CUB [49] datasets. We provide a general introduction to these datasets as follows:

1. SVHN [41]: The Street View House Numbers (SVHN) dataset, a real-world image collection used in Figure 5, features small, cropped digits captured from street view images, often exhibiting variations in rotation. It comprises 73,257 digit images for training and 26,032 for testing, spanning across 10 classes.
2. CIFAR-10 [31]: The CIFAR-10 dataset comprises 60,000 32x32 color images, evenly distributed across 10 classes with each class containing 6,000 images.
3. CIFAR-100 [31]: The CIFAR-100 dataset consists of 100 classes, each with 600 images. These classes are further divided into 20 superclasses, grouping similar categories together.
4. CINIC-10 [6]: The CINIC-10 dataset, containing a total of 270,000 images, is derived from two sources: ImageNet and CIFAR-10. It is divided into three subsets: training, validation, and testing, with each subset comprising 90,000 images. In our study, the validation subset is not utilized.
5. CUB [49]: The Caltech-UCSD Birds-200-2011 (CUB) dataset is designed for fine-grained visual categorization tasks, featuring 11,788 images across 200 bird subcategories. It is divided into 5,944 training images and 5,794 testing images.

7.2 Networks

In the work, our used networks include VGG [44], ResNet [19], Wide ResNet [58], ResNeXt [53], MobileNet-v2 [20]. Here is a general introduction to these networks as follows:

1. CNN: The model architecture consists of two convolutional layers, the first with $32 \ 5 \times 5$ filters and the second with $64 \ 5 \times 5$ filters, followed by two linear layers containing 384 and 194 neurons, respectively. A softmax layer concludes the architecture. This configuration is commonly utilized in studies [40, 25, 28].
2. VGG [44]: VGG models consist of multiple convolution layers and fully-connected layers for classification, typically with 11, 13, 16, or 19 layers. We do not employ batch normalization in VGG11, VGG13, VGG16, or VGG19 (referred to as VGG11-BN, VGG13-BN, VGG16-BN, and VGG19-BN when batch normalization is used). Additionally, we utilize 8-layer and 9-layer VGG networks without batch normalization (VGG8 and VGG9), as mentioned in [56, 37].
3. ResNet [19]: ResNet, leveraging residual connections and batch normalization, aims for enhanced performance and robustness. We use ResNet-18 and ResNet-32 in this work. The implementation code we use is provided by [25].
4. Wide ResNet [58]: Wide ResNet (WRN), an extension of ResNet, varies in width to offer different capacities. Due to increased training time with wider networks, we explore variations like WRN16x2, 22x2, 28x2, 34x2, 40x2, and 22x3, 22x4, 22x5, 22x6.
5. ResNeXt [53]: ResNeXt substitutes group convolutions for partial convolutions in ResNet. We utilize the pre-trained ResNeXt models available in PyTorch for analyzing weight scope distributions, as they offer aggregated residual transformations.
6. MobileNet-v2 [20]: MobileNet-v2, known for its compact structure suitable for portable devices, is employed in our study us-

ing its pre-trained version available in PyTorch to analyze weight scope distributions.

We will further introduce the datasets and networks used in each application. All our experiments are conducted on a NVIDIA 3090Ti GPU.

8 Experimental Details

8.1 Weight Scope Mismatch

We first explain the experimental setup for the phenomenon of weight scope mismatch in Figure 1, which the influenced factor is the type of optimizer. Specifically, Figure 1 trains VGG8 network on CIFAR-10 dataset with 200 epochs. We first use SGD optimizer to individually train two models, which are named M0 and M1, respectively. Then we use Adam optimizer to train model M2. We select the parameters including “conv1.weight”, “conv3.weight”, “conv3.bias”, “conv5.weight”, and “fc.weight” and show their weight distributions. Other hyperparameters are listed as follows: the weight decay is 10^{-5} , the batch size is 128, the learning rate is 0.03 for SGD optimizer, while the learning rate is 0.0003 for Adam. The momentum for SGD is 0.9 by default. The weight distributions are visualized by seaborn ¹.

