OPTIMIZING LOSS LANDSCAPE CONNECTIVITY VIA NEURON ALIGNMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

The loss landscapes of deep neural networks are poorly understood due to their high nonconvexity. Empirically, the local optima of these loss functions can be connected by a simple curve in model space, along which the loss remains fairly constant. Yet, current path finding algorithms do not consider the influence of symmetry in the loss surface caused by weight permutations of the networks corresponding to the minima. We propose a framework to investigate the effect of symmetry on the landscape connectivity by directly optimizing the weight permutations of the networks being connected. Through utilizing an existing neuron alignment technique, we derive an initialization for the weight permutations. Empirically, this initialization is critical for efficiently learning a simple, planar, low-loss curve between networks that successfully generalizes. Additionally, we introduce a proximal alternating minimization scheme to address if an optimal permutation can be learned, with some provable convergence guarantees. We find that the learned parameterized curve is still a low-loss curve after permuting the weights of one of the endpoint models, for a subset of permutations. We also show that there is small but steady performance gain in performance of the ensembles constructed from the learned curve, when considering weight space symmetry.

1 INTRODUCTION

Loss surfaces of neural networks have been of recent interest in the deep learning community. These surfaces are interesting from a theoretical perspective. Their optimization yields interesting examples of a high-dimensional non-convex problem, where counter-intuitively gradient descent methods successfully converge to non-spurious optima. Practically, recent advancements in several applications have used insights on loss surfaces to justify their approaches. For instance, Moosavi-Dezfooli et al. (2019) investigates regularizing the curvature of the loss surface to increase the adversarial robustness of trained models.

One interesting question about these non-convex loss surfaces is to what extent trained models, which correspond to local minima, are connected. Here, *connection* denotes the existence of a path between the models, parameterized by their weights, along which loss is nearly constant. There has been conjecture that such models are connected asymptotically, with respect to the width of hidden layers. Recently, Freeman & Bruna (2016) proved this for rectified networks with one hidden layer.

When considering the connection between two neural networks, it is important for us to consider what properties of the neural networks are intrinsic. There is a permutation ambiguity in the indexing of units in a given hidden layer of a neural network, and as a result, this ambiguity extends to the network weights themselves. Thus, there are numerous equivalent points in model space that correspond to a given neural network. This creates weight symmetry in the loss landscape. It is possible that the minimal loss paths between a network and all networks equivalent to a second network could be quite different. If we do not consider the best path among this set, we could fail to see to what extent models are intrinsically connected. Therefore, we are interested in investigating the effect of weight symmetry in loss landscape connectivity in an effort to find more optimal curves.

Related Work Freeman & Bruna (2016) is one of the first studies to rigorously prove that one hidden layer rectified networks are asymptotically connected and established relevant bounds. Several recent numerical works have shown that parameterized curves along which loss is nearly constant

can be successfully learned. Concurrently, Garipov et al. (2018) proposed learning Bezier curves and polygonal chains and Draxler et al. (2018) proposed learning a curve using nudged elastic band energy between two models. Gotmare et al. (2018) showed that these algorithms work even for models trained using different hyperparameters, excluding network architecture. Recently, Kuditipudi et al. (2019) analyzed the connectivity between ϵ-dropout stable networks.

The symmetry groups in neural network weight space have long been formally studied (Chen et al., 1993). While permutation ambiguity in the weights has been acknowledged, ostensibly ambiguity due to scaling in the weights has received more attention in research. Numerous regularization approaches based on weight scaling such as in (Cho & Lee, 2017) have been proposed to improve the performance of learned models. More closely related to this work, Brea et al. (2019) studies the existence of *permutation plateaus* in which the neurons in the layer of a network can all be permuted at the same value on the loss surface.

A second line of work studies network similarity. Kornblith et al. (2019) gives a comprehensive review on the topic while introducing centered kernel alignment (CKA) for comparing the behavior of different neural networks. CKA is an improvement over the canonical correlation analysis (CCA) technique introduced in Raghu et al. (2017) and explored further in Morcos et al. (2018). A critical contribution in this direction is the neuron alignment algorithm from Li et al. (2016), which showed empirically that two networks of the same architecture learn a subset of similar feature representations.

Contributions We summarize the main contributions of this work as follows:
1. Inspired by the neuron alignment technique of (Li et al., 2016), we derive an initialization for the weight permutation in order to learn *aligned* curves connecting networks.
2. We apply a proximal alternating minimization (PAM) scheme to split the optimization into iteratively optimizing the permutation of the second model weights and optimizing the curve parameters. We prove convergence of our PAM scheme to a local critical point for feed-forward neural networks which are piece-wise analytic functions and continuously differentiable.
3. We perform empirical experiments on three datasets and architectures affirming that more optimal curves can be learned faster with neuron alignment initialization.
4. We observe a notable improvement in ensemble accuracy for simple networks when constructing ensembles by sampling along the aligned curve as opposed to the unaligned curve or a set of independent models.

For the structure of this paper, we first review pertinent background on curve finding and neuron alignment in Section 2. Then, we introduce our proposed optimization models and algorithms for curve finding up to a weight permutation in Section 3. In Section 4, we discuss our experiments in detail. In Section 5, we explore effect of alignment on the performance of model ensembles along the curve.

2 Background on Connectivity and Alignment

In this section we review the existing approaches for loss optima connectivity and neuron alignment.

Loss Optima Connectivity To learn the minimal loss path connecting two N-dimensional neural networks, $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, we utilize the curve finding approach introduced in (Garipov et al., 2018). Here we search for the path, $\boldsymbol{r} : [0, 1] \mapsto \mathbb{R}^N$, that connects the two models while minimizing the average of the loss function, $\mathcal{L}$, along the path. This problem is formalized in equation 1.

$$\boldsymbol{r}^* = \arg\min_{\boldsymbol{r}} \frac{\int_{t\in[0,1]} \mathcal{L}(\boldsymbol{r}(t))\|\boldsymbol{r}'(t)\|dt}{\int_{t\in[0,1]} \|\boldsymbol{r}'(t)\|dt} \qquad \text{subject to} \quad \boldsymbol{r}(0) = \boldsymbol{\theta}_1, \boldsymbol{r}(1) = \boldsymbol{\theta}_2. \tag{1}$$

For tractability, $\boldsymbol{r}^*$ can be approximated by a parameterized curve $\boldsymbol{r}_\phi$, where ϕ denotes the curve parameters. For instance, as described in Section 4, this paper will be using the quadratic Bezier curve. Computationally, an arclength parameterization, that is $||r'(t)|| = 1$ for all t, is assumed to make optimization more computationally feasible. Note that if the endpoint networks are global minima and a flat loss path does exist, then the optimal objective of equation 1 is unchanged.

An equivalent view under the arclength parameterization is that we are minimizing $\mathbb{E}_{\mathrm{t}\sim U}[\mathcal{L}(\boldsymbol{r}_\phi(t))]$, where U is the uniform distribution on the unit interval. This view is taken in Algorithm 2 in

Appendix B. For clarity, we emphasize that r_ϕ denotes the curve on the loss surface between two networks while $\boldsymbol{r}_\phi(t)$ denotes a point on that curve which is a neural network.

Neuron Alignment We give an overview of the neuron alignment framework in (Li et al., 2016). Given input d drawn from the input data distribution D, let $\boldsymbol{X}^{(1)}_{l,i,:}(\mathrm{d}) \in \mathbb{R}^k$ represent the activation values of channel i in layer l of network $\boldsymbol{\theta}_1$, where k is the number of units in the channel. As an example, a channel could correspond to a unit in a dense layer or a kernel in a convolutional layer, while k would be 1 or the number of pixels in a feature map, respectively.

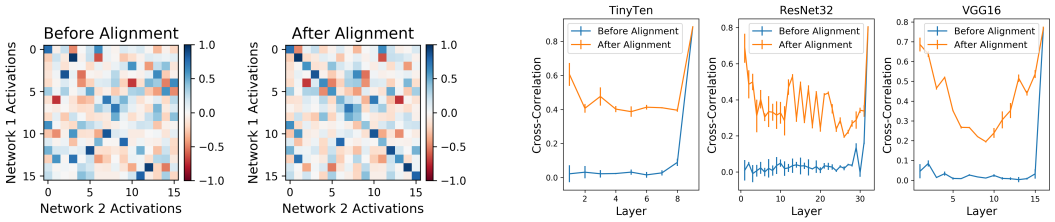
We take the channel mean, $\mu^{(1)}_{l,i}$, to be $\mathbb{E}_{\mathrm{d}\sim D}[\sum_{a=1}^{k} \frac{1}{k} X^{(1)}_{l,i,a}(\mathrm{d})]$ and define the channel deviation, $\sigma^{(1)}_{l,i}$, in an analogous manner. Then the cross correlation between activations in layer l between two networks $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, $\boldsymbol{C}^{(1,2)}_{l,:,:}$, is defined in equation 2.

$$C^{(1,2)}_{l,i,j} = \frac{\mathbb{E}_{\mathrm{d}\sim D}[\sum_{a=1}^{k} \frac{1}{k} \left(X^{(1)}_{l,i,a}(\mathrm{d}) - \mu^{(1)}_{l,i}\right)\left(X^{(2)}_{l,j,a}(\mathrm{d}) - \mu^{(2)}_{l,j}\right)]}{\sigma^{(1)}_{l,i}\sigma^{(2)}_{l,j}} \tag{2}$$

We stress that these networks share the same architecture. To align the activations in layer l between networks $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, the neuron alignment algorithm maximizes the sum of cross-correlation between aligned activations. Equivalently, this finds the permutation, $\boldsymbol{P}_l$, that maximizes the trace of $\boldsymbol{P}_l \boldsymbol{C}^{(1,2)}_{l,:,:}$, which is an instance of the linear assignment problem. We formalize this optimization model in equation 3 below, where K_l represents the index set of activations in layer l.

$$\begin{aligned} \max_{\boldsymbol{P}_l} \quad & \mathrm{trace}(\boldsymbol{P}_l \boldsymbol{C}^{(1,2)}_{l,:,:}) \\ \text{subject to} \quad & \boldsymbol{P}_l 1 = 1, \boldsymbol{P}_l^T 1 = 1, \quad \boldsymbol{P}_l \in \mathbb{Z}_+^{|K_l|\times|K_l|} \end{aligned} \tag{3}$$

The alignment technique is visualized in Figure 1a. This displays the cross-correlation matrix for the TinyTen network and CIFAR100 dataset that we discuss later in Section 4. It is clear that the values along the diagonal are much stronger after alignment. Figure 1b displays the mean cross-correlation at each layer between corresponding neurons. This figure also shows the standard deviation of this signal over a set of 3 network pairs. With this *correlation signature* being consistent over different pairs and being increased highly with alignment, we can feel confident that some subset of highly correlated features are being matched.



