On Model Merging

George Stoica

Committee:
Judy Hoffman Zsolt Kira Alan Ritter

Georgia Tech

October 2023

Contents

1 Introduction

How can the knowledge of one model be transferred into another? Or, more broadly, how can the specializations of several models be transferred between one another?

Synthesizing the knowledge of many models into one is no easy feat, and established approaches come with several limitations. One may propose to ensemble the models, creating a "one" new model using the backbones of the individual ones. Indeed, such approaches have consistently resulted in more robust and stronger predictors over a variety of problems [Sagi and Rokach, 2018]. However, this strong performance comes at a severe cost: maintaining and running all models jointly during test time is required to predict over any data. As models become larger and more complex, the compute and memory overhead required to store such ensembles may quickly become infeasible, rendering them unuseable. One may instead propose knowledge-distillation [Hinton et al., 2015], where one network is retrained on its training data while jointly using the predictions of the other as a "teacher" on the samples. Similar to ensembles, distillation has also been consistently shown to better the skills of student models. However, these approaches rely on having *access* to a labeled set of training data and sufficient compute necessary to *retrain* the model. Moreover, extending a newly distilled model's knowledge by yet a third model on the fly is difficult as the process must repeat. Similar to ensembles, full-distillation may become increasingly infeasible as models become larger and more complex. Now, consider a different setting. Here, an end user (or organization) has access to a set of models each with different specializations that have already been trained on certain data points. For instance, these models could be available on the Huggingface modelhub[*], and trained on diverse datasets. Or, these models could exist on edge-devices (e.g., clients), each with their own propriety data. Imagine the user would like to obtain a model that contains all the skills of each individual one, but does not have the memory to store an ensemble nor has access to the data. This is common in federated learning. Yet, as the field of machine learning continues to expand towards larger models, each trained on its own share of huge swaths of propriety data, this scenario becomes more and more common within our everyday lives. Together, these eventualities motivate the study of other ways of fusing the knowledge of many models into one of very similar (or equal) size.

Over the past several years, mode-connectivity [Garipov et al., 2018] has emerged as a means to tackle this problem. Mode-connectivity seeks an answer to the following foundational question,

Can we preserve the strengths of different models by interpolating between their weights to generate a new model without finetuning on the merged weights?

Assume we have two models Model A and Model B of the same architecture, each trained to solve some problem. Mode connectivity seeks to uncover a path between the parameter sets of Model A and Model B in which every point along the path is a Model C whose parameters are the interpolation of those in Model A and Model B, and which performs just as well as each individually. If such a path exists, then every point along it corresponds to a model which achieves low-loss on the problem (i.e., the path is a low-loss path on the joint loss landscape between two models in parameter space). Notably, the existence of such a path implies that Model A and Model B can be directly merged into a single new model Model C that preserves the original model performances without requiring *any finetuning*. We define this process as ***model merging***.

In this report, we highlight ten works that explore model merging from a plethora of theoretical and practical perspectives. The first five propose interpolation methods based on diverse mode-connectivity principles. We split these works among two sections. The first is titled "Non-Linear Mode Connectivity" (Section 3) and presents two papers that find non-linear low-loss interpolation-paths between models. The second is titled "In Pursuit of Linear Mode Connectivity" (Section 4), and discusses three papers that approach model merging via linear low-loss paths. Of the remaining five papers, four propose interpolation methods that are inspired by mode-connectivity but *not rooted* in it. To this end, we group these works into a third section titled "Model Merging without Mode-Connectivity" (Section 5). Finally, the last paper is not based on mode-connectivity or model-merging. However, it offers interesting ideas that help build upon our understanding of network structures, and spur ideas for diverse future directions in model merging. Thus, we interweave this paper into an extended conclusions and future work section titled "Closing thoughts & Future

[*]https://huggingface.co/models

Directions with the Help of: On the Symmetries of Deep Learning Models and their Internal Representations" (Section 6).

2 Background & Notation

This section introduces relevant background and notation that will be built upon throughout this report.

Notation. We will re-use the same following notation across all remaining sections. Let, Model A and Model B be two well-trained models for a given problem. Further, assume that Model A and Model B both lie in a local-minima (not necessarily the same) on the problem after training. For what follows, we use the term minima, mode, basin and bowl interchangeably. Let $\theta_A \in \mathbb{R}^{|net|}$ be the parameter-set of Model A, where $|net|$ is the number of weights in the model. Similarly, let Model B be parameterized by $\theta_B \in \mathbb{R}^{|net|}$.

If two models are mode-connected, their minima (modes) are connected by a continuous path where every point corresponds to the entire parameter set of a model that achieves low-loss on the respective problem. Denote this path by the function $\phi_\theta^{AB} : [0, 1] \to \mathbb{R}^{|net|}$ parameterized by θ, where $\phi^{AB}(0) = \theta_A$ and $\phi^{AB}(1) = \theta_B$. For notation simplicity, we drop the θ subscript in ϕ_θ^{AB} in the remainder of our report. Put formally, we state that Model A and Model B are mode-connected if,

$$B(\theta_A, \theta_B, \phi^{AB}) \approx 0 \tag{1}$$

$$[\max_{\theta \in \phi^{AB}} \mathcal{L}(\theta) - \text{mean}(\mathcal{L}(\theta_A), \mathcal{L}(\theta_B)] \approx 0 \tag{2}$$

Here, $\mathcal{L}$ denotes the loss (e.g., cross-entropy loss) on the problem over a data partition of choice (e.g., the training set). $B(\theta_A, \theta_B, \phi^{AB})$ is known as the *barrier* [Draxler et al., 2018, Garipov et al., 2018, Frankle and Carbin, 2018] which checks the degree to which ϕ^{AB} is low-loss. Note, that our definition of barrier slightly differs from that found in literature—usually the barrier is instead defined as $B(\theta_A, \theta_B)$—because our definition includes the path. This is intentional, as the works we highlight seek to uncover the best path among many candidates. However, we note that our barrier definition trivially reduces to that found prior work when the path is known ahead of time and thus can be ignored.

On Initializations. Training initialization plays a large role in the easiness of finding low-loss paths between models. It is accepted in the literature that models stemming from the same initialization are likely to lie in the same loss-minima after training [Neyshabur et al., 2020]. In this case—and as we will find in two of our highlighted works, [Wortsman et al., 2022a, Matena and Raffel, 2022]—we can achieve strong merges by just averaging all model parameters. However, when models are trained from different initializations it is *extremely unlikely* that they lie in the same loss-minima. In these cases, we cannot just directly average their parameters, and instead other methods must be employed to produce effective merges. This report focuses almost exclusively on this vastly more challenging setting, where models are trained from different initializations. Unless otherwise stated, you can assume that all models from each paper are trained with different initializations.

On Normalization Layers. Additionally, we note that certain layers (e.g., Batch-normalization) in networks cannot be merged solely via a simple interpolation along a ϕ_θ^{AB}. This is because these typically contain running statistics, which get destroyed under interpolation and cause catastrophic merge performance. The common solution is to instead interpolate as if these were any other layers, but then recompute the statistics with a forward pass on a set of *unlabeled training data* [Draxler et al., 2018, Garipov et al., 2018, Entezari et al., 2022, Jordan et al., 2023]. However, not all normalization layers suffer from this issue. For instance, layer-normalization does not track running statistics and thus can be directly interpolated just as any other layer. For the remainder of this report, we assume unless otherwise specified that all results with models consisting of normalization layers perform this resetting as part of the merging process.

3 Non-Linear Mode Connectivity

This section highlights two works [Draxler et al., 2018, Garipov et al., 2018] that describe methods of finding non-linear low-loss paths between well-trained models, showing they are mode connected.

3.1 Essentially No Barriers in Neural Network Energy Landscape

The primary focus of this work lies in uncovering paths between θ_A and θ_B that achieve the lowest possible barrier. They formalize this goal as,

$$\phi^{AB} = \underset{\substack{\phi \text{ from } \theta_A \text{ to } \theta_B \\ \phi(0)=\theta_A, \phi(1)=\theta_B}}{\arg\min} B(\theta_A, \theta_B, \phi) \tag{3}$$

They define the lowest-barrier path as the minimum energy path (MEP) [Jonsson et al., 1998] and call it ϕ^*. Second, they denote the apex along ϕ^* as the *saddle point* of the path. Given the size of modern neural networks, brute force searching for the optimal path among all possible is computationally intractable. Thus, they instead use an extension of the Nudged Elastic Band (NEB) algorithm [Jonsson et al., 1998], termed the Automated Nudged Elastic Band (AutoNEB) [Kolsbjerg et al., 2016].

3.1.1 Method

The NEB optimization problem. The NEB algorithms determine low-loss paths by iteratively bending a line initially drawn between θ_A and θ_B until no gradients along the path are perpendicular to it. The resultant path is then one of low-loss, with its highest loss-point being an upper-bound for the true saddle point between the minima of Model A and Model B. NEB specifies these paths as a chain of $N + 2$ pivots (each corresponding to a distinct parameter set), with the initial pivot fixed to θ_A and final pivot being θ_B. The path is then "bent" by moving the non-endpoint pivots within the parameter space based on the following gradient descent objective,

$$E(p) = \sum_{i=1}^{N} \mathcal{L}(p_i) + \sum_{i=0}^{N} \frac{1}{2} k ||p_{i+1} - p_i||^2 \tag{4}$$

where p_i corresponds to the ith pivot point, and k is a regularization parameter encouraging pivots to be near to one another. This is important because if k is too low, optimizing $E(p)$ will allow pivots to be very spread apart near areas of high loss, leading to their connected line segment going through these very high loss regions—thereby yielding a ϕ^{AB} of high-barrier. Similarly, too large a k places significantly more emphasis on the proximity of the pivots to one another than each of their losses. This can cause some to occupy high loss regions in an effort to keep the regularization penalty low—thereby also yielding a ϕ^{AB} of high-barrier. Thus, k is a very important hyperparameter to get right.

Optimization with NEB. The NEB algorithms find the best path by optimizing Equation 4 via gradient descent. Specifically, they define the gradient of Equation 4 over each pivot point $(-\nabla_{p_i} E(p))$ as the sum of a "force" derived from the loss and one derived from the segments between pivots. These two individual forces are denoted as F_i^L and F_i^S respectively, and their sum is given by $-\nabla_{p_i} E(p) = F_i^L + F_i^S$. The NEB algorithms compute F_i^L by taking the orthogonal projection of $\nabla \mathcal{L}(p_i)$ onto ϕ^{AB} at p_i. Second, $F_i^S = -k(||p_i - p_{i-1}|| - ||p_{i+1} - p_i||$, and encourages pivots to be similarly distanced in regards to each other. F_i^L always lies perpendicular to the path, whereas F_i^S lies parallel to it. Using F_i^L and F_i^S, optimizing $E(p)$ is an iterative process done in a coordinate-wise fashion. Specifically, each gradient step consists of two stages. The first stage involves computing F_i^L for every pivot (not including the endpoints), and updating the respective pivot by this gradient: $p_i = p_i + \gamma F_i^L$ (where $\gamma \geq 0$ is the step-size). Once gradients for all pivots have been computed, F_i^S is computed for every pivot and is similarly used to update each pivot—thereby redistributing them across the path. Figure 1 illustrates these training operations.

Optimization with AutoNEB. The AutoNEB algorithm improves upon NEB by making one small adjustment. Specifically, it first runs the NEB algorithm for a set number of iterations to achieve the best low-loss pivots. It then iterates through consecutive pivots and computes the barrier along the line connecting them. If the barrier is sufficiently large (above a threshold hyperparameter), the algorithm inserts a set of new pivots between the two. This number of pivots is directly dependent on the size of the barrier. Once these pivots are inserted, the NEB algorithm is reapplied on the new path containing both the original pivots and these new ones.

This procedure is repeated several times, until the lowest-loss path is achieved.

(Auto)NEB Discussion. Unfortunately, this approach does cannot guarantee that the path found is the true optimal MEP path. Due to the non-convex nature of optimizing $E(p)$, the (Auto)NEB algorithms may get stuck in local optimums whose paths are spuriously high-loss. While this implies that any path achieved by (Auto)NEB is an *upper-bound* on the barrier for the true ϕ^* between θ_A and θ_B, the gap between ϕ^* and ϕ^{AB} may be very large. More specifically, if the best path found using (Auto)NEB is itself high-loss, it *does not* imply that θ_A and θ_B are *not* mode connected. Therefore, (Auto)NEB can only be used to determine the extent two models are mode connected if the path it finds is of low-loss.

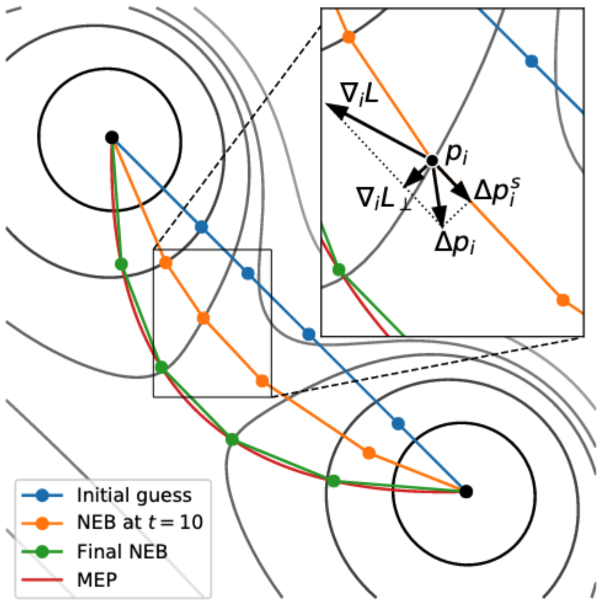


Figure 1: (This figure is a copy of Figure 2 in [Draxler et al., 2018]) Two dimensional loss surface with two minima connected by the optimal MEP in red and the best path found using NEB after convergence. Top-right rectangle showcases the "forces" acting upon a pivot p_i during an optimization iteration.

Partially Mitigating Local Optimas. Interestingly, the authors of [Draxler et al., 2018] partially address the local-optima concern by making an interesting observation: the relationship between MEPs found by (Auto)NEB and each model satisfy the ultrametric property. To explain, suppose we have access to a third model—call it Model C with parameters θ_C—and we have already used (Auto)NEB to find the best paths between all pairs of θ_A, θ_B and θ_C. Let us denote these paths as ϕ^{AB}, ϕ^{AC} and ϕ^{CB} respectively. If it is the case that

$$B(\theta_A, \theta_B, \phi^{AB}) \geq \max(B(\theta_A, \theta_C, \phi^{AC}), B(\theta_C, \theta_B, \phi^{CB})),$$

then we can instead pick ϕ^{AB} to be the concatenation of ϕ^{AB} and ϕ^{CB}. Call this concatenation $\hat{\phi}^{AB}$. By definition of the barrier, we are guaranteed that

$$B(\theta_A, \theta_B, \hat{\phi}^{AB}) = \max(B(\theta_A, \theta_C, \phi^{AC}), B(\theta_C, \theta_B, \phi^{CB})) \leq B(\theta_A, \theta_B, \phi^{AB}).$$

Thus, if the (Auto)NEB algorithm yields a path with poor barrier between two models, we may be able to find a better path by instead connecting these models to a third. However, it is important to note that this is only a partial solution that appears luck-dependent. Applying (Auto)NEB to find a path between a third model does not provide any stronger theoretical guarantees than directly applying the algorithms on the original models. Therefore, these benefits are very empirical and its utility may change dataset to dataset.

Connecting Models with Minimum Spanning Trees. A consequence of the ultrametric property is that the best MEPs computed between models form a Minimum Spanning Tree (MSP). To see this, think of each model parameter set as a vertex in a graph, and the computed MEPs as unidirectional weighted-edges between these vertices with weight equal to the barrier along the path. Once all MEPs between vertices are calculated, we can then use Kruskal's algorithm to obtain the MSP between all vertices (trained models) in the graph. If the barrier of every MEP edge in the MSP is low, we have the very interesting result (and main statement of this paper) that all trained models are connected to one another via a path of low loss. This in turn directly implies they lie on a connected manifold of low-loss, and thus the weights of different models can be interpolated while retaining strong performance on the task.

Putting It All Together. The authors include this MSP idea as the last component in their approach. Specifically, they assume access to n well-trained models, and compute a set of MEPs using AutoNEB between each pair until all trained models are connected via at least one MEP. They then compute the MSP from these MEPs. Afterwards, they delete the path with the largest MEP from the MSP. This results in creating two separate connected graphs. They then use the aforementioned ultrametric property to try and find a path with lower-barrier connecting the disjoint graphs via a different pair of vertices. This process is then repeated until no better MEP between any pair of models exists.

3.1.2 Experiments

While [Draxler et al., 2018] conduct several experiments, in this section I would like to highlight two which together provide a deep understanding of their approach and its characteristics under different settings. The first experiment showcases the behavior of their approach during path-finding. The second experiment shows how their algorithm performs across multiple datasets and models, as well as provides interesting insights that we will carry with us through the remainder of this report.

A Snapshot on Algorithm Behavior. Figure 2 illustrates a reportedly "typical" behavior of their full

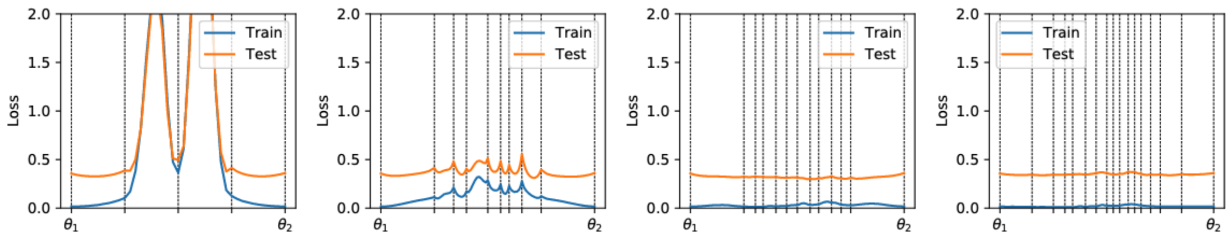


Figure 2: (This figure is a copy of Figure 4 in [Draxler et al., 2018]) "Typical" snapshots of the loss along a path connecting two arbitrary minima through training and as pivots are inserted in high-barrier segments of path. Left: After first NEB optimization, several segments between pivots yield high losses. Second-left: as new pivots are added into these high-loss regions and NEB is re-run, the maximum loss reduces. Second-right: successive updates with AutoNEB reduces MEP barrier. Right: after additional updates, a low-loss path is found. Results obtained using a two differently initialized ResNet20 [He et al., 2016] models on the CIFAR100 [Krizhevsky et al., 2009] dataset.

approach when identifying the MEP between two well-trained models. In this case, they choose two ResNet20 [He et al., 2016] models with different initializations trained on the CIFAR100 [Krizhevsky et al., 2009] dataset. They find MEPs using SGD with momentum set to 0.9 and l^2 regularization with $\lambda = 1e^{-4}$. They then run a four-stage learning procedure, with results after each stage illustrated by the figure. The left-most plot shows the computed MEP after running NEB with four epochs each consisting of 1000 gradient steps and a learning rate of 0.1. We can clearly see that two segments within the computed MEP have extremely high loss, indicating the connected segment is needs to be vastly improved (i.e., through the addition of pivots). The second-left plot shows the resultant MEP after two more NEB epochs using a learning rate of 0.01 and adding new pivots into regions of high loss after every epoch. Notably, we can immediately start to see that their approach is finding a significantly better path between the two models. The computed MEP achieves significantly lower loss (second-right plot) after 4 more epochs with the same learning rate, and finally converges after four more epochs with learning rate of $1e^{-3}$ (right-most plot). Overall, this seems great—their approach is capable of finding paths of very low-loss (low-barrier) between two arbitrary and models with different initializations. That is, they empirically show that two completely independent models are themselves connected by a single path within a very complex and highly-dimensional parameter space (each ResNet20 has tens-of-thousands of parameters). However, it is important to note the cost associated with this finding. Obtaining these results requires up to 14 epochs—each with full access to the training data—and a suite of learning rate schedules to achieve strong results. Moreover, they only show results on variants of the CIFAR dataset; it is not clear whether the same regime would succeed even on slightly more challenging settings such as ImageNet [Deng et al., 2009].

Additional Results Across Datasets and Models. The authors also conduct experiments on the CIFAR10 variant of CIFAR, and using two additional models: shallow-CNNs and the DenseNET [Huang et al., 2016] architecture. For simplicity, we only show results on CIFAR100 here as the results on CIFAR10 are very similar. The exact architectures for each model are not particularly important for the message we would like to present. Instead, it suffices to note that they train several model pairs of the above architectures while varying their depth and hidden layer widths. They then compute the MEP between each pair, and report the results (Figure 3). Interestingly, the figure shows how the computed MEPs change as each network gets deeper and wider across both datasets. Although not explicitly shown, we can consider the barrier to be the difference between each "circle" value and its respective "square" value. Overall, we notice that across *all* architectures:

As networks become wider, the barrier between respective models tends to shrink to near-zero.

This is an interesting and important takeaway which several other works we highlight will independently discover as well. We will build upon this takeaway in future sections, providing more intuitions and a couple caveats as well. In fact, one section will cover a work that proposes a theorem showing a direct correlation between the hidden-state size of models and zero-barrier ([Entezari et al., 2022])! However, the result observed by [Draxler et al., 2018] is a very interesting and valuable first introduction to this phenomenon.

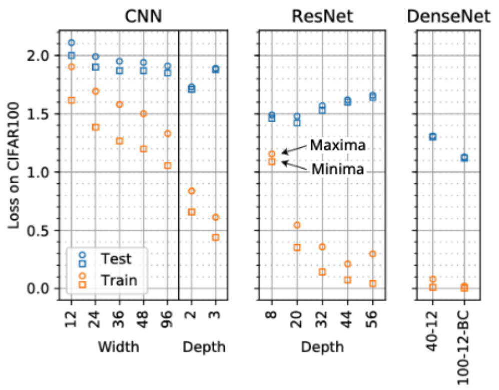


Figure 3: (This figure is a copy of Figure 5 in [Draxler et al., 2018] with two small modifications. First, I have replaced their "barrier" with "maxima". This is because their definition of "barrier" is not correct under the current definitions [Garipov et al., 2018]. Second, I only show the results on CIFAR100 as the story is the same for the other dataset originally shown (CIFAR10).) Results on CIFAR10 and CIFAR100 across CNNs, ResNets and DenseNets [Huang et al., 2016] of differing widths and depths. "Circles" correspond to the maximum loss achieved by the MEP between model pairs. "Squares" correspond to the minimum achieved by the MEPs.

3.1.3 Discussion

It is important to observe that the cost of each NEB algorithm scales by the dataset size and model complexity. This is because solving for the best path ϕ^{AB} requires computing the total loss on the *entire* dataset and all associated gradients, for every pivot in the path. Moreover, as AutoNEB dynamically inserts new pivots into ϕ^{AB} during training, the total cost of a single NEB optimization increases during each iteration of AutoNEB. This can quickly lead to exhorbitant compute requirements as either the dataset size or model complexity increases. This is because each increases both the FLOP count in the forward and backward gradient update passes.

