How to Characterize The Landscape of Overparameterized Convolutional Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

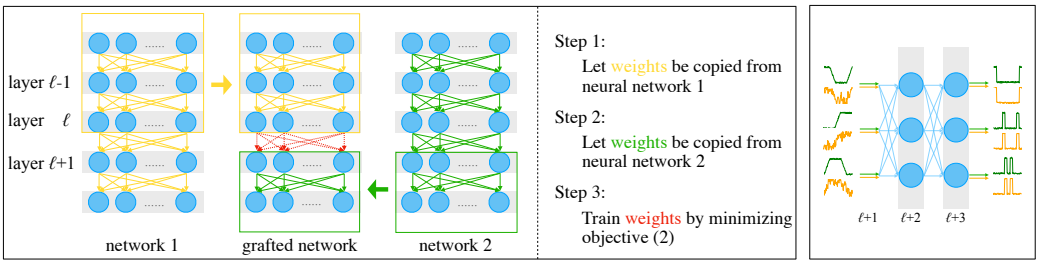
For many initialization schemes, parameters of two randomly initialized deep neural networks (DNNs) can be quite different, but feature distributions of the hidden nodes are similar at each layer. With the help of a new technique called *neural network grafting*, we demonstrate that even during the entire training process, feature distributions of differently initialized networks remain similar at each layer. In this paper, we present an explanation of this phenomenon. Specifically, we consider the loss landscape of an overparameterized convolutional neural network (CNN) in the continuous limit, where the numbers of channels/hidden nodes in the hidden layers go to infinity. Although the landscape of the overparameterized CNN is still non-convex with respect to the trainable parameters, we show that very surprisingly, it can be reformulated as a convex function with respect to the feature distributions in the hidden layers. Therefore by reparameterizing neural networks in terms of feature distributions, we obtain a much simpler characterization of the landscape of overparameterized CNNs. We further argue that training with respect to network parameters leads to a fixed trajectory in the feature distributions.

1 Introduction

A good characterization of the landscape is very important for understanding remarkable successes of DNNs in various domains including computer vision, nature language processing and speech. There have been a lot of research [29, 21, 24, 20, 26, 8] analyzing the landscape of deep neural network in parameter space. On one hand, even one-hidden-layer neural network has complex landscape with exponential number of local minima and saddle points. On the other hand, some interesting empirical phenomena like *mode connectivity*, that low-cost solutions of DNN could be connected by simple path in parameter space, have been found, indicating that the loss landscape is not as complex as we expected. These two conclusions of the landscape in parameter space seem to be contradictory.

We believe that the loss landscape should be investigated in the feature distribution space directly instead of the parameter space as existing studies have done due to the excellent capability of DNN to learn effective feature representations. But our understanding on the feature distribution learned by DNNs is still limited. For example, even to the basic question of to what extent two feature representations learned by different DNNs are essentially the same, various studies such as [23, 32, 19] reached different conclusions due to the redundancy in the feature representations comes from the special properties of DNNs, e.g., the permutation&scale invariance of the trainable parameters, the truncation operation in ReLU, etc. Two feature representations that look quite different at current layer could be essentially the same. As the example illustrated in Fig. 1(b), the difference may have little effect on the features in subsequent layers and could be alleviated, or even eliminated completely after passing the linear transformations and the activation functions ReLU in the subsequent layers. Although several remedies are proposed in recent studies, the redundancy

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.



(a) (b)

Figure 1: (a) The flowchart of neural network grafting. (b) An example of redundancy. The yellow and green curves left represent the features in layer $\ell + 1$ learned in two nets. They look different but are essential similar since they become the same after passing two more layers.

cannot be completely eliminated and different amounts of redundancy may lead to inconsistent conclusions. This discourages the researchers from investigating DNNs in feature distribution space.

In this paper, we start from the problem of inconsistent conclusions above and propose an effective technique named *neural network grafting* (NNG) for comparing feature distributions (see Section 4). We consider the most direct way to identify whether the feature representations learned in two hidden layers of two networks are essentially the same. The key idea is to check whether the representation learned by one network can be used in the other one without significant sacrifice in accuracy, i.e., whether one of these two networks can be grafted onto the other at that hidden layer (Fig. 1(a)). Using NNG, we find that feature distributions learned by two networks with the same architecture but different initilizations are almost the same during the *whole* training process. That means their solution paths, if we view them in the aspect of feature distributions, are the same, while they seem chaotic and quite different under the view of trainable parameters (see Section 6.1).

Our surprising finding on the uniqueness of the solution path implies that NNs become much easier to understand if we view them in the aspect of feature distributions instead of trainable parameters. This motivates us to reparameterize NNs with respect to the feature function distributions to obtain a simpler loss landscape. Specifically, we first propose a new method to reformulate a CNN as an approximation to a continuous CNN, which is obtained by letting the number of channels/hidden nodes in each hidden layer go to infinity. We then extend the emerging technique for reformulating fully connected NNs [4, 5] to continuous CNN to reparameterize it with respect to feature distributions (See Section 5). We show that although the loss of continuous CNN is still non-convex with respect to the trainable parameters, it becomes convex with respect to the feature distributions. Therefore, we obtain a much simper characterization for the loss landscape.

Although our convexity is reformulated in the feature distribution space instead of the trainable parameter space, it has significant implications on NN optimization. In fact, it can be shown that under suitable conditions, the training algorithms (e.g., SGD) of DNNs converge to a solution that is a stationary point of the convex reformulation, as shown in Section B of the Appendix. This excludes bad local minimum solutions for DNN optimization. Moreover, the convexity and unique solution path imply that CNNs are much simpler if we view them in the feature distribution space. All these demonstrate the value of studying DNNs in feature distribution space.

Notations. For a positive integer n, we let $[n]$ to be the set $\{1, 2, \ldots, n\}$. We denote $\|x\|_1$ and $\|x\|$ to be the ℓ_1 and ℓ_2 norm of a vector x in $\mathbb{R}^d$. For any $k \in [d]$, we let x_k be the value of the k-th dimension of x and denote $vec(w)$ to be the operator for reshaping a matrix or tensor w into a vector.

2 Related Work

2.1 Characterization of DNNs' Landscape

One main purpose of the studies on landscape characterization is to explain why DNNs with massive parameters can be trained efficiently in practice. They mainly explore the following subjects:

Geometry of loss surface The training of DNNs depends highly on the network architecture, optimization techniques (e.g., batch normalization (BN)) and some other considerations. Some researchers try to study the impacts of these factors on the geometry of the loss surface. They [8, 21] show that the loss landscapes quickly transit from being nearly convex to being highly chaotic when the network becomes deeper, and residual connections promote surface smoothness and prevent

the explosion of non-convexity, which verifies that skip connection is essential for extremely deep networks. Moreover, [28] demonstrates that the effectiveness of BN comes from its effects on the smoothness of the landscape instead of controlling *covariate shift*.

Properties of local minima These studies attempt to understand the phenomenon observed in practice that local minima found by training algorithms are nearly always good. They show that in idealized settings, the local minima of DNNs have similar loss values [12, 11]. To be precise, [18] proved that linear networks have no sub-optimal local minima. [30] shows that for deep residual network with a single output unit, the objective values of all local minima are no higher than what can be obtained with a linear predictor. However, these existing studies are restricted to strong assumptions of model simplifications, such as linear activation. For nonlinear DNNs in practice, theoretical properties of local minima are still unknown.

Mode Connectivity The phenomenon of mode connectivity implies that the optima of DNNs may have special and simple structures. Some efforts have been made to understand it theoretically. [26] shows that for overparameterized DNNs with piece-wise linear activations, the sublevel sets of their loss functions are connected and bounded. For nonlinear DNNs, [20, 27, 31] prove the existence of these low-cost paths under the assumptions such as the DNNs are dropout stable or noise stable.

Despite the above breakthroughs on characterizing the landscapes of DNNs, many empirical phenomena of DNNs are still poorly understood. Some studies [32, 19] even had inconsistent conclusions, e.g., whether randomly initialized DNNs with the same architecture learn the same representation.

2.2 Overparameterized Deep Learning Theory

The theoretical work about overparameterized NN, motivated by its great empirical success, can be roughly divided into two categories, i.e. mean field limit and neural tangent kernel (NTK) limit, based on whether the network parameters change a lot during training. The NTK limit [17, 10, 14, 22, 13, 2, 3, 6, 7] considers a tiny neighborhood of the initialization, and with an appropriate scaling that approaches infinity, it can be shown that the global minimum is achieved within the tiny neighborhood (called NTK regime) that becomes infinitesimal in the limit. Therefore the neural network can be linearized around the initialization, and the resulting formulation becomes convexity. However, it is observed that NNs within the NTK region does not perform well compared to fully trained NNs using standard methods, which always go out of the NTK regime. Therefore NTK cannot be used to study the standard NN training process and the associated landscape characterization, as we are interested in doing in this paper. To overcome this discrepancy between theory and practice, the mean field limit [24, 9, 25] has been proposed. This is also the approach most related to this paper, because we also consider the standard training process that goes out of the NTK regime. The original mean field framework works with distributions over NN parameters, which is difficult to extend to deep networks. This paper considers the convex reformulation idea in [4, 5], where overprameterized NNs are parameterized using feature distributions of the hidden neurons. We investigate the empirical justifications and their relationship to the theory of convex reformulation.

3 Basics

In this section, we give the detailed formulation of standard CNN, and hereafter we refer to it as discrete CNN in contrast to our continuous CNN in Section 5.1. We notice that fully connected layer can be viewed as a convolutional layer by treating each hidden node as a special channel valued in $\mathbb{R}^{1\times 1}$. Therefore, we can write the fully connected and convolutional layers into a unified form to simplify the formulas. Below we define a $(L+1)$-layer CNN mapping an input into a real valued vector in $\mathbb{R}^K$. The number of hidden nodes/channels in each hidden layer ℓ is denoted to be $m^{(\ell)}$.

