Equivariant Deep Weight Space Alignment

Aviv Navon [* 1] **Aviv Shamsian** [* 1] **Ethan Fetaya** [1] **Gal Chechik** [1 2] **Nadav Dym** [3] **Haggai Maron** [3 2]

Abstract

Permutation symmetries of deep networks make basic operations like model merging and similarity estimation challenging. In many cases, aligning the weights of the networks, i.e., finding optimal permutations between their weights, is necessary. Unfortunately, weight alignment is an NP-hard problem. Prior research has mainly focused on solving relaxed versions of the alignment problem, leading to either time-consuming methods or sub-optimal solutions. To accelerate the alignment process and improve its quality, we propose a novel framework aimed at learning to solve the weight alignment problem, which we name DEEP-ALIGN. To that end, we first prove that weight alignment adheres to two fundamental symmetries and then, propose a deep architecture that respects these symmetries. Notably, our framework does not require any labeled data. We provide a theoretical analysis of our approach and evaluate DEEP-ALIGN on several types of network architectures and learning setups. Our experimental results indicate that a feed-forward pass with DEEP-ALIGN produces better or equivalent alignments compared to those produced by current optimization algorithms. Additionally, our alignments can be used as an effective initialization for other methods, leading to improved solutions with a significant speedup in convergence.

1. Introduction

The space of deep network weights has a complex structure since networks maintain their function under certain permutations of their weights. This fact makes it hard to perform simple operations over deep networks, such as averaging their weights or estimating similarity. It is there-

fore highly desirable to "align" networks - find optimal permutations between the weight matrices of two networks. Weight Alignment is critical to many tasks that involve weight spaces. One key application is model merging and editing (Ainsworth et al., 2022; Wortsman et al., 2022; Stoica et al., 2023; Ilharco et al., 2022), in which the weights of two or more models are (linearly) combined into a single model to improve their performance or enhance their capabilities. Weight alignment algorithms are also vital to the study of the loss landscape of deep networks (Entezari et al., 2022), a recent research direction that has gained increasing attention. Moreover, weight alignment induces an invariant distance function on the weight space that can be used for clustering and visualization.

Since weight alignment is NP-hard (Ainsworth et al., 2022), current approaches rely primarily on local optimization of the alignment objective which is time-consuming and may lead to suboptimal solutions. Therefore, identifying methods with faster run time and improved alignment quality is an important research objective. A successful implementation of such methods would allow practitioners to perform weight alignment in real time which is crucial for several applications. One example of such an application is federated learning setups where weight alignment can improve convergence speed and model quality when models are aligned before weight averaging is performed at each global step (Wang et al., 2020a). Additionally, the ability to align models in real-time is crucial for any task requiring the computation of multiple alignments in a reasonable time, including continual learning setups, weight space mixup (Shamsian et al., 2023), or clustering weight space data.

Following a large body of works that suggested *learning* to solve combinatorial optimization problems using deep learning architectures (Khalil et al., 2017; Bengio et al., 2021; Cappart et al., 2021), we propose the first learning-based approach to weight alignment, called DEEP-ALIGN. DEEP-ALIGN is a neural network with a specialized architecture that is trained to predict high-quality weight alignments for a given distribution of models. A major benefit of our approach is that after a model has been trained, predicting the alignment between two networks amounts to a simple feed-forward pass through the network followed by an efficient projection step, as opposed to solving an optimization problem in other methods.

---
[*]Equal contribution [1]Bar-Ilan University [2]NVIDIA Research [3]Technion. Correspondence to: Aviv Navon <aviv.navon@biu.ac.il>, Aviv Shamsian <aviv.shamsian@biu.ac.il>.

This paper presents a principled approach to designing a deep architecture for the weight alignment problem. We first formulate the weight-alignment problem and prove it adheres to a specific equivariance structure. We then propose a neural architecture that respects this structure, based on newly suggested equivariant architectures for deep-weight spaces (Navon et al., 2023) called Deep Weight Space Networks (DWSNets). The architecture is based on a Siamese application of DWSNets to a pair of input networks, mapping the outputs to a lower dimensional space we call *activation space*, and then using a generalized outer product layer to generate candidates for optimal permutations. Most importantly, and similarly to other equivariant architectures (Zaheer et al., 2017; Kondor & Trivedi, 2018; Cohen et al., 2021), our equivariant design facilitates efficient learning and inference. Specifically, (1) the feedforward computation relies only on simple, efficient blocks (Navon et al., 2023), (2) The inclusion of a parameter sharing scheme in each layer reduces the overall number of parameters, and (3) accounting for symmetries reduces the training set size needed for learning (Figure 4). We demonstrate this efficiency by learning to align 10M+ parameter networks with DEEP-ALIGN using small datasets.

Theoretically, we prove that our architecture can approximate the Activation Matching algorithm (Tatro et al., 2020; Ainsworth et al., 2022), which computes the activations of the two networks on some pre-defined input data and aligns their weights by solving a sequence of linear assignment problems. This theoretical analysis suggests that DEEP-ALIGN can be seen as a learnable generalization of this algorithm. Furthermore, we prove that DEEP-ALIGN has a valuable theoretical property called *Exactness*, which guarantees that it always outputs the correct alignment when there is a solution with zero objective.

Obtaining labeled training data is one of the greatest challenges when learning to solve combinatorial optimization problems. To address this challenge, we generate labeled examples on the fly by applying random permutations and noise to the unlabeled data. We then train our network by combining a supervised loss on these synthetically generated labeled examples and an unsupervised loss on arbitrary pairs of samples. Crucially, both losses do not rely on any externally labeled data.

Our experimental results indicate that DEEP-ALIGN produces better or comparable alignments relative to those produced by slower optimization-based algorithms, when applied to both MLPs and CNNs. Furthermore, we show that our alignments can be used as an initialization for other methods that result in even better alignments, as well as significant speedups in their convergence. Lastly, we show that our trained networks produce meaningful alignments even when applied to out-of-distribution weight space data. In

particular, this is demonstrated by showing the effectiveness of our method in a federated learning setup.

1.1. Previous work

Several algorithms have been proposed for weight-alignment (Tatro et al., 2020; Ainsworth et al., 2022; Peña et al., 2023; Akash et al., 2022). Ainsworth et al. (2022) presented three algorithms: Activation Matching, Weight Matching, and straight-through estimation. Peña et al. (2023) improved upon these algorithms by incorporating a Sinkhorn-based projection method. In part, these works were motivated by studying the loss landscapes of deep neural networks. It was conjectured that deep networks exhibit a property called *linear mode connectivity*: for any two trained weight vectors (i.e., a concatenation of all the parameters of neural architecture), a linear interpolation between the first vector and the optimal alignment of the second, yields very small increases in the loss (Entezari et al., 2022; Garipov et al., 2018; Draxler et al., 2018; Freeman & Bruna, 2016; Tatro et al., 2020). Another relevant research direction is the growing area of research that focuses on applying neural networks to neural network weights. Early methods proposed using simple architectures (Unterthiner et al., 2020; Andreis et al., 2023; Eilertsen et al., 2020). Several recent papers exploit the symmetry structure of the weight space in their architectures (Navon et al., 2023; Zhou et al., 2023a;b; Zhang et al., 2023; Lim et al., 2023). A comprehensive survey of relevant previous work can be found in Appendix A.

2. Preliminaries

Equivariance Let G be a group acting on $\mathcal{V}$ and $\mathcal{W}$. We say that a function $L : \mathcal{V} \to \mathcal{W}$ is *equivariant* if $L(gv) = gL(v)$ for all $v \in \mathcal{V}, g \in G$.

MultiLayer Perceptrons and weight spaces. The following definition follow the notation in (Navon et al., 2023). An M-layer MultiLayer Perceptron (MLP) f_v is a parametric function of the following form:

$$f_v(x) = x_M, \quad x_{m+1} = \sigma(W_{m+1}x_m + b_{m+1}), \quad x_0 = x \tag{1}$$

Here, $x_m \in \mathbb{R}^{d_m}$, $W_m \in \mathbb{R}^{d_m \times d_{m-1}}$, $b_m \in \mathbb{R}^{d_m}$, and σ is a pointwise activation function. Denote by $v = [W_m, b_m]_{m \in [M]}$ the concatenation of all (vectorized) weight matrices and bias vectors. We define the *weight-space* of an M-layer MLP as: $\mathcal{V} = \bigoplus_{m=1}^{M} (\mathcal{W}_m \oplus \mathcal{B}_m)$, where $\mathcal{W}_m := \mathbb{R}^{d_m \times d_{m-1}}$ $\mathcal{B}_m = \mathbb{R}^{d_m}$ and $\bigoplus$ denotes the direct sum (concatenation) of vector spaces. A vector in this space represents all the learnable parameters on an MLP. We define the *activation space* of an MLP as $\mathcal{A} = \bigoplus_{m=1}^{M} \mathbb{R}^{d_m} := \bigoplus_{m=1}^{M} \mathcal{A}_m$. The activation space, as its name implies, represents the concatenation of network
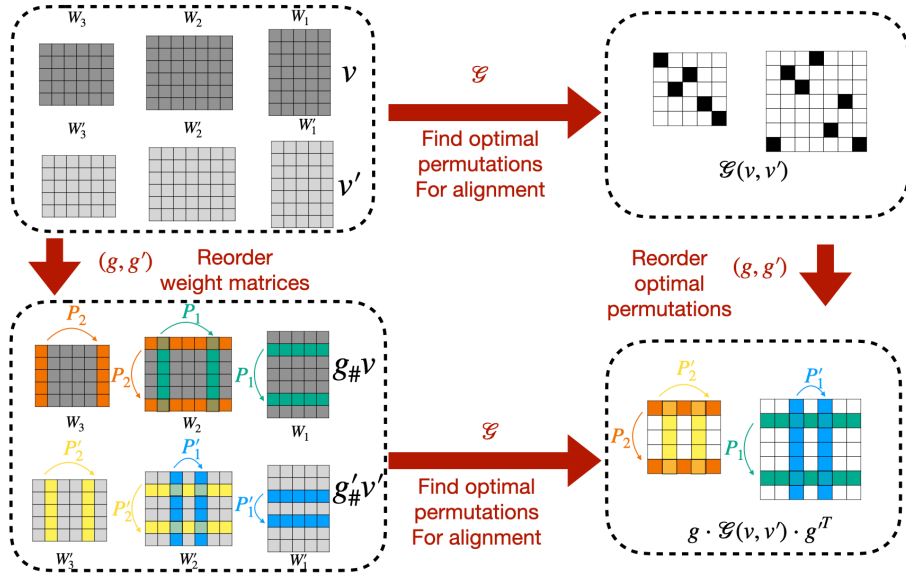
Figure 1. The equivariance structure of the alignment problem. The function $\mathcal{G}$ takes as input two weight space vectors v, v' and outputs a sequence of permutation matrices that aligns them denoted $\mathcal{G}(v, v')$. In case we reorder the input using (g, g') where $g = (P_1, P_2), g' = (P_1', P_2')$, the optimal alignment undergoes a transformation, namely $\mathcal{G}(g_\# v, g_\#' v') = g \cdot \mathcal{G}(v, v') \cdot g'^T$.

activations at all layers. i.e., $\mathcal{A}_m$ is the space in which x_m resides.