Thus a natural question arises, "is it necessary to utilize *all* training data when optimizing ϕ^{AB} for each NEB step?" For instance, do we achieve similar results to running (Auto)NEB on the full data if instead we only utilize a batch of data? If the answer is yes, it would imply that the total compute bound of an NEB optimization is much closer to the model complexity itself, making the algorithm much more compelling.

3.2 Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs

Similar to [Draxler et al., 2018], this next work proposed by [Garipov et al., 2018] seeks to uncover the extent to which different networks trained on the same task are mode-connected. However, their approach is quite different, and they are able to find simpler low-barrier paths connecting arbitrary networks to one another. Specifically, [Garipov et al., 2018] propose a Bayesian approach to path finding with the objective of finding either (1) quadratic Bezier curves or (2) polygonal chains with one bend (pivot) connecting two networks.

Figure 4 provides examples of these paths found by their algorithm for independently trained ResNet164 networks on the CIFAR10 dataset. All plots topilogically show the l_2 regularized cross-entropy train loss surface as a function of a two-dimensional projection of the ResNet164 parameter set. This means that each pixel corresponds to a distinct model, and its color represents its train-loss. For clarity, we have noted each trained model (represented by vertices) by θ_A, θ_B and θ_C respectively. The horizontal axis of each plot is shared, with the vertical axis offering different slices upon the parameter space. The bottom two models (θ_A and θ_B) are the same across all plots. Interestingly, the left-most plot appears to show that each network is not mode connected, as there does not appear to be any path of low-loss connecting their minimas. However, when plotting along different vertical slices (middle and right plots), we can see that θ_A and θ_B are in fact connected via simple low-loss quadratic Bezier and single-pivot polygonal chains respectively. This pictorially echoes the hypothesis we presented from [Draxler et al., 2018]: seemingly disconnected networks (left panel) may in actually be connected by paths (middle and right panels). Moreover, notice how the direct linear path between either minima across all panels in Figure 4 incurs high-loss (i.e., the networks are not mode-connected via a linear path). This speaks to the complexity of the kinds of paths connecting two minima.

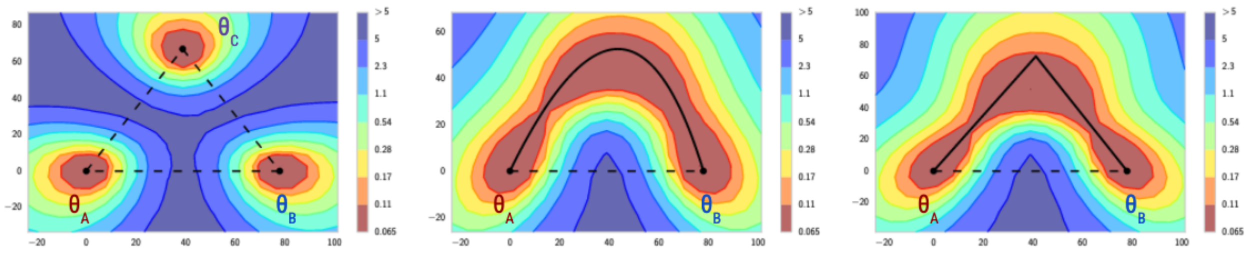


Figure 4: (This figure is a copy of Figure 1 in [Garipov et al., 2018], but with added notation for the plot vertices.) The l_2 regularized cross-entropy loss train surface of a ResNet164 on CIFAR10 as a function of network weights in a atwo-dimensional subspace. The horizontal axis is fixed for each panel, with the vertical axis changing to show different slices along the loss-surface. Left: three optima for different trained networks. Middle: A quadratic Bezier curve connects two networks. Right: A polygonal chain with one bend connects the same two networks. Note that in every panel, the direct linear path (dotted) between two networks incurs high loss.

3.2.1 Method

We now present the method used to uncover such low-loss paths between trained networks. Note that while [Garipov et al., 2018] in effect present two separate methods, the second is an *ensembling* approach which is out of scope in this report as discussed in Section 1.

Following the notation of Section 2, let us assume we have two well-trained models: Model A and Model B. Garipov et al. [2018] propose to find low-loss paths between trained networks by optimizing the expectation of the loss along the path ϕ_θ^{AB} with respect to a uniform distribution on ϕ_θ's domain (i.e., $t \in [0, 1]$):

$$l(\theta) = \int_0^1 \mathcal{L}(\phi_\theta^{AB}(t))dt = \mathbb{E}_{t\sim U(0,1)}\mathcal{L}(\phi_\theta^{AB}(t)) \tag{5}$$

All notation here is as in Section 2, and $U(0, 1)$ is the uniform distribution on $[0, 1]$. To minimize Equation 5, Garipov et al. [2018] randomly sample $\tilde{t}$ from the uniform distribution and then make a gradient step with SGD for θ with respect to the loss $\mathcal{L}(\phi_\theta^{AB}(\tilde{t}))$ until convergence. Notably, each optimization step yields an unbiased estimate of the true gradients of $l(\theta)$:

$$\nabla_\theta \mathcal{L}(\phi_\theta^{AB}(\tilde{t})) \simeq \mathbb{E}_{t\sim U(0,1)}\nabla_\theta \mathcal{L}(\phi_\theta^{AB}(t)) = \nabla_\theta \mathbb{E}_{t\sim U(0,1)}\mathcal{L}(\phi_\theta^{AB}(t)) = \nabla_\theta l(\theta)$$

While ϕ_θ^{AB} can be any curve, Garipov et al. [2018] focus their study on paths made up of polygonal chains with a single pivot, and quadratic Bezier curves. We write the formulation of each below.

Polygonal Chain. Similar to the MEPs found in [Draxler et al., 2018], polygonal chains are piece-wise connected line segments between θ_A and θ_B with pivots. In this work, Garipov et al. [2018] consider chains with just one pivot, which take the following form,

$$\phi_\theta^{AB}(t) = \begin{cases} 2(t\theta + (0.5 - t)\theta_A), & 0 \le t \le 0.5 \\ 2((t - 0.5)\theta_B + (1 - t)\theta), & 0.5 \le t \le 1 \end{cases}$$

Figure 4 illustrates one such chain in the right plot.

Bezier Curve. Bezier curves extend polygonal chains by offering a simple parameterization for *smooth* paths between model parameters. In this work, Garipov et al. [2018] consider quadratic Bezier curves of the following formulation,

$$\phi_\theta^{AB}(t) = (1 - t)^2\theta_A + 2t(1 - t)\theta + t^2\theta_B, \quad 0 \le t \le 1$$

Figure 4 illustrates one such chain in the middle plot.

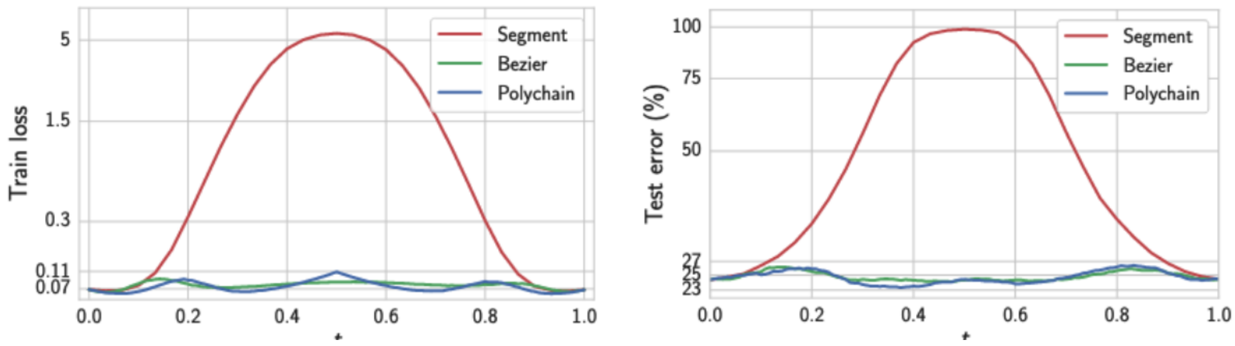


Figure 5: (This figure is a copy of Figure 2 in [Garipov et al., 2018], with the right-mode plot removed as it displays an ensemble only approach.) The l_2 regularized cross entropy train loss (left) and test error (right) as a function of each point on ϕ_θ^{AB} found by the method over two trained ResNet164 models on CIFAR100. "Polychain" corresponds to the single-pivot path version of ϕ_θ^{AB}, "Bezier" corresponds to the quadratic Bezier path for ϕ_θ^{AB}, and "Segment" denotes the direct-linear path between θ_A and θ_B.

3.2.2 Experiments

[Garipov et al., 2018] conduct several experiments on the CIFAR100 and CIFAR10 datasets with a diverse suite of model architectures including DenseNets, VGGs [Simonyan and Zisserman, 2014], ResNets, and Wide-ResNets [Zagoruyko and Komodakis, 2016]. Of these, we choose to highlight results from an experiment using two ResNet164 models with different initializations and trained on the CIFAR100 dataset, as the others follow similar trends. Figure 5 illustrates these results. In each plot the x-axis, labeled as t, represents a distinct location (i.e., model) along the path given by ϕ_θ^{AB}, and the y-axis denotes either its train loss (left-plot) or test-error (right-plot). Thus, $t = 0$ corresponds to $\phi_\theta^{AB}(0) = \theta_A$ and $t = 1$ corresponds to $\phi_\theta^{AB}(1) = \theta_B$.

Overall, the left-panel echoes the results in Figure 4. First, the direct linear path (labeled as "Segment") between θ_A and θ_B immediately incurs very high loss, suggesting that the two networks are not mode-connected via a linear segment. Second, we observe that the polygonal chain achieves significantly lower losses throughout the length of its path, with the Bezier curve yielding the lowest overall error. Notably, based on these results, we could confidently state that Model A and Model B are mode-connected via either a polygonal chain with a single pivot or a quadratic Bezier curve. The right-plot displays very similar trends, but on the test-error instead. One interesting area of note is to observe that the train-loss does not always correspond to high/low test-error. For instance, the polygonal-chain path achieves its highest train-loss near $t = 0.5$, yet this same point appears to yield amongst the lowest test-error. Conversely, we also find that in some regions of t (e.g., around $t = 0.8$), small increases in train-loss (left-plot) correspond to several percentage point increases in test-error (right-plot).

3.2.3 Discussion

Overall, Garipov et al. [2018] present a significantly more appealing and simpler approach to identifying low-loss paths between arbitrary neural networks trained on a task. I really like the approach of this work. The first the result in Figure 4 clearly illustrates how models may seem isolated (left-panel) within each others parameter-loss spaces, yet may actually be connected via simple paths running along different parameter-hyperplanes (middle and right plots). Moreover, they observe that as two models are combined more evenly along the path, the kinds of predictions induced by their merger *change* while still achieving low-loss [Garipov et al., 2018]. Notably, this implies that models along these curves are not degenerate parameterizations (e.g., simple scalings of θ_A or θ_B), but instead have *meaningfully different representations* that can yield *better performance* than the original models. This can be very promising for merging models trained on different configurations of private data. If they can be connected via one such low-loss path, then we may be able to interpolate them along the curve and generate new models that act as if they were trained on *all the data*. However, [Garipov et al., 2018] is not without limitations. Unfortunately, their approach suffers from the same issues presented in Section 3.1.3.

3.3 Limitations of Non-Linear Paths

Ideally, we would like networks to be linearly mode-connected, as it directly implies that we could simply *average* their weights to obtain a model which preserves their strengths. This would in turn, enable us to quickly merge any existing models trained for arbitrary problems, and obtain new models to solve these problems *without needing any training.* Indeed, both [Garipov et al., 2018] and [Draxler et al., 2018] (note how the linear path connecting the two models in Figure 1 is high-loss) observe that by nature a low-loss linear path does not exist between minima. Therefore, they each propose more algorithms to find more complex paths, but which require substantial training and data. While they are able to uncover low-loss paths, the overhead required to find them makes their usage limited for real-world applications as we may not have access to original training data or large compute. Thus, we would really like to find models that are linearly mode-connected (i.e., the linear path is low-loss). *How do we do this?* Unfortunately, there is no guarantee that it is possible. However, it turns out that there are ways we can *transform* trained models such that they are *more likely to be linearly mode-connected.* In the next section, we present three works that illustrate how this can be done.

4 In Pursuit of Linear Mode Connectivity

In this section, we show how to transform arbitrary models to improve their likelihoods of being linearly mode connected (LMC), with the help of [Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023].

4.1 What Does Linear Mode Connectivity Really Mean?

If two models (e.g., Model A and Model B) are linearly mode connected (LMC), there is a direct linear path (e.g., ϕ_{θ}^{AB}) between them that achieves approximately *zero-barrier.* This means the path takes the form,

$$\phi^{AB}(t) = t\theta_A + (1-t)\theta_B. \tag{6}$$

This specification is sufficient to define LMC according to its original definition by [Frankle et al., 2020] and our barrier definition in Equation 2. However, Entezari et al. [2022] propose the slight modification,

$$B(\theta_A, \theta_B, \phi^{AB}) = \max_{t\in[0,1]} \mathcal{L}(\phi^{AB}(t)) - [t\mathcal{L}(\theta_A) + (1-t)\mathcal{L}(\theta_B)] \tag{7}$$

With LMC when $B(\theta_A, \theta_B, \phi^{AB}) \approx 0$. Overall, Equation 7 is extremely similar to our original barrier definition (Equation 2) with the path defined by Equation 6. In fact, it is actually equivalent when $\mathcal{L}(\theta_A) = \mathcal{L}(\theta_B)$. However, Entezari et al. [2022] observe that the modified barrier (Equation 7) is more appropriate because it assigns zero barrier to the loss achieved by a merged model, if it is changing linearly between θ_A and θ_B regardless of whether $\mathcal{L}(\theta_A) = \mathcal{L}(\theta_B)$. In this section, Entezari et al. [2022], Jordan et al. [2023] utilize this new barrier definition, while Ainsworth et al. [2023] use the definition from [Frankle and Carbin, 2018].

Implications of LMC. The existence of LMC between two models is quite powerful. If two models are linearly mode connected, then their parameters not only *lie in the same loss-basin*, but are also related linearly. This means that we can directly average their parameters without *any* negative performance, while simultaneously obtaining a model that captures the knowledge from both original models. Moreover, LMC enables efficient merging of *more than two models.* This is because if several models are LMC, then all these models lie in the same loss-basin and thus we can directly average their parameters together. And we can do all of this *without any* training data or gradient updates!

We now introduce several works that explore the existence of LMC within neural networks.

4.2 The Role of Permutation Invariance in Linear Mode Connectivity of Neural Networks

I believe the best way to introduce this work, written by Entezari et al. [2022], is to state their seminal conjecture:

“*If the permutation invariance of neural networks is taken into account, SGD solutions will likely be linearly mode connected.*”

More concretely, given two networks (e.g., Model A and Model B), they posit that we can likely find a permutation of one model's (e.g., Model B) parameters such that the resulting permuted model is LMC with the first (e.g., Model A). The implications are immense if this conjecture holds. First, it directly implies that we can merge two models by directly averaging their weights *so long as* the correct permutation is known. Second, it means that we do not have to resort to any gradient-based path finding approaches like in [Draxler et al., 2018, Garipov et al., 2018] because the linear-path is always defined as in Equation 6. Entezari et al. [2022] strive to support their conjecture through extensive experimentation and analysis. While I believe that their results unfortunately stop-short of effectively demonstrating this hypothesis, they nonetheless show clear trends where enabling permutations of model parameters lowers the linear-barrier between two networks.

4.2.1 Method

Before describing the exact approach, let us first define what we mean by “permuting the parameters” and “permutation invariance.” First, Entezari et al. [2022] considers permutations to act on the hidden units of each layer within a neural network. This means that when applying permutations, each layer of a network (e.g., the i^{th} layer parameters W_i) is replaced with some $P_iW_iP_{i-1}^{-1}$ where $P_i, P_{i-1} \in \mathcal{P}$ are permutation matrices ($\mathcal{P}$ is the set of permutation matrices), where the permutation matrix P_i acts on the output neurons of W_i and P_{i-1}^{-1} acts on its input neurons. How and why does this happen? To explain, assume the following very simple network: $f(x) = W_2(W_1x)$. If we permute W_1 with some P_1, then $f(x)$ will now be $\hat{f}(x) = W_2(P_1W_1x)$. Now, unfortunately because W_1 was permuted, it's output neurons are no longer aligned with the input neurons of W_2. Thus, we need to also align the input neurons of W_2 to match the new order in W_1. Fortunately, we can achieve this by simply multiplying W_2 is the inverse of P_1, yielding the function: $\hat{f}(x) = W_2P_1^{-1}(P_1W_1x)$. Extrapolating beyond this two-layer network, we can see that every permutation on a layers output must in turn have its inverse applied on the next layer. Note that the permutations applied on the input-neurons of the first layer and output-neurons of the last layer of a network are assumed to be the identity.

Now, what is interesting is that if P_1 is not the identity then $\hat{f}$ and f in our simple example both have *different weights*:f has $\{W_1, W_2\}$ while $\hat{f}$ has $\{W_2P_1^{-1}, P_1W_1\}$. That is, they are technically *different* models. Yet, they are they output the same results for any x: $\hat{f}(x) = W_2P_1^{-1}(P_1Wx) = W_2((P_1^{-1}P)W_1x) = W_2(W_1x) = f(x)$. That is, they are *functionally equal*. This illustrates a fundamental property of permutations: neural networks with elementwise non-linearities (e.g., such as ReLU) are functionally-invariant to permutations. In other words, permuting the parameters of a network does not change its functional output—it behaves as the same function.

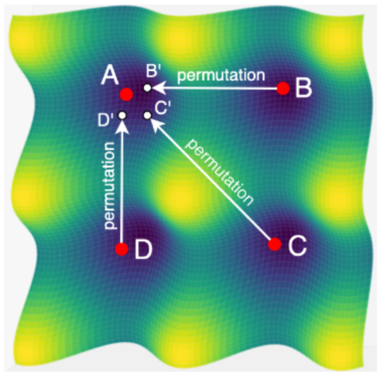


Figure 6: (Copy of left figure in Figure 1 of [Entezari et al., 2022]). Schematic plot of four minima (labeled A,B,C,D) along with valid permutations (B', C', D') which taking them from discrete basins to the same one as A. Moreover, observe that each is now LMC with one another.

The Core Idea and Theorem. Herein lies the core idea of this work: given two networks, we can permute the weights of one such that the new (though functionally equivalent) model lies in the same loss-basin as the other and further the two are LMC. Figure 6 illustrates this idea exactly. The plot is similar Figure 4 and portrays the train-loss surface for four trained networks (A, B, C and D). Initially, we observe that each network appears to be in separate minima, with no clear low-loss path between them (i.e., they are mode disconnected). However, with the right permutations we can map B, C and D to B', C' and D', which are now LMC with A. From there, we can directly average the weights of all networks and achieve a strong model that benefits from the strengths of each. Entezari et al. [2022] package this intuition up into the neat conjecture I introduced their work with. Moreover, they even provide *theoretical support* (albeit extremely limited) for this conjecture. Specifically, they introduce a *theorem* which states that,

> *For any two randomly initialized neural networks of two-layers, the likelihood that there exists a permutation to make them LMC increases (and asymptotically approaches 1) as a function of the hidden state size.*

In other words, two single-hidden-layer networks with random weight (i.e., not trained) are likely to be LMC as the size of their hidden state size (width) grows. Although extremely limited as it only involves very simple networks with random parameters, it is a very nice result which helps support their (and others) empirical findings throughout their field. For instance, it directly supports one of the conjectures of Draxler et al. [2018] that we spoke about in Section 3.1.2.

Theorem Limitation. Yet, I want to highlight another limitation their theorem by focusing on the *rate at which LMC increases.* Specifically, in Figure 7, I am showing the rate at which the barrier between two two-layer models reaches zero (corresponding to them being LMC with probability 1) as a function of their hidden-state size. Interestingly, while we do observe that these networks do approach zero-barrier as their hidden state size increases, the rate at which the barrier decreases is extremely slow. To illustrate, even when the hidden-state size is 2048 ($> 200\times$ the input dimension in my example of 10), the rate of barrier decrease is only 0.7278. Unfortunately, this rate only becomes worse if the input dimensions increase while keeping the hidden-state width fixed. Thus, we observe that the hidden-state size realistically must be *orders of magnitude* larger than the input-dimension in order to achieve acceptable rates of barrier decrease. Nonetheless however, the theorem does provide theoretical grounding to support their conjecture and support for their empirical findings.

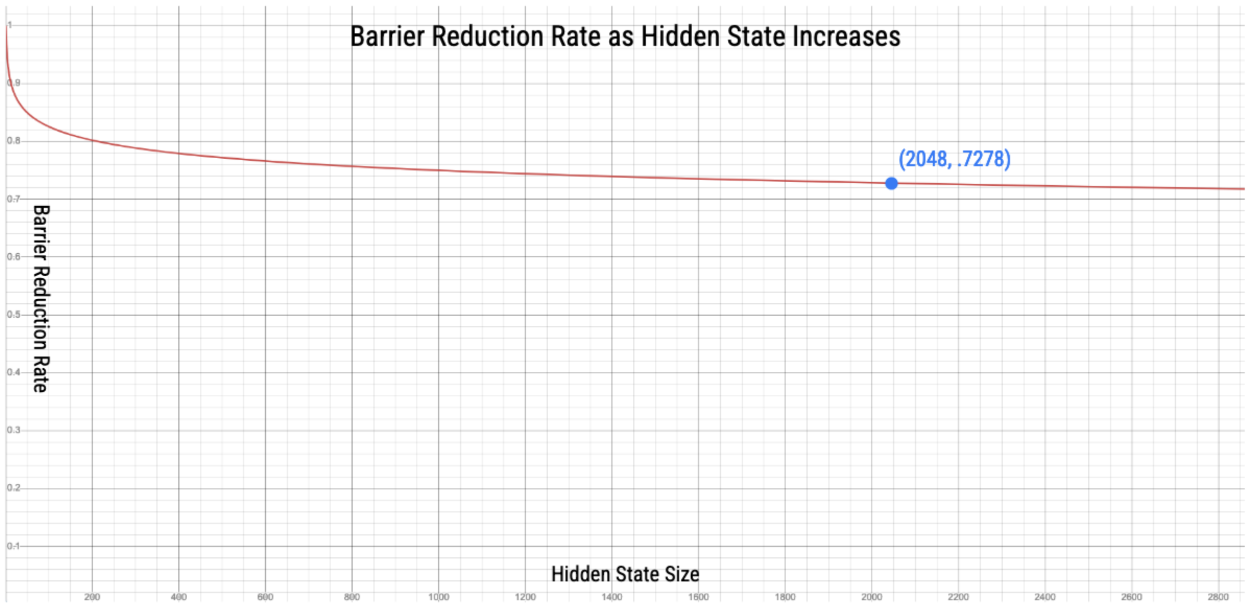


Figure 7: Plot I have created showing the rate at which barrier between two networks in the theorem of [Entezari et al., 2022] reaches zero. The x-axis represents the hidden-state size and the y-axis is associated with the rate of decrease in barrier. The coordinate shows the rate for barrier decrease when the hidden size is 2048: only .7278.