We denote all the trainable parameters in the network as $\theta = (w^{(1)}, \dots, w^{(L)}, u)$, which will be specified later. First, given an input $x \in \mathbb{R}^{W^{(0)}\times H^{(0)}\times C}$, let $m^{(0)} = C$ and define the nodes in the input later as $\hat{f}_j^{(0)}(\theta; x) = x_{:,:,j}$ with $j \in [m^{(0)}]$. Then, for the hidden layer $\ell \in [L]$, define the output of its j-th hidden node/channel $\hat{f}_j^{(\ell)}(\theta; x) \in \mathbb{R}^{W^{(\ell)}\times H^{(\ell)}}$ with $j \in [m^{(\ell)}]$ as

$$\hat{f}_j^{(\ell)}(\theta; x) = h^{(\ell)}\left(\hat{g}_j^{(\ell)}(\theta; x)\right) \text{ with } \hat{g}_j^{(\ell)}(\theta; x) = \frac{1}{m^{(\ell-1)}} \sum_{k=1}^{m^{(\ell-1)}} \hat{f}_k^{(\ell-1)}(\theta; x) * w_{j,k}^{(\ell)},$$

where $w_j^{(\ell)} = [w_{j,1}^{(\ell)}, \ldots, w_{j,m^{(\ell-1)}}^{(\ell)}] \in \mathbb{R}^{a^{(\ell)} \times b^{(\ell)} \times m^{(\ell-1)}}$ is the j-th kernel, $h^{(\ell)} = \mathcal{P}^{(\ell)} \circ \psi^{(\ell)}$ with $\psi^{(\ell)}$ being the activation function and $\mathcal{P}^{(\ell)}$ being the pooling operator if layer ℓ is followed by a pooling layer otherwise $\mathcal{P}^{(\ell)}$ is an identity mapping. For the top layer, we denote u_j to be a vector valued in $\mathbb{R}^K$, then the output $\hat{f}(\theta; x) \in \mathbb{R}^K$ can be defined as $\hat{f}(\theta; x) = \frac{1}{m^{(L)}} \sum_{j=1}^{m^{(L)}} u_j \hat{f}_j^{(L)}(\theta; x)$. The goal of training a CNN is to minimize the following objective function:

$$\hat{Q}(w, u) = J(\hat{f}) + \hat{R}(w, u), \text{ with } \hat{R}(w, u) = \sum_{\ell=1}^{L} \lambda^{(\ell)} \hat{R}^{(\ell)}(w^{(\ell)}) + \lambda^{(u)} \hat{R}^{(u)}(u), \tag{1}$$

where $J(\hat{f}) = \mathbb{E}_{x,y} \phi(\hat{f}(x), y)$ with $\phi(\cdot, \cdot)$ being the loss function and $\hat{R}(w, u)$ is the the regularizer with $\lambda^{(\ell)}$ and $\lambda^{(u)}$ being its non-negative hyper-parameters. In this paper, we focus on the following $\ell_{1,2}$ regularier proposed in [4] due to its ability in learning efficient features:

$$\hat{R}^{(\ell)}(w^{(\ell)}) = \frac{1}{m^{(\ell-1)}} \sum_{k=1}^{m^{(\ell-1)}} r_2 \left(\frac{1}{m^{(\ell)}} \sum_{j=1}^{m^{(\ell)}} r_1 \left(w_{j,k}^{(\ell)} \right) \right), \quad \hat{R}^{(u)}(u) = \frac{1}{m^{(L)}} \sum_{j=1}^{m^{(L)}} r^{(u)}(u_j),$$

where $r_1(w) = \|vec(w)\|_1$, $r_2(w) = w^2$ and $r^{(u)}(u) = \|u\|_2^2$.

4 Neural Network Grafting

In this section, we introduce a new method called *neural network grafting* (NNG) to check whether two different CNNs learn similar feature representations at each layer. The intuition is that if two networks θ_1 and θ_2 learn similar feature representations at layer ℓ, then the features of θ_1 at layer ℓ can be used in θ_2 without significant error increasing, that is, the first ℓ layers of θ_1 with the trained parameters in them can be grafted onto the last $L - \ell$ layers of θ_2 by only training the parameters at the joint. As shown in Fig.1(a), NNG is conducted by first copying parameters from θ_1 and θ_2 to construct a new network $\tilde{\theta}_{(1,2)} = (\theta_1^{(1)}, \cdots, \theta_1^{(\ell)}, \tilde{\theta}_{(1,2)}^{(\ell+1)}, \theta_2^{(\ell+2)}, \cdots, \theta_2^{(L)}, u_2)$ and then training the parameters $\tilde{\theta}_{(1,2)}^{(\ell+1)}$ to align the neurons at the joint by minimizing the following feature matching loss:

$$\min_{\tilde{\theta}_{(1,2)}^{(\ell+1)}} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{m^{(\ell+1)}} \|\hat{g}_k^{(\ell+1)}(\tilde{\theta}_{(1,2)}; x_i) - \hat{g}_k^{(\ell+1)}(\theta_2; x_i)\|_2^2. \tag{2}$$

Detailed steps are given in Alg.1 in Appendix. We could imagine that if the representation at layer ℓ in θ_1 is similar to that of θ_2, then the loss in (2) would be low after grafting, and the final validation error of $\tilde{\theta}_{(1,2)}$ will be similar to that of θ_2 and otherwise it will be significantly higher than θ_2.

Remark 1. *In our grafting process, we minimize the feature difference at layer $\ell + 1$ instead of minimize the final loss to compare the feature representations more directly.*

Notice that both our NNG and previous metrics (e.g., [23], [19] and [32]) can address the case when two feature representations are quite different but can be matched after a linear transformation. However, our NNG can address at least another important case that existing metrics cannot deal with. To be precise, if two feature representations are quite different but the difference is not important for subsequent layers, they would be regarded to be different by previous metrics while they are essentially similar. Such difference could be alleviated and even eliminated completely by, for example, the small weight and truncation of ReLU in subsequent layers. An example is shown in Fig. 1(b), where green and yellow curves represent feature representations of network θ_2 and $\tilde{\theta}_{(1,2)}$ respectively. Detailed quantitative evaluation for it is presented in Fig. 2(c).

5 Landscape Characterization

In this section, we present our landscape characterization method. We first reformulate an overparameterized CNN as an approximation to a continuous CNN, which is obtained by letting the number of channels/hidden nodes in each hidden layer go to infinity (Section 5.1). Then we reformulate it with respect to the feature distributions to obtain a simpler landscape characterization (Section 5.2).

5.1 Continuous CNN

Following the mean-field limit of overparameterized NN, below we give the definition of continuous CNN by letting the number of the channels/hidden nodes in each layer of a discrete CNN go to infinity. Firstly, for the input layer we denote $\mathcal{Z}^{(0)} = [C]$ to be its channel space corresponding to the C channels of the input x. We let $\rho^{(0)}$ be a probability measure on $\mathcal{Z}^{(0)}$ and unless otherwise specified, in this paper, we fix $\rho^{(0)}$ to be a uniform distribution on $\mathcal{Z}^{(0)}$. For each $z^{(0)} \in \mathcal{Z}^{(0)}$, we let

$$f^{(0)}(\rho, z^{(0)}; x) = x_{:,:,z^{(0)}}.$$

For layer $\ell \in [L]$, let $\mathcal{Z}^{(\ell)}$ be the space of the hidden nodes/channels in layer ℓ equipped with probability measure $\rho^{(\ell)}$. Denote $w(z^{\ell}, z^{(\ell-1)}) \in \mathbb{R}^{a^{(\ell)} \times b^{(\ell)}}$ to be the weight/kernel connecting the nodes/channels $z^{(\ell-1)} \in \mathcal{Z}^{(\ell-1)}$ and $z^{(\ell)} \in \mathcal{Z}^{(\ell)}$, the output of hidden nodes/channels can be defined as

$$f^{(\ell)}(\rho, z^{(\ell)}; x) = h^{(\ell)}\left(g^{(\ell)}(\rho, z^{(\ell)}; x)\right),$$

$$\text{where } z^{(\ell)} \in \mathcal{Z}^{(\ell)} \text{ and } g^{(\ell)}(\rho, z^{(\ell)}; x) = \int f^{(\ell-1)}(\rho, z^{(\ell-1)}; x) * w(z^{(\ell)}, z^{(\ell-1)}) d\rho^{(\ell-1)}(z^{(\ell-1)}).$$

At last, by denoting $u(z^{(L)}) \in \mathbb{R}^K$ to be the weight connecting the node $z^{(L)}$ and the final output, then the final (fully connected) output layer is given by

$$f(\rho, u; x) = \int u(z^{(L)}) f^{(L)}(\rho, z^{(L)}; x) d\rho^{(L)}(z^{(L)}).$$

The objective function then takes form of

$$Q(\rho, u, w) = J(f) + R(\rho, u, w) \tag{3}$$

$$\text{where } R(\rho, u, w) = \sum_{\ell=1}^{L} \lambda^{(\ell)} R^{(\ell)}(w, \rho, w) + \lambda^{(u)} R^{(u)}(\rho, u, w)$$

$$\text{with } R^{(\ell)}(\rho, w) = \int r_2\left(\int r_1\left(w(z^{(\ell)}, z^{(\ell-1)})\right) d\rho^{(\ell)}(z^{(\ell)})\right) d\rho^{(\ell-1)}(z^{(\ell-1)}),$$

$$R^{(u)}(\rho, u) = \int r^{(u)}\left(u(z^{(L)})\right) d\rho^{(L)}(z^{(L)}).$$

Discussion: Discrete CNN can be constructed from a continuous CNN by sampling $m^{(\ell)}$ hidden nodes/channels $\{z_1^{(\ell)}, \ldots, z_{m^{(\ell)}}^{(\ell)}\}$ for each layer ℓ according to the probability measure $\rho^{(\ell)}$ and letting the parameter connecting $z_i^{(\ell)}$ and $z_j^{(\ell-1)}$ be $w(z_i^{(\ell)}, z_j^{(\ell-1)})$ for all $i \in [m^{(\ell)}]$ and $j \in [m^{(\ell-1)}]$. From [4] we know that when $m^{(\ell)} \to \infty$ for all $\ell \in [L]$, the final output of discrete CNN converges to that of the continuous one. Therefore, discrete CNN is an approximation to a continuous CNN.

5.2 Reformulate Continuous CNNs

The uniqueness of the solution path, i.e., the unique feature distribution evolution path, discovered by our technique NNG (Sections 4 and 6.1) implies that DNNs could be much simpler if we view them with respect to feature distributions. Therefore, we reformulate continuous CNN with respect to the distribution of pre-activation of neurons, i.e., feature distribution, to eliminate the redundancy caused by weights and obtain a simpler loss landscape characterization, which is inspired from the emerging techniques [4, 5] for reformulating fully connected DNNs. First, we define

$$\mathcal{V}^{(0)} = \{v^{(0)} : v^{(0)} = [x_{:,:,i}^1, \ldots, x_{:,:,i}^N], i \in [C]\}, \text{ and } p^{(0)}(v^{(0)} = [x_{:,:,i}^1, \ldots, x_{:,:,i}^N]) = 1/C, i \in [C].$$

For $\ell \in [L]$, we denote $v^{(\ell)}$ to be the pre-activation of neuron $z^{(\ell)}$ at N training samples, i.e., $v^{(\ell)} = [g^{(\ell)}(\rho, z^{(\ell)}; x_1), \ldots, g^{(\ell)}(\rho, z^{(\ell)}; x_N)] \in \mathbb{R}^{W^{(\ell)} \times H^{(\ell)} \times N}$ and let $\mathcal{V}^{(\ell)}$ be the space of $v^{(\ell)}$,i.e., $\mathcal{V}^{(\ell)} = \{v^{(\ell)} : z^{(\ell)} \in \mathcal{Z}^{(\ell)}\}$. We denote $p^{(\ell)}(v^{(\ell)})$ to be the probability density function of $v^{(\ell)}$. For any $v^{(\ell-1)}$ and $v^{(\ell)}$, we let $w(v^{(\ell)}, v^{(\ell-1)}) \in \mathbb{R}^{a^{(\ell)} \times b^{(\ell)}}$ be the weight connecting them. Problem (3) can then be reformulated as

$$\min_{w,p,u} \frac{1}{N} \sum_{i=1}^{N} \phi\left(f(x_i), y_i\right) + R(w, p, u) \tag{4}$$

$$s.t.\ 1)\ \sum_{v^{(0)}\in\mathcal{V}^{(0)}} v_i^{(0)} * w(v^{(1)}, v^{(0)})p^{(1)}(v^{(1)})p^{(0)}(v^{(0)}) = p^{(1)}(v^{(1)})v_i^{(1)}, i \in [N];$$

$$2)\ \int h^{(\ell-1)}(v_i^{(\ell-1)}) * w(v^{(\ell)}, v^{(\ell-1)})p^{(\ell)}(v^{\ell})p^{(\ell-1)}(v^{(\ell-1)})dv^{(\ell-1)} = v_i^{(\ell)}p^{(\ell)}(v^{\ell}), \ell \geq 2, i \in [N];$$

$$3)\ \int p^{(\ell)}(v^{(\ell)})dv^{(\ell)} = 1 \text{ and } p^{(\ell)}(v^{(\ell)}) \geq 0,$$

where $f(x_i)$ and $R(w, p, u)$ are defined as

$$f(x_i) = \int h^{(L)}(v_i^{(L)})u(v^{(L)})p^{(L)}(v^{(L)})dv^{(L)},\ R(w,p,u) = \sum_{\ell=1}^{L} \lambda^{(\ell)}R^{(\ell)}(w,p) + \lambda^{(u)}R^{(u)}(u,p),$$

$$\text{with } R^{(1)}(w,p) = \sum_{v^{(0)}\in\mathcal{V}^{(0)}} r_2\left(\int r_1\left(w(v^{(1)}, v^{(0)})p^{(0)}(v^{(0)})p^{(1)}(v^{(1)})\right)dv^{(1)}\right)/p^{(0)}(v^{(0)}),$$

$$R^{(\ell)}(w,p) = \int r_2\left(\int r_1\left(w(v^{(\ell)}, v^{(\ell-1)})p^{(\ell-1)}(v^{(\ell-1)})p^{(\ell)}(v^{(\ell)})\right)dv^{(\ell)}\right)/p^{(\ell-1)}(v^{(\ell-1)})dv^{(\ell-1)},$$

$$\ell \geq 2 \text{ and } R^{(u)}(u,p) = \int r^{(u)}\left(u(v^{(L)})p^{(L)}(v^{(L)})\right)/p^{(L)}(v^{(L)})dv^{(L)}.$$