(a) Cross-correlation between neurons

(b) Mean cross-correlation at each layer

Figure 1: **(1a)** Cross-correlation between the activations in the first layer of a TinyTen model for CIFAR100. The plot on the left uses the original indices of the second network, while the plot on the right uses the reindexing of the second model consistent with alignment to the first. **(1b)** The mean cross-correlation between corresponding units for each layer before and after alignment. The standard deviation of this correlation signature over a set of different network pairs is displayed.

3 Optima Connectivity Considering Weight Symmetry

We clarify the idea of weight symmetry in a neural network. $\boldsymbol{\theta}_1$ is a neural network on the loss surface parameterized by its weights. A permutation P_l can be seen as a permutation on the index set of channels in layer l, K_l. For simplicity suppose we have an L layer feed-forward network

with activation function σ, weights $\{W_l\}_{l=1}^L$, and input X_0. Then the weight permutation ambiguity becomes clear when we introduce the following set of permutations:

$$\boldsymbol{Y} := \boldsymbol{W}_L P_{L-1}^T \circ \sigma \circ \boldsymbol{P}_{L-1}\boldsymbol{W}_{L-1}\boldsymbol{P}_{L-2}^T \circ \sigma \circ \boldsymbol{P}_{L-2}\boldsymbol{W}_{L-2}\boldsymbol{P}_{L-3}^T ... \circ \sigma \circ \boldsymbol{P}_1\boldsymbol{W}_1\boldsymbol{X}_0 \tag{4}$$

Then we can define the network weight permutation $\boldsymbol{P}$ as the block diagonal matrix, blockdiag$(\boldsymbol{P}_1, \boldsymbol{P}_2, ..., \boldsymbol{P}_{L-1})$. Additionally, $\boldsymbol{P\theta}$ denotes the network parameterized by the weights $[\boldsymbol{P}_1\boldsymbol{W}_1, \boldsymbol{P}_2\boldsymbol{W}_2\boldsymbol{P}_1^T, ..., \boldsymbol{W}_L\boldsymbol{P}_{L_1}^T]$.

Note that we omit permutations $\boldsymbol{P}_0$ and $\boldsymbol{P}_L$, as the input and output of neural networks have a fixed ordering, so they correspond to the identity $\boldsymbol{I}$. As an example, when classifying images, the input channels have a fixed ordering such as RGB and the index of each logit corresponds to a given class. Without much difficulty this framework generalizes for more complicated architectures. We discuss this for residual networks in Appendix D.

3.1 Curve Finding with Symmetry

From equation 4, it becomes clear that the networks $\boldsymbol{\theta}_1$ and $\boldsymbol{P\theta}_1$ share the same structure and intermediate outputs up to indexing. Taking weight symmetry into account, we can find a curve connecting two networks up to symmetry with the following model.

$$\begin{aligned} \phi^*, \boldsymbol{P}^* = \arg\min_{\phi, \boldsymbol{P}} \quad & \mathbb{E}_{t\sim U}[\mathcal{L}(\boldsymbol{r}_\phi(t))] \\ \text{subject to} \quad & \boldsymbol{r}_\phi(0) = \boldsymbol{\theta}_1, \boldsymbol{r}_\phi(1) = \boldsymbol{P\theta}_2, \quad \boldsymbol{P} = \text{blockdiag}(\boldsymbol{P}_1, \boldsymbol{P}_2, ..., \boldsymbol{P}_{L-1}) \\ & \boldsymbol{P}_l \in \Pi_{|K_l|} \text{ for } l \in \{1, 2, ..., L-1\} \end{aligned} \tag{5}$$

3.1.1 Proximal Alternating Minimization Model

The problem formulation in equation 5 is fairly complicated. Theoretically, it is not easy to analyze. Computationally, approaching the problem directly with first order methods could be computationally intensive as we need to store gradients of ϕ and $\boldsymbol{P}$ simultaneously. The problem can be more easily addressed using the method of proximal alternating minimization (PAM) (Attouch et al., 2010). The PAM scheme involves iteratively solving the two subproblems in equation 6. Here we let $Q(\phi, \boldsymbol{P})$ denote the objective function in equation 5. We also only consider parameterized forms of $\boldsymbol{r}$ that satisfy the endpoint constraints for any combination of ϕ and $\boldsymbol{P}$. For generality, we let $\mathcal{R}$ denote a regularization term on ϕ.

$$\begin{cases} \boldsymbol{P}^{k+1} = \arg\min_{\boldsymbol{P}} & Q(\phi^k, \boldsymbol{P}) + \frac{1}{2\nu_P}||\boldsymbol{P} - \boldsymbol{P}^k||_2^2 \\ \text{such that} & \boldsymbol{P}_l \in \Pi_{|K_l|} \text{ for } l \in \{1, 2, ..., L-1\} \\ & \boldsymbol{P} = \text{blockdiag}(\boldsymbol{P}_1, \boldsymbol{P}_2, ..., \boldsymbol{P}_{L-1}) \\ \phi^{k+1} = \arg\min_{\phi} & Q(\phi, \boldsymbol{P}^{k+1}) + \mathcal{R}(\phi) + \frac{1}{2\nu_\phi}||\phi - \phi^k||_2^2 \end{cases} \tag{6}$$

The traditional curve finding algorithm is equivalent to solving the PAM scheme with a very large value of ν_P. In fact, we are able to prove some local convergence results for a certain class of networks.

Theorem 3.1 (Convergence)**.** *Let $\{\phi^{k+1}, \boldsymbol{P}^{k+1}\}$ be the sequence produced by equation 6. Assume that $\boldsymbol{r}_\phi(t)$ corresponds to a feed-forward neural network with activation function σ for $t \in [0, 1]$. Assume that $\mathcal{L}$, r_ϕ, and σ are all piece-wise analytic functions in C^1 and locally Lipschitz differentiable in ϕ and $\boldsymbol{P}$. Additionally, assume $\mathcal{R}$ is piece-wise analytic in the primal variables and bounded below. Then the following statements hold:*

1. $Q(\phi^{k+1}, \boldsymbol{P}^{k+1}) + \mathcal{R}(\phi^{k+1}) + \frac{1}{2\nu_\phi}||\phi^{k+1} - \phi^k||_2^2 + \frac{1}{2\nu_P}||\boldsymbol{P}^{k+1} - \boldsymbol{P}^k||_2^2 \leq Q(\phi^k, \boldsymbol{P}^k) + \mathcal{R}(\phi^k), \forall k \geq 0$
2. $\{\phi^k, \boldsymbol{P}^k\}$ *converges to a critical point of* $Q(\phi, \boldsymbol{P}) + \mathcal{R}(\phi)$

Proof See Appendix C

Remark Theorem 3.1 does not extend to neural networks with ReLU activation functions. In Appendix C, we address a technique utilizing this theorem for learning a curve connecting rectified networks while still generating a sequence of iterates with monotonic decreasing objective value.

3.2 NEURON ALIGNMENT AS AN INITIALIZATION FOR CURVE FINDING

In spite of convergence guarantees, we still require a good initialization as the loss landscape is non-convex. This is critical for avoiding convergence to a non-global optima. Neuron alignment introduced in (Li et al., 2016) is able to match subsets of similar feature representations. We believe that the permutation on the network weights induced by neuron alignment could be meaningful enough to be a good initialization for P.

In practice, we solve the linear sum assignment problem formulated in Equation 3 using the Hungarian algorithm. See Kuhn (1955) for further reading on the Hungarian algorithm. Algorithm 1 summarizes the process for efficiently computing a permutation of the network weights from neuron alignment. For an L layer network with a maximum layer width of M, the running time of all needed linear assignments is $\mathcal{O}(LM^3)$. This is on the order of the running time associated with one iteration of forward propagation.