Furthermore, we explore the following key factors that may influence the weight scope of parameters:

1. **Learning Rate:** we use 0.03 for M0 and M1, while we use 0.001 for M2.
2. **Weight Decay:** we use 10^{-5} for M0 and M1, while we use 10^{-3} for M2.
3. **Batch Size:** we use 32 for M0 and M1, while we use 1024 for M2.
4. **Optimizer:** we use SGD for M0 and M1, while we use Adam for M2.
5. **Feature Noise:** we add gaussian noise to the input features, sampled from the gaussian distribution $\mathcal{N}(1.0, \epsilon^2)$. We set $\epsilon = 0.0$ for M0 and M1, while we set $\epsilon = 0.1$ for M2.
6. **Label Noise:** the label of each training sample is flipped to a random class with a probability of p . We set $p = 0.0$ for M0 and M1, indicating no label flipped. We set $p = 0.1$ for M2.
7. **Dataset Size:** a fraction q of data is randomly selected to train the model. We set $q = 1.0$ for M0 and M1, while we set $q = 0.1$ for M2.
8. **Label Imbalance:** we split the training data into two parts. The first part contains 90% samples of the first five classes and 10% samples of the last five classes, while the second part contains other samples. We train M0, M1 on the first part, while we train M2 on the second part.

Aside from the investigated factor, the other hyperparameters are set as same as Figure 1, ensuring the training data remains original without any noise or sampling. Then, we could plot figures for each factor which are similar as Figure 1. Figure 13 and Figure 14 illustrates different dataset size or learning rate similarly lead to lead to weight scope mismatch and impact model interpolation.

Then we provide explanations for each of the columns in Table 1 .

1. **KL D.:** this column computes the KL divergence of weight distributions between M0 and M2. More precisely, for each layer, we model the weight distribution as a Gaussian distribution and compute the KL divergence. The average result is then reported. Notably, the KL divergence between M0 and M1 is negligible, nearly

¹ <https://seaborn.pydata.org/generated/seaborn.violinplot.html>

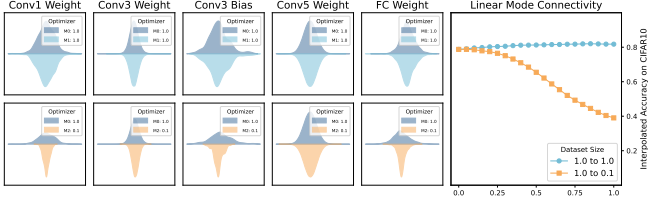


Figure 13: Different training conditions (data size) result in partially different weights and affect model interpolation.

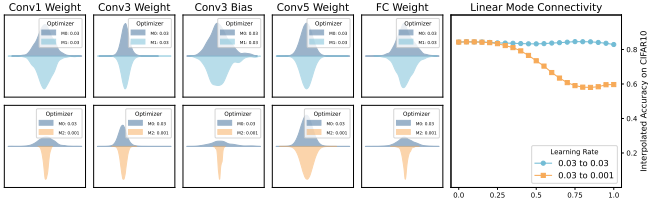


Figure 14: Different training conditions (learning rate) result in partially different weights and affect model interpolation.

zero, and therefore, we do not present the KL divergence between these models. This emphasizes that the divergence between models trained under identical conditions may approach zero.

2. $Ba(=)$: this column calculates the barrier of the interpolation curve between model M0 and M1, and the definition of the barrier is provided in Equation 7 where we use the test error instead of the training loss.
3. $Ba(\neq)$: this column calculates the barrier of the interpolation curve between M0 and M2, and the definition of the barrier is in Equation 7 where we use the test error instead of the training loss.
4. **Diff**: it is the difference between “ $Ba(\neq)$ ” and “ $Ba(=)$ ”, i.e., “ $Ba(\neq) - Ba(=)$ ”.

8.2 Different Weight Initialization

In Figure 3, we use two different popular initialization methods to train VGGBN16 models. The initialization method contains Kaiming uniform and Kaiming normal [18]. Both of them show that the weight distribution follows a Gaussian distribution.

8.3 Manual Scaling

Figure 4 illustrates the impact of mismatch in manually-designed weight scopes on linear interpolation. Our investigation covers pairs such as VGG8 on CIFAR10, VGGBN16 on CIFAR100, and Pre-trained MobileNetV2 (MV2) on CUB. Specifically, we focus on the shallow layers, namely “block0.0” for VGG8 and VGGBN16, and “backbone.0.1.conv.1” for MV2, which represent the first convolution layers in their respective networks. Additionally, the study extends to deep layers: “block3.0” for VGG8, “block4.6” for VGGBN16, and “backbone.0.17.conv.2” for MV2, identified as the penultimate convolution layers. The consequent effects on linear interpolation are detailed in Figure 15. It is also observed that linear interpolation between two models with different weight distributions leads to deteriorated results.