The weights w in problem (4) are determined by $p^{(\ell)}$ via a series of constraints in 1) and 2), hence the network can be regarded as being parameterized by $p^{(\ell)}$ without w. Therefore, the redundancy from w in networks parameterized via w is eliminated. One issue with Problem (4) is that it is still non-convex due to its non-convex constraints. Fortunately, we find that by reparameterizing the above colored items according to Eqn.(5), the constraints above become either convex or linear, and then problem (4) can be rewritten into a simpler convex form. We summarized the main point in the following theorem and details are presented in Appendix A.

Theorem 1. *If we change the variables u and w as follows:*

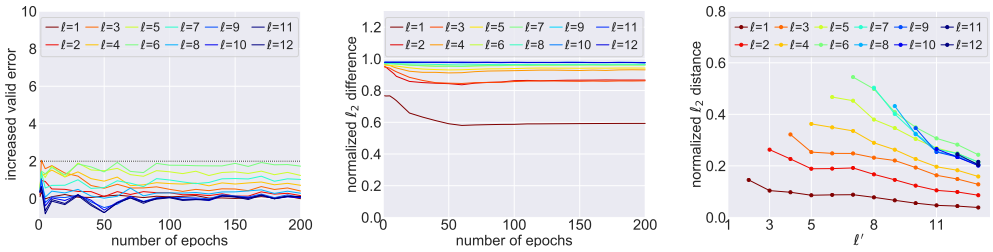
$$\begin{cases} \text{(i) } \tilde{w}(v^{(\ell)}, v^{(\ell-1)}) = w(v^{(\ell)}, v^{(\ell-1)})p^{(\ell)}(v^{(\ell)})p^{(\ell-1)}(v^{(\ell-1)}), \ell \in [L]; \\ \text{(ii) } \tilde{u}(v^{(L)}) = u(v^{(L)})p^{(L)}(v^{(L)}); \end{cases} \tag{5}$$

then problem (4) becomes a convex optimization problem with respect to $\tilde{w}, p$ and $\tilde{u}$.

Discussion: To the best of our knowledge, we give the first global convex formulation in Theorem 1 for continuous CNNs, i.e., a much simpler loss landscape characterization than existing methods. Although we find this convexity in feature distribution space instead of trainable parameter space, this would not overshadow its significant value. For one thing, as empirically shown in Section 6.1, the solution paths of DNNs during training, if we view them in the feature distribution space, are unique although they seem chaotic and quite different in the parameter space, which indicates that DNNs could be more understandable and simpler if we analyze them from feature distribution space. For another, the unique solution path implies that the training algorithms (e.g., SGD) could essentially work on the feature distribution space instead of the parameter space, which together with the convexity naturally explain why one usually does not observe bad local minima in practice. In fact, it can be shown that under suitable conditions, the training algorithms (e.g., SGD) of DNNs converge to a solution that is a stationary point of the convex reformulation, as shown in Section B of the appendix. All these demonstrate the value of studying DNNs in feature distribution space.

6 Experiments

We start from using our proposed technique NNG to show that two overparameterized CNNs with same architecture and initialization method, though have different initializations at the beginning of training, learn the unique solution path during the whole training process, which could be explained by our theory in Section 5.1. Then we give some empirical evidences for the convexity of overparameterized CNN by showing the uniqueness of its optimal solution and visualizing its the loss landscape in Section 6.2. All our empirical findings are consistent across a range of architectures and datasets, we only present the results on CIFAR-10 with VGG-16 below and postpone the results on other architectures and datasets to appendix. In appendix, we also provide results to show that



(a) (b) (c)

Figure 2: (a) The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2 when grafted at layer ℓ and (b) the normalized ℓ_2 distance between the weights $\theta_1^{(\ell)}$ and $\theta_2^{(\ell)}$. These values are plotted at different training stages. (c) The mean ℓ_2 distance (Eqn. (6)) between feature function $f^{(\ell')}(\theta_2; x)$ and $f^{(\ell')}(\tilde{\theta}_{(1,2)}; x)$ for each $\ell' > \ell$, where x enumerates the training data. $\tilde{\theta}_{(1,2)}$ is grafted at layer ℓ.

VGG is overparameterized enough, i.e., a good approximation for the corresponding continuous VGG, demonstrating the applicability of our results. We use $\ell_{1,2}$ regularizer, and save intermediate checkpoints of NN parameters at time-step $t \in \{1, 2, 5, 8\} \bigcup \{10k : k \in \mathbb{N}^+\}$ in the entire section.

6.1 Uniqueness of Solution Path during Training

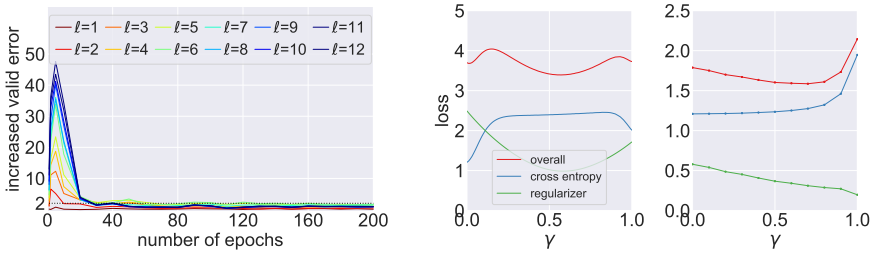
We first demonstrate that the solution paths are the same over the whole training process if we view them in terms of feature distributions, although they seem chaotic and quite different if viewed from the trainable parameters. Firstly, two VGG-16 are trained from two different initializations using the same initializer. The same initializer is used to guarantee the nets are initialized with almost the same feature distribution. For each checkpoint pair of θ_1 and θ_2 saved at the same time-step, we graft θ_1 to θ_2 at each layer and plot the increased validation error of $\tilde{\theta}_{(1,2)}$. For comparison, we also calculate the distance between the parameters in the checkpoints of θ_1 and θ_2. The details for calculating the distance is in Section D of appendix. The results are shown in Fig. 2.

The low increased validation errors of $\tilde{\theta}_{(1,2)}$ at all time-steps in Fig. 2(a) indicate that the feature distributions learned by θ_1 and θ_2 at different layers are equal to each other along the 'training trajectory'. At the same time, the normalized ℓ_2 distances between θ_1 and θ_2 in Fig. 2(b) show that the RMSE between two weight matrices after alignment and unit-variance normalization is above 0.6 for all layers and time-steps, which implies that these two weight matrices are not equivalent under row/column exchange operation. A natural explanation is that the training algorithms (e.g., SGD) essentially work on feature distribution space instead of parameter space, and the evolution of the feature distributions during training follows some differential equation. Since the initializer gives almost the same initializations to the differential equation, the solution paths would be very similar during the whole process of solving this equation, i.e., the training process.

Compared with existing metrics which directly compare the feature representations at each layer, our NNG technique could partly alleviate the effect of redundancy. A direct empirical investigation is shown in Fig. 2 (c), where the mean ℓ_2 distance between feature functions of nets $\tilde{\theta}_{(1,2)}$ and θ_2 at layer ℓ' is defined as:

$$\frac{1}{m^{(\ell')}} \sum_{k=1}^{m^{(\ell')}} \frac{1}{\sqrt{s_k}} \sqrt{\sum_{i=1}^{n} \frac{1}{n} \|\hat{f}_k^{(\ell')}(\tilde{\theta}_{(1,2)}; x_i) - \hat{f}_k^{(\ell')}(\theta_2; x_i)\|_2^2}, \tag{6}$$

and $s_j = \sum_{i=1}^{W^{(\ell')}} \sum_{j=1}^{H^{(\ell')}} \hat{\text{var}}[\left(\hat{f}_k^{(\ell')}(\theta_2; X)\right)_{i,j}]$ while $\hat{\text{var}}$ stands for the estimated variance. For example, we could see that if we graft the sixth layer of θ_1 to seventh layer of θ_2 (line denoted as $\ell = 6$), the ℓ_2 distance between feature functions of $\tilde{\theta}_{(1,2)}$ and that of θ_2 in layer 7 is very large. But this error significantly decreases as it propagates to the output layer. Such phenomenon indicates that a large fraction of error introduced by grafting at layer ℓ is caused by redundancy, which could be largely cancelled out by, for example, the small weight and the truncation of activation function in each layer $\ell' > \ell$. Moreover, we also show that successful graft is non-trivial and provide more interesting empirical findings about NNG in Appendix.



(a) (b)

Figure 3: (a) The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2 over the training process, where θ_1 and θ_2 are trained with different initialization methods. (b) The overall/cross entropy/regularizer loss of $\hat{\theta}_\gamma = \gamma\theta_1 + (1-\gamma)\theta_2$ (left), $\bar{\theta}_\gamma$ (right).

6.2 The Evidence of Convexity

We further design some experiments to support our claim that the landscape of overparameterized CNN is convex with respect to the distribution of feature functions at each layer. To be specific, we consider two network θ_1 and θ_2 with different initialization methods. In this section, we use two types of variance scaling initializer [15] with uniform (*he_uniform*) and normal (*he_normal*) distribution for weights in θ_1 and θ_2 respectively. Because θ_1 and θ_2 use different initialization methods, their initial feature distribution at each layer is quite different, the training journey of θ_1 and θ_2 then follow two different paths. We are interested in the following two questions: (1) Is the end points of the two paths are the same? (2) From the perspective of feature distribution, does linear interpolation between two points in these two paths lead to a convex loss curve? It will strongly support our claim of convexity of landscape if the answers to these two questions are 'yes'.

Uniqueness of Optimal Solution To answer the first question, we make use of our NNG technique again to check whether two intermediate checkpoints of θ_1 and θ_2 with same time-step have similar feature distributions. To be specific, for saved checkpoints, we graft each layer ℓ of θ_1 onto layer $\ell+1$ of θ_2 and the results are in Fig. 3(a). We can see that the increased valid error of $\tilde{\theta}_{(1,2)}$ grafted at each layer is very high in the beginning and decreases quickly as training process goes on, implying that the feature distributions learned by θ_1 and θ_2 start from different points and converge to a same point.