Data: Trained Neural Networks $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, Subset of Training Data $\mathbf{X}_0$
Result: Aligned Neural Networks $\boldsymbol{\theta}_1$ and $\boldsymbol{P\theta}_2$
Initialize $\boldsymbol{P\theta}_2 := [\hat{\boldsymbol{W}}_1^2, \hat{\boldsymbol{W}}_2^2, ..., \hat{\boldsymbol{W}}_L^2]$ as $[\boldsymbol{W}_1^2, \boldsymbol{W}_2^2, ..., \boldsymbol{W}_L^2]$ for $k \in \{1, 2, ..., L-1\}$;
for *each layer* l *in* $\{1, 2, ..., L-1\}$ **do**
  for *each network* j *in* $\{1, 2\}$ **do**
    compute activations, $\mathbf{X}_l^{(j)} = \sigma \circ \boldsymbol{W}_l^j \mathbf{X}_{l-1}^{(j)}$;
    for each element in the batch, vectorize $\mathbf{X}_l^{(j)}$ if applicable ;
    compute, $\boldsymbol{Z}_l^{(j)}$, the Z-score normalization of the activations ;
  end
  compute the correlation matrix, $\boldsymbol{C}_l^{(1,2)} = \boldsymbol{Z}_l^{(1)} \boldsymbol{Z}_l^{(2)T}$;
  compute $\boldsymbol{P}_l$ by solving the assignment problem associated with $\boldsymbol{C}_l^{(1,2)}$ using the Hungarian algorithm ;
  update $\hat{\boldsymbol{W}}_l^2 \to \boldsymbol{P}_l \hat{\boldsymbol{W}}_l^2, \quad \hat{\boldsymbol{W}}_{l+1}^2 \to \hat{\boldsymbol{W}}_{l+1}^2 \boldsymbol{P}_l^T$
end

Algorithm 1: Permutation Initialization via Neuron Alignment

We note that recent work has been skeptical about the quality of complete bipartite matchings between activations (Wang et al., 2018). However, because of the non-linearity of the activation function, a general correspondence between activation units does not necessarily induce a correspondence on the network weights. Thus, we restrict ourselves to the case of bipartite matching.

4 EXPERIMENTS

Datasets In our experiments, we trained neural networks to classify images from CIFAR10 and CIFAR100 (Krizhevsky et al., 2009), as well as STL10 (Coates et al., 2011). The loss function is the cross entropy loss on the softmax of the logits output by the networks. 20% of the images in the training set are used for computing alignments between pairs of models. We augment the data using color normalization, random horizontal flips, random rotation, and random cropping to prevent the models from overfitting on the training set.

Architectures Three different model architectures are used. Table 1 summarizes relevant properties of these architectures. The first architecture considered was the TinyTen model introduced in Kornblith et al. (2019). TinyTen is a narrow 10 layer convolutional neural network that uses batch-normalization, ReLU activations, and global average pooling. This is a useful model for concept testing and allows us to gain insight to networks that are underparameterized. We also include ResNet32 (He et al., 2016) in our experiments to understand the effect of skip connections on curve finding with alignment. Details on how to compute the alignment for ResNet architectures is included in Appendix D as it does not have a simple feed-forward structure. VGG16-BN is the third architecture that we considered in our experiments (Simonyan & Zisserman, 2014). VGG16 has significantly more parameters compared to other models. We chose this set of architectures for the varying properties and because of their prevalence in numerical experiments in related literature.

Table 1: Properties of models used in this study

Model	Number of Parameters	Depth	Accuracy		
			CIFAR10	CIFAR100	STL10
TinyTen	86,778	10	88.7 ± 0.2	58.1 ± 0.5	73.8 ± 0.3
ResNet32	466,906	32	92.9 ± 0.2	67.1 ± 0.5	76.5 ± 0.3
VGG16-BN	15,253,578	16	93.1 ± 0.2	70.9 ± 0.3	72.5 ± 1.5

Quadratic Bezier curves All curves are parameterized as quadratic Bezier curves. Bezier curves are popular in computer graphics as they can be defined by their *control points*. In the current study, we refer to endpoint models as $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ as well as the control point, θ_c. Then $\boldsymbol{r}$ is defined in equation 7

$$\boldsymbol{r}_\phi(t) = (1-t)^2\boldsymbol{\theta}_1 + 2(1-t)t\boldsymbol{\theta}_c + t^2\boldsymbol{\theta}_2. \tag{7}$$

Important properties of the quadratic Bezier curve include $\boldsymbol{r}(0) = \boldsymbol{\theta}_1$, $\boldsymbol{r}(1) = \boldsymbol{\theta}_2$, $\boldsymbol{r}'(0) = 2(\boldsymbol{\theta}_c - \boldsymbol{\theta}_1)$, and $\boldsymbol{r}'(1) = 2(\boldsymbol{\theta}_2 - \boldsymbol{\theta}_c)$. Then $\boldsymbol{\theta}_c$ is the learnable parameter in ϕ. Of course one could consider more complicated curve parameterizations. In practice, we find a simple curve to be enough for our experiments, and consider the learning of a planar curve along which loss is nearly constant to be significant in itself.

4.1 Training Curves

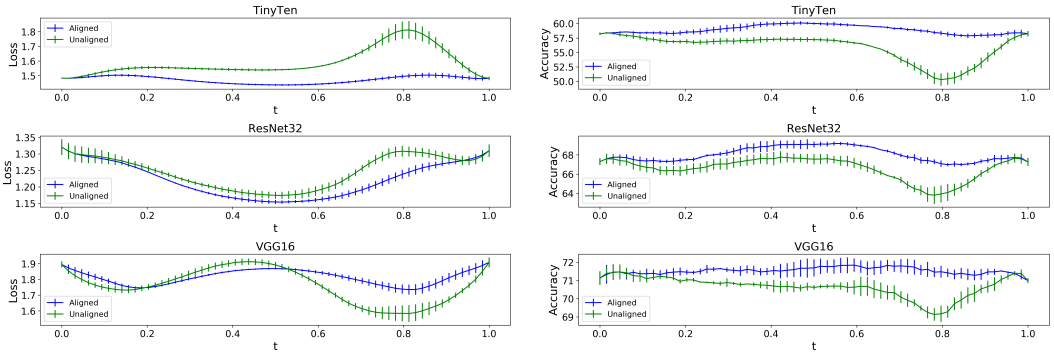
For each architecture, we train 12, 6, and 6 different models using different random initializations for CIFAR10, CIFAR100, and STL10 respectively. Thus we have 6 or 3 independent model pairs for a dataset. We learn four classes of curves:

- Unaligned, $\boldsymbol{r}(\phi)$: Solution of Algorithm 2 with standard initialization
- Aligned, $\boldsymbol{r}(\phi, \boldsymbol{P}_{Corr})$: Solution of Algorithm 2 with neuron alignment initialization
- Unaligned PAM, $\boldsymbol{r}(\phi, \boldsymbol{P}_{PAM})$: Solution of PAM scheme with default initialization
- Aligned PAM, $\boldsymbol{r}(\phi, \boldsymbol{P}_{PAM,Corr})$: Solution of PAM scheme with neuron alignment initialization

Note that we only learned PAM curves for the TinyTen networks. In our experiments we find that PAM and standard curve finding with aligned initialization attain similar empirical performance. Therefore, for large models (ResNet32 and VGG16) we only implement standard curve finding with aligned initialization due to its efficiency. We train two sets of each curve class. One set involves the curves learned when the random seed for curve finding is fixed for all model pairs. The other set consists of the curves learned when the random seed is different for each model pair. We find that the learned curves for different seeds are similar up to reindexing the endpoints. For Figures 2, 3, and 4, we use the first set of curves so that interesting geometric features on the loss surface are not averaged out. For tables and other figures, we use the second set of curves as they are more general.

Table 2: The average accuracy along the curve with standard deviation is reported for each combination of dataset, network architecture, and curve class. All curves are quadratic Bezier curves.

Model	Endpoints	CIFAR10	CIFAR100	STL10
TinyTen	Unaligned	87.2 ± 0.2	56.0 ± 0.2	73.7 ± 0.4
	Aligned	$\mathbf{88.6 \pm 0.1}$	58.7 ± 0.2	$\mathbf{74.1 \pm 0.3}$
	Unaligned PAM	87.1 ± 0.1	56.0 ± 0.3	73.0 ± 0.4
	Aligned PAM	88.5 ± 0.1	$\mathbf{59.0 \pm 0.1}$	74.0 ± 0.5
ResNet32	Unaligned	92.4 ± 0.2	66.5 ± 0.2	76.6 ± 0.2
	Aligned	$\mathbf{92.9 \pm 0.2}$	$\mathbf{67.7 \pm 0.1}$	$\mathbf{76.7 \pm 0.2}$
VGG16	Unaligned	93.0 ± 0.1	70.7 ± 0.1	74.5 ± 1.0
	Aligned	$\mathbf{93.3 \pm 0.1}$	$\mathbf{71.6 \pm 0.1}$	$\mathbf{74.7 \pm 0.8}$



(a) (b)

Figure 2: Test loss (left) and accuracy (right) of the learned quadratic Bezier curve between model endpoints trained on CIFAR100. Results are compared for aligned (blue) and unaligned (green) curves.

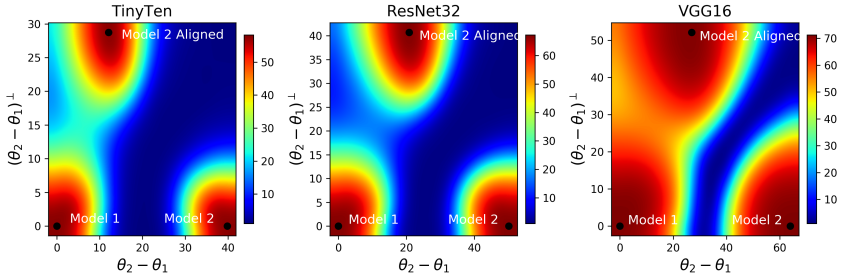


Figure 3: Test accuracy on CIFAR100 across the plane containing $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, and $\boldsymbol{P\theta}_2$, where $\boldsymbol{P}$ is set using neuron alignment. This shows the two different intializations used in our curve finding experiments. The default initialization, $\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1$, and the aligned initialization, $\boldsymbol{P\theta}_2 - \boldsymbol{\theta}_1$.