Permutations are a Needle in a Haystack. Additionally, finding the "right permutation" is no trivial task. This is principally due to the factorial growth of permutations with the size of hidden units within each layer. For instance, even a layer as small as 16 dimensions has a total of $\approx 2 \cdot 10^{13}$ valid permutations! And this is just for a single layer; the number scales multiplicatively with the depth of the network. With a candidate space this large, finding the right permutation via brute force is computationally infeasible. Therefore, Entezari et al. [2022] investigate other methods of finding these permutations.

The Approach. The problem of finding the correct permutation is a variant of the Traveling Salesman Problem which belongs to the class of NP-hard optimization problems. However, simulated annealing (SA) is often the method of choice for solving these problems [Entezari et al., 2022], and so Entezari et al. [2022] adapt for their problem as well. SA can be described as iteratively searching for the best permutation among a set of candidates to minimize an objective function, where the candidate pool gets more focused after each iteration according to best achieved objective value. After convergence, the permutation achieving the lowest objective value is returned. In [Entezari et al., 2022], the set of candidate permutations are all valid permutations across all layers of a network. Once a permutation is sampled from the set, it is applied to a model (e.g., Model B) and then the resulting weights are directly averaged with the weights of a second (e.g., Model A) to obtain a new merged model (e.g, Model C). The quality of the chosen permutation is then measured by the loss over the training set of the merged Model C. If this loss is lower than that of a previously sampled permutation, then this new permutation is stored as the "correct" one, and the loop repeats.

Now, it is important to note that SA is not guaranteed always find the right permutation—even if one does exist. In fact, it may instead fail, leading to permutations that do not move models into the same basin and thus do not make them LMC. Such a failure inevitably leads to high linear-barriers and poor merges. Unfortunately, as we will see in the experiments section, SA almost never finds a permutation to make two models LMC, exemplifying the difficulty of this problem.

4.2.2 Experiments and Discussion

Traditional Experiments. Entezari et al. [2022] conduct extensive experiments to observe how diverse (though simple) neural networks tend to become more (or less) LMC as their architecture changes across multiple datasets. Specifically, they investigate LMC via permutations on MLP and CNN models with no residual connections on the MNIST [LeCun and Cortes, 2010], SVHN [Netzer et al., 2011], CIFAR10 and CIFAR100 datasets. By default their MLP has just one hidden layer with 2^{10} neurons, and they refer to the CNN as "Shallow CNN" with two-convolution layers and equal hidden dimension to the MLP model. However, they experiment with increasing both the hidden state sizes (widths) of these networks, and their depth while keeping their width fixed. They train several models of varying depth and width on all the aforementioned datasets using SGD, all with different initializations.

An Alternative Experiment. Now, Entezari et al. [2022] also examine the extent to which permutations enable models to become LMC via an alternative angle. They propose this in an attempt to circumvent degenerate cases when SA fails to find a good permutation and thus still show a link between permutations and LMC. In this second setting, they refer to all the above trained models as "Real World" and treat them as a collective set (denoted as $\mathcal{S}$). Then, they create a second set, labeled "Our Model" and denoted by $\mathcal{S}'$, of networks which consist of one trained network of any kind (e.g., a Shallow-CNN) and random permutations of it. Thus, every model in the "Our Model" set is functionally equivalent to each other. Their objective is then to show that $\mathcal{S}$ and $\mathcal{S}'$ are similar to one another in terms of barrier behavior. They claim that doing this would imply that for every model in $\mathcal{S}$, one should be able to find another in the set that that it is LMC to the former via permutation, thereby proving their conjecture.

Results. Figure 8 showcases the results of both experiment types together. There is a lot of information in this figure. The top cell shows four panels, each plotting the train-data barrier for different networks without taking permutations into account. This is equivalent to interpolating along the direct linear path between two networks, as is pictorially shown in our discussion of [Draxler et al., 2018, Garipov et al., 2018]. The top-left-most panel shows how the barrier changes for the MLP as its width increases across the four datasets, for both the "Real World" and "Our Model" sets. Overall, we observe two important things.

First, zero-barrier is achieved as the width increases for the small MNIST dataset, with similar (though slower) behavior on SVHN. Note how the more complex CIFAR datasets do not exhibit this behavior. Due to my own research, I can point to this as occurring as a result of data complexity. More specifically, we consistently observe that as the model capacity increases with respect to the difficulty of the dataset, it becomes over-parameterized. This in turn increases the likelihood that arbitrary neurons between two equally over-parameterized networks are "redundant" or highly similar. When this happens, we are far more likely to be able to simple interpolate between them without any cost to barrier, thereby increasing the probability of LMC. Thus, width and data complexity go hand-in-hand.

Second, we observe that models from the "Real World" and "Our Model" sets effectively achieve the same

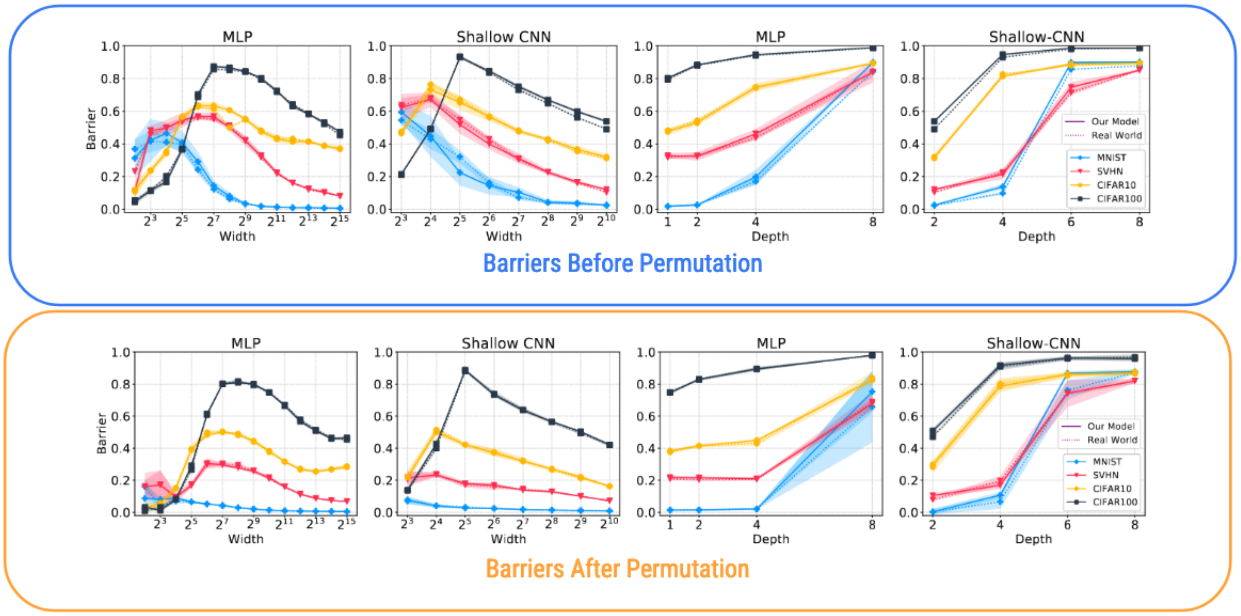


Figure 8: (Altered version of Figures 5 and 7 from [Entezari et al., 2022]) Top: several plots showing barrier as a function of width and depth with different networks in both $\mathcal{S}$ and $\mathcal{S}'$. Bottom: several plots showing barrier as a function of width and depth using the same networks as top among $\mathcal{S}$ and $\mathcal{S}'$. Results are the mean over 10 different runs and shaded regions are the variance.

barriers throughout width changes. Entezari et al. [2022] use this to imply that the two sets (i.e., $\mathcal{S}$ and $\mathcal{S}'$) appear equivalent, and thus models in the "Real World" are permutatable with LMC. However, I believe this result is as convincing. What it does show is that models that *are exactly LMC via permutations* do tend to yield similar barrier with other models from $\mathcal{S}$. However, it does not mean that arbitrary neural networks are themselves LMC via permutation.

Similar to the top-left plot, the top-second-left-panel shows that the Shallow-CNN follows the trends of the single-layer MLP as width increases. The sole difference appears to be that the CNN model achieves low barrier slower than the MLP model. This could be explained by CNNs containing fewer parameters than their fully-connected dense layer counterparts in MLPs [LeCun et al., 2010].

Turning our attention to the top-right plots (the second and first from the top-right), we interestingly observe that as the same networks get deeper (keeping the width of each hidden layer 2^{10}), their barriers shoot up. We observe this for both kinds of models, across both $\mathcal{S}$ and $\mathcal{S}'$, and across all four datasets. This directly suggests that while network width tends to decrease the barrier, depth plays an opposing role. In fact, it appears that depth plays a *significantly greater adversarial effect* on the barrier than width. Indeed, with additional experiments conducted with deeper models such as the VGG and ResNet architectures, they observe that increasing the width by similar scales has *almost no effect* on reducing the barrier before permutations! In fact (as we will soon see in the next paper: [Ainsworth et al., 2023]) when networks are very deep, their widths must be *many times larger* in order to be able to achieve LMC.

Now, the barrier story improves when permutations are taken into account (bottom plots). The bottom figures show how the linear-barrier changes between the same models in the top plots for the same experimental settings, but when permutations are taken into account via the SA algorithm. Overall, we observe that for simple datasets (e.g., MNIST and SVHN) LMC is achievable significantly faster with increased width than without permutations. Together with the observation that the barriers from $\mathcal{S}$ and $\mathcal{S}'$ are overlapping, it demonstrates how permutations help tie these neural networks to be LMC. Additionally, we observe that for the same datasets, permutations yield lower linear-barriers as the networks get deeper (right plots). Specifically, note the substantially increased variance when applying permutations: some are able to yield much lower barriers!

However, when looking at the more complex datasets such as CIFAR10 and CIFAR100, we find that the permutations achieved by SA only negligibly reduce the barrier. While this result is not ideal, I believe it speaks to our earlier discussion between model capacity and dataset complexity. That is, it could imply that the networks utilized are not sufficiently expressive to have high likelihood of LMC even under permutations.

Despite these shortcomings, the authors overall provide compelling results that permutation-invariance does appear to reduce the linear-barrier between arbitrary models—and can even make them LMC under certain conditions! They conduct an extensive study across different architecture-families on four benchmark datasets while varying width, depth, and whether permutations are allowed. They show how width tends to improve the barrier, while depth acts very adversarially. However, in addition to the limitations we find in their core analysis, their results are further limited by the restriction to investigating permutation-invariance on simple networks. That is, they do not show experiments using permutation-invariance on more realistic networks such as the VGG and ResNets. Together with the inconclusive LMC results with MLPs and Shallow-CNNs, it is difficult to extrapolate much from their work in terms of broader applicability. Indeed, the question still partially remains: are realistic networks LMC when permutation-invariance is taken into account? E.g., are ResNets LMC via permutation? Fortunately, as we will now see in the next work [Ainsworth et al., 2023], the answer can be yes.

4.3 Git-Rebasin: Merging Models Modulo Permutation-Symmetries

This work by Ainsworth et al. [2023] is the first to show the existence of linear mode connectivity (LMC) in modern neural networks such as ResNets on the same datasets studied by Entezari et al. [2022]. Specifically, the extend Entezari et al. [2022] by proposing three new algorithms for finding the "right" permutations connecting two arbitrary neural networks together, and show that they enable LMC on CIFAR10 with ResNet20s. Additionally, they conduct two more interesting experiments. The first suggests the existence of LMC between networks is a function of SGD rather than a particular network architecture. This implies that LMC may exist across *any* type of neural network. The second shows that networks trained on *completely separate data* can be merged via their methods into one model that outperforms the original ones. That being said, I would like to caution the broader implications of these results; they each have their own shortcomings and limitations. We now first describe their method. Second, we highlight and discuss their experiments.

4.3.1 Method

Recall the typical method setting: we are given two trained networks (eg., Model A and Model B) and would like to measure whether they are mode-connected. Ainsworth et al. [2023] closely follow the formulation presented in [Entezari et al., 2022] and search for linear mode connectivity between the two networks via permutations. To this end, they propose three distinct methods for finding the "right permutations" for each model. We summarize these below.

4.3.1.1 Matching Activations

The first method is motivated by the intuitions of [Hebb, 1949, Li et al., 2015]: networks with similar feature activations accomplish the same task. In extension, Ainsworth et al. [2023] argue that if the hidden-units of two networks can be matched up to one another—so that their activations are similar—there may be a linear relationship between their weights, making the two networks LMC. Concretely, let Z_l^A, Z_l^B be the feature activations from the l^{th} layer of Model A and Model B. Assume that layer-l contains a d hidden units, and we have computed activations for n training examples. Then, $Z_l^A, Z_l^B \in \mathbb{R}^{d\times n}$. Ainsworth et al. [2023] then propose to find the permutations that align Z_l^B with Z_l^A by optimizing the following objective at each layer,

$$P_i = \arg\min_{P\in\mathcal{P}} \sum_{i=1}^{n} ||Z_l^A - PZ_l^B||_F^2 = \arg\max_{P\in\mathcal{P}} \langle\, P, Z_l^A (Z_l^B)^T \rangle_F \qquad (8)$$

where $\mathcal{P}$ is the set of all valid permutations as in [Entezari et al., 2022], and $||\cdot||_F$ denotes the Frobenius norm. While Equation 8 may appear challenging to solve, Ainsworth et al. [2023] actually find that it constitutes a "linear assignment problem" (LAP) [Bertsekas, 2001]. Moreover, LAP problems can readily be solved by efficient and practical polynomial-time algorithms, and Ainsworth et al. [2023] make use of these in their solution. Additionally, note that matching activations using this method enables *all* layer-permutations between the two models to be solved *independently* of one another. This allows the matching activations approach to be even faster, as permutations for each layer can be solved in parallel, and then all can be applied to Model B at once. Yet, this comes at the cost: training data (though without labels) is required to find these optimal permutations. Despite this shortcoming, many following approaches we will discuss rely on activations to transform and merge models.

4.3.1.2 Matching Weights

Do we really need access to training data? This second—and their most well known—method seeks to answer this question by instead determing the best permutation using only model weights themselves. Specifically, let $W_1^A, \ldots, W_L^A$ denote the layer-weights from the first to last (L) layer in Model A, and let Model B be parameterized in the same way. For simplicity, assume the bias is contained in the weight matrix. The intuition is to think of rows within each weight as individual features (units). Then, if row-i in W_1^A is $\approx$ to row-j in W_1^B, the two rows should be associated. A similar line of reasoning can be used to associate the columns of two network weights as well. Together, this intuition offers a way to think about how the input-space (i.e., weight column-vectors) and output-space (i.e., weight row-vectors) of a layer in Model B can be associated with the input/output space in the corresponding layer of Model B Extending this idea to every layer we can express the optimization problem in terms of the full weights,

$$\underset{P_1,\ldots,P_{L-1}}{\arg\max}\ \langle W_1^A, P_1 W_1^B\rangle + \langle W_2^A, P_2 W_2^B P_1^T\rangle + \ldots + W_L^A, W_L^B P_{L-1}^T\rangle \tag{9}$$

where P_i are valid permutations. Note that P_L is taken to be the identity matrix as in [Entezari et al., 2022], and thus only permutations up to layers $L-1$ need to be computed. Unfortunately, solving this problem for all layer-permutations jointly is not possible in polynomial-time. Indeed, it is a variant of the bilinear assignments problem which is NP-hard and non-polynomial whenever $L > 2$ (i.e., in all realistic networks). However, Ainsworth et al. [2023] make the clever observation that if we instead fix all permutations but one—say P_i—and solve Equation 9 for just unfixed permutation (P_i), then the optimization problem reduces to an LAP! We illustrate it below,

$$P_i = \underset{P_i}{\arg\max}\langle W_i^A, P_i W_i^B P_{i-1}^T\rangle + \langle W_{i+1}^A, P_{i+1} W_{i+1}^B {P_{i-1}}^T\rangle_F$$

$$P_i = \underset{P_i}{\arg\max}\langle P_i, W_i^A P_{i-1}(W_i^B)^T + (W_{i+1}^A)^T P_{i+1} W_{i+1}^B\rangle_F \tag{10}$$

This conveniently leads to a coordinate descent algorithm. Specifically, we create a loop where each iteration first samples random permutations for all but permutations but P_i. Second, keeping all the random permutations fixed, Ainsworth et al. [2023] solve Equation 10 using an LAP solver, and save its best permutation. Third, we choose another layer—say j—and randomly sample permutations for all layers that we have not yet computed permutations for. Note that since we have already computed the permutation for layer-i, we do not assign it a random permutation this time, but rather use P_i. We then repeat step two and continue: each step fixing all permutations but one, computing its optimal permutation and storing it, until convergence. In this case, convergence means that recomputing each P_i in each successive pass does not change the permutation.

This approach has two unique benefits. First, this method-version can be solved on the order of *seconds*. Second, and perhaps more importantly, is that this variant does not require any access to data of any kind! This can be especially useful in settings where we may want to merge the information of two models trained using completely distinct data, but do not have access to the data itself. *All* prior approaches we have covered in this report are immediately unusable in this setting as each requires access to at least some training data. However, with direct weight-matching we can align two models and easily combine them without this constraint.

Of course, lack of knowledge of even a little bit of the problem's data-distribution can severely handicap the ability to find the optimal permutation. I believe a simple way to see this (which I have found through my own research) is that computing permutations based on the network-weights alone puts equal emphasis on each *basis* component of each layer. In other words, weight-matching treats every basis-direction in the network weights equally. However, the benefit of using data (i.e., an activations-based approach) is that it naturally (de-)emphasizes certain basis components of each layer. Thus, data provides a signal towards which parts of the layer-input/output space are actually meaningful. This in effect helps steer the activation-based algorithms to find permutations that better focus on these regions, thereby leading to better network alignment and higher probability of LMC. Overall though, if weight-matching achieves LMC between two networks, it would be very powerful.

4.3.1.3 The Straight-Through Estimator

Inspired by (at the time) recent successes of straight-through estimators [Bengio et al., 2013], Ainsworth et al. [2023] also propose a method that adapts them to permutation selection. Notably, this method finds the best permutation by optimizing an objective function with SGD and requires access to the full training data, making it extremely expensive. Despite the additional cost, it does not yield significant improvements over first two methods—and even the authors Ainsworth et al. [2023] discount its usefulness. Thus, we simply mention it for completeness—especially as it is present in some of the experiments we will highlight—but it is not necessary to go into exacting detail of its implementation.

4.3.2 Experiments and Discussion

In this section, we highlight three sets of experiments that each showcase different interesting takeaways regarding the existence of LMC when permuting two networks.

LMC Across Models. The first, and among the most compelling results from this work are in demonstrating that LMC can exist for ResNet models of sufficient width. Specifically, Ainsworth et al. [2023] show that two ResNet20 (with 32× width multiplier) models trained on CIFAR10 with different initializations achieve LMC when one is permuted using the weight-matching algorithm (Section 4.3.1.2). Moreover, this is achieved with the weight-matching algorithm, so without using *any* task data, Thus, it *does* support the conjecture from [Entezari et al., 2022] in that even more complex architectures can be LMC via permutation! These results are captured by the second-right panel in Figure 9.

However, this result does have obvious concerns. First, CIFAR10 is a very simple dataset and a 32× width multiplier adds several orders of magnitude more parameters to a standard ResNet20 model. This makes it vastly over-parameterized for the CIFAR10 dataset. Second, this is missing from the main paper (found in the appendix), but *all their results* with ResNet models swap their batch-normalization layers for layer-norm, which does not track running statistics. One the one hand, this change is necessary if we want to avoid using data—like in the weight-matching approach—because the lack of running stats means no forward pass to reset statistics is necessary. One the other-hand, it is no longer a standard ResNet model, and so is not a complete result on finding LMC with ResNets.

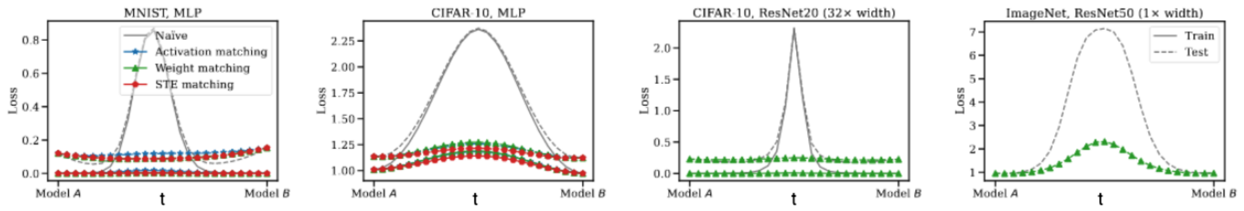


Figure 9: (Copied from Figure 2 of [Ainsworth et al., 2023]) Loss along the linear-path between models trained on MNIST, CIFAR10 and ImageNet [Deng et al., 2009]. In all cases, methods from [Ainsworth et al., 2023] significantly improve upon naively interpolating two networks. Left: Results of merging two 3-layer MLPs with 512 hidden-size in each layer. Second-left: Results from merging MLPs with the same architecture as left-panel on CIFAR10. Second-right: Results of merging ResNet20 models with 32× standard width on CIFAR10. Right: Results of merging ResNet50s with standard width on ImageNet.

In addition, observe that LMC is found for trained MLPs of different initializations on the MNIST dataset (Figure 9 left). Each MLP contains three 512-unit hidden layers. In fact, two approaches—weight matching and the straight through estimator (Section 4.3.1.3)—can obtain *better* models from interpolation than the original trained ones. This further shows how the existence of LMC—and by extension low-loss paths—between networks enables us to interpolate them and generate models that can improve upon each. Second, note how when MLPs of the same architecture are merged on the CIFAR10 dataset, no permutation achieving LMC is found. This corroborates the intuition that dataset difficulty compared to model capacity plays an important role in determining whether LMC is achievable via permutations. Lastly, we observe that for the more complex ImageNet dataset, Ainsworth et al. [2023] are unable to find LMC between ResNet50s (this time with batch-normalization) using any method. While this may appear like a poor result, permutations still lower the barrier by a significant margin compared to without them (i.e., the naive direct linear-path between models). Moreover, it still shows how permutations may be utilized to achieve better merged models than we might obtain if we simply averaged their weights directly. Thus, these results on the whole showcase promise in that permutations can be utilized to bring us closer to LMC and guaranteed strong model-mergers.

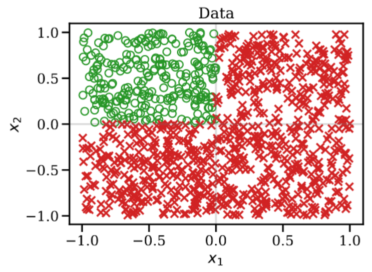


Figure 10: (Copied from Figure 7 of [Ainsworth et al., 2023]) Visualization of the toy two-dimensional classification problem. Red is data from one class, and green is data of the other.