Loss Landscape Visualization We next consider to visualize the loss curve of linear interpolation between two points in feature distribution view. Given two NNs θ_1 and θ_2 with feature distributions denotes as p_1 and p_2, the intuitive idea of constructing a NN $\bar{\theta}_\gamma$, which has feature distribution $\gamma p_1 + (1-\gamma)p_2$, is to make sure the pre-activation set of $\bar{\theta}_\gamma$ is close to the mixture of that of θ_1 and θ_2 with ratio $\gamma/(1-\gamma)$, where the pre-activation set of θ on training set is defined as

$$\hat{\mathcal{V}}_\theta^{(\ell)} = \left\{ v_j^{(\ell)} = [\hat{g}_j^\ell(\theta; x_1), \ldots, \hat{g}_j^\ell(\theta; x_N)] : j \in [m^{(\ell)}] \right\}.$$

An optimization-based algorithm is designed to construct $\bar{\theta}_\gamma$, i.e., Alg.2 in Appendix. We let θ_1 and θ_2 to be VGG-16 trained after 2 epochs using *he_uniform* and *he_normal* initializers, attaining 19% and 57% accuracy respectively, and plot the loss of $\bar{\theta}_\gamma$ when γ varies. The result is shown in Fig. 3 (b). Compared with the non-convex loss curve of linear interpolation in weight space (left), the linear interpolation in feature distribution space (right) leads to a convex loss curve.

7 Conclusions

In this work, we presented an experimental technique NNG, which can be used to determine whether two NNs learn similar features. Extensive experimental results with NNG suggest that overparameterized CNN learns a fixed trajectory in feature distributions. We then proposed a framework to reformulate the loss w.r.t. the feature distribution at each layer, to explain the phenomenon. This reformulation explains convex loss landscape and fixed feature distribution trajectory for SGD. Because of their better theoretical and empirical properties, we argue that NN loss landscapes should be characterized with respect to the feature distribution space rather than the parameter space.

8 Broader Impact

The authors feel that the broader impact seems not applicable to this work. The reason is that this is a research paper, which provides new insights to help people better understand deep neural networks and can motivate more efficient training algorithms in the future. But it is hard to say who may benefit and who may be put at disadvantage from this research.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Z. Allen-Zhu, Y. Li, and Y. Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *arXiv preprint arXiv:1811.04918*, 2018.

[3] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, 2019.

[4] anonymous. Convex formulation of overparameterized deep neural networks. *arXiv*, 2019.

[5] anonymous. Modeling from features: a mean-field framework for over-parameterized deep neural networks. *In progress*, 2020.

[6] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

[7] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, 2019.

[8] R. Balestriero, R. Cosentino, B. Aazhang, and R. Baraniuk. The geometry of deep networks: Power diagram subdivision. In *Advances in Neural Information Processing Systems*, pages 15806–15815, 2019.

[9] L. Chizat and F. Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in neural information processing systems*, pages 3036–3046, 2018.

[10] L. Chizat, E. Oyallon, and F. Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems*, pages 2933–2943, 2019.

[11] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204, 2015.

[12] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[13] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, 2019.

[14] S. S. Du, X. Zhai, B. Poczos, and A. Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representation*, 2019.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2016.

[16] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *International Conference on Learning Representations*, 2017.

[17] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[18] K. Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594, 2016.

[19] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

[20] R. Kuditipudi, X. Wang, H. Lee, Y. Zhang, Z. Li, W. Hu, R. Ge, and S. Arora. Explaining landscape connectivity of low-cost solutions for multilayer nets. In *Advances in Neural Information Processing Systems*, pages 14574–14583, 2019.

[21] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[22] Y. Li and Y. Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, 2018.

[23] Y. Li, J. Yosinski, J. Clune, H. Lipson, and J. E. Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *FE@ NIPS*, pages 196–212, 2015.

[24] S. Mei, A. Montanari, and P.-M. Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[25] P.-M. Nguyen. Mean field limit of the learning dynamics of multilayer neural networks. *arXiv preprint arXiv:1902.02880*, 2019.

[26] Q. Nguyen. On connected sublevel sets in deep learning. *arXiv preprint arXiv:1901.07417*, 2019.

[27] Q. Nguyen, M. C. Mukkamala, and M. Hein. On the loss landscape of a class of deep neural networks with no bad local valleys. *arXiv preprint arXiv:1809.10749*, 2018.

[28] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[29] F. Schilling. The effect of batch normalization on deep convolutional neural networks, 2016.

[30] O. Shamir. Are resnets provably better than linear predictors? In *Advances in neural information processing systems*, pages 507–516, 2018.

[31] L. Venturi, A. S. Bandeira, and J. Bruna. Spurious valleys in two-layer neural network optimization landscapes. *arXiv preprint arXiv:1802.06384*, 2018.

[32] L. Wang, L. Hu, J. Gu, Z. Hu, Y. Wu, K. He, and J. Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *Advances in Neural Information Processing Systems*, pages 9584–9593, 2018.

Supplemental Material: How to Characterize The Landscape of Overparameterized Convolutional Neural Networks

This appendix can be divided into four parts. To be precise,

1. Section A gives the detailed proof for Theorem 1;
2. Section B gives high-level proof of the claim that SGD will converge to the stationary point of our proposed convex formulation.
3. Section C shows omitted algorithms and calculation methods in the paper, including (1) algorithm of *neural network grafting* (2) algorithm to construct NN with mixture features of two networks.
4. Section D presents experimental configuration of this paper;
5. Section E provides more experimental evidences of the uniqueness of solution path, feature redundancy and the uniqueness of optimal solution.
6. Section F provides evidence to show that standard VGG is overparamterized enough. To be specific, we show that the feature distribution learned by standard VGG, have no difference though the *entire* training process with respect to the one learned by a larger VGG, which has same architecture except has twice number of channels/hidden notes at each layer.
7. Section G presents some additional interesting results found by our proposed NNG in comparing the feature distributions of NN at different cases, including: 1) comparing feature distributions learned in different layer ℓ_1 and ℓ_2; 2) comparison of feature distribution learned with two different datasets; 3) one of θ_1 and θ_2 is not fully trained. These results not only show that our proposed NNG metric could differentiate the case when two feature distributions for comparison is different, but also provide some very interesting empirical phenomena we find by using our NNG technique.

A Proof for Theorem 1

We first present the new formulation of Problem (4) after reprameterizing it according to Eqn.(5) below:

$$\min_{\tilde{w},p,\tilde{u}} \frac{1}{N}\sum_{i=1}^{N}\phi\left(f(x_i),y_i\right)+\tilde{R}(\tilde{w},p,\tilde{u}), \tag{7}$$

$$s.t.\ 1)\int h^{(\ell-1)}(v_i^{(\ell-1)})*\tilde{w}(v^{(\ell)},v^{(\ell-1)})dv^{(\ell-1)}=v_i^{(\ell)}p^{(\ell)}(v^{(\ell)}),\ \ 2\le\ell\le L, i\in[N];$$

$$2)\sum_{v^{(0)}\in\mathcal{V}^{(0)}} v_i^{(0)}*\tilde{w}(v^{(1)},v^{(0)})=v_i^{(1)}p^{(1)}(v^{(1)}), i\in[N];$$

$$3)\int p^{(\ell)}(v^{(\ell)})dv^{(\ell)}=1 \text{ and } p^{(\ell)}(v^{(\ell)})\ge 0,$$

where the reformualted $f(x_i)$ and $\tilde{R}(\tilde{w},p,\tilde{u})$ take forms of

$$f(x_i)=\int h^{(L)}(v_i^{(L)})\tilde{u}(v^{(L)})dv^{(L)},\ \tilde{R}(\tilde{w},p,\tilde{u})=\sum_{\ell=1}^{L}\lambda^{(\ell)}\tilde{R}^{(\ell)}(\tilde{w},p)+\lambda^{u}\tilde{R}^{(u)}(\tilde{u},p),$$

$$\text{with } \tilde{R}^{(1)}(\tilde{w},p)=\sum_{v^{(0)}\in\mathcal{V}^{(0)}} r_2\left(\int r_1\left(\tilde{w}(v^{(1)},v^{(0)})\right)dv^{(1)}\right)/p^{(0)}(v^{(0)}),$$

$$\tilde{R}^{(\ell)}(\tilde{w},p)=\int r_2\left(\int r_1\left(\tilde{w}(v^{(\ell)},v^{(\ell-1)})\right)dv^{(\ell)}\right)/p^{(\ell-1)}(v^{(\ell-1)})dv^{(\ell-1)},\ell\geq 2,$$
$$\tilde{R}^{(u)}(\tilde{u},p)=\int r^{(u)}\left(\tilde{u}(v^{(L)})\right)/p^{(L)}(v^{(L)})dv^{(L)}.$$

To prove Theorem 1, we need to prove that problem (7) is convex. First, we need the following lemma.

Lemma 1. *$\frac{x^2}{y}$ is convex on $(x,y)\in[0,+\infty)\times(0,+\infty)$.*

The lemma above can be verified by the definition of convex function directly. Now we turn to give the detailed proof of Theorem 1.

Proof. Firstly, it is easy to verify that all the constraints 1) to 3) are convex *w.r.t.* $\tilde{w},\tilde{u}$ and p.

For the objective function, we have that the loss ϕ is convex *w.r.t.* $\tilde{u}$. Therefore, we only need to prove the regularizer $\tilde{R}(\tilde{w},p,\tilde{u})=\sum_{\ell=1}^{L}\lambda^{(\ell)}\tilde{R}^{(\ell)}(\tilde{w},p)+\lambda^u\tilde{R}^{(u)}(\tilde{u},p)$ is convex *w.r.t.* $\tilde{w},\tilde{u}$ and p. Below, we will prove each item in $\tilde{R}$ is convex. We fist denote

$$\tilde{\Psi}^{(\ell)}(\tilde{w},p;v^{(\ell-1)})=\int r_1\left(\tilde{w}(v^{(\ell)},v^{(\ell-1)})\right)dv^{(\ell)},$$

then:
(1) For the item $\tilde{R}^{(1)}(\tilde{w},p)$, recall that

$$\tilde{R}^{(1)}(\tilde{w},p)=\sum_{v^{(0)}\in\mathcal{V}^{(0)}}r_2\left(\tilde{\Psi}^{(1)}(\tilde{w},p;v^{(0)})\right)/p^{(0)}(v^{(0)})$$

Since we pick $r_1(w)=\|vec(w)\|_1$ in this paper, we can have that $\tilde{\Psi}^{(\ell)}$ is non-negative and convex.

Moreover, we notice that $p^{(0)}$ is constant and $r_2(w)=w^2$ in this paper, it is easy to know that $\tilde{R}^{(1)}(\tilde{w},p)$ is convex.