4.1.1 Alignment as an Initialization for Curve Finding

First, we investigate the effects of using neuron alignment as an initialization for curve finding. That is, we are determining some weight permutation $\boldsymbol{P}$ and then finding the curve between networks $\boldsymbol{\theta}_1$ and $\boldsymbol{P\theta}_2$. We are interested in finding better initializations for curve finding because if it is effective enough, we could forgo the use of more complicated optimization schemes such as proximal alternating minimization.

The test loss and accuracy along the learned curves for CIFAR100 are shown in Figure 2. This loss comprises of only the cross-entropy components and ignores regularization terms. The corresponding Fourier transform of the loss along the curve for assessing curve smoothness is displayed in Figure 11. We observe that, as expected, the accuracy at each point along the aligned curve outperforms the unaligned curve in terms of accuracy, while the loss along the curve is also smoother with neuron alignment. We are comparing loss and accuracy at the curve parameter, t. Noteworthy is the prominent presence of the accuracy barrier along the unaligned curve around t at 0.8 for all models. This accuracy barrier corresponds to a clear loss barrier for Tiny-10 and ResNet32. In contrast, for VGG16 there is a valley in the loss function at this point on the unaligned curve with worse generalization performance. Overall, we find that loss along the aligned curves varies more smoothly as seen in Figure 11, and this leads to better generalization of the interpolated models.

Also we notice that the size of the loss barrier depends on the extent of overparameterization in the model. The highest loss barrier is encountered for TinyTen networks trained on CIFAR100 data, while the loss barrier is practically nonexistent for VGG16 networks trained on STL10. This behavior aligns closely with claims in (Freeman & Bruna, 2016). We do expect the gain in performance from alignment to decrease as the parameterized curves are allowed to become more complex. How-

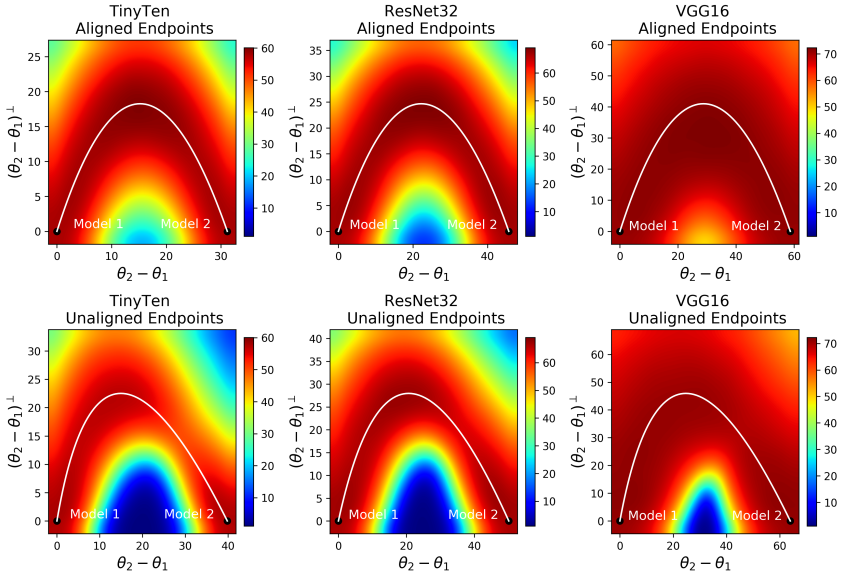
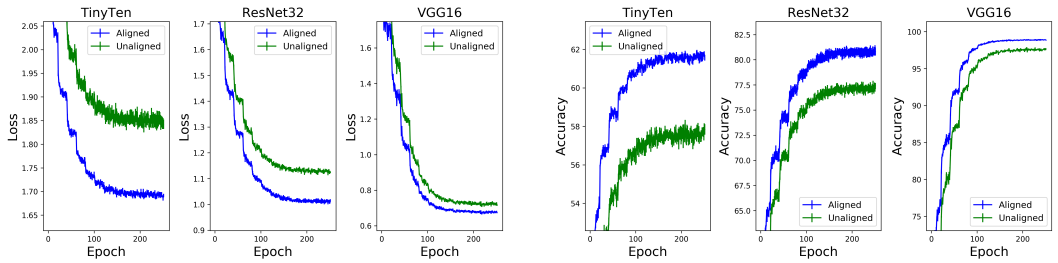


Figure 4: Test accuracy on CIFAR100 across the plane containing bezier curve, $\boldsymbol{r}_\phi(t)$



(a) CIFAR100 training loss

(b) CIFAR100 training accuracy

Figure 5: The training loss and training accuracy for learning the quadratic Bezier curve between model endpoints. These are compared for aligned and unaligned curves.

ever, simpler parameterized curves are appealing from a modeling perspective and they are faster to train with the curve finding algorithm.

Figure 3 displays the planes which contains the initializations for curve finding. It is clear that the aligned initialization has better objective value. The planes containing the learned curves are displayed in Figure 4. These are the planes containing $\boldsymbol{\theta}_1$, $\boldsymbol{P}\boldsymbol{\theta}_2$, and $\boldsymbol{\theta}_c$, although the control point is out of bounds of the figure. The axis is determined by Gram-Schmidt orthonormalization. The midpoint of the aligned curve is essentially equidistant from each endpoint while this does not hold for the unaligned curve. We also note the points on the unaligned curves that generalize poorly correspond to points on the loss plane with noticeably higher curvature as seen in Figure 12a. This observation is in line with the commonly held notion that smoother minima on loss surfaces generalize better, though this notion has recently been called into question (Dinh et al., 2017).

From a practical point of view, the neuron alignment initialization for determining the permutation $\boldsymbol{P}$ may be enough and avoids performing more complicated optimization. We see this by noting the relative flatness of the accuracy along the aligned curves in Figure 2b. Additionally, Figure 5 indicates much faster convergence when learning ϕ using neuron alignment as initialization, which is quite impressive. For example, the aligned curve takes 100 epochs less to achieve the training accuracy that the unaligned curve converges to, when TinyTen is used on CIFAR100. Even for VGG16, the aligned curve reaches the milestone 40 epochs earlier. Additionally, there is clearly a gap in the accuracy that the curves converge to, with the aligned curve always outperforming

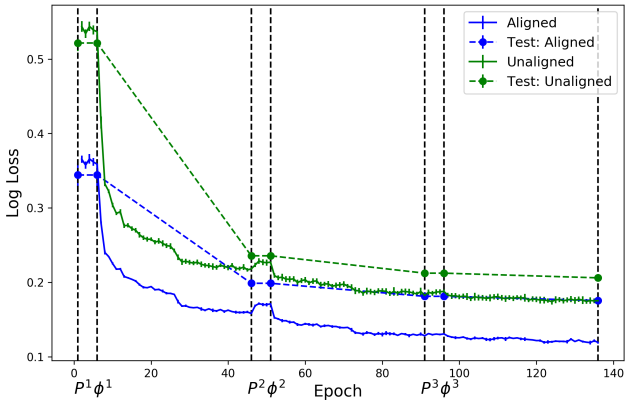


Figure 6: Log loss over a run of the proximal alternating minimization scheme on TinyTen for CIFAR100. The scheme consists of 5 epochs of projected SGD to solve the permutation subproblem, followed by 40 epochs of SGD to solve the curve parameter subproblem. Vertical lines denote the change in different subproblem iterations. Note that while the loss plateaus during the permutation subproblem, the permutation is still changing non-trivially.

the unaligned one, while the underparameterized architectures receive a more significant accuracy boost.

4.1.2 Proximal Alternating Minimization

Proximal alternating minimization provides a comprehensive formulation for learning the weight permutation $\boldsymbol{P}$ directly, coupled with some convergence guarantees. However, We find its empirical performance is similar to standard curve finding, and we advocate the latter for practical use due to computational efficiency.

The first question to address is how to effectively deal with the constraints in equation 6, $\{\boldsymbol{P}_l \in \Pi_{|K_l|}\}_{i=1}^{L-1}$. This constraint is both an integral and an orthogonality constraint, and thus not straightforward to handle. In this experiment, we solve an iteration of the permutation subproblem using projected stochastic gradient descent. In our permutation subproblem algorithm, after a batch in training, each $\boldsymbol{P}_l$ is projected onto the set of doubly stochastic matrices, $\mathcal{D}_{|\mathcal{K}_{\updownarrow}|}$, via the Sinkhorn algorithm (see Knight (2008)). Then after each epoch of training, each $\boldsymbol{P}_l$ is projected onto the set of permutations, $\Pi_{|K_l|}$. This allows us to explore the convex relaxation of the feasible set during an epoch of training before projecting back onto the integral feasible set. Within this subproblem iteration, we use a cosine annealing learning rate to allow convergence to a local critical point.

Figure 6 shows the average log loss while learning the curve connecting two TinyTen networks for the CIFAR100 dataset. We alternate between performing 5 epochs of projected SGD to solve the permutation subproblem and 40 epochs of SGD to solve the curve subproblem (see equation 6). Altogether, we perform 3 outer iterations. As a note, when implementing PAM in practice, we do not typically solve an iteration of the subproblem completely, instead we attempt to reduce the objective by a reasonable amount. This approach helps in avoiding over-iterating. Most notable in the plot is the failure of the loss function to meaningfully decrease during an iteration of the permutation subproblem. Indeed, Table 2 shows that the numerical implementation of PAM is not able to significantly increase the accuracy of the learned curve compared to previous curve finding algorithms, which can be attributed to the highly non-convex nature of equation 5.