LMC is a Function of SGD. In an effort to better understand what network properties may cause the emergence of LMC, Ainsworth et al. [2023] create two artificial methods for a two dimensional classification task. Figure 10 visualizes this problem. The data is such that you can easily fit two two-hidden-layer MLP architecture with ReLU non-linearities to exactly solve this problem using discrete (non-SGD) weights:

$$f_A(x) = \begin{bmatrix}-1 & -1\end{bmatrix} \sigma\left(\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix} \sigma\left(\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix} x + \begin{bmatrix}1\\0\end{bmatrix}\right) + \begin{bmatrix}1\\0\end{bmatrix}\right)$$

$$f_B(x) = \begin{bmatrix}-1 & -1\end{bmatrix} \sigma\left(\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \sigma\left(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} x + \begin{bmatrix}0\\1\end{bmatrix}\right) + \begin{bmatrix}0\\1\end{bmatrix}\right)$$

Notice that these two networks are *distinct* (i.e., they are not permutations of one another), they each perfectly classify the data of the toy problem, and they are *non-SGD* solutions. Interestingly, Ainsworth et al. [2023] show that *no permutation* exists that brings f_A and f_B to be LMC. This suggests that success stemming from LMC via permutations is an inherent property of SGD training, rather than network architecture. Although the setting is limited, it is a comforting result because it suggests that we may be able to find LMC behavior across *any* architecture—so long as it's wide enough.

LMC with Different Datasets. Another very nice experiment we highlight is one where Ainsworth et al. [2023] explore combining models trained on *separate datasets*. More specifically, they split the CIFAR100 dataset into two disjoint subsets. The first, dataset A, contains 20% of the examples from the first 50 classes in the train set, and 80% of the examples from the last 50 classes in the train set. The second, dataset B, contains the remaining 80% examples from the first 50 classes and similarly the last 20% examples from the last 50 classes in the train set. Thus, datasets A and B are disjoint but crucially are still apart of the same overarching classification task. Moreover, the two datasets share the *same output labels*. This setting is very interesting because it is the first we have studied in this report that actually takes privacy into consideration. We can imagine for example a case where an organization has multiple separate datasets (with their own biases) that cannot be combined due to regulatory or privacy concerns. In this case, training separate models on each dataset is feasible, but not jointly on all data in aggregate. Thus, a success in this setting can pose decent implications in real-world regulatory settings.

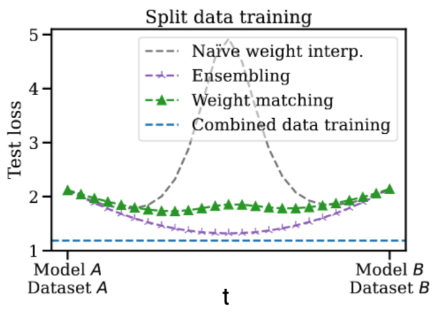


Figure 11: (Copied from Figure 5 of [Ainsworth et al., 2023]) Models trained on disjoint datasets can be merged with lower test loss.

Figure 11 illustrates the results from merging two ResNet20 (widths unknown) models trained on dataset

A and B respectively with different initalizations. The figure shows the test-loss on the y-axis and the interpolation amount on the x-axis. Overall, we find that training one model on all the data in aggregate achieves the best performance, followed closely by the ensemble. These are both important metrics to consider in the disjoint dataset setting, as they show the effective upperbound on what we may hope to achieve with model merging. Ideally, we would like to design approaches that take us as close as possible to these two methods after weight-interpolation. While the naive weight interpolation performs poorly, we observe that the weight-matching approach seems to perform surprisingly well. That is, models interpolated along its path yield better test-loss than the original models, and without using any data for matching! If we restrict ourselves to just these results using test-loss, they suggest that weight-matching can be an effective tool for combining networks in privacy settings.

However, observe how the story changes if we instead plot the same models but showing the test-accuracy (Figure 12). When the test accuracy is considered, we conclusively find that weight-matching *almost-never* achieves a better model than the original ones. Thus, it is important to take results showing mode connectivity with *just loss* with a grain of salt, as loss is *not always* a good indicator of strong performance. Does this mean LMC—and by extension mode connectivity—is meaningless? Not exactly, even in Figure 12 we notice that some small regions in along the linear-path between the two networks achieve better accuracy than the individual models. Similarly—speaking from my own research—strong test performance across linear model-interpolations tends to indicate LMC in the loss-space. So the two are directly tied to each other. Moreover, as we will show next, there exist methods (e.g., [Jordan et al., 2023]) that use LMC to perform very well on downstream tasks as well.

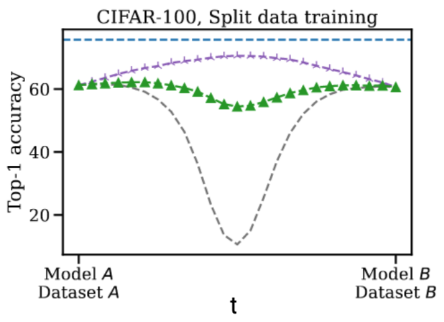


Figure 12: (Copied from Figure 10 of [Ainsworth et al., 2023]) Models trained on disjoint datasets can be merged with lower test accuracy.

4.4 REPAIR: REnormalizing Permuted Activations for Interpolation Repair

The authors of this work, Jordan et al. [2023], significantly improve upon the findings of [Entezari et al., 2022] (and by extension [Ainsworth et al., 2023]) by identifying and addressing a prevalent issue with model interpolation. Specifically, they observe that the hidden-units of interpolated models undergo *variance collapse*, even when their source models are permuted such that their features match (e.g., through SA of [Entezari et al., 2022]). That is, they find that the variance of hidden-layer unit activations across many models progressively decay as we move deeper into their networks, with the activations of later layers becoming nearly constant [Jordan et al., 2023]. The left-panel of Figure 13 illustrates this phenomenon. From the figure, we notice that the activation variances of successively deep hidden-layers of merged models reduce significantly compared to the activation-variances of the original networks (diamonds) regardless of the network architecture. To this end, Jordan et al. [2023] propose a novel algorithm, termed "REPAIR" which acts on top of existing permutation-matching approaches (e.g., such as those from [Entezari et al., 2022, Ainsworth et al., 2023]) to rescale the hidden-unit activation variances post interpolation back to that of the original networks'. This significantly improves the model-interpolation accuracy and barriers (middle and right panels in Figure 13).

Why Does Variance Collapse Occur? [Jordan et al., 2023] use a simple example to help explain why variance collapse occurs. consider a hidden unit/channel in the first layer of an interpolated network (e.g., Model C) obtained from two aligned networks (e.g., Model A and Model B). This unit will be functionally equivalent to the linear interpolation between the respective units in the original networks. Specifically, if we represent this unit's preactivation as X_C and those of the original models as X_A and X_B respectively, then $X_C = tX_A + (1-t)X_B$ for some $t \in [0, 1]$ following our prior notation. If the correlation between X_A and X_B is ≈ 0.4 and $std(X_A) \approx std(X_B)$ as [Jordan et al., 2023] state is typical for two aligned networks, then they show that $Var(X_C) \approx 0.7Var(X_A)$. So for a single hidden unit in the very first layer of the network, a drop of 30% in its activation variance can be expected after interpolation. Although their analysis cannot be rigorously extended to deeper layers in a network, this error is nonetheless expected to compound with depth.

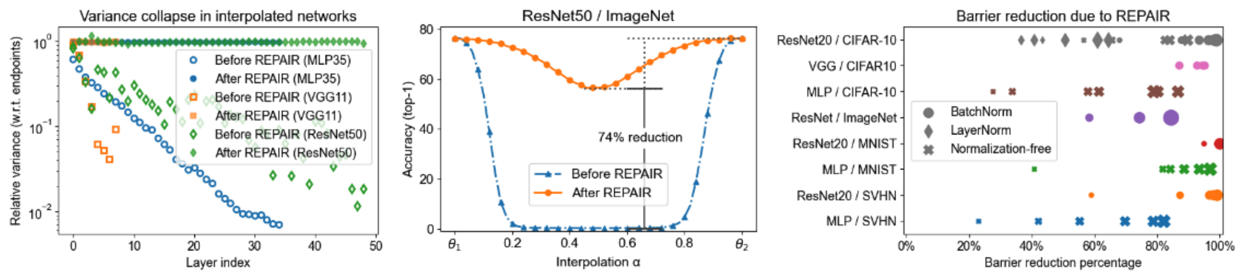


Figure 13: (Copied from Figure 1 of [Jordan et al., 2023]) Each plot shows different aspects from merging two independently trained models with different initializations after permutation-alignment. Left: The variance of activations in interpolated networks progressively collapses as the layers get deeper. Average variance across each layer is reported, and normalized by that of the corresponding in the original two endpoint models. Their method, termed "REPAIR" corrects this phenomenon by rescaling the variances back to the sizes of the original networks after permuting them via an auxiliary method. Middle: REPAIR significantly reduces the barrier of the interpolated ResNet50 model on the challenging ImageNet dataset compared to simply permuting networks and then interpolating them. Right: REPAIR reduces the interpolation barrier across many different architectures, training sets and normalization layers. Layer markers indicate wider networks.

A Different Matching Algorithm. Before describing their core method, we first note that Jordan et al. [2023] utilize a different permutation-finding approach than [Entezari et al., 2022, Ainsworth et al., 2023]. Similar to the activation-matching approach of [Ainsworth et al., 2023] (Section 4.3.1.1), Jordan et al. [2023] also use the activations from unlabeled training data in their matching algorithm. However, they instead compute the pairwise correlations between the activations of each hidden unit in one model with that in another, and search for the permutation that maximizes this correlation along the diagonal. Borrowing from the notation introduced in Section 4.3.1.1, we formally write their optimization objective as,

$$P = \arg\max_{P \in \mathcal{P}} \langle P, \mathrm{corr}(Z_l^A, Z_l^B) \rangle_F \tag{11}$$

where "corr" computes the pairwise correlations. We note that this criterion and approach is not theirs: it is instead introduced by [Li et al., 2015]. Similar to the matching-activations criterion from [Ainsworth et al., 2023], this objective is also a variant of an LAP and can thus be solved very efficiently and quickly with the Hungarian Method.

We now discuss the REPAIR method.

4.4.1 Method

[Jordan et al., 2023] proposes two variations to fix variance collapse in hidden-unit activations—both called REPAIR. However, their first method consistently performs worse than their second and further is at best only an *approximate* solution to the the issue at hand [Jordan et al., 2023]. In fact, it appears to me that the first method is only meant to motivate the second. Thus, we opt to describe the second variant in full detail, as it is used across all experiments shown below.

First, let us explain the objectives of REPAIR. Assume the same example from our "Why Does Variance Collapse Occur?" paragraph. That is, we are given two trained networks Model A and Model B that are aligned to one another via some auxiliary permutation-solver, and we further interpolate these models to obtain a Model C. Jordan et al. [2023] would like scale specific units to prevent variance collapse. For instance, if the units from the example occurred from a VGG network, Jordan et al. [2023] aims to correct the preactivations of every convolutional layer. Similarly, for ResNets Jordan et al. [2023] attempt to correct both these convolutional preactivations and the outputs from each residual block. With these in mind, the goal of REPAIR is to make the following two conditions hold:

$$\mathbb{E}[X_C] = (1-t)\cdot\mathbb{E}[X_A] + t\cdot\mathbb{E}[X_B], \tag{12}$$
$$std(X_C) = (1-t)\cdot std(X_A) + t\cdot std(X_B). \tag{13}$$

Now, the first condition written as Equation 12 holds by default due to linearity of expectation. However, [Jordan et al., 2023] claim that $std(X_C) \ll \min(std(X_A), std(X_B))$ before any corrections due to variance collapse. And importantly, they intuit the collapse only worsens in deeper layers. Thus, REPAIR aims to re-scale the interpolated model's standard-deviation.

The Forward-pass Exact Variant. [Jordan et al., 2023] choose to name this REPAIR method as the "forward-pass exact" variant because it utilizes a forward-pass (plus a few invasive other components) to exactly induce Equation 13. However, doing this is non-trivial and requires several steps. The first involves calculating $\mathbb{E}[X_A], \mathbb{E}[X_B], std(X_A), std(X_B)$ for each selected channel/hidden-unit in the original networks Model A and Model B respectively. [Jordan et al., 2023] do this by defining a PyTorch layer that computes these statistics on the fly with forward passes on the training data. They then apply these layers at each layer (i.e., by invasively overwriting the layer to include the wrapper) in Model A and Model B where they would like to track the statistics. They then compute a forward pass. Afterwards, they define a different PyTorch wrapper module that wraps a given layer by adding a batch-normalization layer after it. Jordan et al. [2023] then apply this wrapper module to the same set of layers as in the original networks—but this time on Model C. After all this, Jordan et al. [2023] linearly interpolate all parameters between Model A and Model B as standard to obtain Model C. However, they set the mean and standard-deviation of each added batch-normalization layer in Model C using the computed statistics from the tracking layers in Model A and Model B. Specifically, they assign the mean and standard-deviation values following Equations 12-13. Finally, they reset the batch-normalization statistics of *all* batch-norms (including the new ones) with another forward pass on the training data (as in prior work).

This approach is the most invasive with respect to the architectures of each original models, requiring several layer injections and added parameters. However, Jordan et al. [2023] make explicit note that the added batch-normalization parameters can be fused into the convolutional layers they lie on top of using [Nenad, 2018]. While Jordan et al. [2023] do not do this in any of their experiments nor include code for this, REPAIR nonetheless appears to be able to remove these added parameters.

4.4.1.1 An Observed Limitation: Merging > 2 Models.

One particular limitation in this approach though has to do with setting where we would like to merge *more than two models* together. For simplicity, let us assume we would like to interpolate three models: Model A, Model B and Model C. This is simple enough (though not perfect) with [Entezari et al., 2022, Ainsworth et al., 2023]: we can pick one model and then permute the parameters of the other two models to align with it. Then, we can directly average all parameters of all models. I note this approach isn't perfect because no matter how many models we would like to merge, (1) we can only align *two* models at a time and (2) must choose one model ahead of time to merge all others too. To explain further, imagine that Model B and Model C are LMC via permutation, but neither is LMC to Model A. If we choose to align Model B and Model C to Model A, we may get worse results than if we picked Model B as the model to align to. Despite this shortcoming, once we have aligned the models we can directly interpolate their parameters.

However, it is unknown how we would apply REPAIR in such a setting. It seems the only solution is to first choose two models to merge (e.g., Model A and Model B) and fix their hidden-unit variances via the method. Then it seems that the next step would be to fuse the added batch-normalizations into the preceding convolution layers, merge this new interpolated model with Model C, and finally reapply REPAIR. However, it is easy to see how this solution may lead to even worse results than before: Model C will be aligned to the interpolated model from Model A and Model B. If the latter two are not LMC, then aligning Model C will be very noisy and yield poor merge performance. Instead, I believe that we would ideally like to be able to compute the exact variance rescaling of a joint interpolation from all three models. This would allow us to bypass this sub-optimal sequential merging approach. Unfortunately however, [Jordan et al., 2023] do not consider this generalization of their method and so do not include any experiments/intuition for addressing this setting. Thus, this could be an interesting direction for future work.

4.4.2 Experiments and Discussion

We now highlight three experiments from [Jordan et al., 2023]. One aspect I really enjoy about the reported experiments from [Jordan et al., 2023] is that nearly all are in terms of accuracy—not loss as in [Ainsworth et al., 2023].

REPAIR Closes the Gap on LMC. Figure 14 showcases how REPAIR improves the test-accuracies of interpolated models throughout interpolation amount. The results are for ResNet18 and ResNet20 models trained on the CIFAR10 datasets, each with different initializations. In the left-panel we observe near constant accuracy for any interpolation amount after utilizing REPAIR for the ResNet18 model. Comparing this result with the hump caused by pre-REPAIR interpolation, it seems like REPAIR is capable of causing LMC to emerge betwen models that would otherwise not appear connected. Similarly, the right-panel showcases the accuracy curve on a ResNet20 with width multiplier of 8×. Notably, we find that REPAIR can improve the accuracy of an interpolated model by a significant amount when using batch-normalization. This is especially significant as Jordan et al. [2023] appear to achieve near equal results to [Ainsworth et al., 2023] (Figure 9), but with 4× less width multiplier! And this is with a ResNet20 using batch-normalization, as is standard. Additionally, we observe that REPAIR can improve the accuracy gap when architectures use layer-normalization instead. Note, we still have the same weakness as in [Ainsworth et al., 2023]: these models are severely over-parameterized for this task. Overall though, these results indicate that REPAIR helps close the gap on showing LMC between models.

Additionally, Figure 15 displays similar results when using ResNets for the much harder ImageNet classification task. Here, we clearly do not achieve LMC via any interpolation, yet REPAIR is consistently able to significantly improve the accuracy compared to without using it (left-panel). We also observe that REPAIR significantly improves ImageNet accuracy when the ResNet size and width are increased (right-panel). These results seem to support the Entezari et al. [2022]'s conjecture regarding model width and LMC: increase in model width corresponds to higher likelihood of LMC.

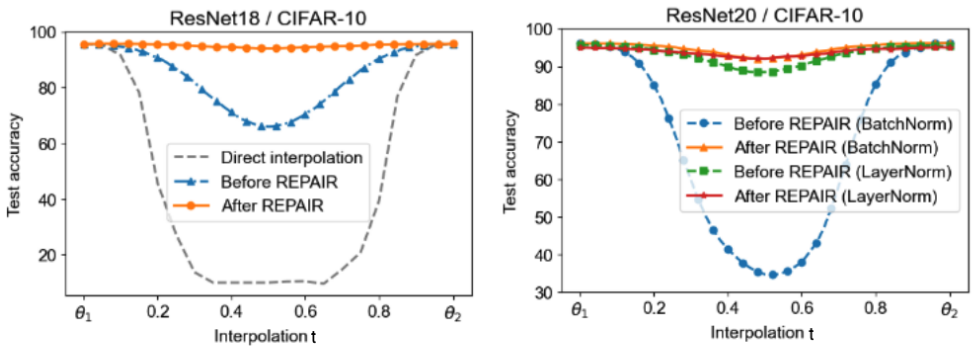


Figure 14: (Concatenation of Figures 3 (middle) and 4 (right) from [Jordan et al., 2023]) These plots show test accuracy as a function of interpolations of ResNet models. Left: REPAIR appears to achieve near-equal accuracy to original models for all interpolated versions. This seems to indicate that LMC is attainable with REPAIR, even when it doesn't appear to exist ("Before REPAIR"). Right: REPAIR improves test-accuracy along the linear-interpolation path than when it's not employed. Similar to (left), REPAIR achieves near constant accuracy throughout interpolation, indicating a significantly better barrier towards LMC. Note this ResNet20 has a width multiplier of just 8×. This is 4× smaller than that needed by [Ainsworth et al., 2023] to report LMC!

REPAIR Address Depth Barrier. The second result I really want to highlight is in Figure 16. This figure shows how the accuracy barrier changes as a function of VGG *depth* before and after REPAIR. For the first time, we observe that model-depth *does not lead* to poor barrier for approaches that use REPAIR. Now, the extrapolation of this is limited as both the VGG network and the CIFAR10 dataset are very simple. However, REPAIR may nonetheless be useful for bypassing the adversarial impacts of increasing depth first found in [Entezari et al., 2022].

LMC with Different Datasets. Finally, Jordan

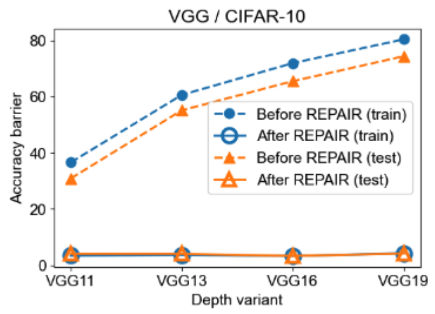


Figure 16: (Copy of Figure 6 (right) from [Jordan et al., 2023]) Accuracy barrier from interpolating between VGG networks of increased depths.

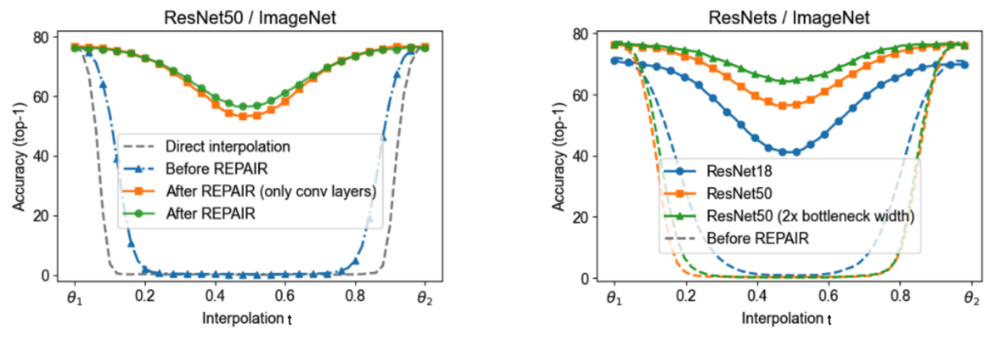


Figure 15: (Copy of Figure 5 from [Jordan et al., 2023]) These plots show test accuracy as a function of interpolations of ResNet models on the ImageNet dataset. Left: ResNet50 interpolated-models achieve significantly better accuracy after REPAIR than before. Note that it is likely that the reason why the performance before REPAIR is so poor is because the batch-norm running stats are not reset. Right: increasing the size and width of ResNets yields better interpolation-accuracies with REPAIR. Similar to the left-plot, a (though unknown) lack of batch-norm running stats reset may be the reason behind such poor interpolations before REPAIR.

et al. [2023] conduct the same experiment as in [Ainsworth et al., 2023] (See "LMC with Different Datasets" in Section 4.3.2), but using REPAIR applied to standard batch-norm networks. While Jordan et al. [2023] do not mention which models they use, I presume they choose the same ResNet20 models used by [Ainsworth et al., 2023] in their experiment. Notably, applying REPAIR to these networks yields a best interpolated loss of 1.30 at $t \approx 0.3$. This is substantially better than the best loss reported by [Ainsworth et al., 2023] of 1.73 at the same interpolation t. While Jordan et al. [2023] also include accuracy results in their appendix, it is very difficult to identify whether REPAIR or [Ainsworth et al., 2023] achieve better interpolation results. This is because the plots of [Jordan et al., 2023] and [Ainsworth et al., 2023] do not share the same y-axis scales, nor do they start from the same value. While it appears that Jordan et al. [2023] improves performance over [Ainsworth et al., 2023], it is ultimately impossible to conclusively ascertain because of their plots' scale discrepancies. Thus, it is difficult to extrapolate indications in favor of Jordan et al. [2023] or [Ainsworth et al., 2023] for this experiment. Nonetheless, we observe that REPAIR can be utilized to good effect even in this regulatory regime—so long as access to some shared data is allowed (which [Ainsworth et al., 2023] explicitly do not use).