Symmetries of weight spaces. The permutation symmetries of the weight space are a result of the equivariance of pointwise activations: for every permutation matrix P we have that $P\sigma(x) = \sigma(Px)$. Thus for example, a shallow network defined by weight matrices W_1, W_2 will represent the same function as the network defined by PW_1, W_2P^T, since the permutations cancel each other. The same idea can be used to identify permutation symmetries of general MLPs of depth M. In this case, the weight space's symmetry group is the direct product of symmetric groups for each intermediate dimension $m \in [1, M-1]$ namely, $S_{d_1} \times \cdots \times S_{d_{M-1}}$. For clarity, we formally define the symmetry group as a product of matrix groups: $G = \Pi_{d_1} \times \cdots \times \Pi_{d_{M-1}}$, where Π_d is the group of $d \times d$ permutation matrices (which is isomorphic to S_d). For $v \in \mathcal{V}$, $v = [W_m, b_m]_{m \in [M]}$, a group element $g = (P_1, \ldots, P_{M-1})$ acts on v via a group action $v' = g_\#(v)$, where $v' = [W_m', b_m']_{m \in [M]}$ is defined by:

$$W_1' = P_1 W_1, W_M' = W_M P_{M-1}^T, \text{ and}$$
$$W_m' = P_m W_m P_{m-1}^T, \forall m \in [2, M-1]$$
$$b_1' = P_1 b_1, b_{M'} = b_M, \text{ and}$$
$$b_m' = P_m b_m, \forall m \in [2, M-1].$$

By construction, v and $v' = g_\#(v)$ define the same function $f_v = f_{v'}$. The group product $g \cdot g'$ and group inverse $g^{-1} = g^T$ are naturally defined as the element-wise matrix product and transpose operations $g \cdot g' = (P_1 P_1', \ldots, P_M P_M')$, $g^T = (P_1^T, \ldots, P_m^T)$. Note that the

elementwise product and transpose operations are well defined even if the P_m and P_m' matrices are not permutations.

DWSNets. In this paper, we use DWSNets (Navon et al., 2023) as weight space encoders in DEEP-ALIGN. This architecture is denoted by $F : \mathcal{V} \to \mathcal{V}$ and is of the following form:

$$F = L_k \circ \sigma \circ L_{k-1} \circ \sigma \cdots \circ \sigma \circ L_1,$$

where L_i are linear G-equivariant layers employing specific parameter sharing schemes (Ravanbakhsh et al., 2017) for processing weight spaces, and $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear function applied elementwise. This is similar to the design of many previous equivariant architectures (Zaheer et al., 2017; Hartford et al., 2018; Maron et al., 2018; 2020; Wang et al., 2020b). Each such layer $L : \mathcal{V} \to \mathcal{V}$ takes as input representations of *all* the weights and biases which we denote as $v \in \mathcal{V}$, $v = [W_m, b_m]_{m \in [M]}$ and outputs new representations based on all the input weights and biases. We note that any other G-equivariant weight space network (Zhou et al., 2023b; Zhang et al., 2023; Zhou et al., 2023a; Lim et al., 2023) can be used seamlessly instead of DWSNets.

3. The weight alignment problem and its symmetries

The weight alignment problem. Given an MLP architecture as in equation 1 and two weight-space vectors $v, v' \in \mathcal{V}$, where $v = [W_m, b_m]_{m \in [M]}, v' = [W_m', b_m']_{m \in [M]}$, the weight alignment problem is defined as the following opti-
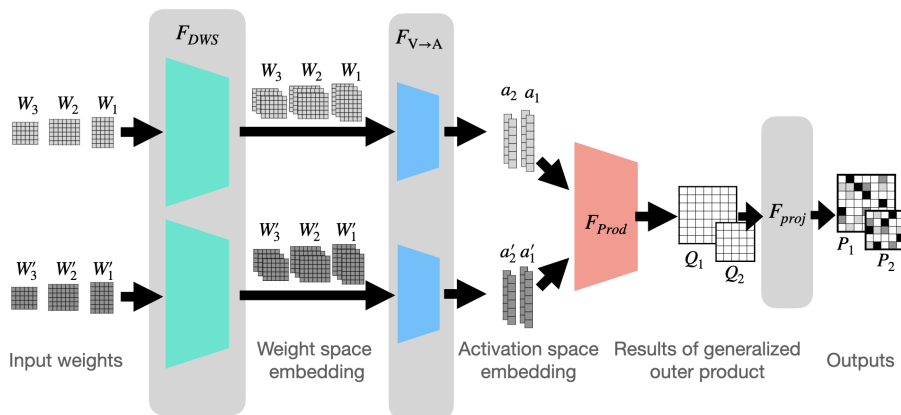
Figure 2. Our architecture is a composition of four blocks: The first block, F_{DWS} generates weight space embedding for both inputs. The second block $F_{\mathcal{V} \to \mathcal{A}}$ maps these to the activation spaces. The third block, F_{Prod}, generates square matrices by applying an outer product between the activation vector of one network to the activation vectors of the other network. Lastly, the fourth block, F_{Proj} projects these square matrices on the (convex hull of) permutation matrices.

mization problem:

$$\mathcal{G}(v, v') = \operatorname{argmin}_{k \in G} \|v - k_\# v'\|_2^2 \qquad (2)$$

In other words, the problem seeks a sequence of permutations $k = (P_1, \ldots, P_{M-1})$ that will make v' as close as possible to v. The optimization problem in equation 2 always admits a minimizer since G is finite. For some (v, v') it may have several minimizers, in which case $\mathcal{G}(v, v')$ is a set of elements. To simplify our discussion we will sometimes consider the domain of $\mathcal{G}$ to be only the set $\mathcal{V}^2_{\text{unique}}$ of pairs (v, v') for which a unique minimizer exists. On this domain we can consider $\mathcal{G}$ as a function to the unique minimizer in G, that is $\mathcal{G} : \mathcal{V}^2_{\text{unique}} \to G$.

Our goal in this paper is to devise an architecture that can learn the function $\mathcal{G}$. As a guiding principle for devising this architecture, we would like this architecture to be equivariant to the symmetries of $\mathcal{G}$. We describe these symmetries next.

The symmetries of $\mathcal{G}$. One important property of the function $\mathcal{G}$ is that it is equivariant to the action of the group $H = G \times G$ which consists of two independent copies of the permutation symmetry group for the MLP architecture we consider. Here, the action $h = (g, g') \in H$ on the input space $\mathcal{V} \times \mathcal{V}$ is simply $(v, v') \mapsto (g_\# v, g'_\# v')$, and the action of $h = (g, g') \in H$ on an element $k \in G$ in the output space is given by $g \cdot k \cdot g'^T$. This equivariance property is summarized and proved in the proposition below and visualized using the commutative diagram in Figure 1: applying $\mathcal{G}$ and then (g, g') results in exactly the same output as applying (g, g') and then $\mathcal{G}$.

Proposition 3.1. *The map $\mathcal{G}$ is H-equivariant, namely, for all $(v, v') \in \mathcal{V}^2_{\text{unique}}$ and $(g, g') \in H$,*

$$\mathcal{G}(g_\# v, g'_\# v') = g \cdot \mathcal{G}(v, v') \cdot g'^T$$

The function $\mathcal{G}$ exhibits another interesting property: swapping the order of the inputs v, v' corresponds to inverting the optimal alignment $\mathcal{G}(v, v')$:

Proposition 3.2. *Let $(v, v') \in \mathcal{V}^2_{\text{unique}}$ then $\mathcal{G}(v', v) = \mathcal{G}(v, v')^T$.*

Extension to multiple minimizers. For simplicity the above discussion focused on the case where $(v, v') \in \mathcal{V}^2_{\text{unique}}$. We can also state analogous claims for the general case where multiple minimizers are possible. In this case we will have that the equalities $g \cdot \mathcal{G}(v, v') \cdot g'^T = \mathcal{G}(gv, g'v')$ and $\mathcal{G}(v, v')^T = \mathcal{G}(v', v)$ still hold as equalities between subsets of G.

Extension to other optimization objectives. In Appendix B we show that the equivariant structure of the function $\mathcal{G}$ occurs not only for the objective in equation 2, but also when the objective $\|v - k_\# v'\|_2^2$ is replaced with any scalar function $E(v, k_\# v')$ that satisfies the following properties: (1) E is invariant to the action of G on both inputs; and (2) E is invariant to swapping its arguments.

4. DEEP-ALIGN

4.1. Architecture

Here, we define a neural network architecture $F = F(v, v'; \theta)$ for learning the weight-alignment problem.
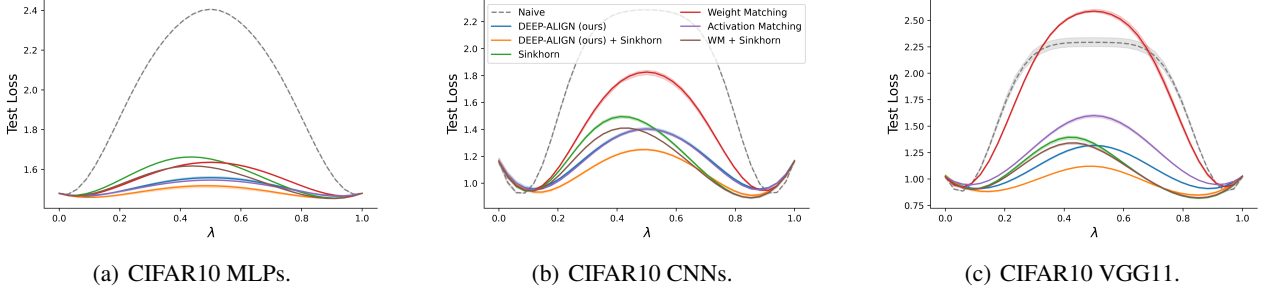
(a) CIFAR10 MLPs.

(b) CIFAR10 CNNs.

(c) CIFAR10 VGG11.

Figure 3. Merging image classifiers: the plots illustrate the values of the loss function used for training the input networks when evaluated on a line segment connecting v and $g_\# v'$, where g is the output of each method. Values are averaged over all test images and networks and 3 random seeds.

The output of F will be a sequence of square matrices $(P_1, \ldots, P_{M-1})$ that represents a (sometimes approximate) group element in G. In order to provide an effective inductive bias, we will ensure that our architecture meets both properties: 3.1,3.2, namely $F(g_\# v, g'_\# v') = g \cdot F(v, v') \cdot g'^T$ and $F(v, v') = F(v', v)^T$.

The architecture we propose is composed of four functions:

$$F = F_{proj} \circ F_{prod} \circ F_{\mathcal{V} \to \mathcal{A}} \circ F_{DWS} : \mathcal{V} \times \mathcal{V}' \to \bigoplus_{m=1}^{M-1} \mathbb{R}^{d_m \times d_m},$$

where the equivariance properties we require are guaranteed by constructing each of the four functions composing F to be equivariant with respect to an appropriate action of $H = G \times G$ and the transposition action $(v, v') \mapsto (v', v)$. In general terms, we choose F_{DWS} to be a siamese weight space encoder, $F_{\mathcal{V} \to \mathcal{A}}$ is a siamese function that maps the weight space to the activation space, F_{prod} is a function that performs (generalized) outer products between corresponding activation spaces in both networks and F_{proj} performs a projection of the resulting square matrices on the set of doubly stochastic matrices (the convex hull of permutation matrices). The architecture is illustrated in Figure 2. We now describe our architecture in more detail.

Weight space encoder . $F_{DWS} : \mathcal{V} \times \mathcal{V}' \to \mathcal{V}^d \times \mathcal{V}'^d$, where d represents the number of feature channels, is implemented as a Siamese DWSNet (Navon et al., 2023). This function outputs two weight-space embeddings in $\mathcal{V}^d$, namely, $F_{DWS}(v, v') = (\mathcal{E}(v), \mathcal{E}(v'))$, for a DWS network $\mathcal{E}$. The Siamese structure of the network guarantees equivariance to transposition. This is because the same encoder is used for both inputs, regardless of their input order. The G-equivariance of DWSNet, on the other hand, implies equivariance to the action of $G \times G$, that is $(\mathcal{E}(g_\# v), \mathcal{E}(g'_\# v')) = (g_\# \mathcal{E}(v), g'_\# \mathcal{E}(v'))$.