However, the permutation $\boldsymbol{P}$ is found to change non-trivially during the permutation subproblem iterations. For some iterations, we find as much as 80% of all columns of $\boldsymbol{P}$ have been permuted from the previous iteration. This observation indicates finding some set of permutations $\{\boldsymbol{P}(i)\}_{i\in I}$, for which $\boldsymbol{\theta}_1$ and $\boldsymbol{P}(i)\boldsymbol{\theta}_2$ can be optimally connected with curves that have the same control point, $\boldsymbol{\theta}_c$. This result provides a striking geometric insight from this empirical experiment and is aligned with the notion of *permutation saddles* formalized in (Brea et al., 2019). We still see that when PAM

is initialized using neuron alignment, it converges to a more optimal curve, implying the permutation itself is still critical for finding the optimal connectivity.

5 Constructing Ensembles Along Aligned Curves

We further investigate if the diversity of models along the curve suffers due to alignment through the lens of constructing network ensembles. We consider the simple ensemble that performs classification by averaging the probability distributions output by the individual networks. In our experiment, we look at four cases. The ensemble formed by considering the curve endpoints, the ensembles formed by the curve endpoints and the midpoint, and an ensemble of the curve endpoints and a third independent model. We consider ensembling only on the CIFAR100 dataset with results for the aligned and unaligned case being summarized in Table 3. We see that at best, ensembles constructed by sampling along the unaligned curve perform as well as the independent ensemble. When constructing ensembles by sampling along the aligned curve, those outperform the independent ensemble for the TinyTen and Resnet32 case and show comparable performance for VGG16. The enhanced performance is most obvious on TinyTen. This result makes sense since ensembling has been reported to lead to better performance increase for simpler networks (Ju et al., 2018). This result is encouraging as ensembles of simple models are common in practice.

Table 3: Accuracy of model ensembles constructed from the curve connecting trained models on the CIFAR100 dataset. Standard deviation is reported as well.

curve parameter t	Curve Ensembles (%)			Independent Ensembles (%)
	$\{0.0, 1.0\}$	$\{0.0, 0.5, 1.0\}$		
		Unaligned	Aligned	
TinyTen	61.23 ± 0.36	61.39 ± 0.25	$\mathbf{62.40 \pm 0.30}$	61.76 ± 0.01
ResNet32	71.35 ± 0.30	72.13 ± 0.12	$\mathbf{72.33 \pm 0.12}$	72.03 ± 0.08
VGG16	74.00 ± 0.17	74.69 ± 0.22	$\mathbf{74.91 \pm 0.10}$	74.88 ± 0.06

6 Discussion

We generalize the curve finding algorithm to an optimization framework that removes the weight symmetry ambiguity when learning a curve connecting two neural networks on the loss surface. This can be solved via a proximal alternating minimization algorithm. We further prove its algorithmic convergence guarantees under additional conditions. Empirically, we show that neuron alignment can be used to successfully and efficiently learn to optimize connections between modes of neural nets. Therefore, addressing the ambiguity of weight symmetry is critical for learning planar curves on the loss surface along which accuracy is mostly constant. Our results hold true over a range of datasets and network architectures. With neuron alignment, these less complicated curves can be trained in less epochs than before and to higher accuracy. We also see a modest increase in the performance of ensembling by sampling along the aligned curve.

Future work will include gaining a deeper *theoretical understanding* of how neuron alignment affects curve finding dynamics and understanding the relationship of these nearly constant loss curves to *information geometry*.

References

Hédy Attouch, Jérôme Bolte, Patrick Redont, and Antoine Soubeyran. Proximal alternating minimization and projection methods for nonconvex problems: An approach based on the kurdyka-łojasiewicz inequality. *Mathematics of Operations Research*, 35(2):438–457, 2010.

Johanni Brea, Berfin Simsek, Bernd Illing, and Wulfram Gerstner. Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape. *arXiv preprint arXiv:1907.02911*, 2019.

An Mei Chen, Haw-minn Lu, and Robert Hecht-Nielsen. On the geometry of feedforward neural network error surfaces. *Neural computation*, 5(6):910–927, 1993.

Minhyung Cho and Jaehyung Lee. Riemannian approach to batch normalization. In *Advances in Neural Information Processing Systems*, pp. 5225–5235, 2017.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1019–1028. JMLR. org, 2017.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International Conference on Machine Learning*, pp. 1308–1317, 2018.

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

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pp. 8789–8798, 2018.

Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. Using mode connectivity for loss landscape analysis. *arXiv preprint arXiv:1806.06977*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Cheng Ju, Aurélien Bibaut, and Mark van der Laan. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *Journal of Applied Statistics*, 45(15):2800–2818, 2018.

Philip A Knight. The sinkhorn–knopp algorithm: convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1):261–275, 2008.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Rohith Kuditipudi, Xiang Wang, Holden Lee, Yi Zhang, Zhiyuan Li, Wei Hu, Sanjeev Arora, and Rong Ge. Explaining landscape connectivity of low-cost solutions for multilayer nets, 2019.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *Iclr*, 2016.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9078–9086, 2019.

Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, pp. 5727–5736, 2018.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pp. 6076–6085, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

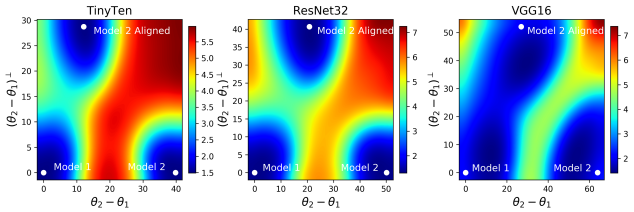
Liwei Wang, Lunjia Hu, Jiayuan Gu, Yue Wu, Zhiqiang Hu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation, 2018.

Yangyang Xu, Ioannis Akrotirianakis, and Amit Chakraborty. Proximal gradient method for huberized support vector machine. *Pattern Analysis and Applications*, 19(4):989–1005, 2016.

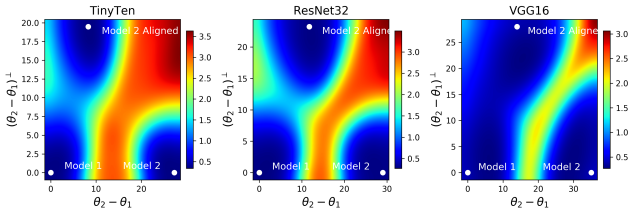
Supplementary Material for Optimizing Loss Landscape Connectivity via Neuron Alignment

A Additional Figures

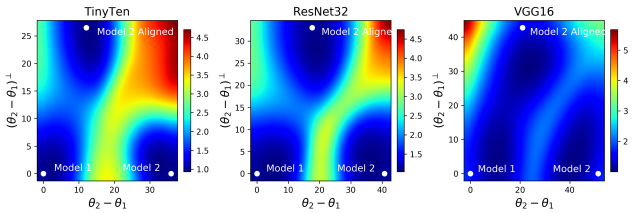
A.1 Planes Containing Linear Initializations



(a) CIFAR100

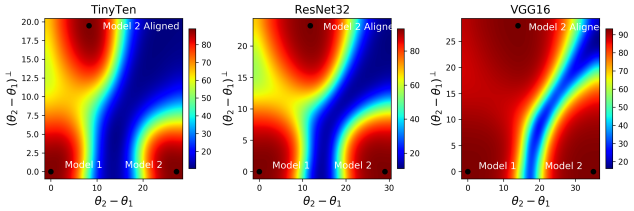


(b) CIFAR10

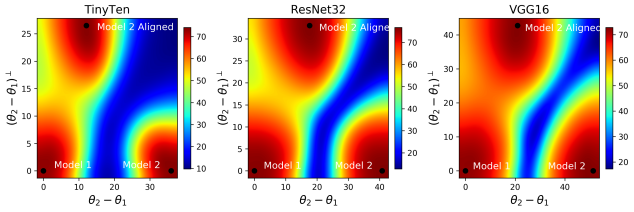


(c) STL10

Figure 7: Test loss on plane containing $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, and $\boldsymbol{P\theta}_2$.



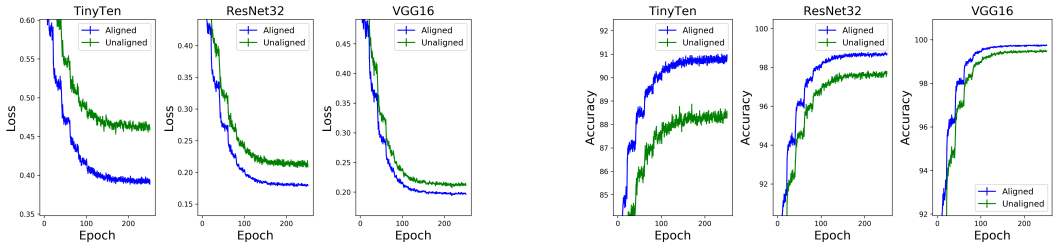
(a) CIFAR10



(b) STL10

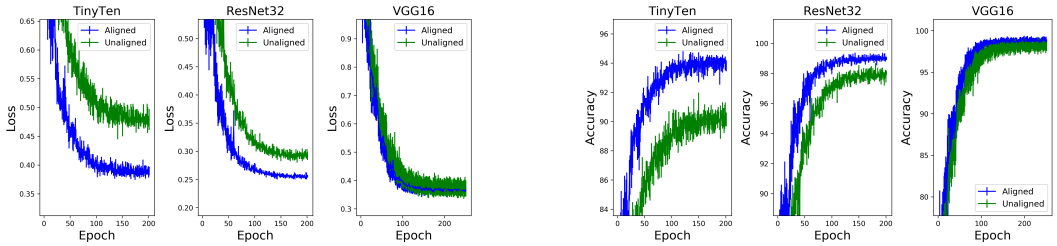
Figure 8: Test accuracy on plane containing $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, and $\boldsymbol{P\theta}_2$.