(2) When $\ell\geq 2$, $\tilde{R}^{(\ell)}(\tilde{w},p)$ takes the form of

$$\tilde{R}^{(\ell)}(\tilde{w},p)=\int\frac{r_2\left(\tilde{\Psi}^{(\ell)}(\tilde{w},p;v^{(\ell-1)})\right)}{p^{(\ell-1)}(v^{(\ell-1)})}dv^{(\ell-1)}.$$

We denote

$$I(\tilde{w},p)=\frac{r_2\left(\tilde{\Psi}^{(\ell)}(\tilde{w},p;v^{(\ell-1)})\right)}{p^{(\ell-1)}(v^{(\ell-1)})},$$

and let $p_\gamma=\gamma p_1+(1-\gamma)p_2$ and $\tilde{w}_\gamma=\gamma\tilde{w}_1+(1-\gamma)\tilde{w}_2$ for any $p_1,p_2,\tilde{w}_1,\tilde{w}_2$ and $\gamma\in[0,1]$. Then, we have

$$\begin{aligned}
&I(\tilde{w}_\gamma,p_\gamma)=\frac{r_2\left(\tilde{\Psi}^{(\ell)}(\tilde{w}_\gamma,p_\gamma;v^{(\ell-1)})\right)}{p_\gamma^{(\ell-1)}(v^{(\ell-1)})}\\
\leq&\frac{r_2\left(\gamma\tilde{\Psi}^{(\ell)}(\tilde{w}_1,p_1;v^{(\ell-1)})+(1-\gamma)\tilde{\Psi}^{(\ell)}(\tilde{w}_2,p_2;v^{(\ell-1)})\right)}{p_\gamma^{(\ell-1)}(v^{(\ell-1)})}\\
\leq&\gamma\frac{r_2\left(\tilde{\Psi}^{(\ell)}(\tilde{w}_1,p_1;v^{(\ell-1)}))\right)}{p_1^{(\ell-1)}(v^{(\ell-1)})}+(1-\gamma)\frac{r_2\left(\tilde{\Psi}^{(\ell)}(\tilde{w}_2,p_2;v^{(\ell-1)}))\right)}{p_2^{(\ell-1)}(v^{(\ell-1)})}\\
=&\gamma I(\tilde{w}_1,p_1)+(1-\gamma)I(\tilde{w}_2,p_2).
\end{aligned}\tag{8}$$

The first inequality in (8) holds since $\tilde{\Psi}$ is convex and the the second one comes from Lemma 1. Thus $I(\tilde{w},p)$ is a convex function of $(\tilde{w},p)$ and therefore $\tilde{R}^{(\ell)}(\tilde{w},p)$ is convex.

(3) We denote $\tilde{u}_\gamma = \gamma\tilde{u}_1 + (1-\gamma)\tilde{u}_2$, $p_\gamma = \gamma p_1 + (1-\gamma)p_2$ for any $\tilde{u}_1, \tilde{u}_2, p_1, p_2$ and $\gamma \in [0,1]$ and recall that

$$\tilde{R}^{(u)}(\tilde{u}, p) = \int r^{(u)}\left(\tilde{u}(v^{(L)})\right)/p^{(L)}(v^{(L)})dv^{(L)}.$$

Hence, we have

$$\begin{aligned}
&\tilde{R}^{(u)}(\tilde{u}_\gamma, p_\gamma) = \int \frac{\left(\|\tilde{u}_\gamma(v^{(L)})\|\right)^2}{p_\gamma^{(L)}(v^{(L)})}dv^{(L)} \\
\leq &\int \frac{\left(\gamma\|\tilde{u}_1(v^{(L)})\| + (1-\gamma)\|\tilde{u}_2(v^{(L)})\|\right)^2}{p_\gamma^{(L)}(v^{(L)})}dv^{(L)} \\
\leq &\int \gamma\frac{\|\tilde{u}_1(v^{(L)})\|^2}{p_1^{(L)}(v^{(L)})} + (1-\gamma)\frac{\|\tilde{u}_2(v^{(L)})\|^2}{p_2^{(L)}(v^{(L)})}dv^{(L)} \\
= &\gamma\tilde{R}^{(u)}(\tilde{u}_1, p_1) + (1-\gamma)\tilde{R}^{(u)}(\tilde{u}_2, p_2).
\end{aligned}$$

The last inequality comes from Lemma 1 and therefore $\tilde{R}^{(u)}(\tilde{u}, p)$ is a convex function of $(\tilde{u}, p)$

The proof is complete. □

B Convergence Analysis

We would like to study the relationship of the convex reformulation and gradient descent in the original parameter space for continuous DNN. We consider the constrained formulation of continuous DNN in (4) and its convex relaxation in (5), with the reparameterization of (w, u) by $(\tilde{w}, \tilde{u})$.

In the continuous DNN formulation (4), the network is parameterized with respect to (w, p, u). However, the typical optimization using SGD or GD modify the network parameters (w, u) but does not modify p directly. It can be easily seen that changing (w, u) modifies p accordingly.

When the continuous NN converges, then it reaches a point (w_*, p_*, u_*) which is stationary with respect to the neural network parameters. That is, if we modify (w_*, u_*), the objective value does not decrease (however, we are not allowed to modify the distribution p_* directly because it is not an explicit part of the DNN parameters, but rather it is an hidden parameter induced by the constraints).

Nevertheless the following result shows that under suitable conditions, this point is also stationary with respect to the distributional parameter p_*. That is, if we vary (w_*, p_*, u_*) arbitarily, as long as the new parameters satisfy the constraints, then the objective value does not decrease.

Since the same claim holds for the reparameterization $(\tilde{w}_*, p_*, \tilde{u}_*)$ in (5). This result shows that when the continuous DNN converges, it converges to a solution of the convex reformulation, and this establishes a connection between the standard DNN optimization method using SGD, and our convex reformulation.

Theorem 2. *Consider an arbitrary point (w_*, p_*, u_*) of the constrained formulation of continuous NN in* (4)*, such that varying the DNN network parameters (w_*, u_*) cannot decrease the objective function. Assume that $p_*^{(\ell)}(v) \neq 0$ for all v and ℓ, then $(\tilde{w}_*, p_*, \tilde{u}_*)$ is a stationary point of the convex reformulation reparameterized by* (5)*.*

Proof. (High-level Sketch) We present a highlevel proof that is easier to understand. A more rigorous treatment requires heavy notations in differential equations and functional analysis, which we shall avoid in this version.

The key argument is to show that all varations of $(\tilde{w}_*, p_*, \tilde{u}_*)$ that satisfy the constraints can be achieved by varying (w_*, u_*) of the continuous DNN parameters, which is commonly used in gradient descent optimization algorithms.

Let the objective function be $\tilde{Q}(\tilde{w}, p, \tilde{u}) = Q(w, p, u)$.

To simplify notations, we denote the DNN weights on the ℓ-th layer $w(v^{(\ell)}, v^{(\ell-1)})$, and the reparameterized weight $\tilde{w}(v^{(\ell)}, v^{(\ell-1)})$ by $w^{(\ell)}$ and $\tilde{w}^{(\ell)}$.

Note that we have

$$\langle \nabla_{\tilde{u}} \tilde{Q}(\tilde{w}_*, p_*, \tilde{u}_*), \Delta\tilde{u} \rangle = 0$$

for all $\Delta\tilde{u}$, since $Q(w_*, p_*, u_*)$ is a stationary point with respect to network parameter u and $p_* > 0$. Here the inner product is with respect to integration over $v^{(L)}$.

Since the constraints at layer ℓ involve the weight $\tilde{w}^{(\ell)}$ and density $p^{(\ell)}$, therefore to prove the desired result, we only need to show that for each ℓ:

$$\langle \nabla_{\tilde{w}^\ell} \tilde{Q}(\tilde{w}_*, p_*, \tilde{u}_*), \Delta\tilde{w}_*^{(\ell)} \rangle + \langle \nabla_{p^{(\ell)}} \tilde{Q}(\tilde{w}_*, p_*, \tilde{u}_*), \Delta p_*^{(\ell)} \rangle = 0, \tag{9}$$

under the infinitesimal transformation (with other layers fixed):

$$\tilde{w}_*^{(\ell)} \to \tilde{w}_*^{(\ell)} + \epsilon \Delta\tilde{w}_*^{(\ell)}, \qquad p_* \to p_*^{(\ell)} + \epsilon \Delta p_*^{(\ell)},$$

when $\epsilon \to 0$, with the constraint

$$\int h^{(\ell-1)}(v_i^{(\ell-1)}) \Delta\tilde{w}_*^{(\ell)}(v^{(\ell)}, v^{(\ell-1)})\, dv^{(\ell)} = v^{(\ell)} \Delta p_*^{(\ell)}(v^{(\ell)}).$$

Here inner product with respect to $\Delta\tilde{w}_*^{(\ell)}$ is integration with respect to $(v^{(\ell)}, v^{(\ell-1)})$, and inner product with respect to $\Delta p_*^{(\ell)}$ is is integration with respect to $v^{(\ell)}$.

Since the constraint is linear, we can decompose the subspace into two subspaces. Case 1 is $\Delta p_*^{(\ell)}(v^{(\ell)}) = 0$, and Case 2 is $\Delta p_*^{(\ell)}(v^{(\ell)}) \neq 0$ but $\int \Delta p_*^{(\ell)}(v^{(\ell)}) dv^{(\ell)} = 0$ and

$$\Delta\tilde{w}_*^{(\ell)}(v^{(\ell)}, v^{(\ell-1)}) = \tilde{w}_*(v^{(\ell)}, v^{(\ell-1)}) \Delta p_*^{(\ell)}(v^{(\ell)}) / p_*^{(\ell)}(v^{(\ell)}). \tag{10}$$

It is easy to check that the constraints are satisfied in Case 2.

Now to show (9), we only need to show it under these two subspaces, as detailed below.

Case 1: In this case, we can let

$$\Delta w_*^{(\ell)}(v^{(\ell)}, v^{(\ell-1)}) = \Delta\tilde{w}_*^{(\ell)}(v^{(\ell)}, v^{(\ell-1)}) \frac{1}{p^{\ell-1}(v^{(\ell-1)}) p^{(\ell)}(v^{(\ell)})},$$

and consider a change of network parameter

$$w_*^{(\ell)} \to w_*^{(\ell)} + \Delta w_*^{(\ell)}.$$

This change of parameter does not change u_* and p_* in the resulting neural network. Since $Q(w_*, p_*, u_*)$ is stationary in w_*, it follows that

$$\langle \nabla_{w_*^{(\ell)}} Q(w_*, p_*, u_*), \Delta w_*^{(\ell)} \rangle = 0,$$

which implies (9).

Case 2: In this case, we consider a change of DNN parameter $w^{(\ell)}$, which leads to

$$w_*^{(\ell)}(v^{(\ell)}, v^{(\ell-1)}) \to w_+^{(\ell)}(v_+^{(\ell)}, v^{(\ell-1)}) = w_*^{(\ell)}(v_+^{(\ell)}, v^{(\ell-1)}) + \epsilon g_\ell(v^{(\ell)}) \omega(v^{(\ell-1)}),$$

where variable v_+ is induced by

$$v_+^{(\ell)} = v^{(\ell)} + \epsilon \cdot g_\ell(v^{(\ell)}),$$

where we simply pick an arbitrary vector function $\omega(v)$ so that

$$\int h^{(\ell-1)}(v_i^{(\ell-1)}) \omega_i(v^{(\ell-1)}) p^{(\ell-1)}(v^{(\ell-1)}) dv^{(\ell-1)} = 1.$$

With the above transformation, we have the constraint:

$$\int h^{(\ell-1)}(v_i^{(\ell-1)}) * w_+(v_+^{(\ell)}, v^{(\ell-1)}) p_+^{(\ell)}(v_+^{(\ell)}) p^{(\ell-1)}(v^{(\ell-1)}) dv^{(\ell-1)} = p_+^{(\ell)}(v_+^{(\ell)}) [v_+^{(\ell)}]_i, \ell \geq 2, i \in [N].$$

The corresponding change of distribution is from $p^{(\ell)}(v)$, to $p_+^{(\ell)}(v_+)$, which is equivalent to the distribution of $p^{(\ell)}(v)$ under the change of variable $v \to v_+$, corresponding to the DNN resulted from a change of network parameter $w_*^{(\ell)} \to w_+^{(\ell)}$.