4.5 Complexity and Mode Connectivity

Interestingly, we have consistently observed in the past several papers that as we move to more complex models and difficult datasets, achieving mode-connectivity is likely. Indeed, the last two papers covered ([Ainsworth et al., 2023, Jordan et al., 2023]), both can only sometimes achieve LMC in certain settings. Yet, despite not always obtaining mode-connected interpolation paths between models, they show that merging along these paths (e.g., by averaging) while retaining performance *is still possible*. We explore this idea further in the next section.

5 Model Merging without Mode Connectivity

In this section, we present four papers that propose methods for interpolating between trained models without requiring them to be mode-connected. However, they still rely on mode-connectivity principles. Namely, they first align different networks to one another to increase the chance of LMC (like in [Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023]), but then obtain a merged model by interpolating their weights.

5.1 Model Fusion via Optimal Transport

This work, by Singh and Jaggi [2020], proposes an alignment method inspired by optimal transport to merge arbitrary numbers of models together. Unlike the prior papers covered thus far in our report, this work stands out in three important ways. First, their optimal transport alignment measure allows for soft-alignment between the parameters of models. This allows for much more general matching than with permutation matrices. Namely, multiple weights in one model can be aligned with a single (or more) parameters in a second model, instead of only allowing 1-1 matching like in permutations. Second, they are the first method (of the past five works) that can merge models with different widths together. Third, they are the first (of the last five works) to study model merging across models trained on different label sets.

5.1.1 Method

Many of the previous works we have highlighted (e.g.,[Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023]) assume that the neurons in each layer of different networks have one-to-one correspondences. More concretely, let us have two trained models (e.g, Model A and Model B) with different initializations. These prior works assume that one neuron in each layer (e.g., or Model B) is similar to exactly one neuron in the same layer of the other model (e.g., Model A) (i.e., there is a one-to-one correspondence between the neurons). Consequently, these works propose permutation-based approaches to exactly align one neuron in one model to one neuron in the other. However, Singh and Jaggi [2020] observe that neuron correspondences between these same neural networks actually tend to be softer: many neurons in one model (e.g., model B) can be aligned to one (or more) in another (e.g., Model A). For such cases, permutation-based feature matching approaches are less-optimal as they fail to capture the full relationships between weights of different models. Thus, Singh and Jaggi [2020] propose a novel feature matching approach based in Optimal Transport (OT), which generalizes permutation-based approaches by allowing for *soft-correspondences* between the features of two networks.

Allowing for soft-matches between networks can be especially useful when we want to merge networks of different sizes (i.e., widths) together. In these cases, all permutation approaches cannot be used. However, the many-to-one or many-to-many assignments induced by soft-matching enables either thinner networks to be stretched to the size of larger one—or vice versa—to merge them. Yet, OT methods are not without their flaws. Of particular note is that unlike permutations, they do not preserve function-invariance. Thus, methods transformed by OT are not functionally equivalent to their original versions, and can instead yield widely different predictions over data. If not careful, these resulting models may be significantly worse than their original counterparts. Interestingly though not stated in the paper, Singh and Jaggi [2020] seem keenly aware of these problems. In fact, despite proposing an OT soft-alignment algorithm to match the features of two networks, Singh and Jaggi [2020] instead use permutations whenever possible.

Singh and Jaggi [2020] propose two methods to find the best feature matches between models, called activation-based alignment and weight-based alignment respectively. However, they refer to both only as "OT" in their experiments. Interestingly, activation-based alignment takes a very similar form to the activation matching of [Ainsworth et al., 2023] (Section 4.3.1.1), despite being published two years earlier. In fact, the two methods are functionally and notationally equivalent except in two minor ways. First, [Singh and Jaggi, 2020] use layer pre-activation (i.e., before non-linearities) whereas Ainsworth et al. [2023] use post-activation features. Second, Singh and Jaggi [2020] state they use OT to find a transformation matrix with soft-correspondences (i.e., floating-point values), rather than an LAP solver to find permutations. However, this latter difference does not always exist. In fact, [Singh and Jaggi, 2020] utilize a permutation-based approach with 1-1 feature matchings for all experiments involving merging networks with the same width. Otherwise, they use the Python Optimal Transport package to compute alignments when merging models of different widths. Thus, in these settings, Singh and Jaggi [2020] and Ainsworth et al. [2023] share almost identical feature-matching calculators.

The weight-based alignment is nearly identical to the activation-based alignment method. The sole difference is that the weights between the networks are directly used instead of the pre-activations. The rest is exactly the same.

Given two networks Model A and Model B, Singh and Jaggi [2020] compute these transformations for each layer sequentially. That is, they first pick a reference model (e..g, Model A). Second, they use one of the two methods described above to find the "right" transformation of the parameters in the first layer of

Model B to those in the first layer of Model A. For clarity, let this transformation be denoted by T, and the parameters in both models be $W_1^A, \ldots, W_L^A$ and $W_1^B \ldots, W_L^B$ respectively (assume both models have L layers). Third—exactly like in [Ainsworth et al., 2023] (Section 4.3.1.2), they apply T to the rows of W_1^B, and T^T to the columns of W_2^B. Fourth, they repeat step (2) for the second-layer and continue until transformations for all layers have been computed. While a single-pass through all the layers is sufficient to find strong transformation matrices for Model B, they find the best results can be achieved with 2-3 passes.

Merging > 2 Models. Singh and Jaggi [2020] also explicity describe how OT can be utilized to align and merge more than just two models. The procedure is straightforward, and simply requires one model to be chosen as a reference. Afterwards, the remaining models are independently aligned to it using the same aforementioned sequential approach. We note however, that this suffers from the same drawbacks of prior work discussed in detail in Section 4.4.1.1.

5.1.2 Experiments and Discussion

We now highlight two types of experiments from [Singh and Jaggi, 2020]. The first involves merging models that were trained on different input and *not completely identical* output distributions. The second experiment involves standard model merging, but also includes results from merging *many models* together.

5.1.2.1 Merging Models from Different Distributions.

I would like to preface this experiment with the following. While exciting, this experiment is unfortunately very confusingly and ambiguously worded. Thus, even after spending a significant amount of time parsing their text and analyzing their figures, I can only give a very educated guess as for what these experiments show. However, even with this uncertainty, the experiment I am about to highlight seems very interesting from my understanding.

In this experiment, Singh and Jaggi [2020] assume a federated-style setting where we would like to merge two models (e.g., Model A and Model B) trained on different data and having different learned skills. In their case, Singh and Jaggi [2020] focus the problem where one model (e.g., Model A) has basic knowledge of all the skills Model B is an expert in, but Model A is itself an expert in one skill that Model B does not possess. One way Singh and Jaggi [2020] argue this might manifest, is through federated-learning, Here, Model B may a server with a strong skill set for a range of problems (e.g., a general language model), while Model A may be a client application that has been trained to solve a particular set of tasks (e.g., personalized keyword prediction). In this case, we would like to add all the skills of Model A to Model B by merging the models, as we cannot train Model B on the data used to train Model A.

The actual experiment is as follows. Singh and Jaggi [2020] attempt to merge two MLP models (each containing 3 layers with 100, 200 and 400 dimensions respectively) that are trained on two non-i.i.d distributions (call these Data A and Data B respectively) of the MNIST dataset. One dataset, Data A, is chosen to contain 100% of all the training data for one digit (e.g., digit Y), and X% of the training data for each of the remaining 9 digits in the MNIST dataset. In contrast, Data B does not contain any data for the chosen digit (i.e., 4), but does contain the remaining $(100 - X)$% of the training data split across the remaining 9 digits of the MNIST dataset. This way, the authors argue that a Model A trained on Data A will be an "expert" at classifying images of the digit Y, while still retaining some knowledge for classifying the remaining 9 digits. This is in contrast to a Model B trained on Data B, which would (hopefully) be significantly better at classifying the same 9 digits, at the expense of not knowing how mark the missing digit.

Figure 17 illustrates the results from merging models trained across different variations of this setting. All plots show test accuracy on the entire MNIST dataset (i.e., all ten digits) over different interpolations of Model A and Model B. Although not mentioned (found only by deciphering a footnote reference in their main paper), the OT method used is activation-based alignment and only considers Data B when computing alignments. For the left and middle panels, $Y = 4$ and $X = 10, 5$ respectively. Using OT significantly outperforms the naively averaging the parameters of trained models across every interpolation. In particular, note that several interpolations even improve upon the performance of the original base models.

Interestingly however, when Model A is trained with fewer data of the cumulative output space (i.e., middle-panel), its merge is less fruitful when evaluating over the entire test set (containing all ten digits). I believe we might be able to explain this result with the following observation from my own research. Note that, Model A achieves significantly lower performance (middle) than when it has more data (left). From my

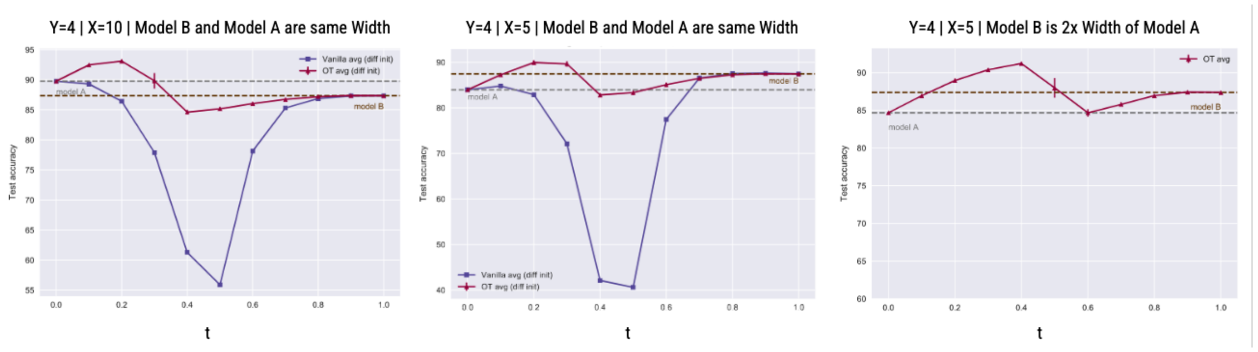


Figure 17: (Mixture of Figure 2 (a), Figure S7 (b), Figure 4 from [Singh and Jaggi, 2020]). I have relabeled them for consistency and clarity.) Test Accuracy from interpolating between two MLPs trained on non-i.i.d. MNIST data. Notation is as in 5.1.2.1. Left: Data A contains all of digit 4 data, and 10% of the remaining data, and Data B contains the rest. Boths models are the same width. OT achieves significantly better interpolations than naive weight averaging. Middle: same setting as (left-panel) but now Data A only contains 5% of the non-digit-4 data, while Data B contains the rest. OT is still significantly better than naive weight averaging. Right: same setting as (middle-panel) but now Model B is twice the width of Model A. OT is able to achieve strong interpolations between the two models.

own research experience, I consistently find that strong-performing merges between models are only really possible when the models themselves are strong predictors for a task. One potential reason for this, is that poor-performing models often do not lie in a minima and thus aligning to/from them induces a lot of noise when merging, which in turn can deteriorate accuracy.

The right-panel of Figure 17 is a very similar setting to the middle-panel, but with Model B now being 2× the width of Model A. In this setting where *none of our previously described works support*, OT can still be used to interpolate well between models.

Overall, these results are very interesting because they show that even when models are trained on different label-sets, they can still be merged with at worst a slight drop in overall-performance. Moreover, we even find that merging models with different widths is feasible without performance drop. However, it is important to remember the simplicity of this setting: it involves merging MLPs on the MNIST dataset—where even a two layer network can achieve near perfect performance. It is not clear how well these results scale as the dataset and problem becomes more complex. For instance, would this work on ImageNet [Deng et al., 2009] when say 30% of the classes are unknown to Model B? Or what if Data A and Data B truly contained different label sets (i.e., Data A contains data from the first 500 ImageNet classes, and Data B only contains data from the remaining 500 classes)? These larger scale experiments are not considered by [Singh and Jaggi, 2020], and would be an important line of future research.

5.1.2.2 Standard Model Merging

The results in this section involve standard model merging: two (or more) models trained on the same data with different initalizations are aligned, their weights are averaged and the resulting model is evaluated. We highlight two types of experiments Singh and Jaggi [2020] conduct in this setting.

Merging Two Models. Table 1 illustrates their results when merging two networks with different initalizations on the CIFAR10 dataset. In the figure, "M_A" and "M_B" are the original trained models, "Prediction Avg." denotes their ensemble with logit-averaging, "Vanilla Avg." corresponds to naively averaging their weights, and "OT" refers to averaging the models after aligning them with either alignment or weight-based matching (it is not specified which). Each cell block contains two types of values. The floating point numbers are accuracies, while the numbers with "×" denote how much more efficient each approach is over maintaining all models. Additionally, Singh and Jaggi [2020] experiment with VGG11 and ResNet18. Overall, we find that OT consistently outperforms naive-weight averaging, and in the case of VGG11 is very close to that of the ensemble, yet both architecture variants do not improve upon the original model performances. Singh and Jaggi [2020] attribute this to the underlying difficulty of the alignment problem and the greediness of their approach. This seems fair, and compounded by the fact that these models may not be LMC to begin with via *any* function-preserving transformation.

Dataset + Model	M_A	M_B	Prediction avg.	Vanilla avg.	OT avg.
CIFAR10 + VGG11	90.31	90.50	91.34	17.02	85.98
	1 ×		1 ×	2 ×	2 ×
CIFAR10 + ResNet18	93.11	93.20	93.89	18.49	77.00
	1 ×		1 ×	2 ×	2 ×

Table 1: (Copy of Table 1 [Singh and Jaggi, 2020]). Test Accuracies on merging multiple model families with different initializations trained on the same data for CIFAR10 dataset. "Prediction Avg" refers to an ensemble with logit averaging, "Vanilla Avg." denotes naive weight averaging, and "OT" is their method. Merging results from two VGG11s are listed first, followed by results with ResNet18s below. Below each accuracy-value is the amount of compute saved by each approach compared to maintaining all the models.

CIFAR10+ VGG11	Individual Models	Prediction avg.	Vanilla avg.	OT avg.
Accuracy	[90.31, 90.50, 90.43, 90.51]	91.77	10.00	73.31
Efficiency	1 ×	1 ×	4 ×	4 ×
Accuracy	[90.31, 90.50, 90.43, 90.51, 90.49, 90.40]	91.85	10.00	72.16
Efficiency	1 ×	1 ×	6 ×	6 ×

Table 2: (Copy of Table S9 from [Singh and Jaggi, 2020]). Test Accuracies on merging multiple VGG models with different initializations trained on the same data for CIFAR10 dataset. "Prediction Avg" refers to an ensemble with logit averaging, "Vanilla Avg." denotes naive weight averaging, and "OT" is their method. The second row-block contains results from merging four VGG11's together, while the second is the result of merging 6. Below each accuracy-value is the amount of compute saved by each approach compared to maintaining all the models.

Merging > 2 Models. Table 2 illustrates results when merging more than two VGG11 networks trained on the same data with different initializations together on CIFAR10. Overall, we observe that even when combining the information of many methods, OT is still able to retain decent performance. Note how the vanilla averaging achieves random accuracy in this setting, illustrating its challenge.

Overall, for the time (recall this work is from a year before [Entezari et al., 2022]), these results were very promising. Merging models with different initializations is an extremely challenging task, even when they share the same training data. Yet despite this, OT is able to retain good performance after averaging weights via their alignment measures. In fact, they find decent performance even when combining the weights of many models together—something even recent work has issues with (e.g., [Ainsworth et al., 2023]). Yet, these experiment settings are indeed limited. The CIFAR10 dataset is relatively simple, and VGG11 networks are small. It would be very interesting to see how OT performs on more challenging datasets such as ImageNet and with much larger models. Promising results in these settings would be very compelling as to the applicability of the proposed approach in the real-world.

5.2 Federated Learning with Matched Averaging

Interestingly, Wang et al. [2020] propose a hybrid approach for model merging in the federated learning setting. Specifically, unlike our prior work which proposes to merge all the parameters of the models we would like to combine at once, Wang et al. [2020] instead propose to merge only a few layers at a time. Once merged, they are frozen and swapped with the original layers used in merging from the original models, Afterwards, each model is re-trained with gradient updating all remaining parameters while keeping the merged-layers frozen. This process is repeated, itereatively merging more and more together.

The Full Federated Learning Setting. A federated-learning paradigm typically consists of two sets of models. The first set contains a group of client models (e.g., edge devices) that are trained on isolated datasets one accessible to a particular model. The second set contains one global model (e.g., a data center) which is used to store the aggregate information of all client models. One way to aggregate information may be by directly averaging the weights of all client models (e.g., [McMahan et al., 2017]). Herein lies the principal challenge in federated learning. Each client model has access to a disjoint dataset to be used to train them, while the global model *does not* have access to *any* of the data. One setting in which this may occur is that of privacy, where each edge device contains particular information that cannot be distributed to others. We would like to aggregate all the knowledge stored in across all client models together into one global model that retains all the information. However, due to our constraints, we can only do this using the client's weights alone. That is, none of the activation matching approaches we have covered thus far (e.g., [Ainsworth et al., 2023, Jordan et al., 2023, Singh and Jaggi, 2020]) can be used; only weight-matching is allowed (e.g., as in Section 4.3.1.2).

To address this problem, federated-learning typically involves a series of training rounds broken up into three stages. In the first stage, the set of client models train on their respective datasets independently for a set number of epochs. In the second stage, the client models are merged altogether to create the global model without using any data. In the third stage, the global model is copied into each client model for the first stage. This is considered one round, of potentially many.

5.2.1 Method

In this work, Wang et al. [2020] propose to improve federated model merging with a hybrid approach by leveraging permutation-invariance (introduced in Section 4.2.1), allowing the global model to have larger widths than the clients. For ease of understanding, we discuss each components sequentially, with each building upon the former.

Notation. Before explaining the core approach, we first introduce some useful notation to simplify the method. Let us assume a federated-learning setting consisting of J client models and one global model—denoted as Global Model. Further, assume that all models have the same architecture (e.g., are VGG9) with L total layers, and let the client models have the same width. Let the weight of layer l of client model j be denoted by $W_l^j \in \mathbb{R}^{d_l \times d_{l-1}}$, where d_l, d_{l-1} are the hidden-state dimensions for layer-l and the previous layer respectively. Similarly, let the same layer in the global model be $W_l^G \in \mathbb{R}^{g_l \times g_{l-1}}$, where $g_l \geq d_l, g_{l-1} \geq d_{l-1}$ are the hidden-state dimensions for the current and preceding layers respectively. Finally, let $P_l^j \in \mathcal{S}$ be an arbitrary permutation on the l^{th} layer of client model j, and $\mathcal{P}$ be the set of valid permutations following the notation in Section 4.2.1.

Leveraging Permutation Invariance. The first component of this method involves finding the best permutations to align each client model to the global model. We formalize this objective by the following:

$$P_l^1, \dots P_l^J = \underset{P_l^1, \dots P_l^J}{\arg\max} \sum_{j=1}^{J} \langle P_l^j, C_l^j \rangle_F, \qquad\qquad W_l^G = \frac{1}{J} \sum_{j=1}^{J} P_l^j W_l^j,$$

$$C_l^j = \begin{bmatrix} c([W_l^j [P_{l-1}^j]^T]_1, [W_l^G]_1) & \dots & c([W_l^j [P_{l-1}^j]^T]_1, [W_l^G]_{g_l}) \\ \vdots & \ddots & \\ c([W_l^j [P_{l-1}^j]^T]_{d_l}, [W_l^G]_1) & & c([W_l^j [P_{l-1}^j]^T]_{d_l}, [W_l^G]_{g_l}) \end{bmatrix} \tag{14}$$

where $C_l^j \in \mathbb{R}^{d_l \times g_l}$ is a "cost-matrix" containing the pair-wise costs between rows in $W_l^j [P_{l-1}^j]^T$ and W_l^G respectively that are computed by some cost function (e.g., euclidean distance) $c(\cdot, \cdot)$. $[P_{l-1}]^T$ is the already computed permutation matrix for the previous layer. For now, assume that $g_l = d_l$ and $g_{l-1} = d_{l-1}$. Immediately, we notice three things. First, (for hopefully easier reading) Equation 14 is very similar to Equation 10, but with merging many models at once. Second, we observe that the parameters of the global model (e.g., W_l^G) are simply the average of the permuted-parameters across all client models. Third—and perhaps most confusingly—Equation 14 attempts to find the optimal permutations to match each client model weight to the respective global model weight, but the global model weights are themselves composed of the best permutations from each client. This is akin to the chicken and egg problem. Wang et al. [2020] address this problem via a very similar process used by Ainsworth et al. [2023].Specifically, they optimize Equation 14 in two stages. First, they fix all permutations except for that of one client (e.g., P_l^j). This turns Equation 14 into a standard LAP that they solve using the Hungarian algorithm. They then iteratively solve for all other client permutations, keeping all permutations but one fixed each time. Second, once all client permutations are found, they recompute W_l^G as in Equation 14.

Increasing Global Model Width. Of particular note, Wang et al. [2020] observe that client models are trained on heterogeneous amounts of data in federated learning settings. As a result, it is possible that certain client models may not share any features with a global model being constructed by the other clients. In this case, forcing a feature from the client to match to another in the global model can destroy the information from each and degrade performance. Thus, to avoid such "poor" matches, Wang et al. [2020] introduce a "slack" term to Equation 14 to allow for "partial-permutations." Based on strength (or lack thereof) of feature similarities between a client and the global model (e.g., thresholding the cost by some ϵ), these partial-permutations allow the client to *add* neurons to the global model, thereby extending its width. However, in an effort to keep the global model comparably sized to each client, they add a penalty-cost (e.g., via some function $f(i)$) for each neuron extension. Mathematically, we can alter Equation 14 as follows,

$$P_l^1,\dots P_l^J = \underset{P_l^1,\dots P_l^J}{\arg\max} \sum_{j=1}^{J} \langle P_l^j, C_l^j \rangle_F, \qquad\qquad W_l^G = \frac{1}{J}\sum_{j=1}^{J} P_l^j W_l^j,$$

$$C_l^j = \left[\begin{array}{ccc|ccc} c([W_l^j[P_{l-1}^j]^T]_1, [W_l^G]_1) & \dots & c([W_l^j[P_{l-1}^j]^T]_1, [W_l^G]_{g_l}) & \epsilon + f(g_l+1) & \dots & \epsilon + f(g_l+d_l) \\ \vdots & \ddots & & \vdots & \ddots & \\ c([W_l^j[P_{l-1}^j]^T]_{d_l}, [W_l^G]_1) & & c([W_l^j[P_{l-1}^j]^T]_{d_l}, [W_l^G]_{g_l}) & \epsilon + f(g_l+1) & \dots & \epsilon + f(g_l+d_l) \end{array}\right]. \quad (15)$$

Wang et al. [2020] follow the specifications of [Yurochkin et al., 2019] for defining $c(\cdot,\cdot)$, ϵ and $f(i)$. Note that Equation 15 is exactly the same as Equation 14, except that $C_l^j \in \mathbb{R}^{d_l \times (g_l+d_l)}$. In addition, each P_l^j in Equation 15 is no longer a permutation matrix in the classical sense. Instead, it is is of size $\mathbb{R}^{d_l \times (g_l+d_l)}$. However, its functionality is similar: taking $P_l^j W_l^j$ implies permuting the weights of W_l^j to align with the weights of the global model learned from the other clients and padding with "dummy" neurons having zero-weights. We can solve Equation 15 using the same process described for Equation 14, with one small change. Specifically, we compute W_l^G as before, but this truncate its dimensions by removing all zero-rows. This way, we ensure $d_l \leq g_l \leq J \cdot d_l$, or that g_l is at most the concatenation of all client model weights at layer-l.