A.2 TRAINING



(a) Training loss (b) Training accuracy

Figure 9: CIFAR10



(a) Training loss (b) Training accuracy

Figure 10: STL10

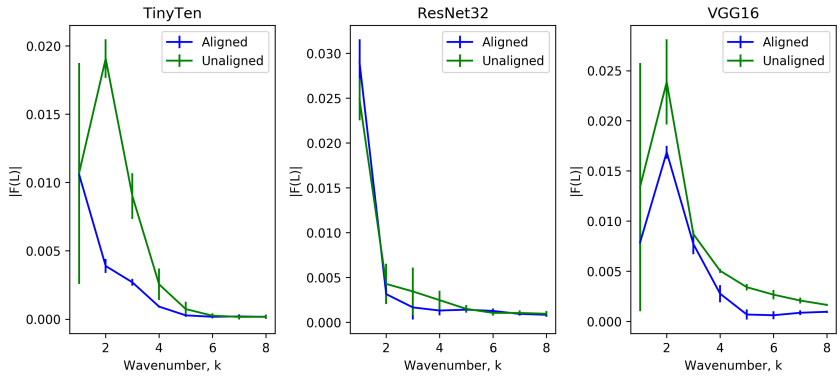
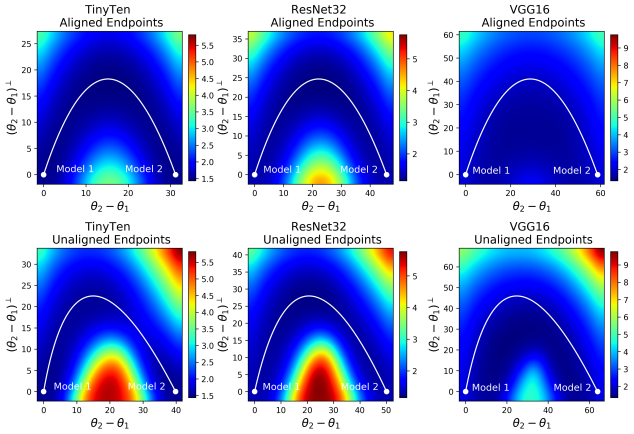
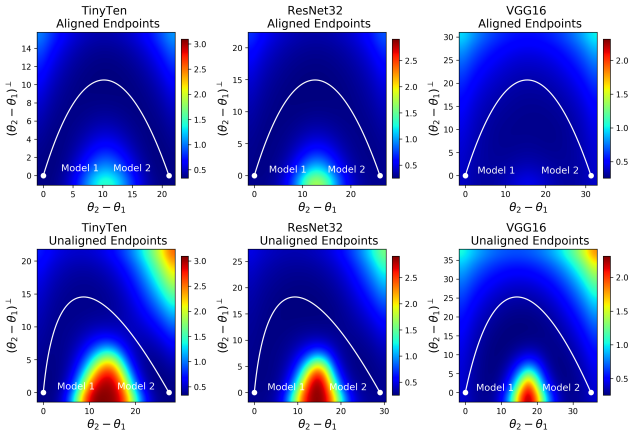


Figure 11: Fourier transform of CIFAR100 loss curve

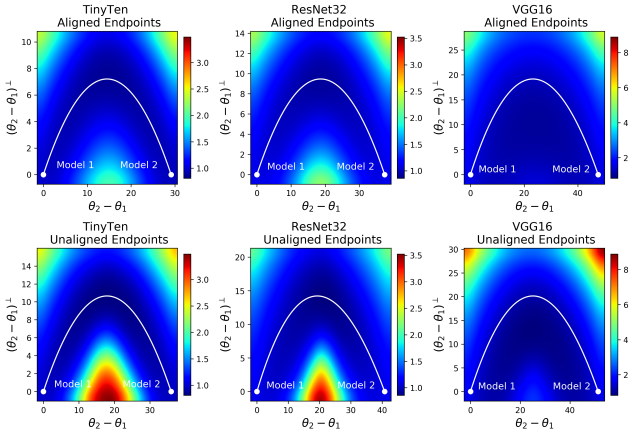
A.3 PLANES CONTAINING LEARNED BEZIER CURVES



(a) CIFAR100

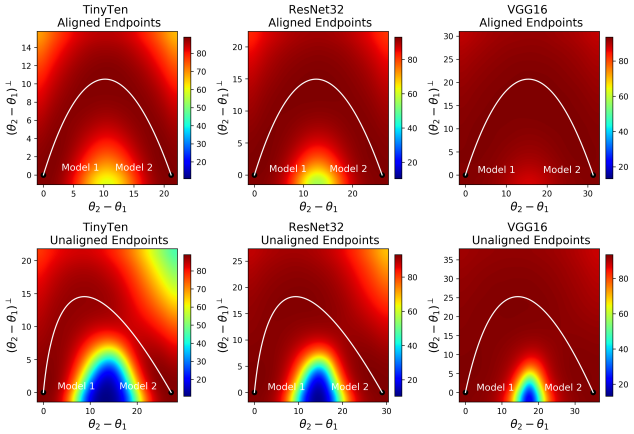


(b) CIFAR10

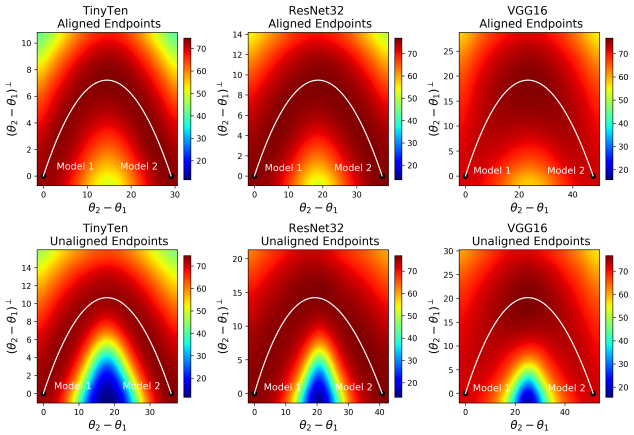


(c) STL10

Figure 12: Test loss on plane containing learned curve, $\boldsymbol{r}_\phi(t)$.



(a) CIFAR10



(b) STL10

Figure 13: Test accuracy on plane containing learned curve, $\boldsymbol{r}_\phi(t)$.

B ALGORITHMS

This section contains algorithms described in Section 2.

Data: Two trained models, $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$
Result: A curve, $\boldsymbol{r}_\phi$, connecting $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ along which loss is flat
Initialize $\boldsymbol{r}_\phi(t)$ as $\boldsymbol{\theta}_1 + t(\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1)$;
while *not converged* **do**
  for *batch in dataset* **do**
    sample point t_0 in $[0, 1]$;
    compute loss $L(\boldsymbol{r}_\phi(\mathrm{t}_0))$;
    optimization step on network $\boldsymbol{r}_\phi(\mathrm{t}_0)$ to update ϕ ;
  end
end

Algorithm 2: Curve Finding

C PROOFS

For the following proofs, we first establish and more rigorously define some terminology. We first discuss an important abuse of notation. For clarity the parameterized curve connecting networks

under some permutation $\boldsymbol{P}$ that has been written as $\boldsymbol{r}_\phi(t)$ will now sometimes be referred to as $\boldsymbol{r}(t;\phi,\boldsymbol{P})$.

Feed-forward neural networks In this section, we will be analyzing feed-forward neural networks. We let $\boldsymbol{X}_0 \in \mathbb{R}^{m_0 \times d}$ be the input to the neural network, d samples of dimension m_0. Then we let $\boldsymbol{W}_i \in \mathbb{R}^{m_i \times m_{i-1}}$ denote the network weights mapping from layer $l-1$ to layer l. Additionally, σ denotes the pointwise activation function. Then we can express the output of a feed-forward neural network, $\boldsymbol{Y}$, as:

$$\boldsymbol{Y} := \boldsymbol{W}_L\sigma \circ \boldsymbol{W}_{L-1}\sigma \circ \boldsymbol{W}_{L-2}...\sigma \circ \boldsymbol{W}_1\boldsymbol{X}_0 \tag{8}$$

To include biases, $\{\boldsymbol{b}_i\}_{i=1}^L$, we simply convert to homogeneous coordinates,

$$\hat{\boldsymbol{X}}_0 = \begin{bmatrix}\boldsymbol{X}_0 \\ 1\end{bmatrix}, \qquad \hat{\boldsymbol{W}}_i = \begin{bmatrix}\boldsymbol{W}_i & \boldsymbol{b}_i \\ \boldsymbol{0} & 1\end{bmatrix}, \qquad \hat{\boldsymbol{Y}} = \begin{bmatrix}\boldsymbol{Y} \\ 1\end{bmatrix} \tag{9}$$

In all proofs, these terms are interchangeable.

Huberized ReLU The commonly used ReLU function is defined as $\sigma(t) := \max(0,t)$. However, this function is not in C^1 and hence not locally Lipschitz differentiable. This makes conducting analysis with this function difficult. Thus, we will approach studying it through the lens of the huberized ReLU function, defined as:

$$\sigma_\delta(t) := \begin{cases} 0 & \text{for } t \le 0 \\ \frac{1}{2\delta}t^2 & \text{for } 0 \le t \le \delta \\ t - \frac{\delta}{2} & \text{for } \delta \le t \end{cases} \tag{10}$$

It is clear that σ_δ is a C^1 approximation of σ such that $||\sigma - \sigma_\delta||_\infty = \frac{\delta}{2}$. Using huberized forms of loss functions for analysis is a fairly common technique such as in (Xu et al., 2016) which studies huberized support vector machines.