Mapping the weight space to the activation space. The function $F_{\mathcal{V} \to \mathcal{A}} : \mathcal{V}^d \times \mathcal{V}'^d \to \mathcal{A}^d \times \mathcal{A}'^d$ maps the weight spaces $\mathcal{V}^d, \mathcal{V}'^d$ to the corresponding Activation Spaces (see

preliminaries section). There are several ways to implement $F_{\mathcal{V} \to \mathcal{A}}$. As the bias space, $\mathcal{B} = \bigoplus_{m=1}^{M} \mathcal{B}_m$, and the activation space have a natural correspondence between them, perhaps the simplest way, which we use in this paper, is to map a weight space vector $v = (w, b) \in \mathcal{V}^d$ to its bias component $b \in \mathcal{B}^d$. We emphasize that the bias representation is extracted from the previously mentioned weight space encoder, and in that case, it depends on and represents both the weights and the biases of the input. This operation is again equivariant to transposition and the action of $G \times G$, where the action of $G \times G$ on the input space is the more complicated action (by $(g_\#, g'_\#)$) on $\mathcal{V} \times \mathcal{V}$ and the action on the output space is the simpler action of $G \times G$ on the activation spaces.

Generalized outer product. $F_{prod} : \mathcal{A}^d \times \mathcal{A}'^d \to \bigoplus_{m=1}^{M} \mathbb{R}^{d_m \times d_m}$ is a function that takes the activation space features and performs a *generalized outer product* operation as defined below:

$$F_{prod}(a, a')_{m,i,j} = \phi([a_{m,i}, a'_{m,j}])$$

where the subscripts m, i, j represent the (i, j)-th entry of the m-th matrix, and $a_{m,i}, a'_{m,j} \in \mathbb{R}^d$ are the rows of a, a'. Here, the function ϕ is a general (parametric or nonparametric) symmetric function in the sense that $\phi(a, b) = \phi(b, a)$. In this paper, we use $\phi(a, b) = s^2 \langle a / \|a\|_2, b / \|b\|_2 \rangle$ where s is a trainable scalar scaling factor. The equivariance with respect to the action of $G \times G$ and transposition is guaranteed by the fact that ϕ is applied elementwise, and is symmetric, respectively.

Projection layer. The output of F_{prod} is a sequence of matrices $Q_1, \ldots, Q_{M-1}$ which in general will not be permutation matrices. To bring the outputs closer to permutation matrices, F_{proj} implements a approximate projection onto the convex hull of the permutation matrices, i.e., the space of doubly stochastic matrices. In this paper, we use two different projection operations, depending on whether the network is in training or inference mode. At training time, to ensure differentiability, we implement F_{proj} as

Table 1. *CNN image classifiers*: Results on aligning CIFAR10 and STL10 CNN image classifiers.

	CNN (CIFAR10)		CNN (STL10)		VGG11 (CIFAR10)		VGG16 (CIFAR10)	
	Barrier ↓	AUC ↓	Barrier ↓	AUC ↓	Barrier ↓	AUC ↓	Barrier ↓	AUC ↓
Naive	1.12 ± 0.01	0.52 ± 0.00	1.00 ± 0.00	0.65 ± 0.00	1.27 ± 0.04	0.73 ± 0.02	1.13 ± 0.04	0.77 ± 0.03
Weight Matching	0.66 ± 0.02	0.17 ± 0.01	0.85 ± 0.00	0.45 ± 0.00	1.56 ± 0.01	0.75 ± 0.00	2.07 ± 0.10	1.31 ± 0.07
Activation Matching	0.23 ± 0.01	$\mathbf{0.00 \pm 0.00}$	0.47 ± 0.00	0.25 ± 0.00	0.57 ± 0.01	0.20 ± 0.00	2.33 ± 0.04	1.12 ± 0.02
Sinkhorn	0.31 ± 0.01	$\mathbf{0.00 \pm 0.00}$	0.36 ± 0.00	0.16 ± 0.00	0.36 ± 0.00	0.04 ± 0.00	0.92 ± 0.05	0.21 ± 0.02
WM + Sinkhorn	0.24 ± 0.00	$\mathbf{0.00 \pm 0.00}$	0.31 ± 0.00	0.14 ± 0.00	0.31 ± 0.02	0.02 ± 0.01	0.87 ± 0.03	0.20 ± 0.00
DEEP-ALIGN	0.23 ± 0.01	$\mathbf{0.00 \pm 0.00}$	0.38 ± 0.01	0.18 ± 0.00	0.29 ± 0.00	0.05 ± 0.00	0.67 ± 0.01	0.26 ± 0.01
DEEP-ALIGN + Sinkhorn	$\mathbf{0.08 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.23 \pm 0.00}$	$\mathbf{0.09 \pm 0.00}$	$\mathbf{0.09 \pm 0.01}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.31 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$

an approximation of a matrix-wise projection Q_m to the space of doubly stochastic matrices using several iterations of the well-known Sinkhorn projection (Mena et al., 2018; Sinkhorn, 1967). Since the set of doubly stochastic matrices is closed under the action of $G \times G$ on the output space, and under matrix transposition, and since the Sinkhorn iterations are composed of elementwise, row-wise, or column-wise operations, we see that this operation is equivariant as well. At inference time, we obtain permutation matrices from Q_i by finding the permutation matrix P_i which has the highest correlation with Q_i, that is $P_i = \arg\max_{P \in S_{d_i}} \langle Q_i, P \rangle$, where the inner product is the standard Frobenious inner product. This optimization problem, known as the *linear assignment problem*, can be solved using the Hungarian algorithm.

As we carefully designed the components of F so that they are all equivariant to transposition and the action of $G \times G$, we obtain the following proposition:

Proposition 4.1. *The architecture F satisfies the conditions specified in 3.1,3.2, namely for all $(v, v') \in \mathcal{V} \times \mathcal{V}$ and $(g, g') \in H$ we have: $F(g_{\#}v, g'_{\#}v') = g \cdot F(v, v') \cdot g'^T$ and $F(v, v') = F(v', v)^T$.*

4.2. Data generation and Loss functions

Generating labeled data for the weight-alignment problem is hard due to the intractability of the problem. Therefore, we propose a combination of both unsupervised and supervised loss functions where we generate labeled examples synthetically from unlabeled examples, as specified below.

Data generation. Our initial training data consists of a finite set of weight space vectors $D \subset \mathcal{V}$. From that set, we generate two datasets consisting of pairs of weights for the alignment problem. First, we generate a labeled training set, $D_{\text{labeled}} = \{(v^j, v'^j, t^j)\}_{j=1}^{N_{\text{labeled}}}$ for $t^j = (T_1^j, \ldots, T_{M-1}^j) \in G$. This is done by sampling $v^j \in D$ and defining v'^j as a permuted and noisy version of v^j. More formally, we sample a sequence of permutations $t \in G$ and define $v'^j = t_{\#}f_{\text{aug}}(v^j)$, where f_{aug} applies several weight-space augmentations, like adding binary and Gaussian noise, scaling augmentations for ReLU networks, etc. We then set the

label of this pair to be t. In addition, we define an unlabeled dataset $D_{\text{unlabeled}} = \{(v^j, v'^j)\}_{j=1}^{N_{\text{unlabeled}}}$ where $v^j, v'^j \in \mathcal{V}$.

Loss functions. The datasets above are used for training our architecture using the following loss functions. The labeled training examples in D_{labeled} are used by applying a cross-entropy loss for each row $i = 1, \ldots, d_m$ in each output matrix $m = 1, \ldots, M-1$. This loss is denoted as $\ell_{\text{supervised}}(F(v, v'; \theta), t)$. The unlabeled training examples are used in combination with two unsupervised loss functions. The first loss function aims to minimize the alignment loss in equation 2 directly by using the network output $F(v, v'; \theta)$ as the permutation sequence. This loss is denoted as $\ell_{\text{alignment}}(v, v', \theta) = \|v - F(v, v'; \theta)_{\#}v'\|_2^2$. The second unsupervised loss function aims to minimize the original loss function used to train the input networks on a line segment connecting the weights v and the transformed version of v' using the network output $F(v, v'; \theta)$ as the permutation sequence. Concretely, let $\mathcal{L}$ denote the original loss function used to train the weight vectors v, v', the loss is defined as $\ell_{\text{LMC}}(v, v', \theta) = \mathcal{L}(\lambda v + (1 - \lambda)F(v, v'; \theta)_{\#}v')$ for λ sampled uniformly $\lambda \sim U(0, 1)$[1]. This loss is similar to the STE method in (Ainsworth et al., 2022) and the differentiable version in (Peña et al., 2023). Our final goal is to minimize the parameters of F with respect to a linear (positive) combination of $\ell_{\text{alignment}}, \ell_{\text{LMC}}$ and $\ell_{\text{supervised}}$ applied to the appropriate datasets described above.

5. Theoretical analysis

Relation to the activation matching algorithm. In this subsection, we prove that our proposed architecture can simulate the activation matching algorithm, a heuristic for solving the weight alignment problem suggested in (Ainsworth et al., 2022). In a nutshell, this algorithm works by evaluating two neural networks on a set of inputs and finding permutations that align their activations by solving a linear assignment problem using the outer product matrix of the activations as a cost matrix for every layer $m = 1, \ldots, M - 1$.

Proposition 5.1. *(*DEEP-ALIGN *can simulate activation*

---

[1]This loss function satisfies the properties as described in Section 3 when taking expectation over λ.
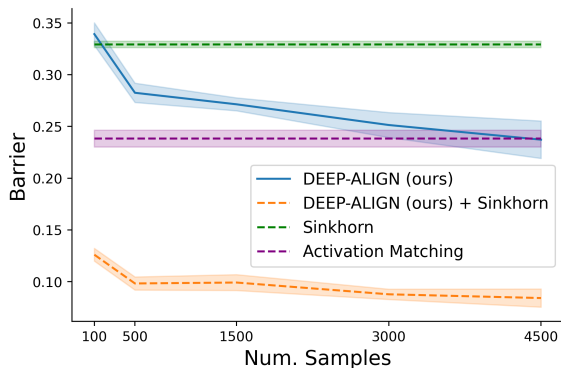
Figure 4. Sample size: The test barrier for aligning CIFAR10 CNN classifiers with a varying number of training examples.

Figure 5. Runtime comparison: DEEP-ALIGN is significantly more efficient at inference compared baseline methods.

matching) *For any compact set $K \subset \mathcal{V}$ and $x_1, \ldots, x_N \in \mathbb{R}^{d_0}$, there exists an instance of our architecture F and weights θ such that for any $v, v' \in K$ for which the activation matching algorithm has a single optimal solution $g \in G$ and another minor assumption specified in the appendix, $F(v, v'; \theta)$ returns g.*

This result offers an interesting interpretation of our architecture: the architecture can simulate activation matching while optimizing the input vectors $x_1, \ldots, x_N$ as a part of their weights θ.