Using the Fokker-Planck equation, we have

$$p_+^{(\ell)}(v) = p_*^{(\ell)}(v) - \epsilon \nabla (p_*^{(\ell)}(v) g_\ell(v)) + o(\epsilon),$$

and let

$$\Delta p_*^{(\ell)}(v) = p_+^{(\ell)}(v) - p_*^{(\ell)}(v) = -\epsilon \nabla (p_*^{(\ell)}(v) g_\ell(v)) + o(\epsilon).$$

It follows that we can keep $\tilde{w}^{(\ell+1)}$ unchanged by setting

$$w_*^{(\ell+1)}(v^{(\ell+1)}, v^{(\ell)}) \to w_*^{(\ell+1)}(v^{(\ell+1)}, v^{(\ell)}) \frac{p^{(\ell)}(v^{(\ell)})}{p_+^{(\ell)}(v^{(\ell)})}.$$

Note also that

$$\int h^{(\ell-1)}(v_i^{(\ell-1)}) * w_*(v_+^{(\ell)}, v^{(\ell-1)}) p_+^{(\ell)}(v_+^{(\ell)}) p^{(\ell-1)}(v^{(\ell-1)}) dv^{(\ell-1)} = p_+^{(\ell)}(v_+^{(\ell)})[v_+^{(\ell)}]_i, \ell \geq 2, i \in [N].$$

Therefore

$$\int h^{(\ell-1)}(v_i^{(\ell-1)}) * \Delta \tilde{w}'_*(v_+^{(\ell)}, v^{(\ell-1)}) = 0,$$

where

$$\Delta \tilde{w}'_*(v_+^{(\ell)}, v^{(\ell-1)}) = [w_+(v_+^{(\ell)}, v^{(\ell-1)}) - w_*(v_+^{(\ell)}, v^{(\ell-1)})] p_+^{(\ell)}(v_+^{(\ell)}) p^{(\ell-1)}(v^{(\ell-1)}).$$

Now case 1 implies that

$$\langle \nabla_{\tilde{w}} \tilde{Q}(\tilde{w}_*, p_*, \tilde{u}_*), \Delta \tilde{w}'_* \rangle = 0. \tag{11}$$

Note that the stationarity of the DNN network with respect to a change of the network parameter $w_*^{(\ell)}$ implies that as $\epsilon \to 0$:

$$\langle \nabla_w Q(w_*, p_*, u_*), \Delta w_* \rangle + \langle \nabla_{p^{(\ell)}} Q(w_*, p_*, u_*), \Delta p_*^{(\ell)} \rangle = o(\epsilon),$$

and $\Delta w_* = w_+ - w_*$. This equation in the convex reformulation is equivalent to

$$\langle \nabla_{\tilde{w}^{(\ell)}} \tilde{Q}(\tilde{w}_*, p_*, \tilde{u}_*), [\tilde{w}_+^{(\ell)} - \tilde{w}_*^{(\ell)}] \rangle + \langle \nabla_{p^{(\ell)}} Q(\tilde{w}_*, p_*, \tilde{u}_*), \Delta p_*^{(\ell)} \rangle = 0.$$

By substracting (11), we obtain

$$\langle \nabla_{\tilde{w}^{(\ell)}} \tilde{Q}(\tilde{w}_*, p_*, \tilde{u}_*), \Delta \tilde{w}_*^{(\ell)} \rangle + \langle \nabla_{p^{(\ell)}} Q(\tilde{w}_*, p_*, \tilde{u}_*), \Delta p_*^{(\ell)} \rangle = 0,$$

where $\Delta \tilde{w}_*$ is given by (10). This implies (9). It follows by letting $\epsilon \to 0$ that for Case 2, (9) holds for all

$$\Delta p_*^{(\ell)}(v) = -\nabla (p_*^{(\ell)}(v) g_\ell(v)).$$

Since $g_\ell(v)$ is arbitrary, it is easy to verify that this implies that (9) holds for Case 2 with all possible $\Delta p_*^{(\ell)}(v)$ such that

$$\int \Delta p_*^{(\ell)}(v) dv = 0.$$

This proves the desired result. □

C Missing Algorithms in Main Text

C.1 Neural Network Graft

Here we present the detailed steps of our technique *neural network grafting* (NNG) in Algorithm 1. It takes two neural network θ_1 and θ_2 as input and output a grafted neural network $\theta_{(1,2)}$. In order to keep the notations simple to demonstrate our idea, we assume θ_1 and θ_2 have the same number of layers and the graft algorithm is to compare the feature distributions of layer ℓ in θ_1 and θ_2, i.e. graft layer ℓ in θ_1 onto layer $\ell + 1$ in θ_2. It should be emphasized that Algorithm 1 can be applied to the case where θ_1 and θ_2 have different numbers of hidden nodes, i.e., $m_1^{(\ell)} \neq m_2^{(\ell)}$ in each layer, which is used in Section F, and the case where θ_1 and θ_2 have different numbers of hidden layers. Moreover, NNG can also even be used to compare different layers, for example, compare layer ℓ_1 in θ_1 and layer ℓ_2 in θ_2 by grafting layer ℓ_1 in θ_1 onto layer $\ell_2 + 1$ in θ_2. We don't explicitly write down the algorithm of those cases and the details can be found in Section D (comparing layer with different hidden nodes) and Section G.1 (comparing different layers).

Algorithm 1 Neural Network Graft at layer ℓ

Input: Dataset $\{(x_i, y_i)\}_{i=1}^N$ and two networks θ_1 and θ_2 with $\theta_i = (\theta_i^{(1)}, \dots, \theta_i^{(L)}; u_i), i = 1, 2$.

Allocate a grafted network $\tilde{\theta}_{(1,2)}$, whose front ℓ layers' hidden size is same with that of front ℓ layers in θ_1 (i.e. $m_{(1,2)}^{(\ell')} = m_1^{(\ell')}$ for $\ell' \in [\ell]$), and back $L - \ell$ hidden layers' size is same with that of back $L - \ell$ layers in θ_2 (i.e. $m_{(1,2)}^{(\ell')} = m_2^{(\ell')}$ for $\ell + 1 \le \ell' \le L$).
Copy weights from θ_1 and θ_2 to $\tilde{\theta}_{(1,2)}$, i.e., $\tilde{\theta}_{(1,2)} = (\theta_1^{(1)}, \cdots, \theta_1^{(\ell)}, \tilde{\theta}_{(1,2)}^{(\ell+1)}, \theta_2^{(\ell+2)}, \cdots, \theta_2^{(L)}, u_2)$.

Train $\tilde{\theta}_{(1,2)}^{(\ell+1)}$ by solving the following problem

$$\min_{\tilde{\theta}_{(1,2)}^{(\ell+1)}} \sum_{i=1}^{N} \sum_{k=1}^{m^{(\ell+1)}} \|\hat{g}_k^{(\ell+1)}(\tilde{\theta}_{(1,2)}; x_i) - \hat{g}_k^{(\ell+1)}(\theta_2; x_i)\|_2^2. \tag{12}$$

Return: The grafted network $\tilde{\theta}_{(1,2)}$ for evaluation

C.2 Feature Distribution Fusion

Here we present Algorithm 2, the detailed optimization-based algorithm to construct a new neural network $\bar{\theta}_\gamma$ from two nets θ_1 and θ_2. The feature distribution of $\bar{\theta}_\gamma$ can be approximately equal to $\gamma p_1 + (1 - \gamma) p_2$, where p_1 and p_2 are the feature distributions of θ_1 and θ_2, respectively.

Algorithm 2 Feature Distribution Fusion

Input: Training set $\{(x_i, y_i)\}_{i=1}^N$ and the sets $\hat{\mathcal{V}}_i^{(\ell)}$ of two CNNs with $\ell \in [L]$ and $i = 1, 2$.
for $\ell = 1, \dots, L$ **do**
Construct a set $\hat{\mathcal{V}}_\gamma^{(\ell)}$ by sampling $\gamma m^{(\ell)}$ and $(1-\gamma)m^{(\ell)}$ points from $\hat{\mathcal{V}}_1^{(\ell)}$ and $\hat{\mathcal{V}}_2^{(\ell)}$, respectively.

Train $\bar{w}_\gamma^{(\ell)}$in the ℓ-th layer of VGG $\bar{\theta}_\gamma$ by minimizing:

$$L(\bar{w}_\gamma^{(\ell)}) = \frac{1}{m^{(\ell)} N} \sum_{i=1}^{N} \sum_{v_j^{(\ell)} \in \hat{\mathcal{V}}_\gamma^{(\ell)}} \|\hat{g}_j^{(\ell)}(\bar{\theta}_\gamma; x_i) - v_{j,i}^{(\ell)}\|_2^2 + \lambda^{(\ell)} \hat{R}^{(\ell)}(\bar{w}_\gamma^{(\ell)}). \tag{13}$$

end for
Train the output layer by solving:

$$\min_{\bar{u}} \frac{1}{N} \sum_{i=1}^{N} \phi(\hat{f}(\bar{\theta}_\gamma; x_i), y_i) + \lambda^{(u)} \hat{R}^u(\bar{u}).$$

Return: The trained network $\bar{\theta}_\gamma$.

D Experimental Configuration

In this section, we present the experimental settings and implementation details. We use same regularization parameter λ for all layers, i.e., $\lambda^{(1)} = \cdots = \lambda^{(L)} = \lambda^{(u)} = \lambda$.

Training θ_1 and θ_2 We denote θ_1 and θ_2 to be two standard VGG-16 without batch normalization, since batch normalization can change the structure of the networks. The hidden sizes of the last two fully-connected layers are both 4096. All models are implemented using Tensorflow [1] and we follow the same data augmentation as [16]. Models are trained using stochastic gradient descent with momentum 0.9. We fix all of the hyperparameters summarized in Table 1 and only tune the initial learning rate and the regularization parameter. Since training neural network with large regularization parameter is difficult to converge using the standard initial learning rate 0.01, we vary

it in $\{1e-2, 5e-3, 2e-3, 1e-3\}$ and select the largest one that could produce robust training results (actually it could attain lowest validation error for that).

Table 1: Hyper-parameters used for training VGG-16

Hyper-parameter	Value
Batch size	64
Epochs to anneal lr	60
Anneal rate	0.2
Warm-up epochs	1

Training $\tilde{\theta}_{(1,2)}$ grafted at adjacent layers To compare the feature functions learned at layer ℓ of θ_1 and θ_2, we need to train a grafted network $\tilde{\theta}_{(1,2)}$ connecting two adjacent layers, i.e., layer ℓ of θ_1 and layer $\ell+1$ of θ_2. The detailed training process is as follows: we first train two networks θ_1 and θ_2 using previous configurations and then we train the weight matrix and bias between grafted layer ℓ and $\ell+1$ of the network $\tilde{\theta}_{(1,2)}$ using our proposed Algorithm 1 and copy all other parameters from θ_1 and θ_2. We use Adam Optimizer with initial learning rate $1e-3$ to optimize the objective Eqn (2). The data augmentation is same with as that in training θ_1 and θ_2 and number of epochs for training is 10 for CIFAR-10 and 20 for CIFAR-100.

Training $\tilde{\theta}_{(1,2)}$ grafted at non-adjacent layers To determine whether feature function distributions of θ_1 and θ_2 (θ_1 and θ_2 have the same structure) at two different layers are similar or not, we need to train a network grafted at non-adjacent layer. The training process is similar to that of grafting at adjacent layers. We provide detailed implementations here. Suppose we wish to calculate the similarity of feature distributions at layer ℓ_1 and ℓ_2 in two nets θ_1 and θ_2 respectively (that is to determine whether the last $L-\ell_2$ layers in θ_2 could be grafted on the first ℓ_1 layers in θ_1), we minimize the feature matching loss w.r.t. parameters $w_{(1,2)}$:

$$\frac{1}{N}\sum_{i=1}^{N}\|\mathcal{P}^o\circ\hat{f}^{(\ell_1)}(\theta_1;x_i)*w_{(1,2)}-\hat{f}^{(\ell_2+1)}(\theta_2;x_i)\|_2^2,$$

where N is the number of training samples, o is the number of max-pooling operator in layers $\ell'\in[\ell_1+1,\ell_2]$, $\hat{f}^{(\ell)}(\theta;x)$ and $\hat{g}^{(\ell)}(\theta;x)$ are pre-activation and post-activation feature map in $\mathbb{R}^{W^{(\ell)}\times H^{(\ell)}\times m^{(\ell)}}$.