C.1 Proof of Theorem 3.1

To prove this, we need that our problem meets the conditions required for local convergence of proximal alternating minimization (PAM) described in (Attouch et al., 2010). This requires the following:

1. Each term in the objective function containing only one primal variable is bounded below and lower semicontinuous.
2. Each term in the objective function which contains both variables is in C^1 and is locally Lipschitz differentiable.
3. The objective function satisfies the Kurdyka-Lojasiewicz (KL) property.

First we reformulate the problem so that it becomes unconstrained. Let χ denote the indicator function, where:

$$\chi_C(t) := \begin{cases} 0, & \text{for } t \in C \\ +\infty, & \text{otherwise} \end{cases} \tag{11}$$

Then equation 5 with added regularization is equivalent to:

$$\phi^*, \boldsymbol{P}^* = \arg\min_{\phi,\boldsymbol{P}} \quad Q(\phi,\boldsymbol{P}) + \mathcal{R}(\phi) + \sum_{l=1}^{L-1} \chi_{\Pi_{|K_l|}}(\boldsymbol{P}_l) \tag{12}$$

We now address each requirement for local convergence.

1. By the theorem, $\mathcal{R}$ is assumed to be bounded below and lower semicontinuous. It is easy to see from equation 11 that the sum of indicator functions are bounded below and lower semicontinuous.

2. Now we consider the form of the function, $Q(\phi, \boldsymbol{P})$. It has been defined as

$$\int_{t=0}^{1} \mathcal{L}(\boldsymbol{r}(t; \phi, \boldsymbol{P}))dt$$

We know that $\boldsymbol{r}(t; \phi, \boldsymbol{P})$ corresponds to a feed-forward neural network. Then Q can be expressed as:

$$\int_{t=0}^{1} \mathcal{L}\left(W_L(t; \phi, \boldsymbol{P})\sigma \circ W_{L-1}(t; \phi, \boldsymbol{P})...\sigma \circ W_1(t; \phi, \boldsymbol{P})X_0\right) dt \tag{13}$$

with weight matrices W_i and activation function σ. It becomes clear that for $Q(\phi, \boldsymbol{P})$ to be in C^1 and locally Lipschitz differentiable, the same must be true for $\mathcal{L}$, σ, and $\{W_i\}_{i=1}^{L}$. The first two are true as they are assumptions of the theorem. Since, r_ϕ is in C^1 and locally Lipschitz differentiable in the primal variables, then this is also true for all W_i. Thus, $Q(\phi, \boldsymbol{P})$ is in C^1 and locally Lipschitz differentiable.

3. To satisfy the KL property, it is enough for the objective function to be subanalytic (Attouch et al., 2010). Using similar logic from the last point, since $\mathcal{L}$, σ, and r_ϕ are piece-wise analytic, it follows that $Q(\phi, \boldsymbol{P})$ is also piece-wise analytic. $\mathcal{R}$ is assumed to be piece-wise analytic. The sum of indicator functions is well known to be a semi-algebraic function, and thus a subanalytic function. Additionally, all piece-wise analytic functions are subanalytic functions. As a sum of subanalytic functions, the objective function is a subanalytic function and satisfies the KL property.

C.2 Considering Rectified Networks

Theorem 3.1 does not extend to the class of rectified networks. However, we are still interested in contructing a sequence of iterates $\{\phi^k, \boldsymbol{P}^k\}$ such that the objective value, $\mathbb{E}_{t\sim U}[\mathcal{L}(\boldsymbol{r}(t; \phi^k, \boldsymbol{P}^k))]$, is monotonic decreasing. The following theorem will introduce a technique for constructing such a sequence.

Lemma C.1 ($\mathcal{L}$ restricted to possible network outputs is Lipschitz continuous)**.** *For a feed-forward neural network, assume that $\mathcal{L}$ is continuous and that the neural network input, X_0, is bounded. Additionally, assume that the spectral norm of all weights, $\{W_i\}_{i=1}^{L}$, is bounded above by K_W, and the activation function, σ, is continous with $||\sigma|| \leq 1$. Let S_Y denote the set of Y where*

$$\begin{aligned} Y &= \boldsymbol{W}_L\sigma \circ \boldsymbol{W}_{L-1}\sigma \circ \boldsymbol{W}_{L-2}...\sigma \circ \boldsymbol{W}_1\boldsymbol{X}_0 \\ &\textit{such that} \quad ||\boldsymbol{W}_i||_2 \leq K_W \quad \forall i \in \{1, 2, ..., L\} \end{aligned} \tag{14}$$

Then $\mathcal{L}$ restricted to the set S_Y is Lipschitz continuous with some Lipschitz constant K.

Proof. Since X_0 is bounded, it follows that there exists some constant K_X such that $||X_0|| \leq K_X$. Since, the spectral norm of W_1 is bounded above by K_W, it is easy to see that $||W_1X_0|| \leq K_WK_X$. Now since the pointwise activation function is a non-expansive map, it immediately follows that $||\sigma \circ W_1X_0|| \leq K_WK_X$. Following this process inductively, we see that the network output, Y, is bounded and that:

$$||Y|| \leq K_W^L K_X \tag{15}$$

Since Y is arbitrary, it follows that this is a bound for S_Y. Then we can restrict $\mathcal{L}$ to the ball in $\mathbb{R}^{m_L \times d}$ of radius $K_W^L K_X$. This ball is compact and $\mathcal{L}$ is continuous, so it follows that $\mathcal{L}$ restricted to this ball is Lipschitz continuous. Thus, there exists some Lipschitz constant K. Clearly, S_Y is contained in this ball. Therefore, $\mathcal{L}$ is Lipschitz continuous on the set of all possible network outputs with Lipschitz constant K. □

Let θ_1 and θ_2 be feed-forward neural networks with ReLU activation function. Assume that $\mathcal{L}$ and r_ϕ are piece-wise analytic functions in C^1 and locally Lipschitz differentiable. Assume that $\mathcal{R}$ is piece-wise analytic, lower semi-continuous, and bounded below. Assume that the maximum network width at any layer is M units. Additionally, assume that the network weights have a spectral norm bounded above by K_W, and that this is a hard constraint when training the networks.

Create the parameterized curve $\boldsymbol{r}_\delta(t; \phi, \boldsymbol{P})$ by substituting the huberized ReLU function, σ_δ, into all ReLU functions in $\boldsymbol{r}(t; \phi, \boldsymbol{P})$. We refer to the objective values associated with these curves as $Q_\delta(\phi, \boldsymbol{P})$ and $Q(\phi, \boldsymbol{P})$ respectively.

Theorem C.2 (Monotonic Decreasing Sequence for Rectified Networks)**.** *For a feed-forward network, assume the above assumptions have been met. Additionally, assume that X_0 is bounded, so that $\mathcal{L}$ restricted to the set of possible network outputs is Lipschitz continuous with Lipschitz constant K_L by Lemma C.1. Now generate the sequence $\{\phi^k, \boldsymbol{P}^k\}$ by solving equation 6 for $r_\delta(t; \phi, \boldsymbol{P})$. On this sequence impose the additional stopping criteria that*

$$\frac{1}{2\nu_\phi}||\phi^{k+1} - \phi^k||_2^2 + \frac{1}{2\nu_P}||\boldsymbol{P}^{k+1} - \boldsymbol{P}^k||_2^2 \geq K_L\sqrt{M}\frac{\delta}{2}\sum_{i=1}^{L-1} K_W^i \qquad \forall k \geq 0. \tag{16}$$

Then, the sequence of curves $\boldsymbol{r}(t; \phi^k, \boldsymbol{P}^k)$ connecting rectified networks has monotonic decreasing objective value.

Proof. First we consider the approximation error from replacing σ with σ_δ. It is straightforward to see that

$$\max_t |\sigma(t) - \sigma_\delta(t)| \leq \frac{\delta}{2}. \tag{17}$$

Then it follows that for any input $\boldsymbol{x}$,

$$||\sigma \circ W_1\boldsymbol{x} - \sigma_\delta \circ W_1\boldsymbol{x}||_2 \leq \sqrt{M}\frac{\delta}{2}.$$

Since the spectral norm of W_i are bounded above by K_W, then we see that

$$||W_2\sigma \circ W_1\boldsymbol{x} - W_2\sigma_\delta \circ W_1\boldsymbol{x}||_2 \leq K_W\sqrt{M}\frac{\delta}{2}.$$

Now notice that

$$\begin{aligned} ||\sigma \circ W_2\sigma \circ W_1\boldsymbol{x} - \sigma_\delta \circ W_2\sigma_\delta \circ W_1\boldsymbol{x}|| \leq\ & ||\sigma \circ W_2\sigma \circ W_1\boldsymbol{x} - \sigma \circ W_2\sigma_\delta \circ W_1\boldsymbol{x}|| \\ & + ||\sigma \circ W_2\sigma_\delta \circ W_1\boldsymbol{x} - \sigma_\delta \circ W_2\sigma_\delta \circ W_1\boldsymbol{x}||. \end{aligned} \tag{18}$$

Since the ReLU function is a non-expansive map, it must be that the first term is bounded above by the previous error, $K_W\sqrt{M}\frac{\delta}{2}$. The second term corresponds once again to the error associated with the huberized form of the ReLU function, $\sqrt{M}\frac{\delta}{2}$. Thus the total error can be bounded by $(K_W + 1)\sqrt{M}\frac{\delta}{2}$.