Exactness. We now discuss the *exactness* of our algorithms. An alignment algorithm is said to be exact on some input (v, v') if it can be proven to successfully return the correct minimizer $\mathcal{G}(v, v')$. For NP-hard alignment problems such as weight alignment, exactness can typically be obtained when restricting it to 'tame' inputs (v, v'). Examples of exactness results in the alignment literature can be found in (Aflalo et al., 2015; Dym & Lipman, 2017; Dym, 2018). The following proposition shows that (up to probability zero events) when v, v' are exactly related by some $g \in G$, our algorithm will retrieve g *exactly*:

Proposition 5.2 (DEEP-ALIGN **is exact for perfect alignments**)**.** *Let F denote the* DEEP-ALIGN *architecture with non-constant analytic activations and $d \geq 2$ channels. Then, for Lebesgue almost every $v \in \mathcal{V}$ and parameter vector θ, and for every $g \in G$, we have that $F(v, g_{\#}v, \theta) = g$.*

6. Experiments

In this section, we evaluate DEEP-ALIGN on the task of aligning and merging neural networks. To support future research and the reproducibility of our results, we made our source code and datasets publicly available at: `https://github.com/AvivNavon/deep-align`.
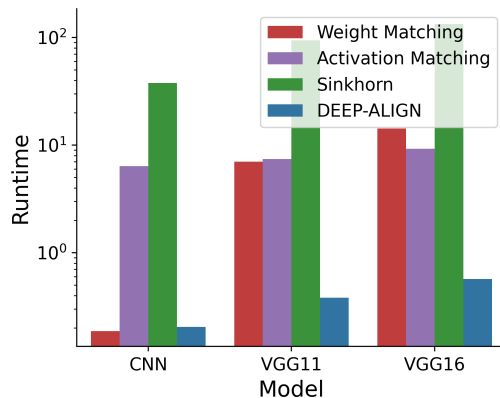
Evaluation metrics. We use the standard evaluation metrics for measuring model merging (Ainsworth et al., 2022; Peña et al., 2023): Barrier and Area Under the Curve (AUC). For two inputs v, v' the Barrier is defined by $\max_{\lambda \in [0,1]} \psi(\lambda) \equiv \mathcal{L}(\lambda v + (1-\lambda)v') - (\lambda \mathcal{L}(v) + (1-\lambda)\mathcal{L}(v'))$ where $\mathcal{L}$ denote the loss function on the original task. Similarly, the AUC is defined as the integral of ψ over $[0, 1]$. Lower is better for both metrics. Following previous works (Ainsworth et al., 2022; Peña et al., 2023), we bound both metrics by taking the maximum between their value and zero.

Compared methods. We compare the following approaches: (1) *Naive*: where two models are merged by averaging the models' weights without alignment. The (2) *Weight matching* and (3) *Activation matching* approaches proposed in (Ainsworth et al., 2022). (4) *Sinkhorn* (Peña et al., 2023): This approach directly optimizes the permutation matrices using the task loss on the line segment between the aligned models (denoted $\mathcal{C}_{Rnd}$ in (Peña et al., 2023)). (5) *WM + Sinkhorn*: using the weight matching solution to initialize the Sinkhorn method. (6) DEEP-ALIGN: Our proposed method described in Section 4. (7) DEEP-ALIGN + *Sinkhorn*: Here, the output from the DEEP-ALIGN is used as an initialization for the *Sinkhorn* method.

Experimental details. Our method is first trained on a dataset of weight vectors and then applied to unseen weight vectors at test time, as is standard in learning setups. In contrast, baseline methods are directly optimized using the test networks. For the *Sinkhorn* and DEEP-ALIGN + *Sinkhorn* methods, we optimize the permutations for 1000 iterations. For the *Activation Matching* method, we calculate the activations using the entire train dataset. We repeat all experiments using 3 random seeds and report each metric's mean and standard deviation. For full experimental details see Appendix E.
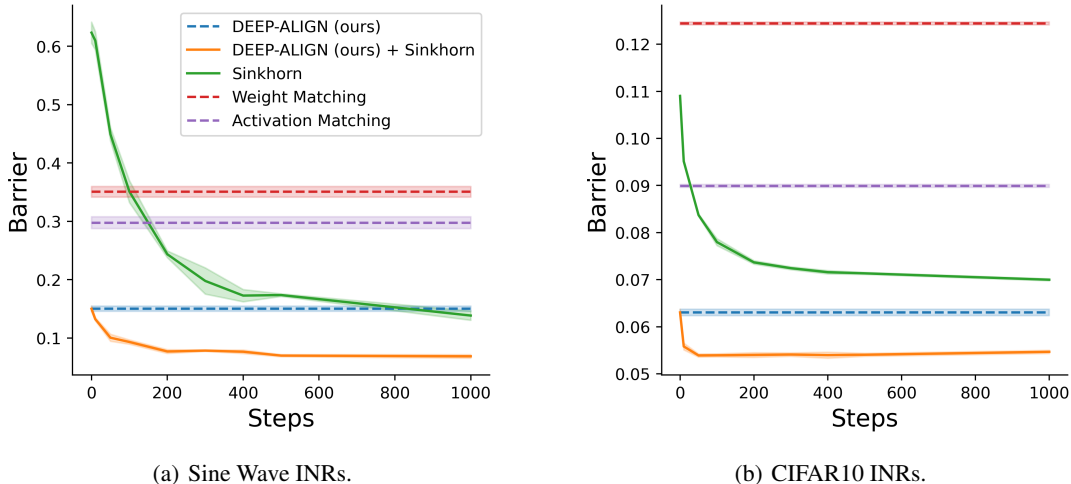
(a) Sine Wave INRs. (b) CIFAR10 INRs.

Figure 6. *Aligning INRs*: The test barrier vs. the number of Sinkhorn iterations (relevant only for *Sinkhorn* or DEEP-ALIGN + *Sinkhorn*), using (a) sine wave and (b) CIFAR10 INRs. DEEP-ALIGN outperforms baseline methods or achieves on-par results.

6.1. Results

Aligning classifiers. Here, we evaluate our method on the task of aligning image classifiers. We use six network datasets. Two datasets consist of MLP classifiers for MNIST and CIFAR10, and four datasets consist of CNN classifiers trained using CIFAR10 and STL10. This collection forms a diverse benchmark for aligning NN classifiers. The results are presented in Figure 3, Table 1 and Table 5. The alignment produced through a feed-forward pass with DEEP-ALIGN performs on par or outperforms all baseline methods. Initializing the Sinkhorn algorithm with our alignment (DEEP-ALIGN + Sinkhorn) further improves the results, and significantly outperforms all other methods. For the CNN alignment experiments, we report the averaged alignment time using 1K random pairs (Figure 5), and show that DEEP-ALIGN is significantly more efficient compared to all baselines when aligning large networks. Importantly, we fix the number of training examples for all convolution network architectures (CNN, VGG11, and VGG16). Our results demonstrate that DEEP-ALIGN scales well to large networks without the need to increase the dataset size. Furthermore, in Figure 4 we provide barrier results for training our method with a varying number of training networks. DEEP-ALIGN produces alignments with on-par quality to the Sinkhorn method, with only ~ 100 training samples.

Aligning INRs. We use two datasets consisting of implicit neural representations (INRs). The first consists of Sine waves INRs of the form $f(x) = \sin(ax)$ on $[-\pi, \pi]$, where $a \sim U(0.5, 10)$, similarly to the data used in (Navon et al., 2023). We fit two views (independently trained weight vectors) for each value of a starting from different random initializations. The task is to align and merge the two INRs.

We train our network to align pairs of corresponding views.

The second dataset consists of INRs fitted to CIFAR10 images. We fit five views per image. The results are presented in Figure 6. DEEP-ALIGN, performs on par or outperforms all baseline methods. Moreover, using the output from the DEEP-ALIGN to initialize the Sinkhorn algorithm further improves this result, with a large improvement over the Sinkhorn baseline with random initialization.

Aligning networks for Federated Learning. DEEP-ALIGN allows to perform multiple alignments efficiently at inference. Federated learning (FL, McMahan et al. (2017)) is one setup in which the need for numerous alignments arises. In FL, the goal is to construct a unified model from multiple networks trained on separate and distinct datasets. One of the most frequently used approach for FL is FedAvg (McMahan et al., 2017), where the local models are averaged in weight space to form a joint global model. Here, we evaluate a version of FedAvg in which we align the local models before averaging them. This approach was shown beneficial in recent works (Wang et al., 2020a). We use the CIFAR10 and STL10 datasets with varying federation sizes (number of clients). To simulate a realistic scenario, we employ a pretrained DEEP-ALIGN network trained using an OOD dataset, e.g., we train DEEP-ALIGN on STL10 for the CIFAR10 experiment and vice versa. Importantly, Sinkhorn and activation matching methods are inapplicable in this setup since no data is available on the server on which the model averaging is performed. Moreover, even if some data is available, the runtime of these methods makes them impractical for the FL setup (Figure 5). However, the Sinkhorn-L_2 method with L_2-loss on the relaxed alignment objective of Eq. 2, can be applied in this setup. The results

Table 2. Federated learning using the CIFAR10 and STL10 dataset, with a varying number of clients.

	CIFAR10			STL10		
	50	100	200	10	25	50
Naive (FedAvg)	67.87 ± 2.71	63.15 ± 2.08	59.92 ± 1.66	46.82 ± 2.84	47.08 ± 0.51	43.51 ± 1.82
Weight Matching	68.05 ± 2.35	63.38 ± 2.59	59.23 ± 1.29	47.73 ± 1.75	46.87 ± 0.73	44.44 ± 1.02
Sinkhorn-L_2	67.80 ± 3.49	63.11 ± 2.11	60.22 ± 1.92	48.01 ± 0.72	46.10 ± 0.84	45.33 ± 0.95
DEEP-ALIGN	$\mathbf{69.86 \pm 1.13}$	$\mathbf{66.31 \pm 1.19}$	$\mathbf{65.52 \pm 1.78}$	$\mathbf{49.48 \pm 2.23}$	$\mathbf{48.22 \pm 1.08}$	$\mathbf{48.14 \pm 0.88}$

presented in Table 2 show DEEP-ALIGN outperforms the Naive (i.e., FedAvg) and weight-matching baselines. Additionally, we provide results for merging models trained using disjoint data splits in Appendix F.

7. Conclusion

Limitations. One limitation of our approach is the need for pretraining a network. We show, however, that this process requires only a small number of networks and does not require any labeled examples. Another limitation is that the current architecture is specific for a given input network architecture. This limitation relates to the specific weight-space encoder we utilize in this work, DWSNet. This could be mitigated by modifying the DWS encoder (see Section F), or by replacing it with more recent GNN-based weight space encoders like Zhang et al. (2023); Lim et al. (2023). Such approaches show promising results for training and generalizing over diverse network types.

Summary. We investigate the challenging problem of weight alignment in deep neural networks. The key to our approach, DEEP-ALIGN, is an equivariant architecture that respects the natural symmetries of the problem. DEEP-ALIGN is the first architecture designed for weight alignment. At inference time, our method aligns unseen network pairs without the need to perform expensive optimization. DEEP-ALIGN, performs on par or outperforms optimization-based approaches while significantly reducing the runtime or improving the quality of the alignments. Furthermore, we demonstrate that the alignments of our method can be used to initialize optimization-based approaches.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

References

Aflalo, Y., Bronstein, A., and Kimmel, R. On convex relaxation of graph isomorphism. *Proceedings of the National Academy of Sciences*, 112(10):2942–2947, 2015.

Ainsworth, S. K., Hayase, J., and Srinivasa, S. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

Akash, A. K., Li, S., and Trillos, N. G. Wasserstein barycenter-based model fusion and linear mode connectivity of neural networks. *arXiv preprint arXiv:2210.06671*, 2022.

Andreis, B., Bedionita, S., and Hwang, S. J. Set-based neural network encoding. *arXiv preprint arXiv:2305.16625*, 2023.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Velickovic, P. Combinatorial optimization and reasoning with graph neural networks. *CoRR*, abs/2102.09544, 2021.

Cohen, T. et al. *Equivariant convolutional networks*. PhD thesis, Taco Cohen, 2021.

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. Essentially no barriers in neural network energy landscape. In *International conference on machine learning*, pp. 1309–1318. PMLR, 2018.

Dym, N. Exact recovery with symmetries for the doubly stochastic relaxation. *SIAM Journal on Applied Algebra and Geometry*, 2(3):462–488, 2018.

Dym, N. and Lipman, Y. Exact recovery with symmetries for procrustes matching. *SIAM Journal on Optimization*, 27(3):1513–1530, 2017.