Putting the two components together. Wang et al. [2020] find the optimal permutations by iteratively optimizing Equation 15 over each layer via a three stage process. First, all client permutations for the first layer (e.g., P_1^j) are computed. Note that P_0^j is the Identity as it is on the input dimension. Then, the first and second layer weights of each client are transformed using these respective permutations. Second, W_l^G is calculated as in Equation 15. Third, the client permutations of the next layer are calculated using the transformed second-layer weights from the first stage. This process is repeated for all layers, until the last layer where W_L^G is set to be weighted average of each W_L^J according to the fraction of training data each client utilized. Finally, the fully merged Global Model is achieved, and ready to be evaluated.

(1) Partial Merge, (2) Freeze, (3) Train, (4) Repeat. Unfortunately, Wang et al. [2020] do not observe strong results when using the above algorithm. In particular, they observe that the approach starts to perform increasingly poorly on downstream tasks as the model complexity and depth increases. Thus, Wang et al. [2020] propose a modification to the approach.

Namely, rather than merging all client models into the global model across all layers at once, they instead propose to merge "few" layers at a time. Specifically, they first gather only the first layer from the client models and apply their two-component algorithm on just this layer to obtain the merged first layer in the global model. Then, they copy the computed global model weights into the corresponding (i.e., first) layers of each client model. Simultaneously in this step, they compress the hidden-units of the global model (i.e., g_1) to the same number of hidden-dimensions of the clients (i.e., d_1). They do not specify how they do this in the paper. However, I would guess that this is done by applying the same P_1^j computed to match a client model-j to the global model, on the global model instead: $[P_1^j]^T W_1^G$. Second, they freeze these newly copied weights for each client, and re-train the client models on the same local dataset, only updating the remaining unfrozen weights. I believe this is done to try and correct any information lost from merging the client models together. Altogether, they term this approach "FedMA" (with communication).

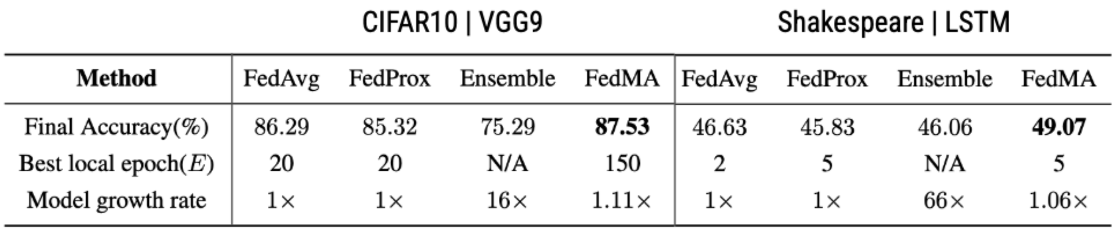
Method Discussion. I find this hybrid approach to be a very interesting and fresh take amongst the model-merging literature that we have not seen yet in this report. Although we haven't discussed this in detail with the prior permutation-based approaches we have covered [Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023]—even [Singh and Jaggi, 2020] to an extent—permutation-invariance is not preserved under merging. Concretely, it is true that we can permute the weights of one model arbitrarily to obtain a new model that is functionally equivalent to the original [Entezari et al., 2022]. However, model-merging is a fundamentally lossy operation as the elementwise interpolation between the weights of two models does not preserve the functional-invariance of either.

We can illustrate this by considering a very simple one layer MLP with a ReLU activation: $f(x) = \text{ReLU}(Wx)$. Define two models as $f^A(x) = \text{ReLU}(W^A x)$ and $f^B(x) = \text{ReLU}(W^B x)$ respectively. Finally, choose some arbitrary permutation P to match W^B to W^A. Then we can interpolate between the two and obtain the merged model: $f(x) = \text{ReLU}([tW^A + (1-t)W^B]x) = \text{ReLU}(tW^A x + (1-t)W^B x) \overset{?}{=} t\text{ReLU}(W^A x) + (1-t)\text{ReLU}(W^B x) = tf^A(x) + (1-t)f^B(x)$. Indeed, the above "$\overset{?}{=}$" is only guaranteed to be "=" for any x when the interpolation amount $t = \{0, 1\}$ (i.e., no interpolation is performed). Although this is a simple example of just one layer, it is easy to see how *every* layer in a deep network surfers from this issue and may lead to cascading errors as networks become deeper. In fact, this may be one of the reasons it is so rare for a merged model to achieve ensemble accuracy (what the last expression above mimicks).

The hybrid approach proposed by Wang et al. [2020] is very interesting to me because it partially bypasses this issue. Since after merging each layer, the clients can then retrain over these fixed merged-layers in an effort to adapt their respective models to any information lost in the merge. However, the approach is not without serious limitations. Most notably, it induces a huge compute overhead. Unlike any of the previous works we have covered in this report, FedMA requires an entire training loop after *every* merge. Although the method does not update the weights of any merged layers (they are frozen), it nonetheless significantly increases the cost and time of overall merging. This may not be a huge issue in the federated-learning setting, where model-retraining is an integral part of a "round." However, it becomes infeasible in scenarios where we would like to merge a set of very-large models (e.g., CLIP variants [Ramé et al., 2022]), each containing a particular set of skills into one with all of them (e.g., something similar to that of [Singh and Jaggi, 2020]—Section 5.1.2.1). In such cases, it may be extremely expensive to retrain each model after every merge operation. Thus, I believe it is appropriate to consider FedMA as a federated-learning only approach.

5.2.2 Experiments and Discussion

We highlight two experiments from [Wang et al., 2020]. The first showcases the capabilities of FedMA for a standard federated learning benchmark. The second, demonstrates its ability to correct for *biases* induced by merging different client models together.



	CIFAR10 | VGG9				Shakespeare | LSTM			
Method	FedAvg	FedProx	Ensemble	FedMA	FedAvg	FedProx	Ensemble	FedMA
Final Accuracy(%)	86.29	85.32	75.29	**87.53**	46.63	45.83	46.06	**49.07**
Best local epoch(E)	20	20	N/A	150	2	5	N/A	5
Model growth rate	1×	1×	16×	1.11×	1×	1×	66×	1.06×

Table 3: (Copy of Tables 1 & 2 from [Wang et al., 2020]). Test Accuracy on CIFAR10 and Shakespeare datasets with VGG9 and LSTM respectively. "Best local epoch" denotes the number of epochs each client is trained for. "Model growth rate" is the size of global model after all training is completed. "FedAvg" and "FedProx" refer to two prior works. FedAvg directly averages the weights of each client model in each round, while FedProx [Li et al., 2020] adds a proximal term to client loss functions to limit gradient updates from getting too far from global model weights. FedMA is the [Wang et al., 2020] method.

A Standard Federated Learning Benchmark. This experiment consists of two federated-learning tasks on CIFAR10 and the Shakespeare dataset. For CIFAR10, Wang et al. [2020] use 16 client models and heterogeneously split the training data across each client. They do this by sampling from a distribution over all classes and clients, where each sample-value is the fraction of data from a particular class that a particular client will have access to. For the Shakespeare dataset, they employ 66 clients consisting of 1-layer LSTM networks and randomly distributed the training data among all clients. Table 3 shows the results of these experiments. Overall, we observe that FedMA significantly outperforms prior work, including the ensemble while requiring a fraction of the compute. Moreover, FedMA is able to achieve its performance while only minorly increasing the global model width (e.g., 1.11× for CIFAR10). However, it is important to note that FedMA requires 9× the training as all the comparison methods since it has to re-train the client network after every layer merge (there are 9 layers in VGG9). Coupled with requiring 5× the total client training epochs per round as the other works (e.g., 150 vs. 20), FedMA is a costly method.

Handling Data Bias. In this experiment Wang et al. [2020] seek to simulate a setting where the data clients have access to is biased, as may occur in the real-world. This can be problematic because although each client model be affected by its biased data (as its "test set" is also biased), the global model will inevitably contain all model biases in aggregate. If care is not take to mitigate these biases, the global model may in turn generalize very poorly in an unbiased setting. Wang et al. [2020] argue that FedMA is uniquely positioned to address this issue head-on. Although not explained in the paper, I believe we can expect this behavior because any aggregate biased induced in merging a layer can be ameliorated by the successive client training on the layers it feeds into done as part of the FedMA algorithm. In this way, the clients can individually assuage the aggregate biases of merged layers throughout the merging process.

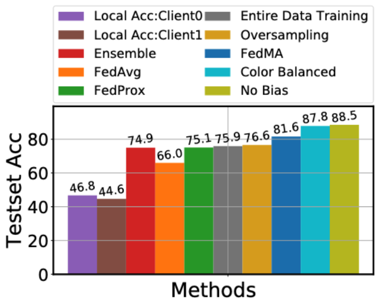


Figure 18: (Copy of Figure 4 from [Wang et al., 2020]). Test Accuracy on a skewed version of CIFAR10 using VGG9 models.

Wang et al. [2020] study this problem by artificially skewing the CIFAR10 dataset. Specifically. they randomly select 5 classes and make 95% of the training images in those classes be grayscale. They then set only 5% of the images across the remaining 5 classes grayscale. They also alter the test-set to contain 50% grayscale images for each class (unbiased grayscaling for each class). They then split the training data from the grayscale dominant and color dominant classes into two clients (called Client0 and client1) for their federated experiments. They compare FedMA to FedAvg and FedProx using these clients. Additionally, they train four more models on all the data in different settings, to gain a holistic understanding over the benefits of FedMA. First, they train a model (called Entire Data Training) using all the altered training data as you would in standard classification. Second, they train a model (called Oversampling) using all the altered training data, but over-sample the under represented image kinds from each class such there are equal proportions of grayscale and color images. Third, they train a model (called Color Balanced) using a color-balanced version of the altered training data where each class has an equal proportion of grayscale and color images. Fourth, they train and test a model (called No Bias) using all the original data as in standard classification. Figure 18 illustrates the results.

Overall, FedMA outperforms both other federated methods, and most of the other comparison benchmarks. In fact, it only is beaten by the Color Balanced and No Bias models, which is exected as each forms the upper bound. Color Balanced is the uppoerbound for this simulated biased task, while No Bias is the upperbound for an unbiased model trained (and evaluated) without any biased data. As described above, I believe this is expected behavior, and speaks the unique benefits FedMA has from being able to address any issues caused by merging each layer. However, the method remains significantly more expensive than any of its compared models. Moreover, similar to our analysis for prior works, this is a very simplified setting for bias: it is conducted on the small CIFAR10 dataset with small VGG9 networks. It would be very interesting to observe how FedMA performs on more challenging tasks such as ImageNet.

5.3 Model Soups: Averaging Weights of Multiple Fine-Tuned Models Improves Accuracy Without Increasing Inference Time

This work, by Wortsman et al. [2022a], studies model merging when the underlying models share the *same (pretrained) initialization*. This setting deviates from that of the previous works we have covered thus far in this report, as they all study merging models with different initializations. Nonetheless, the same initialization setting is still an important one to consider and understand.

As discussed in Section 2, merging models with the same (pretrained) initialization is typically a significantly easier task than when they have different initializations. This is because fine-tuned models (even when independent) from the same pretrained checkpoint tend to lie in the same loss basin after training (i.e., they are mode connected) [Neyshabur et al., 2020]. As a result, directly interpolating between their parameters is feasible, as even the linear-path between them is likely to induce low (or even zero) barrier. Wortsman et al. [2022a] illustrate this phenomenon with a simple experiment. They independently finetune three CLIP [Radford et al., 2021] ViT-B/32 models [Dosovitskiy et al., 2020] on ImageNet [Deng et al., 2009], and plot them over loss and error surfaces. These surfaces include the ImageNet train loss, ImageNet test error, and the average test error over five out-of-distribution datasets: ImageNet-V2 [Recht et al., 2019], ImageNet-R [Hendrycks et al., 2021a], ImageNet-Sketch [Wang et al., 2019], ImageNet-A [Hendrycks et al., 2021b], and Objectnet [Barbu et al., 2019].

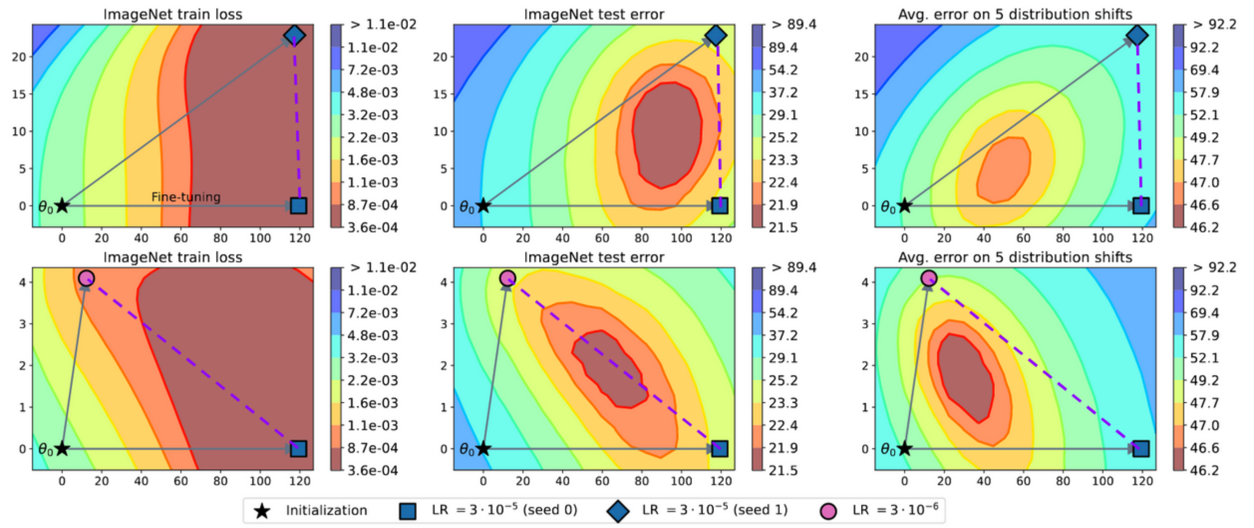


Figure 19: (Copy of Figure 1 from [Wortsman et al., 2022a]. Dashed lines are added by me for explanation purposes.) Loss and error landscapes plotted using different finetuned CLIP [Radford et al., 2021] models on ImageNet. All models have the same pretrain initialization (star) and are shown by either a square, diamond or circle. Left-column: ImageNet train loss surface. Middle-column: ImageNet test error surface. Right-column: Average Error surface the aforementioned OOD datasets.

Figure 19 illustrates the results. First, notice how all models lie in the same bowl across all surfaces, despite being trained with different parameters. Moreover, we observe that the finetuned models in the top-left panel in the figure share the same minima, with those in the bottom-left plot are in a similar situation. Specifically, these finetuned models all appear *mode connected*. In fact, even the pretrained model appears mode connected with the finetuned versions! Second, we consistently observe that interpolating the finetuned models via the direct linear path (dashed purple line) almost always leads to a model that performs *at least as well* as the endpoint models. This suggests that simple weight-averaging can be sufficient to obtain strong performing models from endpoint ones. As a testament to this, we observe in the bottom-middle-panel that averaging the weights of the finetuned models would yield a model *in the minima*. Third, consider that the dashed lines form a triangle with the solid lines. Looking at the angle from the initialization model "vertex" (star), there appears to be a correlation between the angle obtuseness and better merges. Interestingly, Wortsman et al. [2022a] conduct an empirical study comparing this "angle" to downstream merging performances on ImageNet, and observe a strong direct-correlation between obtuseness and performance from merge. I find this to be very interesting. It suggests that the angle can be used as a proxy for estimating how strong (or weak) averaging the parameters of two models will be *without having to evaluate the merged model*.

Inspired by these observations, Wortsman et al. [2022a] propose a simple approach to merge models that stem from the same pretrained initialization using variations of weight-averaging. We will discuss each method in detail in Section 5.3.1, but we emphasize the simplicity of this kind of approach: neither data nor function-preserving transformations are required to achieve a successful merge. Most notably, Wortsman et al. [2022a] demonstrate the efficacy of their approach by achieving state-of-the-art (SoTA) on the ImageNet dataset, *solely* by averaging the weights of existing models.

5.3.1 Method

Wortsman et al. [2022a] propose several very simple methods (all labeled under the umbrella of "model soups") to merge finetuned models from the same pretrained checkpoint. We describe each in their own paragraphs, and finish with a discussion including ideas of simple extensions. For all methods described below, assume we are given a set of M base models that are finetuned with different hyperparameters on a task from the same pretrained checkpoint. Let each be parameterized by $\theta_1, \ldots \theta_M$ respectively and have functional form $f(x, \theta_i)$ respectively. Our objective is to combine these models into one model (termed a "soup") with parameters θ_C. Unless otherwise specified, we assume the soup is simply the average of all model parameters: $\theta_C = \frac{1}{M}\sum_{i=1}^{M} \theta_i$.

Uniform Soups. This is the simplest approach, and is just the aforementioned soup: $\theta_C = \frac{1}{M}\sum_{i=1}^{M} \theta_i$. While very straightforward, this approach may produce a merged model with low accuracy because some finetuned models may not be compatible with *all other* ones. One way this may arise is if some of the M models end up in *different minima* after finetuning. In this case, they will not be mode connected and weight averaging will yield poor results.

Greedy Soups. This is a slightly more complicated approach, but circumvents the issue in uniform soups. In greedy soups, each base model is first evaluated on a held-out validation set (containing no overlap with the training data), and sorted in order of descending accuracy. Then, the method begins sequentially adding each model to the soup (starting from the first model), and compares the validation accuracy of the new soup to that of the old one. If the newly added model improves accuracy, the model is kept in the soup. Otherwise, the model is removed and the old soup is recomputed. The method then moves to the next model in the soup, and repeats until all models have been tried. Note that in the first step, the soup is just the first model in the list so as to always be at least as good as the best performing model. Table 4 illustrates the Greedy soup algorithm.

```
Recipe 1 GreedySoup
  Input: Potential soup ingredients $\{\theta_1, \ldots, \theta_k\}$ (sorted in
  decreasing order of ValAcc($\theta_i$)).
  ingredients $\leftarrow \{\}$
  for $i = 1$ to $k$ do
    if ValAcc(average(ingredients $\cup \{\theta_i\}$)) $\geq$
            ValAcc(average(ingredients)) then
      ingredients $\leftarrow$ ingredients $\cup \{\theta_i\}$
  return average(ingredients)
```

Table 4: (Copy of Algorithm 1 from [Wortsman et al., 2022a])

Greedy Soups, Random List. This is a slight variant of the greedy soup and involves iterating over a randomized list instead of the one based on the heldout validation accuracy. This is mainly used as an ablation method to better understand the utility of ordering by accuracy.

Learned Soup. The learned soup is the most complex and costly method proposed by [Wortsman et al., 2022a]. In the learned soup, mixing coefficients (i.e., the t_i's from our standard report notation) are learned for each base model using the heldout validation set. Wortsman et al. [2022a] propose to do this by maximizing the following objective,

$$\underset{\sum_{i=1}^{M} t_i = 1; \forall t_i \leq 0; \beta \in \mathbb{R}}{\arg\min} \sum_{n=1}^{N} l(\beta \cdot f(x_n, \sum_{i=1}^{M} t_i \theta_i), y_n) \tag{16}$$

where β is a temperature scaling parameter and l is a loss function. Wortsman et al. [2022a] learn these t_i's keeping all model parameters frozen, over three epochs on the held-out validation dataset (consisting of $\{x_n, y_n\}_{i=1}^{N}$ examples) using the AdamW optimizer and constant learning rate of 0.1. In addition, they also explore a "by layer" variant of this learned soup. For this variation, they learn a separate set of mixture coefficients *for each layer* of the network.

	ImageNet	Dist. shifts
Best individual model	80.38	47.83
Second best model	79.89	43.87
Uniform soup	79.97	51.45
Greedy soup	81.03	50.75
Greedy soup (random order)	80.79 (0.05)	51.30 (0.16)
Learned soup	80.89	51.07
Learned soup (by layer)	81.37	50.87
Ensemble	81.19	50.77
Greedy ensemble	81.90	49.44

Table 5: (Copy of Table 3 from [Wortsman et al., 2022a]) Comparison of different soup methods applied on CLIP ViT-B/32 models that are finetuned on ImageNet. For "Greedy Soup (random order)" three random model orders are tried. "Greedy ensemble" refers to the greedy soup but as an ensemble where predictions are averaged instead.

Discussion. Table 5 compares these methods with one another in the same evaluation settings as Figure 19. Overall, we observe that the uniform soup actually yields worse performance than sticking with the best finetuned model, highlighting the aforementioned pitfall it has. However, it appears this is a case-by-case basis, as it also yields the best overall performance when evaluated on the distribution shifts setting. In fact, it appears that greedy soup performs worse than even its variant using a random model list ordering on distribution shifts.

The authors do not explain this phenomenon. However, one reason this may be happening is due to the bias the greedy soup induces and overall-noise. First, greedy soup selectively averages models based on ImageNet heldout validation accuracy. Note, here the heldout set is a 2% partition of the ImageNet training set (with the remaining 98% forming the set used to finetune each base model) Thus, greedy soup is naturally biased to picking the best ImageNet accuracy achievers, thus achieving far better performance than the other two methods on ImageNet. However, these same models may not be the best at generalizing well to the distribution shifts setting. Second, none of the base models were finetuned on the distribution shifts dataset. Thus, there will inherently be some noise and variance when averaging model weights.

Looking at the other results, we observe that the learned soup performs about as well as the greedy soup, with the layer-wise learned soup performing the best on ImageNet. This is expected, as it is the most complex soup method and involves training to pick the best interpolation weights. Overall however, greedy soup performs quite well and is the method of choice for Wortsman et al. [2022a].