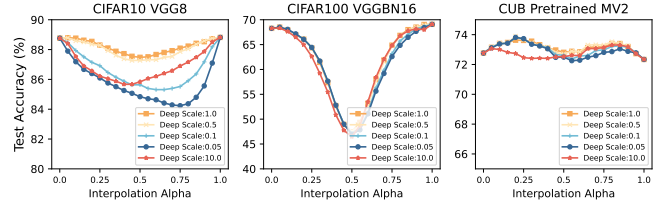


Figure 15: The linear interpolation curves between a standard model (Scale=1.0) and its scaled versions (Scale \neq 1.0) are shown, in which the penultimate convolution layer is rescaled.

8.4 Mode Connectivity

In applying mode connectivity, our method is compared with model averaging (Vanilla), OTFusion [45] and Git-rebasin [3] across different datasets and models. It is important to note that OTFusion and Git-rebasin are designed for different scenarios, necessitating separate comparative analyses. To make a comparison with OTFusion, we conduct experiments with VGG8 on SVHN and VGGBN8 on CIFAR10, as illustrated in Figure 5. Additionally, we present results for MLP on MNIST and ResNet-32 on CIFAR100 in Figure 16. These results collectively demonstrate the benefits of weight scope alignment in enhancing mode connectivity within permutation-based model fusion methods.

Furthermore, we conduct comparisons with Git-rebasin [3] on the CIFAR-10 dataset using ResNet-18 and WRN. In this experiment, we train two models with different random seeds for 100 epochs and then compute their loss barriers on the test set using Equation 7, with a learning rate of 0.01 and a weight decay of 10^{-4} . The experimental results are displayed in Figure 6.

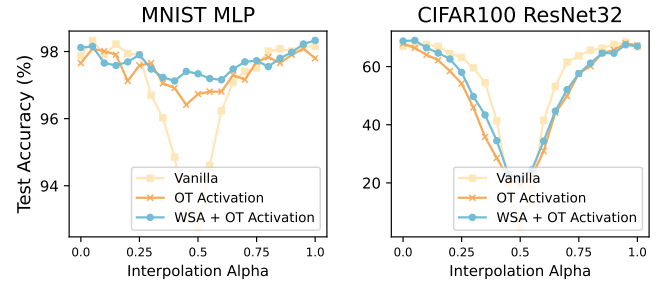


Figure 16: OTFusion on MNIST MLP and CIFAR100 ResNet-32.

We further assess the performance on RESISC45, DTD, GTSRB datasets included in VTAB [59]. Then, we compute the barrier of test accuracy as described in Section 5.2. A lower barrier indicates better interpolated performance. The results below demonstrate that WSA can reduce the barrier across the three datasets.

Table 4: The barrier of test accuracy of ViTs finetuned on three datasets.

Dataset	Baseline	WSA
RESISC45	0.43%	-0.22%
DTD	-0.50%	-0.93%
GTSRB	0.90%	0.79%

8.5 Federated Learning

To simulate various clients, we employ a Dirichlet distribution to allocate data across clients for the CIFAR-10, CIFAR-100, and CINIC-10 datasets [22], utilizing the parameter α to control data heterogeneity. Figure 17 and Figure 18 depict the data distribution among clients for α values of 0.5 and 1.0, respectively, with lower α values indicating increased data heterogeneity. The training convergence curves in 10 clients and 100 clients setting are illustrated in Figure 19.

Our experiments are conducted using the FedExp framework². The experimental setup includes a learning rate of 0.01, weight decay of 10^{-4} , a decay of the learning rate by 0.998 in each round, a maximum gradient norm of 10, a fusion alpha of 1.0, and 20 local steps for all models and datasets. Specifically for FedProx, μ is set to 0.1, 1, and 0.001 for CIFAR-10, CIFAR-100, and CINIC-10, respectively. In the case of FedExp, ϵ is maintained at 0.001 which is identified as the optimal value for all three datasets in [25].

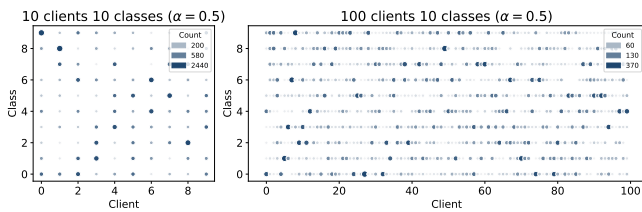


Figure 17: Client distribution of 10 clients and 100 clients sampled from Dirichlet distribution ($\alpha = 0.5$).