Eilertsen, G., Jönsson, D., Ropinski, T., Unger, J., and Ynnerman, A. Classifying the classifier: dissecting the weight space of neural networks. *arXiv preprint arXiv:2002.05688*, 2020.

Entezari, R., Sedghi, H., Saukh, O., and Neyshabur, B. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations*, 2022.

Fey, M., Lenssen, J. E., Morris, C., Masci, J., and Kriege, N. M. Deep graph matching consensus. *arXiv preprint arXiv:2001.09621*, 2020.

Freeman, C. D. and Bruna, J. Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*, 2016.

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., and Wilson, A. G. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018.

Hartford, J., Graham, D., Leyton-Brown, K., and Ravan-bakhsh, S. Deep models of interactions across sets. In *International Conference on Machine Learning*, pp. 1909–1918. PMLR, 2018.

Ilharco, G., Ribeiro, M. T., Wortsman, M., Gururangan, S., Schmidt, L., Hajishirzi, H., and Farhadi, A. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Jordan, K., Sedghi, H., Saukh, O., Entezari, R., and Neyshabur, B. Repair: Renormalizing permuted activations for interpolation repair. *arXiv preprint arXiv:2211.08403*, 2022.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Kondor, R. and Trivedi, S. On the generalization of equivari-ance and convolution in neural networks to the action of compact groups. In *International Conference on Machine Learning*, pp. 2747–2755. PMLR, 2018.

Lim, D., Robinson, J., Zhao, L., Smidt, T., Sra, S., Maron, H., and Jegelka, S. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.

Lim, D., Maron, H., Law, M. T., Lorraine, J., and Lucas, J. Graph metanetworks for processing diverse neural architectures. *arXiv preprint arXiv:2312.04501*, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regu-larization. *arXiv preprint arXiv:1711.05101*, 2017.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

Maron, H., Litany, O., Chechik, G., and Fetaya, E. On learning sets of symmetric elements. In *International conference on machine learning*, pp. 6734–6744. PMLR, 2020.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelli-gence and statistics*, pp. 1273–1282. PMLR, 2017.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learn-ing latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

Mityagin, B. The zero set of a real analytic function. *arXiv preprint arXiv:1512.07276*, 2015.

Navon, A., Shamsian, A., Achituve, I., Fetaya, E., Chechik, G., and Maron, H. Equivariant architectures for learning in deep weight spaces. *arXiv preprint arXiv:2301.12780*, 2023.

Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J. A note on learning algorithms for quadratic assignment with graph neural networks. *stat*, 1050:22, 2017.

Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J. Revised note on learning quadratic assignment with graph neural networks. In *2018 IEEE Data Science Workshop (DSW)*, pp. 1–5. IEEE, 2018.

Peña, F. A. G., Medeiros, H. R., Dubail, T., Aminbeidokhti, M., Granger, E., and Pedersoli, M. Re-basin via implicit sinkhorn differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20237–20246, 2023.

Ravanbakhsh, S., Schneider, J., and Poczos, B. Equivariance through parameter-sharing. In *International conference on machine learning*, pp. 2892–2901. PMLR, 2017.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

Shamsian, A., Zhang, D. W., Navon, A., Zhang, Y., Kofinas, M., Achituve, I., Valperga, R., Burghouts, G. J., Gavves, E., Snoek, C. G., et al. Data augmentations in deep weight spaces. *arXiv preprint arXiv:2311.08851*, 2023.

Sinkhorn, R. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

Stoica, G., Bolya, D., Bjorner, J., Hearn, T., and Hoffman, J. Zipit! merging models from different tasks without training. *arXiv preprint arXiv:2305.03053*, 2023.

Tatro, N., Chen, P.-Y., Das, P., Melnyk, I., Sattigeri, P., and Lai, R. Optimizing mode connectivity via neuron alignment. *Advances in Neural Information Processing Systems*, 33:15300–15311, 2020.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., and Tolstikhin, I. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

Vesselinova, N., Steinert, R., Perez-Ramirez, D. F., and Boman, M. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8:120388–120416, 2020.

Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., and Khazaeni, Y. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*, 2020a.

Wang, R., Albooyeh, M., and Ravanbakhsh, S. Equivariant maps for hierarchical structures. *arXiv preprint arXiv:2006.03627*, 2020b.

Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pp. 23965–23998. PMLR, 2022.

Yan, J., Yang, S., and Hancock, E. R. Learning for graph matching and related combinatorial optimization problems. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 4988–4996. International Joint Conferences on Artificial Intelligence Organization, 2020.

Yu, T., Wang, R., Yan, J., and Li, B. Learning deep graph matching with channel-independent embedding and hungarian attention. In *International conference on learning representations*, 2019.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zanfir, A. and Sminchisescu, C. Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2684–2693, 2018.

Zhang, D. W., Kofinas, M., Zhang, Y., Chen, Y., Burghouts, G. J., and Snoek, C. G. Neural networks are graphs! graph neural networks for equivariant processing of neural networks. 2023.

Zhou, A., Yang, K., Burns, K., Jiang, Y., Sokota, S., Kolter, J. Z., and Finn, C. Permutation equivariant neural functionals. *arXiv preprint arXiv:2302.14040*, 2023a.

Zhou, A., Yang, K., Jiang, Y., Burns, K., Xu, W., Sokota, S., Kolter, J. Z., and Finn, C. Neural functional transformers. *arXiv preprint arXiv:2305.13546*, 2023b.

A. More previous work

Weight space alignment. Several algorithms have been proposed for weight-alignment (Tatro et al., 2020; Ainsworth et al., 2022; Peña et al., 2023; Akash et al., 2022). Ainsworth et al. (2022) presented three algorithms: Activation Matching, Weight Matching, and straight-through estimation. Peña et al. (2023) improved upon these algorithms by incorporating a Sinkhorn-based projection method. In part, these works were motivated by studying the loss landscapes of deep neural networks. It was conjectured that deep networks exhibit a property called *linear mode connectivity*: for any two trained weight vectors, a linear interpolation between the first vector and the optimal alignment of the second, yields very small increases in the loss (Entezari et al., 2022; Garipov et al., 2018; Draxler et al., 2018; Freeman & Bruna, 2016; Tatro et al., 2020; Jordan et al., 2022).

Weight-space networks. A growing area of research focuses on applying neural networks to neural network weights. Early methods proposed using simple architectures such as MLPs or transformers to predict test errors or the hyperparameters that were used for training input networks (Unterthiner et al., 2020; Andreis et al., 2023; Eilertsen et al., 2020).

Recently, (Navon et al., 2023) presented the first neural architecture that is equivariant to the natural permutation symmetries of (MLP) weight spaces and demonstrated significant performance improvements over previous approaches. This architecture, called Deep Weight Space Networks (DWSNets), is composed of multiple linear G-equivariant layers, which were characterized in (Navon et al., 2023), interleaved with pointwise nonlinearities, such as ReLU functions. In other words, a DWSNets is a function $F : \mathcal{V} \to \mathcal{V}$ of the following form:

$$F = L_k \circ \sigma \circ L_{k-1} \circ \sigma \cdots \circ \sigma \circ L_1,$$

where L_i are linear G-equivariant layers and $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear function applied elementwise. This is similar to the design of many previous equivariant architectures (Zaheer et al., 2017; Maron et al., 2018). Each such layer $L : \mathcal{V} \to \mathcal{V}$ takes as input representations of all the weights and biases which we denote as $v \in \mathcal{V}$, $v = [W_m, b_m]_{m \in [M]}$ and outputs new representations based on all the input weights and biases. As common in deep learning architectures, these layers can also handle d dimensional features for each weight and bias, i.e. vectors $v \in \mathcal{V}^d$. After a composition of several such layers, the output weights can be used for the task of interest, or pooled to form a single representation for the input weight space vector. In our case, we use the bias representations produced by the final layer L_k, which are essentially representations of the activation space $\mathcal{A}$.

(Zhou et al., 2023a) proposed a similar approach and an extension to CNN architectures , which was later enhanced by the addition of attention mechanisms (Zhou et al., 2023b). Finally, (Zhang et al., 2023; Lim et al., 2023) proposes modeling the neural networks as computational graphs and applying Graph Neural Networks to them, demonstrating very good results on several weight space learning tasks.

Learning for combinatorial optimization. There exists a large body of research on learning to solve hard combinatorial optimization problems such as TSP and SAT (Cappart et al., 2021; Bengio et al., 2021; Khalil et al., 2017; Vesselinova et al., 2020; Selsam et al., 2018). The key observation behind these works, also shared by the current work, is that even though those problems are computationally intractable in general, there may be efficient methods to solve them for specific problem distributions. In these cases, machine learning can be used to learn to predict these solutions. Specifically relevant to this work are works that suggested learning to solve the graph matching problem (Fey et al., 2020; Zanfir & Sminchisescu, 2018; Yan et al., 2020; Yu et al., 2019; Nowak et al., 2017; 2018) which is a similar alignment problem.

B. Proofs for section 3

Proof of Proposition 3.1. We write $\mathcal{G}(v, v') = \operatorname{argmin}_{k \in G} E(k, v, v')$ with $E(k, v, v') = \|v - k_\# v'\|_2^2$. First, we note that the minimal value of the optimization problem $\mathcal{G}(gv, gv')$ is equal to the minimal value of $\mathcal{G}(v, v')$, namely,

$$\min_{k \in G} E(k, v, v') = \min_{k \in G} E(k, g_\# v, g'_\# v').$$

This is true since

$$\min_{k \in G} E(k, g_\# v, g'_\# v') = \min_{k \in G} \|g_\# v - k_\# g'_\# v'\|_2^2 = \min_{k \in G} \|v - (g^{-1}kg')_\# v'\|_2^2 = \min_{k \in G} \|k_\# v - v'\|_2^2 =$$

and using the fact that $k \mapsto g^{-1}kg'$ is bijective.

Second, we show that if $k^* = \mathcal{G}(v, v')$, or in other words, is a minimizer of $E(k, v, v')$, then $g \cdot k^* \cdot g'^T$ minimizes $E(k, gv, g'v')$ because:

$$E(g \cdot k^* \cdot g'^T, gv, g'v') = \|g_\# v - (g \cdot k^* \cdot g'^T)_\# g'_\# v'\|_2^2 = \|v - (g^T \cdot g \cdot k^* \cdot g'^T \cdot g')_\# v'\|^2 = E(k^*, v, v')$$

$\square$

Proof of Proposition 3.2. Similarly to the proof of the previous proposition, we have $\min_{k \in G} E(k, v', v) = \min_{k \in G} \|v' - k_\# v\|_2^2 = \min_{k \in G} \|k_\#^T v' - v\|_2^2 = \min_{k \in G} E(k, v, v')$. Additionally, plugging in $k = \mathcal{G}(v, v')^T$ achieves this value:

$$E(\mathcal{G}(v, v')^T, v', v) = \|v' - [\mathcal{G}(v, v')^T]_\# v\|_2^2 = \|[\mathcal{G}(v, v')]_\# v' - v\|_2^2 = E(\mathcal{G}(v, v'), v, v')$$

$\square$

Proof of generalization to other objectives. We prove the generalization mentioned in the main text. Namely, that the symmetries of the function $\mathcal{G}$, are shared by any function of the form $\mathcal{G}(v, v') = \operatorname{argmin}_{k \in G} E(v, k_\# v')$ providing that the energy function E satisfies

$$E(gv, gv') = E(v, v') \text{ and } E(v, v') = E(v', v), \quad \forall g \in G, v, v' \in \mathcal{V}.$$

First, we note that the minimal value of the optimization problem $\mathcal{G}(gv, gv')$ is equal to the minimal value of $\mathcal{G}(v, v')$, namely,

$$\min_{k \in G} E(v, k_\# v') = \min_{k \in G} E(g_\# v, k_\# g'_\# v').$$

This is true since

$$\min_{k \in G} E(g_\# v, k_\# g'_\# v') = \min_{k \in G} E(v, g_\#^T k_\# g'_\# v') = \min_{k \in G} E(v, (g^{-1} k g')_\# v')$$

using the fact that $k \mapsto g^{-1} k g'$ is bijective and the G-invariance of E.