Extension Ideas. I would like to end this discussion by proposing a slight variation of greedy soups that is always guaranteed to perform at least as well as greedy-soups, though with higher worst-case compute overhead. Specifically, rather than simply returning the soup after iterating through the model list once, a better version could be to restart the loop but this time only consider adding previously rejected models. These loops can then be repeated a specified number of times, or until the soup no longer changes (i.e., it converges). If run until convergence, the worse case cost is noticeably higher than greedy soups (it is $\frac{M-1}{2}\times$ as costly). However, even running it for one or two extra loops (adding just $M-1$ or $2M-3$ total iterations in the worst case), could be fruitful. This is because all these soup methods (aside from the learned versions) are attempting to "hit" on the minima of whichever loss-basin their base models occupy (e.g., the top-middle panel in Figure 19). While uniform soup does not have any control over where the final soup lands within the error-surface, greedy soup somewhat does as it selectively chooses which models to average—thereby moving the cumulative soup around the loss-surface. Thus, greedy soup stands to benefit from observing more potential paths over which to move its current soup as it increases the overall likelihood that the soup lands in the minima. I think this method would be interesting to try as an experimental extension of [Wortsman et al., 2022a]. Moreover, we could also consider looping on the entire list again each time. This is similar a version Wortsman et al. [2022a] propose (but do not explore) which involves sampling without replacement. The difference in my case is that no sampling would be done. Regardless, this version would enable the soup to put more weight on some models (i.e., by adding them to the soup multiple times) over others. This would add yet more (potentially beneficial) directions the soup can move along the loss-surface.

5.4 Experiments and Discussion

We now highlight and discuss two more experiments from [Wortsman et al., 2022b]. The first consists of merging a series of models finetuned on a vision problem, and the second on a language problem. We describe each in paragraphs below.

Method	ImageNet acc. (top-1, %)	Distribution shifts
ViT-G (Zhai et al., 2021)	90.45	–
CoAtNet-7 (Dai et al., 2021)	90.88	–
Our models/evaluations based on ViT-G:		
ViT-G (reevaluated)	90.47	82.06
Best model in hyperparam search	90.78	84.68
Greedy soup	**90.94**	**85.02**

Table 6: (Copy of Table 1 from [Wortsman et al., 2022a]) ImageNet accuracy and average distribution shifts accuracy for different models and approaches. CoAtNet-7 [Dai et al., 2021] was the current ImageNet SoTA at the time [Wortsman et al., 2022a] was written.

Vision. The vision experiment involves Wortsman et al. [2022a]'s most notable result: SoTA on ImageNet. Specifically, in this experiment Wortsman et al. [2022a] finetune 58 ViT-G/14 models on ImageNet using the same JFT-3B pretrained checkpoint, via different hyperparameter and learning setups. Table 6 shows the results. Wortsman et al. [2022a] find that the final greedy soup contains 14/58 models, and performs *statistically significantly* better than the best individually finetuned model on ImageNet. Moreover, the greedy soup achieves a new SoTA (at the time of writing) on ImageNet.

Language. The language experiment consists of evaluating model soups on the GLUE [Wang et al., 2018] benchmark. More specifically, Wortsman et al. [2022a] finetune models (from the same initialization) on each of four GLUE datasets: MRPC [Dolan and Brockett, 2005], RTE [Dagan et al., 2006], CoLA [Warstadt et al., 2018] and SST-2 [Socher et al., 2013]. The models employed are BERT [Devlin et al., 2019] and T5 [Raffel et al., 2020], and 32 models of each type are train on each of the aforementioned datasets. Table 7 illustrates the results. All models are measured on the standard classification metrics for each dataset (higher is better). Overall, we find that the greedy soup almost always improves upon the best individual model.

Model	Method	MRPC	RTE	CoLA	SST-2
BERT (Devlin et al., 2019b)	Best individual model	88.3	61.0	59.1	92.5
	Greedy soup	88.3 (+0.0)	61.7 (+0.7)	59.1 (+0.0)	93.0 (+0.5)
T5 (Raffel et al., 2020b)	Best individual model	91.8	78.3	58.8	94.6
	Greedy soup	92.4 (+0.6)	79.1 (+0.8)	60.2 (+0.4)	94.7 (+0.1)

Table 7: (Copy of Table 5 from [Wortsman et al., 2022a]) Performance of model soups on four text classification datasets from the GLUE benchmark.

Discussion. These two experiments together demonstrate the applicability and effectiveness of the greedy soups across diverse modalities. However, they further highlight the difference induced by merging models with the *same* initialization. Recall that all the approaches we have discussed involving merging differently initialized models do not achieve anywhere near the performance Wortsman et al. [2022a] demonstrate is possible when models have the same initialization. And this is with just weight averaging! Indeed, initialization plays a critical factor in merging model merging difficulty.

Additionally, one experiment I wish Wortsman et al. [2022a] had included is a study merging several models finetuned on *different tasks* (e.g., datasets) with different kinds of outputs, and then *evaluated* on the same tasks. I think such an experiment would be very interesting, as it would explore the extent to which weight averaging approaches could be used to directly combine specialized models into one model without losing the specializations. Strong results would be particularly compelling, as it would suggests we could quickly obtain "multitask" models from existing ones on-the-fly *simply by averaging their weights*. Importantly, no training or access to any data would be required, making this potentially well suited in a federated/regulatory setting. Note that Wortsman et al. [2022a] do include an experiment consisting of merging several finetuned models on different datasets and then evaluating the merged model in a zero-shot manner only on another dataset (Appendix E in [Wortsman et al., 2022a]). However, the implications of this experiment are clouded by the evaluation dataset (CIFAR100) sharing the *same* image distribution as a dataset one of the merged finetuned models was finetuned on (CIFAR10).

5.5 Merging Models with Fisher-Weighted Averaging

The authors, Matena and Raffel [2022], of this work extend upon [Wortsman et al., 2022a] by viewing model-merging as maximizing the joint-likelihood of the posteriors of model parameters. Interpreting model-merging in this light, they show that direct averaging of parameters across different models is akin to an isotropic Gaussian approximation of their parameter posteriors. Yet they find that isotropism forms a naive approximation, and Matena and Raffel [2022] show how Fisher information can instead by utilized to achieve better calibrated posterior distributions and thus better merges. The authors compare model merging with Fisher information and model-soups (a la [Wortsman et al., 2022a]) on several experiment. Note, like [Wortsman et al., 2022a], Matena and Raffel [2022] assume all models have the *same initialization*.

5.6 Method

Matena and Raffel [2022] focus on a version of model-merging similar to [Wortsman et al., 2022a] where parameters of models with the same architecture and initialization are directly interpolated without transformation (e.g., as opposed to [Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023, Singh and Jaggi, 2020, Wang et al., 2020], etc...). Interestingly, Matena and Raffel [2022] view this model merging as an approximation of maximizing the joint likelihood of model posteriors over their parameters.

Model Soups as Isotropic Merging. To illustrate this joint likelihood perspective, consider the setting of [Wortsman et al., 2022a], where models are directly interpolated (e.g., by uniform averaging) to produce a merged model. Mathematically, we can write this as being given a set of M models with parameters $\theta_1, \dots, \theta_M$, and obtaining a merged model θ_C via $\theta_C = \frac{1}{M}\sum_{i=1}^{M} t_i\theta_i$, where $\sum_{i=1}^{M} t_i = 1; t_i \geq 0, \forall i \in [1, M]$ are model interpolation weights. We now consider the above from the maximization of joint-likelihood and show how the two are equal. Let the posterior distribution over an arbitrary model's (e.g,, model-i) parameters be approximated via an isotropic Gaussian with mean set to the model's learned parameters (i.e., θ_i), and assume that each posterior has an associated weight (e.g, the same $t_i \geq 0$ from before). If we formulate the above posterior distribution for every model's parameters, we can then obtain the best merged parameters θ_C by solving the standard optimization problem: $\theta_C = \arg\max_{\theta_C} \sum_{i=1}^{M} t_i \log p(\theta_C|\theta_i, I)$ where $p(\theta_C|\theta_i, I)$ is the probability distribution of the aforementioned approximate isotropic Gaussian posterior distributions for each model-i. Lastly, I is the identity matrix. Solving the optimization problem can be done in closed-form to yield $\theta_C = \frac{1}{M}\sum_{i=1}^{M} t_i\theta_i$, exactly as in the weighted model-soups version [Wortsman et al., 2022a]. Thus, weighted model averaging can be considered as isotropic Gaussian merging, or just isotropic merging for short.

The Fisher Information Matrix. Matena and Raffel [2022] posit that approximating the joint likelihood via isotropic Gaussian posteriors may be overly simplistic and lead to sub-optimally performing model merges. To address this, they instead approximate the posteriors via the Laplace approximation [Daxberger et al., 2021]. This effectively leads to an approximate Gaussian posterior of the form $p(\theta_C|\theta_i, F_i)$, where the Fisher information matrix F_i replaces the Identity-matrix found in the istropic Gaussians. The Fisher information matrix [Fisher, 1922], F_i for each model, can be computed by taking the average over the outer-products between the full-model gradient and itself over a set of data:

$$F_i = \mathbb{E}_x \left[\mathbb{E}_{y \sim p_\theta(y|x)} [\nabla_{\theta_i} \log p_{\theta_i}(y|x) \cdot (\nabla_{\theta_i} \log p_{\theta_i}(y|x))^T] \right] \tag{17}$$

Unfortunately, computing Equation 17 yields a Fisher information matrix $\in \mathbb{R}^{|\theta_i| \times |\theta_i|}$, which requires an infeasible amount of memory to store. Thus, [Matena and Raffel, 2022] instead only estimate the diagonal of the Fisher matrix via,

$$\hat{F}_i = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{y\sim p_{\theta_i}(y|x_i)}(\nabla_{\theta_i} \log p_{\theta_i}(y|x_i))^2, \tag{18}$$

where $x_1, \ldots, x_n$ are drawn i.i.d from the dataset used to train the model, and $F_i \in \mathbb{R}^{|\theta_i|}$. The expectation of y can be estimated from sampling gradients over $p_{\theta_i}(y|x_i)$ and computing F_i requires N gradients over data.

Fisher Merging. Matena and Raffel [2022] can much more efficiently compute the Fisher information using Equation 18, provided a set of labeled training data. Using these diagonal matrices Matena and Raffel [2022] reformulate the likelihood maximization objective as, $\theta_C = \arg\max_{\theta_C} \sum_{i=1}^{M} t_i \log p(\theta_C|\theta_i, \hat{F}_i)$. Now, similar to the isotropic merging approach, this has the close-form solution:

$$\theta_C = \frac{\sum_{i=1}^{M} t_i \hat{F}_i \odot \theta_i}{\sum_{i=1}^{M} t_i \hat{F}_i} \tag{19}$$

where $\odot$ is the elementwise product. Matena and Raffel [2022] refer to this merging process as Fisher merging.

Unmergeable Parameters. In several cases, models may have task-specific components that are not shared from task-to-task. For instance, this includes classification heads of particular shapes. Matena and Raffel [2022] handle this by only applying Fisher/Isotropic merging on the parameters of models where their architectures are shared, and keep the task-specific heads unchanged.

Fisher Merging Discussion. I find two concerning limitations regarding the Fisher merging algorithm. First, and potentially most obvious, is that computing accurate Fisher representations requires a significant amount of training data. In fact, Matena and Raffel [2022] find that *all the training data* is required to obtain the best Fisher representations, while training on a subset of data yields worse performing merged models across downstream tasks. This issue is compounded by the requirement of having *labeled* data from each task to compute the Fisher information. In the case of federated settings where task-data (especially labeled) is of questionable use, their method cannot be used. The second issue I would like to highlight is that of memory consumption. Matena and Raffel [2022] compute a Fisher information matrix that is of *equal size* to the parameters of each original model. Thus, merging M models via the Fisher merging method actually requires at least $2M$ the memory-compute overhead, compared to an approach such as model-soups Wortsman et al. [2022a] which only requires M. So Fisher merging is $2\times$ as expensive as isotropic averaging in terms of compute required for the merge operation. While this may be okay for the smaller models used in their experiments such as BERT [Devlin et al., 2019], it can quickly become expensive as the model size scales and number of models we want to merge increases.

5.6.1 Experiments and Discussion

We highlight and discuss experiments conducted by Matena and Raffel [2022] comparing Fisher merging with isotropic merging and relevant baselines respective to each task. We use bold paragraph headers to group each experiment.

Same-Task Model Merging. The first experiment we highlight involves merging a set of finetuned models on the same task that all share the same initialization. Matena and Raffel [2022] show the results of merging different BERT-bsaed models Devlin et al. [2019] finetuned on either the RTE [Dagan et al., 2006], MRPC [Dolan and Brockett, 2005] or SST-2 [Dolan and Brockett, 2005] datasets. For each dataset, Matena and Raffel [2022] pick five finetuned BERT checkpoints from the Huggingface datahub which all share the same initialization. They then merge these models using either Fisher merging, Isotropic merging, or the ensemble—where the output predictions of each model are averaged. Note that for both Fisher and Isotropic merging,

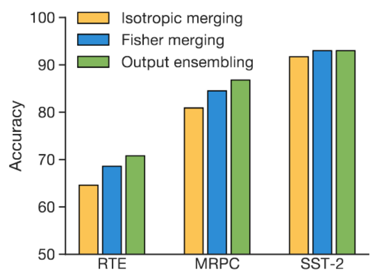


Figure 20: (Copy of Figure 2 from [Matena and Raffel, 2022]). Validation accuracy of merging different groups of five fine-tuned BERT models on the RTE, MRPC and SST-2 datasets respectively.

Matena and Raffel [2022] use uniform interpolation weights (i.e., $t_i = 1/5$) for each model. Figure 20 illustrates the results of these merges. Overall, we can see that Fisher merging significantly outperform isotropic merging (e.g., Model-Soups) across all tasks, and even nearly matches the ensemble performance for SST-2. These results appear strong, showcasing the benefits of utilizing Fisher features to weigh parameters interpolations between base models. However, it is important to remember that Fisher merging is significantly more expensive than either isotropic merging or ensembling in terms of *compute required to merge*. This is because Fisher merging requires computing many gradients, whereas the remaining methods can directly obtain merged models using only the model weights (i.e., without any data).

Robust Fine-Tuning. Another interesting experiment Matena and Raffel [2022] conduct involves a procedure termed "robust fine-tuning" (WiSE-FT) [Wortsman et al., 2022b]. In robust fine-tuning, Wortsman et al. [2022b] show that across extensive scenarios finetuned models can be interpolated with their pretrained checkpoints to achieve both significantly better performance on (1) the finetune task and (2) preserve performance on the pre-training task. This makes WiSE-FT a very simple and efficient merging procedure for general model-finetuning without catastrophic forgetting. Note that not *all interpolations* between the pretrained base model and its finetuned variant are guaranteed to be strong. Thus, a search over interpolation weights (i.e., t_i) is typically required to find the optimal merged-model. Matena and Raffel [2022] conduct WiSE-FT using a pretrained ImageNet [Deng et al., 2009] ViT-B/16 model [Dosovitskiy et al., 2020] and comparitive ViT-B/16s of the same initialization but finetuned on five out-of-domain (OOD) datasets: ImageNet-A [Hendrycks et al., 2021b], ImageNet-R [Hendrycks et al., 2021a], ImageNet Sketch [Wang et al., 2019], ImageNet V2 [Recht et al., 2019] and ObjectNet [Barbu et al., 2019]. They then then perform pairwise merging of the pretrained model to each out-of-domain checkpoint, and average the merged model accuracies over all OOD datasets. Figure 21 compares the results when using Fisher vs Isotropic merging as the interpolation weights between the in-domain model and finetuned model change. Note that because Matena and Raffel [2022] only perform pairwise merging, only one interpolation weight (e.g., t_1) is needed as the second can simply be $t_2 = 1 - t_1$.

Overall the figure displays two interesting results. First, it appears like Fisher merging *on the whole* tends to yield better performances than Isotropic merging across interpolation weights. This is a nice result because it suggests that the actual interpolation value doesn't matter much in terms of whether Fisher is better than Isotropic merging, so we can be more confident that we'll achieve stronger performances when using Fisher in general. Second, this experiment setting appears much harder than the "same-task model merging" setting presented earlier. Indeed, it appears like the gap between Fisher merging and isotropic merging diminishes as task complexity increases.

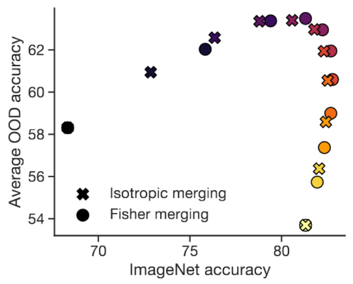


Figure 21: (Copy of Figure 3 from [Matena and Raffel, 2022]). Average Accuracies of WiSE-FT for a set of OOD datasets and the in-distribution ImageNet dataset. Dark to light color corresponds to the interpolation weight progressing from 0 to 1.

Coupling these last two observations with the significant added cost to compute Fisher information, I wonder if Fisher merging may be too costly for the degree of performance uplift. Another question I have is how the story changes/looks like when more than two models are merged at a time. For instance, how does Figure 21 change when *all five* OOD models are merged with the in-distribution model? Do we see a similar trend as the figure? Does performance degrade significantly? Does one task "out-weigh" the others, causing catastrophic forgetting on the remaining datasets? These are all very interesting questions to me, and something I would explore further.

Intermediate-Task Training. Matena and Raffel [2022] also explore a novel setting they call "Intermediate-Task Training." Here, [Matena and Raffel, 2022] conduct an experiment where they consider different BERT-base models (that all share the same pretrain initialization) finetuned over each dataset from the GLUE [Wang et al., 2018] benchmark, that are then finetuned on the RTE task. For instance, one such model may be first finetuned on the MRPC dataset, before then being further finetuned on the RTE dataset. Matena and Raffel [2022] refer to each preliminary GLUE dataset that is used to then finetune for RTE a "donor task."

To my best understanding (this section of their paper was not very clearly phrased in my opinion) Matena and Raffel [2022] then merge each of these new models with their original GLUE finetuned version. To illustrate, one such merge may involve the new RTE model that was originally finetuned on MRPC, with its initialization: the original MRPC finetuned model. Matena and Raffel [2022] aggregate the results from using Fisher and Isotropic merging on this problem, and compare them to (1) performances of the RTE models originally finetuned on other GLUE datasets, and (2) performance from only finetuning on RTE. For both Fisher and Isotropic merging, Matena and Raffel [2022] conduct a grid search to find the best interpolation weights. Figure 22 shows the results. Interestingly, finetuning on a preceding task before RTE only sometimes is better than directly finetuning on RTE itself. However, both Fisher and Isotropic merging always improve over only finetuning on RTE, no matter the donor task. Fisher and Isotropic merging appear to be very close across nearly all donor settings: only for 2/7 donor tasks is Fisher noticeably better than Isotropic merging. Although this is a Fisher merging paper, this result further underscores the effectiveness of istropric merging (i.e., model soups [Wortsman et al., 2022a]) across a variety of settings. It appears that when models share the same initializations, they can be combined with very promising results by simply interpolating between them—even when trained on completely separate tasks.

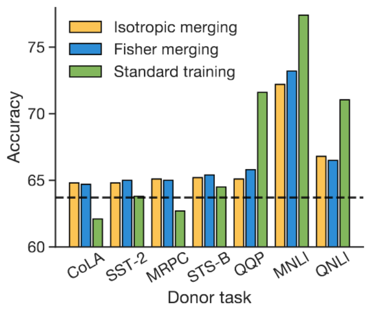


Figure 22: (Copy of Figure 4 from [Matena and Raffel, 2022]). Validation set accuracy on RTE when performing intermediate-task training with datasets from GLUE as the "donor task" to RTE. "Standard training" represents first finetuning on one of the (non-RTE) GLUE datasets and then finetuning on RTE. The dashed line is the accuracy after only finetuning on RTE.

6 Conclusion & Future Directions with the Help of: On the Symmetries of Deep Learning Models and their Internal Representations

Throughout this report, we have extensively covered model merging from two perspectives. In Sections 3-4, we learned how mode connectivity can be utilized as a tool to find interpolation paths between arbitrary models, which yield powerful merged models. Then, in Section 5 we expanded our view to include model merging approaches that were not explicitly rooted in mode connectivity. Overall, the works we have discussed proposed a variety of diverse methods to synthesize the knowledge of different models into one that retains the individual strengths of each. Despite this, they all share one underlying intuition:

A successful merge of M models into one that retains the knowledge of each, requires the internal representations of each model to be *aligned*

In this section, we build upon this intuition with the help of "On the Symmetries of Deep Learning Models and their Internal Representations" by [Godfrey et al., 2022]. Note that this section is styled a little differently than our preceding ones, because [Godfrey et al., 2022] is actually *not* a mode connectivity or model merging paper in the strict sense. Instead, it is a rigorous theoretical study on symmetries and function-invariances within the internal representations of neural networks sharing the same architectures and processed over the same data. While this is a discrepancy, their work consists of several analyses and experiments that collectively support and build upon our understanding of merging efficacy across different models. We highlight each of these below, and how they motivate diverse future explorations. We then finish our report with a conclusion summarizing our findings.

6.1 Measuring Representation Symmetry and Alignment with Intertwiner Groups

The fundamental building block of Godfrey et al. [2022] is the concept of intertwiner groups over network architectures. These groups consist of the set of transformations (e.g., permutations) that can be applied to a networks parameters while preserving function-invariance. Intertwiner groups exist at the layer-level of a network, with each layer specifying a *different* group. Additionally, each group sits *on top* of the layer's activation. We formalize this below using the following notation.

Notation. Let Model A and Model B be two neural networks of the same architecture. For simplicity (though this can be extended to more complex architectures), assume their network structure is that of an MLP with L layers. Let i denote the i^{th} layer, and $g_i(x) = \sigma_i(W_i x + b_i)$ be the function representation of the layer. Here σ_i is the activation (e.g., ReLU), $W_i \in \mathbb{R}^{d_i \times d_{i-1}}, b_i \in \mathbb{R}^{d_i}$ are the weight and bias respectively, and $x \in \mathbb{R}^{d_{i-1}}$ is an arbitrary input. Using g, we can denote the full function of each model as $f^A = g_L^A \circ \cdots \circ g_1^A$ and $f^B = g_L^B \circ \cdots \circ g_1^B$ respectively. For each $1 \leq i < L-1$, we can then decompose f^A as $f^A = f_{>i}^A \circ f_{\leq i}^A$, where $f_{\leq i}^A = f_i^A \circ \cdots \circ f_1^A$ and $f_{>i}^A = f_L^A \circ \cdots \circ f_{i+1}^A$. We define the same for f^B. Further, let $\mathcal{H}_i$ denote the set of invertible matrices $\in \mathbb{R}^{d_i \times d_i}$. Lastly, assume that $\sigma_i = \text{ReLU}\ \forall i$. Although in principle σ_i can be many different activations, Godfrey et al. [2022] focus their analysis on ReLU networks.