Following this inductively, it can be seen that the this error grows geometrically with the number of layers. Additionally, the loss function is Lipschitz continuous when restricted to the set of possible network outputs. So we find the following bounds:

$$\begin{aligned} ||Y - Y_\delta|| &\leq \sqrt{M}\frac{\delta}{2}\sum_{i=1}^{L-1} K_W^i \\ ||\mathcal{L}(Y) - \mathcal{L}(Y_\delta)|| &\leq K_L\sqrt{M}\frac{\delta}{2}\sum_{i=1}^{L-1} K_W^i \end{aligned} \tag{19}$$

Since this bound on the spectral norm must hold for all points on the curve, it follows that $||Q(\phi, \boldsymbol{P}) - Q_\delta(\phi, \boldsymbol{P})||$ is also bounded above by the bound in equation 19.

Then let $\{\phi^k, \boldsymbol{P}^k\}$ be the sequence generated by solving equation 6 using the curve r_δ. σ_δ is a piecewise analytic function in C^1 and is locally Lipschitz differentiable. Additionally, the spectral norm constraint on the weights is semi-algebraic and bounded below, so Theorem 3.1 can be applied. It then follows that

$$\begin{aligned} &Q(\phi^{k+1}, \boldsymbol{P}^{k+1}) + \mathcal{R}(\phi^{k+1}) + \frac{1}{2\nu_\phi}||\phi^{k+1} - \phi^k||_2^2 + \frac{1}{2\nu_P}||\boldsymbol{P}^{k+1} - \boldsymbol{P}^k||_2^2 \\ &\leq Q(\phi^k, \boldsymbol{P}^k) + \mathcal{R}(\phi^k) + K_L\sqrt{M}\delta\sum_{i=1}^{L-1} K_W^i, \qquad \forall k \geq 0 \end{aligned} \tag{20}$$

Thus, $\boldsymbol{r}(t; \phi^k, \boldsymbol{P}^k)$ is a sequence of curves, connecting rectified networks, with monotonic decreasing objective value as long as

$$\frac{1}{2\nu_\phi}||\phi^{k+1} - \phi^k||_2^2 + \frac{1}{2\nu_P}||\boldsymbol{P}^{k+1} - \boldsymbol{P}^k||_2^2 \geq K_L\sqrt{M}\delta \sum_{i=1}^{L-1} K_W^i \qquad \forall k \geq 0$$

Since the above equation is a stopping criterion introduced in the theorem statement, it follows that we have constructed a sequence of curves, connecting rectified networks, with monotonic decreasing objective value. □

D RESIDUAL NETWORK ALIGNMENT

Algorithm 1 applies to networks with a typical feed-forward structure. In this section, we discuss how we compute alignments for the ResNet32 architecture as it is more complicated. It is important to align networks such that the network structure is preserved and network activations are not altered. In the context of residual networks, special consideration must be given to skip connections.

Consider the formulation of a basic skip connection,

$$\boldsymbol{X}_{k+1} = \sigma \circ (\boldsymbol{W}_{k+1}\boldsymbol{X}_k) + \boldsymbol{X}_{k-1} \tag{21}$$

In this equation, we can see that $\boldsymbol{X}_{k+1}$ and $\boldsymbol{X}_{k-1}$ share the same indexing of their units. This becomes clear when you consider permuting the hidden units in $\boldsymbol{X}_{k-1}$ without permuting the hidden units of $\boldsymbol{X}_{k+1}$. It is impossible to do so without breaking the structure of the equation above, where there is essentially the use of an identity mapping from $\boldsymbol{X}_{k-1}$ to $\boldsymbol{X}_{k+1}$.

We consider a traditional residual network that is decomposed into residual blocks. In each block the even layers have skip connections while the odd layers do not. So, we compute the alignment as usual for odd layers. For all even layers within a given residual block, we determine a shared alignment. We do this by solving the assignment problem for the average of the cross-correlation matrix over the even layers in that residual block.

E ALIGNMENT ALONG CURVES

Clearly, alignment is a useful method for learning better flat loss curves between models. An interesting question is how curve finding itself relates to alignment. Until now, we have only considered the alignment between the endpoint models, $\boldsymbol{r}(0)$ and $\boldsymbol{r}(1)$. Now, we consider how points along the curve, $\boldsymbol{r}(t)$, align to the endpoints. To study this numerically, we will use the curve midpoint $\boldsymbol{r}(0.5)$. From Figure 4, we see that this is the point on the quadratic Bezier curve that is roughly linearly connected to both endpoints.

E.1 CORRELATION SIGNATURE

First, we consider how the correlation signature changes along the curve. Figure 14 displays the correlation signature between the curve midpoint and each endpoint in blue. To gain a better understanding of this signature, we require some context. Thus, the correlation signature between the linear midpoint and each endpoint is displayed in green. This allows us to understand how the correlation signature changes over the curve finding optimization. Additionally, we display the correlation signature between the curve midpoint and each endpoint, where the midpoint has been aligned to the given endpoint, in yellow. This essentially gives us context on how highly the midpoint is aligned to each endpoint. This is because the yellow curve acts as an upper bound for the blue curve.

There are several observations to be made about Figure 14. The correlation signature between the endpoint and the curve midpoint is fairly high. For unaligned endpoints, the correlation is only slightly lower than the signature computed when the curve midpoint is aligned to the endpoint. In the case where the endpoints are aligned, the signatures are seen to coincide. This suggests that the curve finding algorithm is finding the quadratic curve along which similar feature representations are being interpolated.

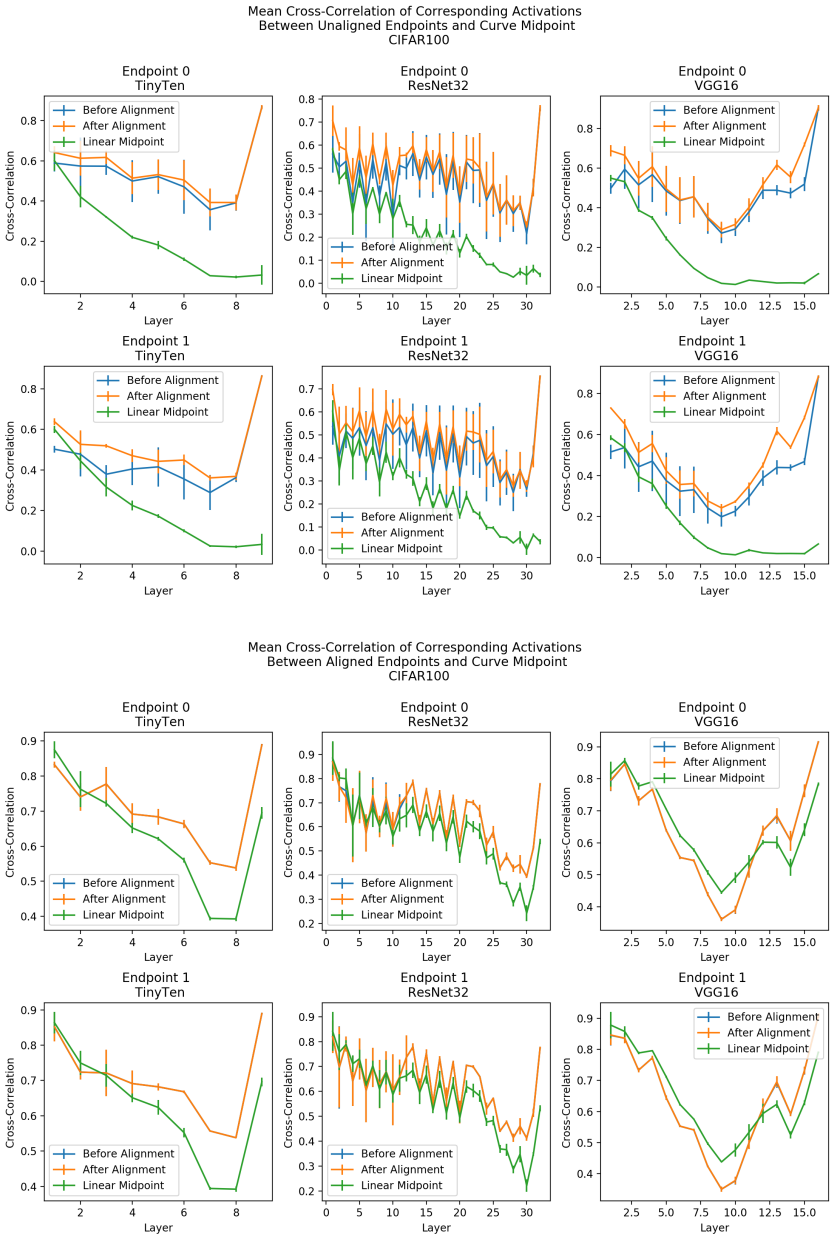


Figure 14: The mean cross-correlation between units in the curve midpoint model and each endpoint model. For context, the mean cross-correlation between the linear midpoint and each endpoint is displayed. Additionally, the mean cross-correlation between the curve midpoint and each endpoint after being aligned to the respective endpoint is displayed.

Concerning the linear midpoint, the correlation at the linear midpoint decays to 0 when endpoints are unaligned as the network goes deeper. When endpoints are aligned, the correlation signature at the linear midpoint is similar to the correlation signature at the curve midpoint. Since these linear connections between the endpoints are the initializations for the curve finding algorithm, this gives some intuition on how alignment works to give a better initialization.