Next, we show that if $k^* = \mathcal{G}(v, v')$, or in other words, is a minimizer of $E(v, k_\# v')$, then $g \cdot k^* \cdot g'^T$ minimizes $E(g_\# v, k_\# g'_\# v')$ This is because: $E(g_\# v, (g \cdot k^* \cdot g'^T)_\# g'_\# v') = E(v, k^*_\# v')$ using the G-invariance of E.

We turn to proving a generalization of 3.2: similarly to the previous argument, we have

$$\min_{k \in G} E(v', k_\# v) = \min_{k \in G} E(k_\#^T v', v) = \min_{k \in G} E(k_\#^T v', v) = \min_{k \in G} E(v, k_\#^T v')$$
$$= \min_{k \in G} E(v, k_\# v')$$

where we used the fact that we can swap the inputs to E and the invariance.

Additionally, plugging in $k = \mathcal{G}(v, v')^T$ achieves this value:

$$E(v', \mathcal{G}(v, v')_\#^T v) = E(\mathcal{G}(v, v')_\# v', v) = E(v, \mathcal{G}(v, v')_\# v')$$

$\square$

C. Proofs for section 5

Relation to activation matching. Here we prove Proposition 5.1. Let the outer product matrix $Z \in \mathbb{R}^{d_m \times d_m}$ be a cost matrix for the activation matching algorithm. We say that Z is ϵ-*friendly* if it has the following property: there exists some i such that for all $j \neq i \langle Z, P_i \rangle > \langle Z, P_j \rangle + \epsilon$ where $P_i, P_j \in \Pi_{d_m}$. Intuitively this condition means that the cost matrix is bounded away from the set of cost matrices for which the are multiple optimal solutions [2]. Let us now state Proposition 5.1 with full details:

---

[2]Formally, one can prove that under any continuous distribution and for any finite number of cost matrices sampled from this distribution, with probability 1, we can find an ϵ such that all the cost matrices are ϵ-friendly.

Proposition C.1 (Full formulation of Proposition 5.1)**.** *For any compact set $K \subset \mathcal{V}$, $x_1, \ldots, x_N \in \mathbb{R}^{d_0}$ and for any $\epsilon > 0$, there exists an instance of our architecture F and weights θ for that architecture such that for any $v, v' \in K$ for which all the cost matrices used by the activation matching algorithm are ϵ-friendly, we have that $F(v, v'; \theta)$ returns exactly the same solution as the activation matching algorithm.*

Proof of Proposition C.1. First, to obtain weights for the DWS network that approximate the activations of both networks on the set of inputs $x_1, \ldots, x_n$, we employ Lemma G.2 from (Navon et al., 2023) (stated below for convenience). We note that this lemma is proved by the approximation of all intermediate activations of the input networks so the proof of this lemma trivially shows that there is a DWSnet that outputs an approximation of all the activations.

Then, we set F_{prod} to calculate outer products in order to generate the outer-product-based cost matrix used in (Ainsworth et al., 2022). It follows that the composition of the DWS network above with the outer product function approximates uniformly their limits, see (Lim et al., 2022) (Lemma 6), and the composition of their limits is exactly the function that takes two weight vectors and computes the cost matrices from (Ainsworth et al., 2022). We have developed an architecture and weights for this architecture that can uniformly approximate the cost matrices used by the activation matching algorithm to any precision. As a final step, we apply our linear assignment projection.

To show that this architecture will return exactly the same solution g of the activation matching algorithm, we use our ϵ-friendly assumption and Lemma C.3 which imply that there is an open ball of radius δ around each cost matrix in which the linear assignment problem is *constant*. We can now use the uniform approximation result from the previous paragraph to approximate all the cost matrices up to δ and get that applying the linear assignment problem to the approximated cost matrices yields the same output as the original cost matrices. $\qquad\square$

Lemma C.2. *[Lemma G.2 in (Navon et al., 2023)] Let $\epsilon > 0$. For any $x_1, \ldots, x_N \in \mathbb{R}^{d_0}$ there exist a DWSNet D_ϵ and weights θ_ϵ such that for any $v \in K$ we have $\|D(v; \theta_\epsilon)_i - f_v(x_i)\|_2 < \epsilon$.*

We note that (Navon et al., 2023) restricted $x_1, \ldots, x_N$ to some compact set but this assumption is not used in their proof.

Lemma C.3. *Let $\epsilon > 0$, $K \subset \mathbb{R}^n$ a compact domain and let $f_i : K \to \mathbb{R}$, $i = 1, \ldots, m$ such that $f_i(x)$ are continuous. Let*

$$S = \{x \in K \mid \exists i \ s.t \ \forall j \neq i \ f_i(x) > f_j(x) + \epsilon\},$$

define $g(x) = argmax_{j=1}^m f_j(x)$, then there exists some $\delta > 0$ such that for any $x \in S$, $g(x)$ is constant in an open ball of radius δ around x.

Proof. Since $f_1, \ldots, f_n$ are finite and continuous on a compact domain, there exists some $\delta > 0$ such that for any $x, y \in K$, $\|x - y\| < \delta$ implies $|f_i(x) - f_i(y)| < \frac{\epsilon}{2}$ for every i. Then for any $x \in S$ such that $\forall j \neq i \ f_i(x) > f_j(x) + \epsilon$, if $\|x - y\| < \delta$, for every $j \neq i$ we have

$$f_i(y) - f_j(y) = (f_i(y) - f_i(x)) + (f_i(x) - f_j(x)) + (f_j(x) - f_j(y)) > 0,$$

which implies that $g(y) = i$. $\qquad\square$

Proof of Proposition 5.2. As mentioned in the statement of the proposition, we assume that F uses analytic non-constant activation functions, and $d \geq 2$ channels. In the proof we consider our standard choice of ϕ as the normalized inner product and set the scaling parameter s to be $s = 1$ for simplicity. We consider the version of F where the last projection layer F_{proj} computes the permutations maximizing the correlation with the outputs $Q_1, \ldots, Q_M$ of the product layer (that is, we consider the version of F used in test time, rather than the version used in training which uses differentiable Sinkhorn iterations). By equivariance of the model, it is sufficient to show that for almost every v, θ the identity matrices are the closest permutations to $(Q_1, \ldots, Q_{M-1})$.

Next, recall that the indices of each $Q_m = Q_m(v, \theta)$ are given by

$$Q_{m,i,j} = \frac{\langle a_{m,i}, a_{m,j} \rangle}{\|a_{m,j}\| \|a_{m,j}\|}$$

where $a_{m,i}$ is a d dimensional vector. In particular, we have that $|Q_{m,i,j}| \leq 1$, and the equality holds if $i = j$. It remains to show that when $i \neq j$ we will have a strict inequality $|Q_{m,i,j}| < 1$ for all m, for almost every v and θ. Equivalently, we will need to show that for all $m = 1, \ldots, M$ and all $i \neq j$, the functions

$$\phi_{m,i}(v, \theta) = \|a_{m,i}\|^2$$

and

$$\psi_{m,i,j}(v, \theta) = \|a_{m,j}\|^2 + \|a_{m,i}\|^2 - \langle a_{m,i}, a_{m,j} \rangle$$

are non-zero for almost all (v, θ). We note that ϕ and ψ are analytic, and the zero set of a non-zero analytic function always has Lebesgue measure zero (see (Mityagin, 2015)). Therefore it is sufficient to show that there exists a single (v, θ) for which $\phi_{m,i}(v, \theta) \neq 0$, and a single (v, θ) for which $\psi_{m,i,j}(v, \theta) \neq 0$.

Let us consider parameter vectors θ as follows: recall that the output of each layer is a sequence of hidden weight matrices $W_m = W_{m,a,b,c}$ and a sequence of hidden bias vectors $b_m = b_{m,i,c}$, where the last index c runs over the channels. We choose θ so that each affine layer will map all matrices W_m to zero, and all bias vectors b_m to a new value b'_m, where $b'_{m,i,1} = b_{m,i,1}$ and $b'_{m,i,c} = 1$ if $c \neq 1$. Since this describes an affine equivariant mapping, and the linear layers in DWS can express all linear equivariant function, the vector θ can indeed be defined to give this function.

With this choice of θ, we will obtain

$$a_{m,i} = \left[\rho^D(b_{m,i}), 1_{d-1} \right] \in \mathbb{R}^d$$

where ρ denotes the activation used in the DWS network, D denotes the depth of the DWS network, and $b_{m,i}$ is the i-th entry of the m-th input bias vector. In particular, it is an entry of v. To conclude the proof it is sufficient to show that we can choose a pair of b_i, b_j such that $\rho^D(b_{m,i}) \neq \rho^D(b_{m,j})$. If this is indeed the case we can immediately deduce that $\psi_{m,i,j}(v, \theta) \neq 0$.

To prove the latter point all we need is to show that ρ^D is not a constant function. Indeed, since ρ itself is non-constant its image contains an interval I, and by analyticity ρ cannot be constant on this interval, so ρ^2 is non-constant. Continuing recursively in this way we can show that ρ^D will not be constant for any D. $\qquad \square$

D. Extending DWSNets to CNNs

In this work, we employed a DWSNet (Navon et al., 2023) as our F_{DWS} block. Since the study in (Navon et al., 2023) focused on MLPs, the original implementation only supports input MLP networks. Here, we provide technical details on the extension to input CNN networks under the DWSNets framework. This requires only two simple adjustments. First, the kernel dimensions are flattened into the feature dimension. Second, the first FC layer (after the last convolution layer) in the input network generally requires special attention. Specifically, denoting the output dimension of the last CNN with d_0 and the dimensions of the first FC layer weight matrix by $d_1 \times d_2$, we reshape the weight matrix to $d_0 \times d_2 \times (d_1/d_0)$, i.e., folding the d_1/d_0 into the feature dimension. This preserves the equivariance to the permutation symmetries of the input network.

E. Experimental Details

In all experiments, we use a 4-hidden layer DEEP-ALIGN network with a hidden dimension of 64 and an output dimension of 128 from the F_{DWS} block. We optimize our method with a learning rate of $5e - 4$ using the AdamW (Loshchilov & Hutter, 2017) optimizer. For all experiments with image classifiers, we train DEEP-ALIGN with all objectives as described in Section 4 ($\ell_{\text{alignment}}, \ell_{\text{LMC}}$ and $\ell_{\text{supervised}}$). For the INR experiments, we drop the $\ell_{\text{alignment}}$ loss since we found adding this loss significantly hurt the performance (see Appendix F).

We use the entire dataset to estimate the activations for the Activation Matching (AM) baseline. For the Sinkhorn baseline, we optimize the permutations using learning rate $1e - 1$ and for 1000 iterations.

When using image datasets, we use the standard train-test split and allocate 10% of the training data for validation.