With this notation in mind, let $G_{\sigma_i}^A$ denote the intertwiner group for a layer f_i^A. We now formally define this group as,

$$G_{\sigma_i}^A := \{U \in \mathcal{H}_i | \exists V \in \mathcal{H}_i \text{ such that } V\sigma_i(U \cdot (W_i^A x + b_i^A)) = f_i^A(x)\} \tag{20}$$

In other words, $G_{\sigma_i}^A$ is the set of invertible linear transformations that can be applied to the i^{th} layer of a network, which can be undone in a functionally-preserving manner by another invertible transformation. It is straightforward to see that the set of permutation matrices $\mathcal{P}$ is a contained by $G_{\sigma_i}^A$, as (1) the operation of any permutation is undone by its inverse and (2) it is invariant to ReLU non-linearities. Note that V functionally depends on U. Thus, Godfrey et al. [2022] introduce a function, ϕ_σ to compute it: $V = \phi(U^{-1})$. ϕ_σ also depends on the type of activation (e.g., ReLU), but is fortunately simple for several popular activations. Table 8 defines both the intertwiner groups and corresponding ϕ_σ's for several popular activation functions.

Activation	G_{σ_n}	$\phi_\sigma(A)$
$\sigma(x) = x$ (identity)	$GL_n(\mathbb{R})$	A
$\sigma(x) = \frac{e^x}{1+e^x}$	Σ_n	A
$\sigma(x) = \text{ReLU}(x)$	Matrices PD, where D has positive entries	A
$\sigma(x) = \text{LeakyReLU}(x)$	Same as ReLU as long as negative slope $\neq 1$	A
$\sigma(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ (RBF)	Matrices PD, where D has entries in $\{\pm 1\}$	abs(A)
$\sigma(x) = x^d$ (polynomial)	Matrices PD, where D has non-zero entries	$A^{\odot d}$

Table 8: (Copy of Table 1 from [Godfrey et al., 2022]) Explicit descriptions of intertwiner groups (G_{σ_i}—denoted by G_{σ_n} here) and ϕ_σ for six different activation functions. $GL_n(\mathbb{R})$ is equivalent to our defined $\mathcal{H}_i$. Here $P \in \mathcal{P}$ is a permutation matrix, D is a diagonal matrix, *abs* is the elementwise absolute value, and $A^{\odot d}$ denotes the elementwise d^{th} power.

Discussion. Overall, we can make several interesting observations. First, we find that across the board, nearly all activations (with the exception of the identity) have very structured function-invariant transformation spaces that pose heavy restrictions on the transformation shape. Similarly, the optimal "undoer" for any chosen transformation can be easily and efficiently computed by ϕ_σ. Second, let us consider these results from the linear mode connectivity (LMC) perspective. By definition, each G_{σ_i} denotes the *set* of function-invariant transformations for each type of activation. Thus, G_{σ_i} *defines* the set of valid-transformations we can apply on an arbitrary model (say some Model B) in the hopes of moving it to the same minima as another model (say Model A) for merging. Recall from Section 4 how [Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023] proposed to map differently initialized models to the same minima via permutations, and all their models used ReLU activations (e.g., ResNet [He et al., 2016]). Well, focusing on the ReLU row, we observe that G_{σ_i} is the set of matrices consisting of PD, where P is a permutation and D is a positive diagonal matrix! So, in other words, the set of transformations considered by [Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023] effectively encompass the set of *all* valid transformations, without being scaled by

D. This tells us two things. The first, is that permutations are very nearly the *optimal* transformations for aligning one model to another when function-invariance is desired under ReLU networks. Second, there *is space* for more complex transformations to be considered.

Future Directions. This set of remaining transformations (i.e., $G_{\sigma_i} - \mathcal{P}$) is entirely dependent on D. This is very interesting to me because we can further think of each entry in D as an element-wise weight on the importance of a particular neuron when it is being aligned to another of a different model. We can illustrate this by looking again at the linear interpolation path for two models presumed LMC via the following function-invariant transformations: $D^A I \in G_{\sigma_i}$ for Model A and $D^B P^B \in G_{\sigma_i}$ for Model B, where I is the identity. Using these transformations, we can denote the LMC interpolation-path by,

$$\begin{aligned}
\theta_C &= t \cdot D^A I \theta_A + (1-t) \cdot D^B P^B \theta_B && (21)\\
&= t \cdot D^A \theta_A + (1-t) \cdot D^B P^B \theta_B && (22)\\
&= \begin{bmatrix} t \cdot D^A_{1,1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & t \cdot D^A_{|net|,|net|} \end{bmatrix} \cdot \theta_A + \begin{bmatrix} (1-t) D^B_{1,1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & (1-t) D^B_{|net|,|net|} \end{bmatrix} \cdot P\theta_B && (23)
\end{aligned}$$

where $|net|$ is the number of parameters in the model. Thus, each D^A, D^B diagonal-element (e.g., $D_{i,i}$) places a *different* weight on the corresponding parameter (e.g., parameter i) of θ_A, θ_B respectively. Interestingly, recall the Fisher information weights from [Matena and Raffel, 2022] that we discussed previously. Under appropriate normalization, these fit under D^A, D^B! Thus, G_{σ_i} encapsulates the search spaces and intuitions over several works we have discussed in this report. Yet, the existence of function-invariant transformations not covered by [Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023] yields space for future explorations that may improve upon their work by closing the gap towards transformations covering all of G_{σ_i}.

6.2 Understanding Internal Representations with Intertwiner Groups and Model Stitching

To gain a better understanding of the differences between the internal representations of distinct models, Godfrey et al. [2022] propose to use model-stitching [Csiszárik et al., 2021, Bansal et al., 2021] with their intertwiner groups. Model-stitching [Csiszárik et al., 2021, Bansal et al., 2021] describes a setting where the layers of two networks (e.g., f^A, f^B) are concatenated together at a specified depth. Usually, this concatenation also involves adding a linear layer to translate from the outputs of one model to the inputs of the other. Concretely, we define model-stitching as follows. We use the same notation as in Section 6.1 and further represent the linear layer by $\varphi : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ to show model-stitching at an arbitrary layer-i: $f^C = f^A_{>i} \circ \varphi \circ f^B_{\leq i}$.

In a typical stitching experiment, f^A, f^B are trained on the same data but with different initializations. The models are then frozen, and stitching is performed at a specified layer, and φ is inserted. The new stitched-model (e.g., f^C is then trained on the same problem, but with gradients only applied to φ. The intuition is that if f^C can recover the accuracy of the original models used in stitching: $\text{Acc}(f^C) \approx \text{Acc}(f^A), \text{Acc}(f^B)$, then f^A and f^B have *similar representations* at layer-i (the stitching layer). Godfrey et al. [2022] propose to run two types of model-stitching experiments where φ differs in each one. The first type of experiment is conducted with φ parameterized as an affine layers as is standard [Csiszárik et al., 2021, Bansal et al., 2021]. For this experiment type, they run one experiment each on φ as a rank-$\{1, 2, 4, \text{full}\}$ affine transformation. On the other hand, the second type of experiment consists of one experiment and restricts φ to be a *linear transformation* drawn from the corresponding G_{σ_i} of layer-i. Note $\sigma_i = \text{ReLU}$ (i.e., $G_{\sigma_i} = G_{\text{ReLU}}$) in this experiment and φ is *linear* as opposed to affine (i.e., there is no bias term). Thus, the stitching does not involve *any translations*. Godfrey et al. [2022] posit that if the stitched model in the second experiment is able to achieve the same performance as f^A, then f^A and f^B are simply permutations scaled by some values of each others weights at layer-i. Intuitively, this makes sense, as φ in the second experiment can only comprise of a permutation and scaling term. Godfrey et al. [2022] measure the performance of the final stitched model relative to the original ones using a metric they refer to as the "stitching penalty":

$[\mathrm{Acc}(f^A) + \mathrm{Acc}(f^B)]/2 - \mathrm{Acc}(f^C)$. We observe that this metric measures the performance degradation caused by stitching at a particular layer-i and using one of the respective φ parameterizations.

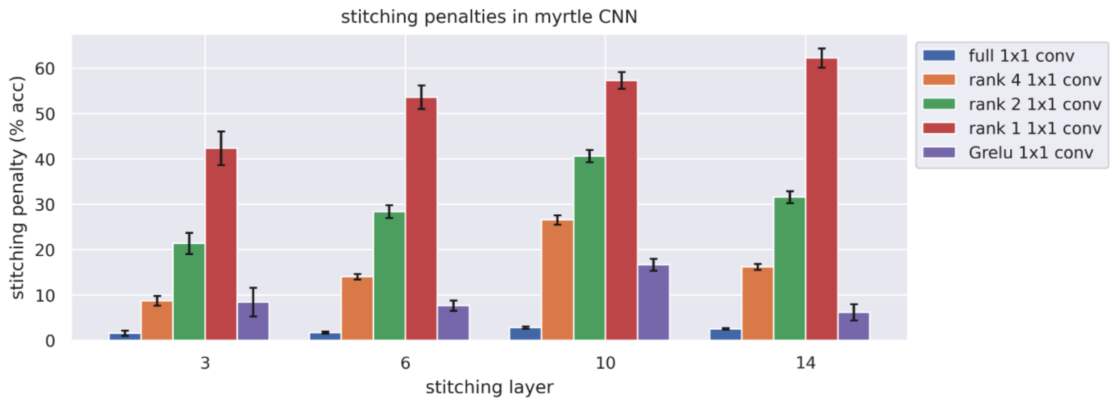


Figure 23: (Copy of Figure 1 from [Godfrey et al., 2022]) Full, reduced rank, and G_{ReLU} stitching penalties (lower is better) for Myrtle CNNs [Page, 2019] on CIFAR10 [Krizhevsky et al., 2009]. Confidence intervals are obtained by evaluating stitching penalties for 32 pairs of models trained with different random seeds. The accuracy of the models was $91.3 \pm 0.2\%$. Godfrey et al. [2022] do not elaborate on how many layers these Myrtle CNNs contain.

Figure 23 illustrates the stitching penalties (lower is better) when stitching at different layers of Mrytle CNNs [Page, 2019] for the two experiment types over the CIFAR10 Krizhevsky et al. [2009] dataset. The results are calculated by averaging the penalties of 32 pairs of models each trained with different seeds, and confidence intervals are shown.

Discussion. The results yield three interesting insights. First, we observe that transformations with sizeable rank are required to successfully stitch across different layers. This suggests that the internal representations eat each layer are sufficiently large rank, with many neurons processing distinct pieces of information. Second, we observe that the overall stitching penalty increases as stitching occurs at deeper layers. This corroborates the findings of Kornblith et al. [2019]: internal representation similarity between networks decreases in later layers. Third, and I would most like to focus on is that while φ with G_{ReLU} always performs worse than the full affine parameterization of φ, it significantly outperforms the rank-restricted variants. This suggests two things. First, f^A, f^B are not simply permutations of each others parameters at any layer. But second, G_{ReLU} is able to account for *a substantial amount* of the internal representations of independently trained networks.

Future Directions. This result reminds me a lot about the permutation barrier/accuracy plots we discussed in Sections 4.3.2-4.4.2. Across each of those plots, we consistently observed how permutation-based approaches accounted for a *significant* portion of the work needed to align two models together. We can think of the stitching penalties in Figure 23 as offering a different way of understanding the barrier plots in those sections. However, if we continue with this perspective, Figure 23 illustrates the permutation (and scaling) symmetries between two networks *at each layer* rather than the *entire* networks as in the barrier plots. Thus, these stitching penalties are in a sense yielding more granular insight on the inter-relationships between the neurons in different models at each layer. Putting the two together, one avenue of exploration I find compelling is to visualize the loss-surfaces around different networks we want to merge *at the layer level*. The central question to understand here is, "if two networks are *not mode connected* via permutation of all their weights, does this mean that they are not mode connected at *any layer*?" Specifically, is there a layer up to which they are connected but after which become disconnected?

One way we can quantitatively measure this is by the following. Suppose want to understand whether the two networks are mode-connected up to an arbitrary layer-i. What we could do, is merge the two networks up the ith layer, and then keep the remaining elements of each network *separate*, thereby creating a multihead model. We can then modify the barrier equation as follows to measure the mode-connectivity for partially merged models. For simplicity, let us assume we are searching for linear mode connectivity (LMC) a la Equation 7. Let $[\cdot;\cdot]$ be the concatenation operation. Let Model A, Model B, and ϕ^{AB} be as defined in Section 2. Let ϕ_i^{AB} only specify a linear-path up until layer-i, and let $\theta^A_{\leq i}, \theta^B_{\leq i}$ denote the parameters up to layer-i of Model A, Model B respectively. Likewise, let $\theta^A_{>i}, \theta^B_{>i}$ denote the parameters after layer-i of Model A, Model B respectively. Finally, let $\theta^A = [\theta^A_{>i}; \theta^A_{\leq i}]$, with θ^B defined the same way. Then we can define the merged components of f^A, f^B by: $\theta^C_{\leq i} = \phi^{AB}_i(t) = t \cdot \theta^A_{\leq i} + (1-t) \cdot \theta^B_{\leq i}$. Putting it all together, we can modify the linear barrier defined in Equation 7 as,

$$B(\theta_A, \theta_B, \phi^{AB}_i) = \max_{t \in [0,1]} t \left(\mathcal{L}([\theta^A_{>i}; \phi^{AB}_i(t)]) - \mathcal{L}(\theta_A)\right) + (1-t) \left(\mathcal{L}([\theta^B_{>i}; \phi^{AB}_i(t)]) - \mathcal{L}(\theta_B)\right) \qquad (24)$$

Note that as i grows to encompass all the layers of each network, Equation 24 reduces to the linear barrier from Equation 7. Notably, Equation 24 can used to estimate the extent to which any two networks are LMC at *any layer* of their networks. Moreover, we can also extend Equation 24 to include paths that start at some $1 \leq j < i$ and go up to i with just a few modifications to the notation. I think gaining a better understanding of the above questions can be critical to designing better merging algorithms. For instance, we might find that while we may not be able to merge two models entirely, we *could* merge them partially (e.g., up-to a specific layer), thereby saving compute (e.g., over an ensemble) while maintaining strong performance. [Stoica et al., 2023] have recently studied this partial merging problem. However, their results offer only preliminary understandings and much more is left to be understood.

6.3 Aggregate Layer Similarities with Intertwiner Groups and Centered Kernel Alignment

In addition to the previous experiments, Godfrey et al. [2022] conduct an analysis measuring the centered kernel alignment (CKA) [Kornblith et al., 2019] with G_{ReLU} (called G_{ReLU}-CKA) between the layers of two ResNet18s [He et al., 2016] trained on CIFAR10 with different random seeds. Figure 24 highlights the results.

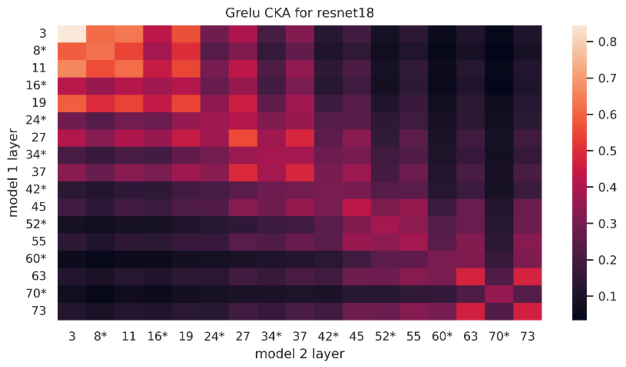


Figure 24: (Copy of Figure 11 (left) from [Godfrey et al., 2022]) G_{ReLU}-CKA for two ResNet18s trained on CIFAR10 with different random seeds. Layers marked with "*" occur inside residual blocks. Results averaged over 16 such pairs of models.

Discussion. Overall, the results seem to function as expected and corroborate [Csiszárik et al., 2021]. Layers at the beginning of a network are significantly higher in similarity between the distinct networks, than the deeper layers. Moreover, it appears that the layers within the residual blocks are *always* less similar when compared between networks than those that occur outside of the blocks. Yet taking the two types of layers into account separately, it appears like the layer-similarities within each type decrease nearly monotonically as we go deeper in the network.

Future Directions. Focusing on the diagonal shows how each layer of one network is structurally similar to the corresponding layer of the second. for me, the diagonal is the most interesting piece of this result, as it gives an aggregate measure of how the layers between two model become less and less similar with regards to function-invariant transformations, as we go deeper into the network. In fact, it appears like we may be able to utilize the diagonal of the G_{ReLU}-CKA alignment measure comparing two layers of distinct neural networks as a proxy to understand whether we should merge them. Moreover, since G_{ReLU} contains the set of *all* function-invariant transformations, G_{ReLU}-CKA serves as a pseudo-upperbound on the amount of alignment we can achieve when transforming one network to another via one of the existing methods like

Entezari et al. [2022], Ainsworth et al. [2023], Jordan et al. [2023]. Similarly, the metric could be used to understand *when to stop merging* (i.e., at what stage the internal representations of two networks become too dissimilar). Thus, G_{ReLU}-CKA could be used in tandem with the modified barrier in Equation 24 to better understand the internal representations between the layers of arbitrary networks, with an explicit focus on determining whether each respective layers should be merged.

6.4 Conclusion

Throughout this report, we have seen many diverse ideas and intuitions across ten works exploring different aspects of model merging. Several works [Draxler et al., 2018, Garipov et al., 2018, Entezari et al., 2022, Ainsworth et al., 2023, Jordan et al., 2023] analyzed the feasibility of model merging from the perspective mode connectivity, whether linear or non-linear. Collectively, they showed how robust paths could be found between arbitrary networks *with different initializations* in which *every* point along the path yielded a performant merged-model. In addition, we saw how other works [Singh and Jaggi, 2020, Wang et al., 2020, Wortsman et al., 2022a, Matena and Raffel, 2022] illustrated that under certain conditions, strong merges could be possible even without knowing if networks were mode connected. Yet, model merging is still in its infancy. Mode connectivity, especially linear mode connectivity (LMC), remains an elusive target even for the best model-merging algorithms currently available. Furthermore, as we illustrated with [Godfrey et al., 2022], literature from other domains can still provide valuable insights for us to better understand the characteristics of when merging is feasible. Even more importantly, it can spur further directions and avenues for future work. Taking all into consideration, model merging is far from solved, yet is a worthy endeavor.

References

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *ICLR*, 2023. 11, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 31, 37, 41, 42, 45

Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ak06J5jNR4. 42

Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper˙files/paper/2019/file/97af07a14cacba681feacf3012730892-Paper.pdf. 33, 39

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013. URL https://api.semanticscholar.org/CorpusID:18406556. 18

Dimitri P. Bertsekas. *Auction algorithmsAuction Algorithms*, pages 73–77. Springer US, Boston, MA, 2001. ISBN 978-0-306-48332-5. doi: 10.1007/0-306-48332-7_15. URL https://doi.org/10.1007/0-306-48332-7˙15. 16

Adrián Csiszárik, Péter Kőrösi-Szabó, Ákos K. Matszangosz, Gergely Papp, and Dániel Varga. Similarity and matching of neural network representations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=aedFIIRRfXr. 42, 44

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33428-6. 36, 38

Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=dUk5Foj5CLf. 36

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux - effortless bayesian deep learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=gDcaUj4Myhn. 37

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 7, 18, 27, 33, 39

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423. 36, 38

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL https://aclanthology.org/I05-5002. 36, 38

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv:2010.11929*, 2020. 33, 39

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *ICML*, 2018. 4, 6, 7, 8, 9, 11, 12, 13, 14, 45

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *ICLR*, 2022. 4, 8, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 28, 31, 37, 41, 42, 45

R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London, A*, 222:309–368, 1922. 37

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv:1803.03635*, 2018. 4, 11

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, 2020. 11

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *NeurIPS*, 2018. 3, 4, 8, 9, 10, 11, 12, 14, 45

Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations. In *NS*, 2022. 40, 41, 42, 43, 44, 45

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2016. 7, 41, 44

Donald O. Hebb. *The organization of behavior: A neuropsychological theory.* Wiley, New York, June 1949. ISBN 0-8058-4300-0. 16

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *ICCV*, 2021a. 33, 39

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CVPR*, 2021b. 33, 39

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL http://arxiv.org/abs/1503.02531. cite arxiv:1503.02531Comment: NIPS 2014 Deep Learning Workshop. 3

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993. 7, 8

Hannes Jonsson, GREG MILLS, and KARSTEN JACOBSEN. *Nudged elastic band method for finding minimum energy paths of transitions*, pages 385–404. 06 1998. doi: 10.1142/9789812839664_0016. 5

Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. REPAIR: REnormalizing permuted activations for interpolation repair. In *ICLR*, 2023. 4, 11, 20, 21, 22, 23, 24, 25, 29, 31, 37, 41, 42, 45

Esben L. Kolsbjerg, Michael N. Groves, and Bjørk Hammer. An automated nudged elastic band method. *The Journal of Chemical Physics*, 145(9):094107, 09 2016. ISSN 0021-9606. doi: 10.1063/1.4961868. URL https://doi.org/10.1063/1.4961868. 5

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. 43, 44

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 7, 43

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/. 14

Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010. doi: 10.1109/ISCAS.2010.5537907. 15

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020. URL https://proceedings.mlsys.org/book/316.pdf. 31

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent Learning: Do different neural networks learn the same representations? *arXiv:1511.07543*, 2015. 16, 21

Michael S Matena and Colin Raffel. Merging models with fisher-weighted averaging. In *NeruIPS*, 2022. 4, 37, 38, 39, 40, 42, 45

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 2017. 29

Markus Nenad. *Fusing batch normalization and convolution in runtime*. 2018. URL https://nenadmarkus.com/p/fusing-batchnorm-and-conv/. 22

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 14

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 512–523. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper˙files/paper/2020/file/0607f4c705595b911a4f3e7a127b44e0-Paper.pdf. 4, 33

David Page. How to train your resnet, Nov 2019. URL https://myrtle.ai/learn/how-to-train-your-resnet/. 43

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 33

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html. 36

Alexandre Ramé, Kartik Ahuja, Jianyu Zhang, Matthieu Cord, Léon Bottou, and David Lopez-Paz. Recycling diverse models for out-of-distribution generalization. *arXiv:2212.10445*, 2022. 31

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? *CoRR*, abs/1902.10811, 2019. URL http://arxiv.org/abs/1902.10811. 33, 39

Omer Sagi and Lior Rokach. Ensemble learning: A survey. *WIREs Data Mining and Knowledge Discovery*, 8 (4):e1249, 2018. doi: https://doi.org/10.1002/widm.1249. URL https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1249. 3

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL https://api.semanticscholar.org/CorpusID:14124313. 10

Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. *NeurIPS*, 2020. 25, 26, 27, 28, 29, 31, 37, 45

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170. 36

George Stoica, Daniel Bolya, Jakob Bjorner, Taylor Hearn, and Judy Hoffman. Zipit! merging models from different tasks without training, 2023. 44

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL https://aclanthology.org/W18-5446. 36, 39

Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, pages 10506–10518, 2019. 33, 39

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *ICLR*, 2020. 28, 29, 30, 31, 32, 37, 45

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018. 36

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022a. 4, 33, 34, 35, 36, 37, 38, 40, 45

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *CVPR*, 2022b. 36, 39

Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *ICML*, 2019. 30

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146. 10