Calculating the distance between the parameters of θ_1 and θ_2 We first assume that the alignment permutation $q^{(\ell)}$, whose j-th component $q_j^{(\ell)}$ represent that neuron j in layer ℓ of network θ_1 is aligned to neuron $q_j^{(\ell)}\in[m^{(\ell)}]$ in layer ℓ of network θ_2, is defined at each layer. We then could use the following statistic to measure the difference of θ_1 and θ_2 in the parameter matrix space at layer ℓ:

$$\sum_{i=1}^{m^{(\ell)}}\sum_{j=1}^{m^{(\ell-1)}}\|(w_2^{(\ell)})_{q_i^{(\ell)},q_j^{(\ell-1)}}-(w_1^{(\ell)})_{i,j}\|_2^2/\left((\|w_1^{(\ell)}\|_2^2+\|w_2^{(\ell)}\|_2^2)/2\right), \tag{14}$$

where $\|x\|_2$ is the ℓ_2 norm of tensor x's flattened vector, $w_1^{(\ell)}, w_2^{(\ell)}$ are weights at layer ℓ, $(w_k^{(\ell)})_{i,j}$ is a scalar for fully-connected layers and is a flattened 9-dim vector for VGG-16. Here we consider a heuristics algorithm to determine $q^{(\ell)}$ from the input layer to the last hidden layer. For the input layer $\ell=0$, the permutation is an identity map, i.e., $q_j^{(0)}=j$, using the nature of image. Now if the alignment permutation at layer $\ell-1$, i.e., $q^{(\ell-1)}$, is determined, the alignment permutation at layer ℓ could be calculated by minimize the objective

$$\min_{q^{(\ell)}}\sum_{i=1}^{m^{(\ell)}}\sum_{j=1}^{m^{(\ell-1)}}\|(w_2^{(\ell)})_{q_i^{(\ell)},q_j^{(\ell-1)}}-(w_1^{(\ell)})_{i,j}\|_2^2, \tag{15}$$

it should be noted that such optimization problem could be written as be a standard problem of minimum weighted bipartite matching, i.e., minimizing the following objective

$$\min_{q^{(\ell)}}\sum_{i=1}^{m^{(\ell)}}d_{j,q_j^{(\ell)}} \tag{16}$$

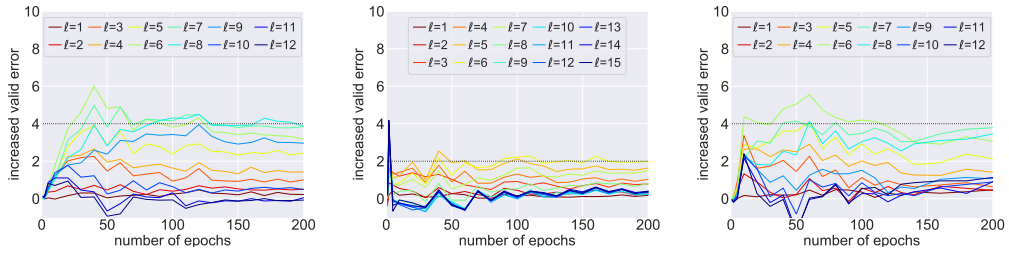


Figure 4: The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2 over the training process. The configuration of θ_1 and θ_2 for each subplot could be found in Table 2

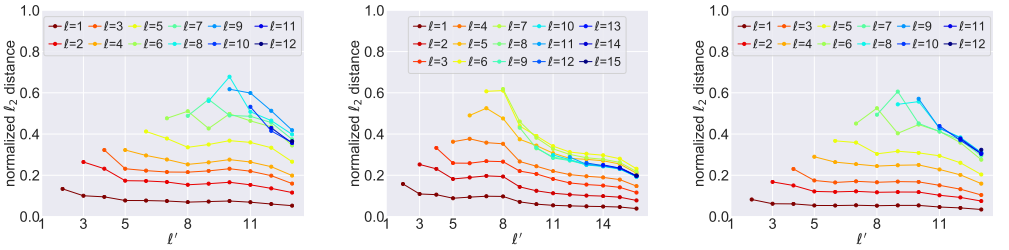


Figure 5: The mean ℓ_2 distance (Eqn. (6)) between feature function $f^{(\ell')}(\theta_2; x)$ and $f^{(\ell')}(\tilde{\theta}_{(1,2)}; x)$ for each $\ell' > \ell$, where x enumerates the training data. $\tilde{\theta}_{(1,2)}$ is grafted at layer ℓ. The configuration of θ_1 and θ_2 for each subplot could be found in Table 2

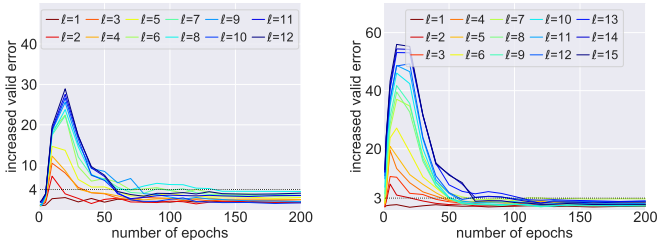


Figure 6: The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2 over the training process. θ_1 and θ_2 are (a) VGG-16 trained on CIFAR-100 and (b) VGG-19 trained on CIFAR-10.

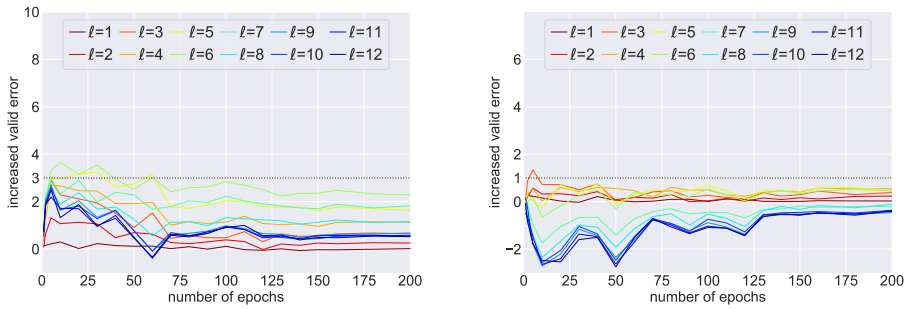
, where $d_{i,j} = \sum_{k=1}^{m^{(\ell-1)}} \|(w_2^{(\ell)})_{i,q_k^{(\ell-1)}} - (w_1^{(\ell)})_{j,k}\|_2^2$. This optimization problem could be solved in $O((m^{(\ell)})^3)$ using Hungarian Algorithm.

E Additional Experiments on Other Datasets, Architectures, and Regularizer

In this section, we show that our empirical findings found by our NNG technique is consistent across different datasets, architectures and regularizers, and provide more explanations. We consider three type of configurations, which is listed in the following Table 2.

Table 2: Configurations for Additional Experiments

Index	Dataset	Architecture	Regularizer
a	CIFAR-100	VGG-16	$\ell_{1,2}$
b	CIFAR-10	VGG-19	$\ell_{1,2}$
c	CIFAR-100	VGG-16	ℓ_2



(a) (b)

Figure 7: The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2 over the training process, where (a) θ_1 is standard VGG-16 and θ_2 has twice number of hidden nodes/channels compared with θ_1 (b) θ_2 is standard VGG-16 and θ_1 has twice number of hidden nodes/channels compared with θ_2. Both of the networks are trained using $\ell_{1,2}$ regularizer on CIFAR-10 dataset.

In the following, we demonstrate that our findings on 1) uniqueness of solution path; 2) the evidence of feature redundancy; 3) evidence of convexity found by NNG are consistent across these configurations. The training settings for θ_1, θ_2, the intermediate saved checkpoints, and training settings for $\tilde{\theta}_{(1,2)}$ are the same with respect to that in main text.

Uniqueness of Solution Path. We first show that the feature distributions learned by θ_1 and θ_2 trained from two different initializations are similar though the entire training process. Fig.4 plots the increased validation error of $\tilde{\theta}_{1,2}$ compared with θ_2 as training goes on. It is observed that the error is bouned in the low-error region, indicating that θ_1 and θ_2 learn similar feature distribution during the entire training process in all of these configurations.

Evidence of Feature Redundancy. In the main text, we claim that a large amount of error, the ℓ_2 difference between features of $\tilde{\theta}_{1,2}$ and that of θ_2 in layer $\ell + 1$ introduced by grafting at layer ℓ is caused by redundancy, which have little effect on the feature function in subsequent layers. We show such phenomenon could also be found in the configurations above. As illustrated in Fig.5, the mean ℓ_2 distance between $\hat{f}^{(\ell')}(\tilde{\theta}_{(1,2)}; x)$ and $\hat{f}^{(\ell')}(\theta_2; x)$ decreases as layer ℓ' increase.

Convergence to Same Point Similar to the settings in the main text, we use our NNG technique to check whether the feature distributions learned in two nets θ_1 and θ_2 intilizied with two different methods (i.e., $he_{uniform}$ and he_{normal}) converge to the same distribution as the training process goes on. The configurations of θ_1 and θ_2 are chosen from (a) and (b) in Table 2, i.e., θ_1 and θ_2 are VGG-16 trained on CIFAR-100, or θ_1 and θ_2 are VGG-19 trained on CIFAR0-10. For saved checkpoints, we graft each layer ℓ of θ_1 onto layer $\ell + 1$ of θ_2 and the results are presented in Fig.6. We can see that the increased valid error of $\tilde{\theta}_{(1,2)}$ grafted at each layer is very high in the beginning (since θ_1 and θ_2 are initialized with two quite different feature distributions) and decreases quickly as training process goes on, and finally almost all the curves finally bounded in low-error region. This implies that the feature distributions learned by θ_1 and θ_2 start from different points and converge to a same point.

F VGG is Over-parameterized Enough in Feature Distribution View

In this section, we conduct experiments to show that VGG-16 are overparameterized enough. To be precise, we verify this point by showing that the feature representations learned by VGG-16 are stable, in a sense that, the learned feature representations would be quite similar if we increase the numbers of channels/hidden nodes in each layer of VGG-16 by 2 times.