MLP classifiers. For this experiment, we generate two wight datasets, consisting of MNIST and CIFAR10 classifiers. Each classifier is a 3-hidden layer MLP with a hidden dimension of 128. The input dimension is 784 for MNIST and 3072 for CIFAR10. We train the classifiers for 5 epochs with a batch size of 128 and learning rate $5e - 3$. Both datasets consist of 10000 networks, split into 8000 for training and 1000 each for validation and testing.
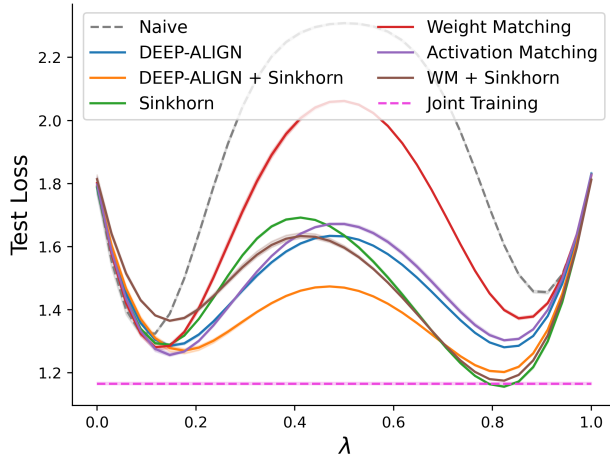
Figure 7. Merging networks trained on distinct subsets of CIFAR10 with different class distributions.

We train DEEP-ALIGN for 25K iterations. Since the F_{DWS} block can grow large when the input dimension to the input network is large, we employ the method proposed in (Navon et al., 2023) to control the number of parameters. Thus, we linearly map the input dimension 784 or 3072 (MNIST or CIFAR10) to 8. See (Navon et al., 2023) for details.

CNN classifiers. For this experiment, we generate four datasets by training classifiers on CIFAR10 and STL10 datasets. We generate two datasets of VGG11 (9M parameters) and VGG16 (15M parameters) networks, trained on CIFAR10. Each dataset consists of 4500 training examples, 100 networks for validation, and 100 for testing.

Table 3. CNN classifiers architecture.

CNN Classifiers Arch.
3x3 Conv 16
3x3 Conv 32
3x3 Conv 32
2x2 MaxPool
3x3 Conv 64
3x3 Conv 64
2x2 MaxPool
3x3 Conv 128
3x3 Conv 128
2x2 MaxPool
Linear (2048, 10)

The CNN datasets consist of CNN networks with 7 convolution layers followed by a fully-connected layer, with a total of 300K parameters. The full architecture is presented in Table 3. For STL10, we apply a sequence of 3 augmentations, first, we random crop 64×64 path, next we resize the patch to 32×32, and finally we apply random rotation drawn from $U(-20, 20)$. We train the CIFAR10 and STL10 classifiers for 20/100 epochs respectively using Adam optimizer with $1e-4$ learning rate. We save the model's checkpoint at the final epoch. Both datasets consist of 5000 networks, split into 4500 for training and 250 each for validation and testing. We train DEEP-ALIGN for 300 epochs.

Sine INRs. To generate the Sine wave dataset, we use the same procedure as in (Navon et al., 2023). Each INR is an MLP with 3 layers, a hidden dimension of 32, and Sine activations. The dataset consists of 2000 sine waves and two INR copies (views) for each sine wave. We use 1800 waves for training, and 100 for validation, and testing.

CIFAR10 INRs. The CIFAR10 dataset consists of $60K$ images. We split the dataset to train, validation, and test with $45K$ / $5K$ / $10K$ samples respectively. For each image, we create 5 independent INR copies. Each INR is a 5-layer MLP with a 32 hidden dimension each followed by sine activations. We optimize the INRs using Adam optimizer with a $1e - 4$ learning rate for $10K$ update steps. We train DEEP-ALIGN for 300 epochs.

Federated Learning. We use two datasets, mainly CIFAR10 and STL10. For CIFAR10, we vary the number of clients from 50 to 200, and use a DEEP-ALIGN network trained on STL10 classifiers. For STL10, we vary the number of clients from 10 to 50, and use a DEEP-ALIGN network trained on CIFAR10 classifiers. For all experiments, we train the joint network for 1000 rounds. We randomly select 5 clients at each round and train the global model for 50 local optimization steps. We then send the local models to the hub for (alignment and) averaging.

Disjoint datasets. We use the same CNN network configuration as in the CNN classifiers experiments, and train 2500 networks for each split (5000 in total). Each network is trained for 20 epochs using the Adam optimizer with learning rate $1e - 4$. We allocate 100 networks from each split for testing and validation, and the remaining 4600 networks for training. We train DEEP-ALIGN for 50K steps.
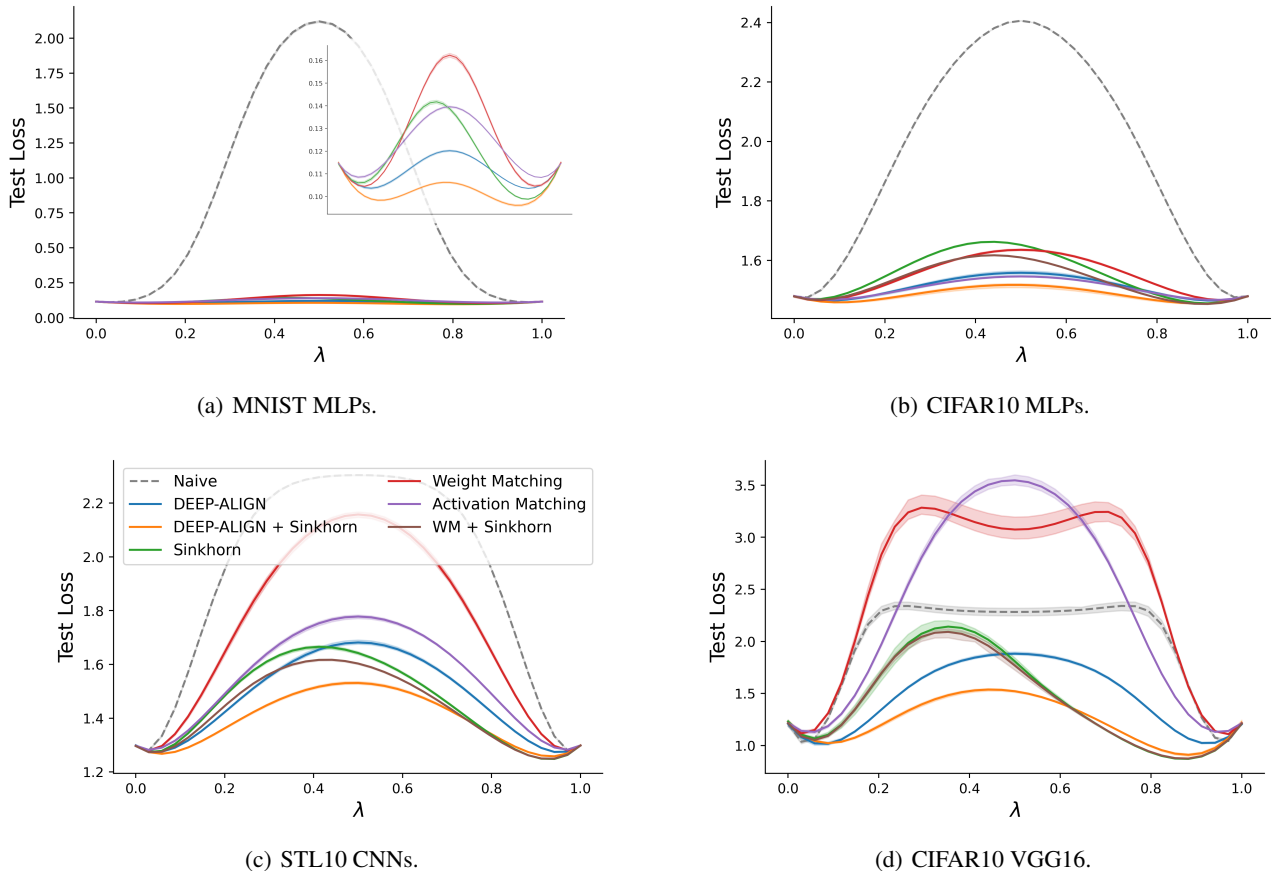
(a) MNIST MLPs.

(b) CIFAR10 MLPs.

(c) STL10 CNNs.

(d) CIFAR10 VGG16.

Figure 8. Additional results for aligning image classifiers.

Time comparison. Prior methods for weight matching, which rely on optimization, often suffer from exhaustive runtime, which may be impractical for real-time applications. In contrast, once trained, DEEP-ALIGN is able to produce high-quality weight alignments through a single forward pass and an efficient projection step. We compare DEEP-ALIGN to baselines by measuring the time required to align a pair of models in the CIFAR10 CNN and VGG classifiers datasets, and report the averaged alignment time using 1000 random pairs on a single A100 Nvidia GPU. The results are presented in Figure 5. DEEP-ALIGN is significantly faster than Sinkhorn and Activation Matching while achieving comparable results. Furthermore, DEEP-ALIGN is on par with Weight Matching w.r.t runtime, yet it consistently generates better weight alignment solutions.

F. Additional Experimental Results

Additional results for aligning classifiers. We provide additional results for aligning MLP and CNN image classifiers. The experiments follow the procedure described in Section 6.1. Table 5 provides results for aligning MLP classifiers trained using the MNIST and CIFAR10 datasets. DEEP-ALIGN outperforms all baseline methods. Additionally, Figure 8 provides LMC results for aligning both MLPs and CNNs trained on the MNIST, CIFAR10, and STL10 datasets. DEEP-ALIGN achieves on-par or improves over baselines, while DEEP-ALIGN + Sinkhorn outperforms all other methods. Notably, for VGG16, while the weight matching and activation matching improve over the naive method in terms of their respective objectives, they achieve poor barrier performance.

Towards diverse input architectures. One important extension of the DEEP-ALIGN framework is the generalization to diverse input networks. As discussed in Section 7, this limitation relates to the DWSNet encoder we utilize and can potentially be mitigated by replacing the DWS encoder with GNN-based weight space encoders like Zhang et al. (2023); Lim et al. (2023). Here, we show, on a small-scale experiment, that by modifying the DWSNet encoder, the DEEP-ALIGN

Table 4. Barrier results for aligning MLP networks with varying depths (3, 4, and 5 layers), trained using the MNIST dataset.

	MNIST Diverse MLPs
	Barrier ↓
Weight Matching	0.04 ± 0.00
Sinkhorn	0.02 ± 0.00
Weight Matching + Sinkhorn	$\mathbf{0.01 \pm 0.00}$
DEEP-ALIGN	0.03 ± 0.00
DEEP-ALIGN + Sinkhorn	$\mathbf{0.01 \pm 0.00}$

Table 5. MLP image classifiers: Results on aligning MNIST and CIFAR10 MLP image classifiers.

	MNIST (MLP)		CIFAR10 (MLP)	
	Barrier ↓	AUC ↓	Barrier ↓	AUC ↓
Naive	2.007 ± 0.00	0.835 ± 0.00	0.927 ± 0.00	0.493 ± 0.00
Weight Matching	0.047 ± 0.00	0.011 ± 0.00	0.156 ± 0.00	0.068 ± 0.00
Activation Matching	0.024 ± 0.00	0.007 ± 0.00	0.066 ± 0.00	0.024 ± 0.00
Sinkhorn	0.027 ± 0.00	0.002 ± 0.00	0.183 ± 0.00	0.072 ± 0.00
WM + Sinkhorn	0.012 ± 0.00	$\mathbf{0.000 \pm 0.00}$	0.137 ± 0.00	0.050 ± 0.00
DEEP-ALIGN	0.005 ± 0.00	$\mathbf{0.000 \pm 0.00}$	0.078 ± 0.01	0.029 ± 0.00
DEEP-ALIGN + Sinkhorn	$\mathbf{0.000 \pm 0.00}$	$\mathbf{0.000 \pm 0.00}$	$\mathbf{0.037 \pm 0.00}$	$\mathbf{0.004 \pm 0.00}$

generalizes to input networks with varying depths (number of layers). Specifically, in this experiment, we apply the DEEP-ALIGN model to MLPs with 3, 4, and 5 layers trained on the MNIST dataset. We implement a variant of the DWSNet encoder in which we share the weight of internal blocks. This allows the encoder to be applied to input networks with varying depths. The results, presented in Tabls 4, show that DEEP-ALIGN outperforms the WM baseline and that DEEP-ALIGN + Sinkhorn achieves on-par results compared to WM + Sinkhorn. These preliminary results demonstrate that the DEEP-ALIGN framework, together with an appropriate DWS encoder, can successfully be applied to diverse input architectures.