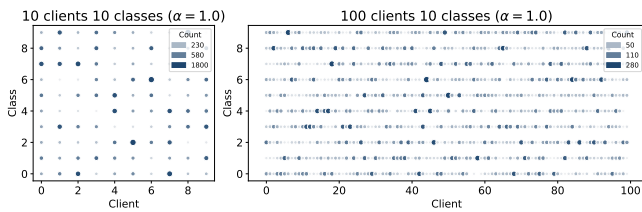


Figure 18: Client distribution of 10 clients and 100 clients sampled from Dirichlet distribution ($\alpha = 1.0$).

Table 5: Hyperparameter sensitivity in FL.

λ	Test Acc (Epoch 100)	Test Acc (Epoch 1000)
0	42.34	63.06
1	47.59	67.52
5	51.99	67.72
10	53.18	66.58
50	52.90	62.68

8.5.1 Additional Experiments

Hyperparameter sensitivity In Equation 10, λ is the hyperparameter that controls the strength of weight scope regularization. We explore the sensitivity of the hyperparameter λ by testing values in [1,

5, 10, 50] within the context of federated learning. The results below indicate that when $\lambda = 5$, the performance is optimal. However, setting λ significantly higher, initially leads a rapid performance increase as shown the performance at the 100-th epoch. However, the final convergence performance is actually lower. Intuitively, applying stronger weight scope regularization may cause the model to lose flexibility, akin to large weight decay.

² <https://github.com/Divyansh03/FedExp>

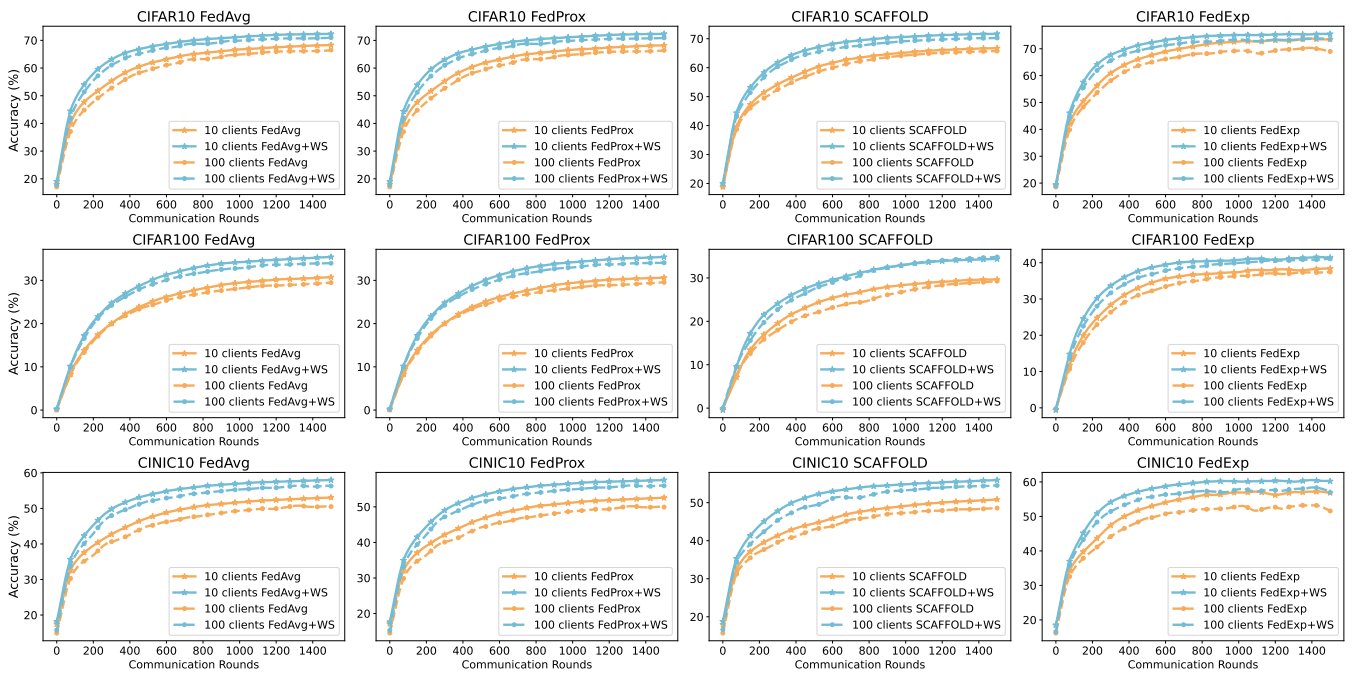


Figure 19: Convergence curve of test accuracy across various settings in FL.