We first train two networks θ_1 and θ_2 on CIFAR-10, where one of θ_1 and θ_2 is a standard VGG-16 and the other is VGG-16 with twice number of hidden nodes/channels compared with the standard one. We compare the feature distributions learned in them during the training process by grafting the checkpoints of θ_1 and θ_2 saved at the same time-step together, just as we did in the experiments for showing the uniqueness of same solution path in the main text. The results are shown in Fig.7. We

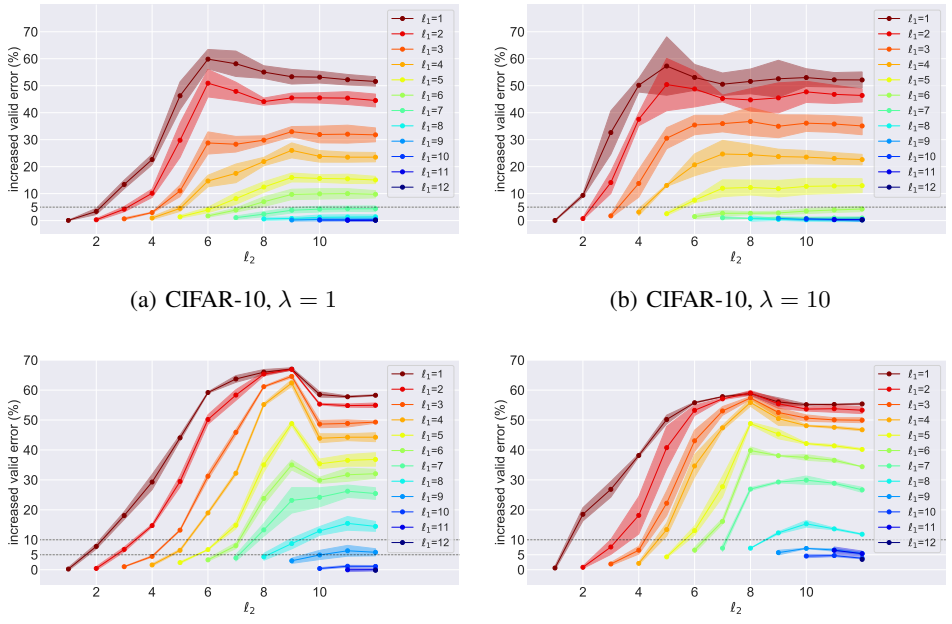
could see that the increased validation of $\tilde{\theta}_{(1,2)}$ compared with θ_2 is accumulated in low-error region for all layers and time-steps, which verifies our claim empirically.

G Additional Findings

G.1 Feature Distributions Comparison between Different Layers

We next use our proposed NNG metric to compare the similarity of feature function distributions between different layers, i.e., layer ℓ_1 of θ_1 and layer ℓ_2 of θ_2. Here θ_1 and θ_2 are neural network trained from two different initializations with same $\ell_{1,2}$ regularizer on same dataset, $\tilde{\theta}_{(1,2)}$ is the network grafted at two non-adjacent layers. To specific, we use the feature representations at layer ℓ_1 in θ_1 to be the input for the feature representations at layer $\ell_2 + 1$ in θ_2 and train the weights and bias parameters using a ℓ_2 regression loss, see Section D for details. The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2 is reported in Fig.8. From Fig. 8a), we can see that for the fully trained VGG-16 on CIFAR10, the feature representations in the layers near the input are relatively different from each other, while those learned in the layers near the output are similar with each other. We can also observe that the feature representations learned in the six layers near the input are quite different from those learned in the subsequent layers, since when two of them are grafted together, the validation error increases significantly by at least 20%. To the best of our knowledge, we are the first to find this property of DNNs in feature learning.

Another observation is the *measured similarity between different layers using our NNG metric is different for small and large regularization parameters*. Since the results and core conclusions are similar for CIFAR-10 and CIFAR-100, we just provide analysis for CIFAR-10 below. To be precise, Fig. 8b) shows that for large regularization parameter $\lambda = 10$, the back seven layers near output are almost the same, while when it comes to small regularization parameter $\lambda = 1$ only five layers near the output are similar.



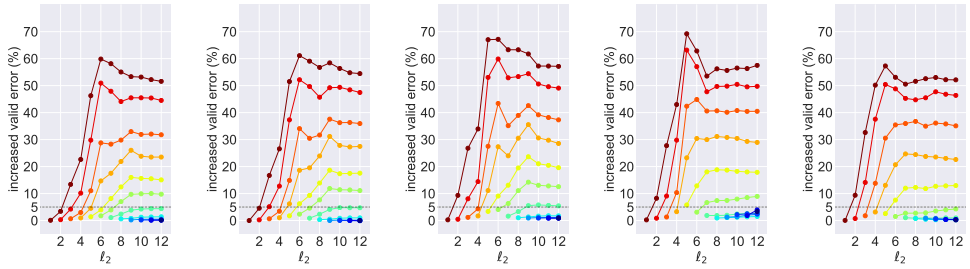
(a) CIFAR-10, $\lambda = 1$

(b) CIFAR-10, $\lambda = 10$

(c) CIFAR-100, $\lambda = 1$

(d) CIFAR-100, $\lambda = 4$

Figure 8: The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2, where θ_1 and θ_2 are trained VGG-16 using same $\ell_{1,2}$ regularizer parameter but different initialization. For each curve denote as ℓ_1, we plots the increased validation error when we directly graft the first ℓ_1 layers in θ_1 on the $\ell_2 + 1$-th layer in θ_2, which demonstrate whether the feature functions at layer ℓ_1 in θ_1 is similar to that at layer ℓ_2 in θ_2. The shadowed regions represent the 95% confidence interval.



(a) $\lambda = 1$ (b) $\lambda = 2$ (c) $\lambda = 4$ (d) $\lambda = 8$ (e) $\lambda = 10$

Figure 9: The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2, where θ_1 and θ_2 are trained VGG-16 using same $\ell_{1,2}$ regularizer parameter but different initialization. For each curve denote as ℓ_1 (we omit the legend of ℓ_1 here and it is same to that in Fig 8), we plots the increased validation error when we directly graft the first ℓ_1 layers in θ_1 on the $\ell_2 + 1$-th layer in θ_2, which demonstrate whether the feature functions at layer ℓ_1 in θ_1 is similar to that at layer ℓ_2 in θ_2.

The above observation indicates that: 1) our proposed NNG metric can differentiate the similarity and difference effectively; 2) larger regularization parameter will promote more layers near the final output of a neural network to learn similar feature representations.

To better demonstrate 2), we further analyze the results about how these curves change when λ varies from 1 to 10, the results are shown in Fig 9. We could draw two observations of curves' changes if we choose 5% as the threshold to differentiate similar or not. One is that the front several layers are similar with their adjacent layers when λ is small, when λ becomes larger, these adjacent layers become different. The other is more layers near the output become similar when λ becomes larger.

G.2 Feature Distributions Learned on Different Datasets

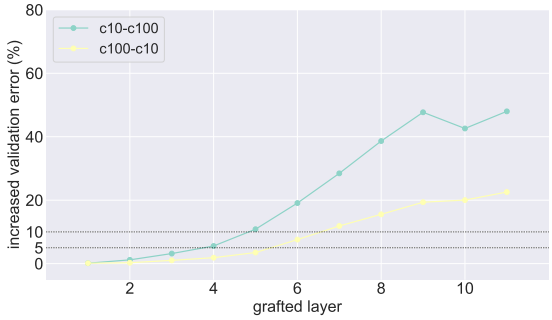
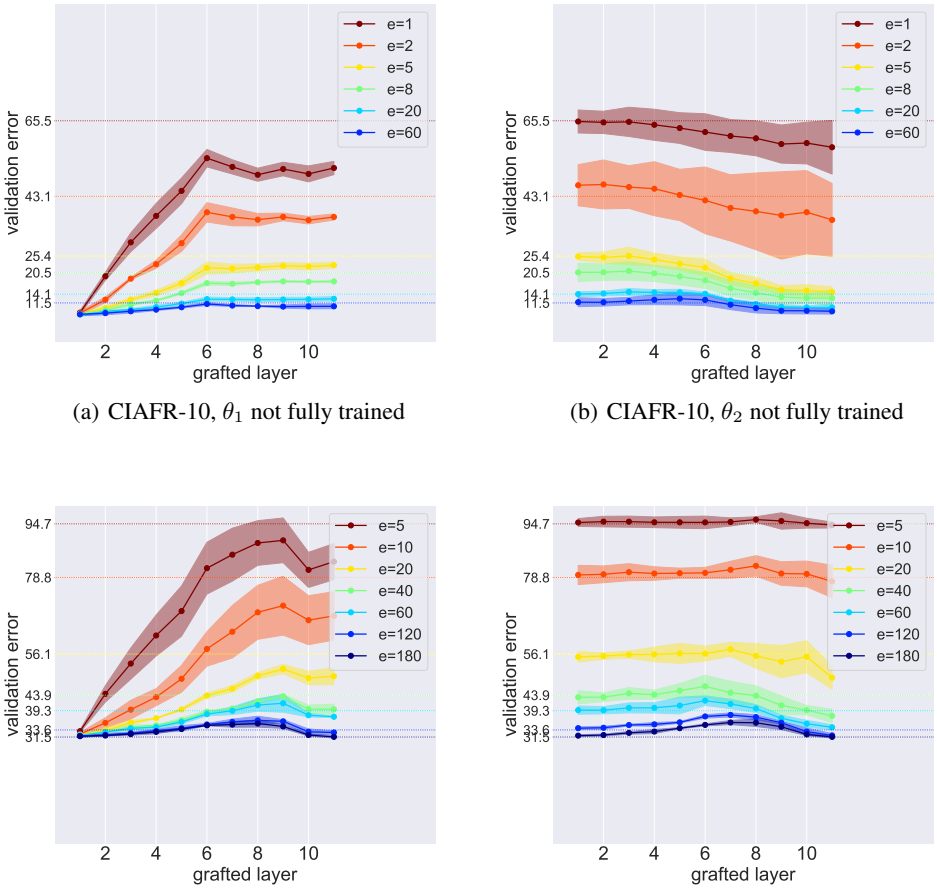


Figure 10: The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2, where θ_1 and θ_2 are trained VGG-16 $\ell_{1,2}$ regularizer with parameter $\lambda = 1$ but different initialization and different dataset. θ_1 and θ_2 are trained with CIFAR-10 and CIFAR-100 respectively for curve 'c10-c100' and are trained with CIFAR-100 and CIFAR-10 respectively for the curve 'c100-c10'

In the section, we investigate the similarity of feature representations learned on different datasets. To be specific, we train two networks θ_1 and θ_2 on CIFAR-10 and CIFAR-100, respectively and compare the feature representations learned by them using our NNG technique. The increased validation error of $\tilde{\theta}_{(1,2)}$ compared with θ_2 is shown in Fig.10, from which we can draw the following conclusions:

1. Our proposed NNG metric can clearly identify whether two feature learned from different dataset are similar or not. We can see that the feature representations learned from different datasets are quite similar in the layers near the input layer while have a large gap in the layers near the final output.

2. We could see that the performance of grafted network $\tilde{\theta}_{(1,2)}$ comprised of the near-input layers of the network trained CIFAR-10 and the near-output layers of the network on CIFAR-100 is significant worse than the one comprised the near-input layers of the network on CIFAR-100 and the near-output layers on CIFAR-10, this could be contributed to two factors: 1) CIFAR-100 task is more difficult than CIFAR-10 task; 2) the feature representation learned on CIFAR-100 are more diverse.



(a) CIAFR-10, θ_1 not fully trained

(b) CIAFR-10, θ_2 not fully trained

(c) CIAFR-100, θ_1 not fully trained

(d) CIAFR-100, θ_2 not fully trained

Figure 11: Increased validation performance of $\tilde{\theta}_{(1,2)}$ compared with θ_2. The shadowed regions represent the 95% confidence interval. For each subplot, $e = [u]$ represents θ_k is the model after u training epochs and the other network is the converged model. The dotted line is the validation error of corresponding θ_k after u training epochs.

G.3 Feature Distributions Learned at Different Stages

In this section, we consider the case where one of θ_1 and θ_2 is not fully trained. Fig 11 shows that: 1) our proposed metric is identifiable to the case when one of θ_1 and θ_2 has not trained to optimal; 2) the performance gap between the grafted NN and the original NN quickly decreases to zero for the previous few layers and slowly decreases for the last few layers. This indicates that the feature functions in the previous few layers converge much faster than that of the last few layers.