Aligning networks trained on disjoint datasets. Following (Ainsworth et al., 2022), we experiment with aligning networks trained on disjoint datasets. One major motivation for such a setup is Federated learning (McMahan et al., 2017). In Federated Learning, the goal is to construct a unified model from multiple networks trained on separate and distinct datasets.

To that end, we split the CIFAR10 dataset into two splits. The first consists of 95% images from classes 0-4 and 5% of classes 5-9, and the second split is constructed accordingly with 95% of classes 5-9. We train the DEEP-ALIGN model to align CNN networks trained using the different datasets. For Sinkhorn and Activation Matching, we assume full access to the training data in the optimization stage. For DEEP-ALIGN, we assume this data is accessible in the training phase. The results are presented in Figure 7. DEEP-ALIGN, along with the Sinkhorn and Activation Matching approaches, are able to align and merge the networks to obtain a network with lower loss compared to the original models. However, our approach is significantly more efficient at inference.

Results for the Activation matching + Sinkhorn baseline. Here, we provide results for the *activation matching (AM) + Sinkhorn* baseline, in which we use the AM solution to initialize the Sinkhorn optimization process. We commit this baseline from the main paper due to its extremely long runtime. Nonetheless, DEEP-ALIGN+Sinkhorn outperforms this runtime expensive baseline, as shown in Table 6.

DEEP-ALIGN as initialization. As discussed in the main text, our approach can be used as initialization for optimization-based approaches, like the Sinkhorn re-basin. Here, provide extended results on using the output of DEEP-ALIGN as the initial value for the alignment problem. We evaluate two previously proposed methods, weight-matching (WM) (Ainsworth et al., 2022) and Sinkhorn re-basin (Peña et al., 2023). Initializing the Sinkhorn method significantly improves the performance under all evaluated datasets. In addition, using DEEP-ALIGN initialization greatly improves the convergence speed. Furthermore, DEEP-ALIGN improves the barrier results of the weight-matching method. Notably, using the DEEP-ALIGN initialization achieves on-par or improved values for the WM objective.
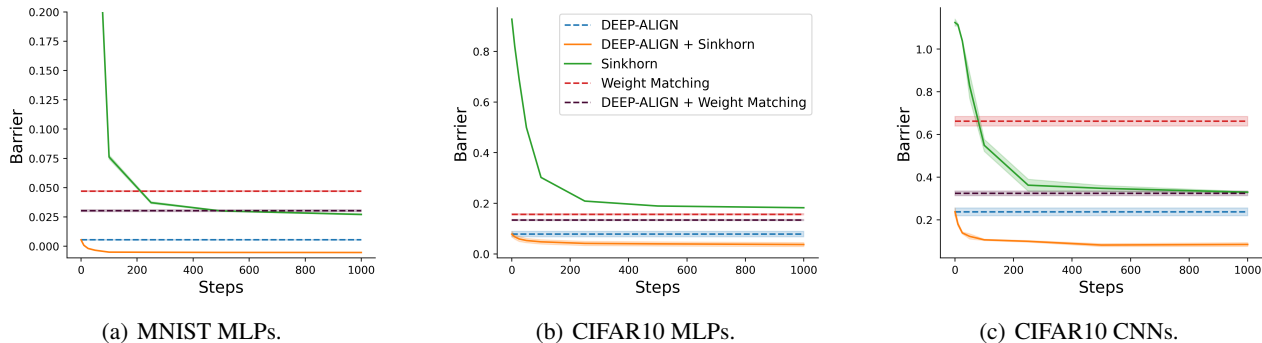
(a) MNIST MLPs.   (b) CIFAR10 MLPs.   (c) CIFAR10 CNNs.

Figure 9. DEEP-ALIGN as initialization: Results for using DEEP-ALIGN as initialization for the optimization-based approaches Sinkhorn re-basin and weight matching.

Table 6. Barrier results for the AM + Sinkhorn baseline. DEEP-ALIGN + Sinkhorn outperforms AM + Sinkhorn in terms of the Barrier metric and runtime.

	CIFAR10 (CNN)	STL10 (CNN)	
	Barrier ↓		Runtime ↓
Activation Matching	0.23 ± 0.01	0.47 ± 0.00	6.38
Sinkhorn	0.31 ± 0.01	0.36 ± 0.00	37.74
AM + Sinkhorn	0.10 ± 0.01	0.26 ± 0.00	$44.12 = 6.38 + 37.74$
DEEP-ALIGN	0.23 ± 0.01	0.38 ± 0.01	**0.20**
DEEP-ALIGN + Sinkhorn	$\mathbf{0.08 \pm 0.00}$	$\mathbf{0.23 \pm 0.00}$	$37.94 = 0.20 + 37.74$

Effect of sample size. We evaluate DEEP-ALIGN on the CIFAR10 CNN classifiers and the sine-wave INRs experiment, using a varying number of training examples. The results are presented in Figure 4 (in the main text) and Figure 10. On the CNNs dataset, DEEP-ALIGN produces alignments with on-par quality to the Sinkhorn method, with only ~ 100 training samples. Using the INRs dataset, DEEP-ALIGN achieves on-par results w.r.t the Sinkhorn method with random initialization using 1800 training samples. On the other hand, for both datasets, initializing the Sinkhorn method with the DEEP-ALIGN alignment shows significant improvement in the test barrier using only 100 training samples. These results show the efficiency of DEEP-ALIGN both in producing model alignments or initializing optimization-based approaches.

Ablation on the DEEP-ALIGN objective. Here, we provide results for DEEP-ALIGN trained with different objectives. Recall that we introduced three objectives (losses) to train DEEP-ALIGN. The first is the supervised loss $\ell_{\text{supervised}}$ computed using a model and its permuted version. The second is $\ell_{\text{alignment}}$ which is the $L2$ loss between the aligned weight vectors, and the third is ℓ_{LMC} which evaluates the original task loss on the line segment between the aligned models. For this ablation study, we use the MNIST and CIFAR10 MLP classifiers along with the CIFAR10 INRs. The results are presented in Table 7. Using only the supervised and alignment loss generally achieves insufficient results in terms of the Barrier metric. Dropping the alignment loss and using the supervised and LMC losses appears to have a minimal impact on the results in the classifier experiments. However, interestingly, including the $\ell_{\text{alignment}}$ in the INR experiment seems to have a detrimental effect on the Barrier results, causing a significant drop in performance. This suggests the alignment loss and the barrier metric are not always well aligned. In these cases it is advised to drop the $\ell_{\text{alignment}}$ loss and optimize DEEP-ALIGN with the $\ell_{\text{supervised}}$ and ℓ_{LMC} losses.

Visualization of predicted permutations. We visualize the predicted permutation obtained using DEEP-ALIGN applied to three test sine-waves INRs. Each network pair consists of an INR and its permuted and noisy version. For clarity, we depict only the first permutation matrix, P_1. The rows of Figure 11 correspond to the three test INRs. The left column represents the output from the F_{prod} layer, which then projected to the set of permutations using F_{prod} (middle column). DEEP-ALIGN is able to perfectly predict the ground truth permutations (right column).
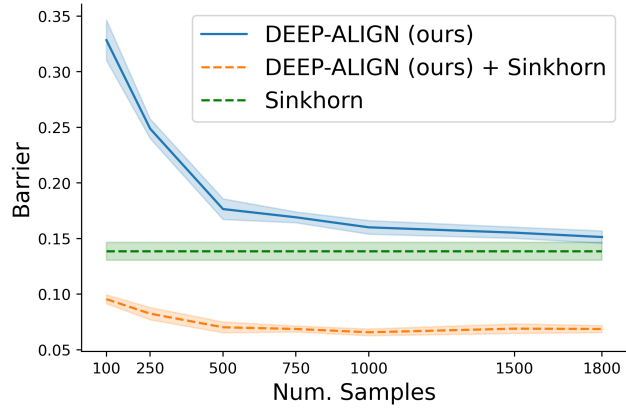
Figure 10. Effect of sample size: DEEP-ALIGN achieves on par results w.r.t the Sinkhorn with 1800 training pairs, while only 100 pairs are sufficient to significantly improve Sinkhorn by initializing the alg. with the DEEP-ALIGN outputs.

Table 7. Optimizing DEEP-ALIGN *using different objectives*: Test Barrier results averaged over 3 random seeds.

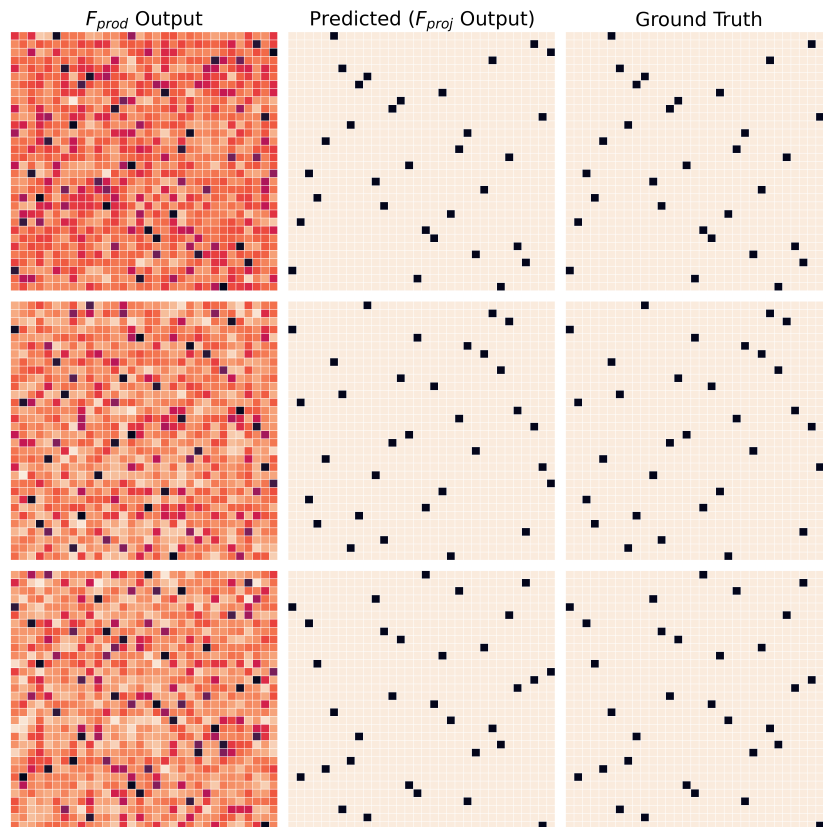|	MNIST MLP CLS	CIFAR10 MLP CLS	CIFAR10 INR
$\ell_{\text{supervised}} + \ell_{\text{LMC}}$	0.007 ± 0.00	$\mathbf{0.070 \pm 0.00}$	$\mathbf{0.063 \pm 0.00}$
$\ell_{\text{supervised}} + \ell_{\text{alignment}}$	0.061 ± 0.00	0.343 ± 0.00	0.127 ± 0.00
$\ell_{\text{supervised}} + \ell_{\text{LMC}} + \ell_{\text{alignment}}$	$\mathbf{0.005 \pm 0.00}$	0.078 ± 0.00	0.087 ± 0.00

Figure 11. Predicted permutation matrices and ground truth permutations for three test sine wave INRs and their permutated and noisy version. DEEP-ALIGN outputs the exact ground truth permutations.