Low-loss connection of weight vectors: distribution-based approaches

Ivan Anokhin [1] **Dmitry Yarotsky** [1]

Abstract

Recent research shows that sublevel sets of the loss surfaces of overparameterized networks are connected, exactly or approximately. We describe and compare experimentally a panel of methods used to connect two low-loss points by a low-loss curve on this surface. Our methods vary in accuracy and complexity. Most of our methods are based on "macroscopic" distributional assumptions, and some are insensitive to the detailed properties of the points to be connected. Some methods require a prior training of a "global connection model" which can then be applied to any pair of points. The accuracy of the method generally correlates with its complexity and sensitivity to the endpoint detail.

1. Introduction

Though loss surfaces of neural networks have a complex shape, it is generally accepted that large networks train well and their performance is not very dependent on the weight initialization, despite apparently different trained values resulting from different initializations (Choromanska et al., 2015). When thinking of the landscape of a complex nonconvex function such as a loss surface, one can imagine different heuristic scenarios for the structure of the bottom of the surface (Baity-Jesi et al., 2018). One scenario is that the loss function has multiple isolated local minima. Another scenario is that there are, in contrast, few local minima, and the sub-level sets of the loss function have only one or a small number of connected components (despite their possibly complex shape). Of course, the second scenario agrees better with the practically observed efficiency of network training by gradient descent. In general, the second scenario is more likely in the setting of overparameterized networks (small networks are known to host numerous isolated local minima, see e.g. (Safran & Shamir, 2017)).

Recent research provides some further evidence in favor of the "connected sublevel set" scenario. A particular easy-to-formulate task that one can analyze both experimentally and theoretically is:

Given two low-loss weight vectors Θ^A, Θ^B, connect them by a low-loss curve. (1)

Recent studies of this connectedness problem can be divided into numerical and theoretical ones. The numerical studies have been performed in (Garipov et al., 2018; Draxler et al., 2018). In these papers, the desired low-loss curves are constructed by numerically optimizing the curves connecting the two given low-loss points. The results show that typically one can find a curve on which the loss value is only slightly worse than at the endpoints.

The theoretical studies rigorously confirm this effect under certain conditions (generally, overparameterization-related). For a single-hidden- layer ReLU network, (Freeman & Bruna, 2016) prove that two weight vectors with loss $\leq l_0$ can be connected by a curve with loss $\leq l_0$ if the number of hidden neurons is sufficiently large. For pyramidal multilayer networks with piecewise linear activation functions, (Nguyen, 2019) proves that in the overparameterized setting (when the size of the first hidden layer is larger than the size of the training set), sublevel sets are connected and unbounded. (Kuditipudi et al., 2019) assume that the model is dropout-stable or noise-stable, and then construct a connecting path with a low loss.

Obviously, these experimental and theoretical works have quite different methodologies. The paths found in the experimental studies are numerically optimized to particular endpoints, and the structure of these optimal paths is not well understood. On the other hand, while the theoretical works offer some explicit rigorous constructions of low-loss paths, it is not clear to which extent they match the experimentally found ones.

Motivated by this discrepancy, in the present paper we adopt a somewhat different point of view on task (1), putting forward this general goal:

Describe universally applicable and possibly simple methods that, given two weight vectors Θ^A, Θ^B produce connecting curves of possibly low loss.

[1] Skolkovo Institute of Science and Technology, Moscow. Correspondence to: Ivan Anokhin <i.anokhin.mm@gmail.com>, Dmitry Yarotsky <d.yarotsky@skoltech.ru>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

With this goal in mind, we propose a panel of methods of different complexity and accuracy, bridging the gap between the above numerical and theoretical studies.

In contrast to the numerical optimization of (Garipov et al., 2018; Draxler et al., 2018), we aim to construct the connecting curve by a more-or-less direct prescription (to a varying degree, depending on the method). While the above numeric optimization papers demonstrate but do not explain the connectedness phenomenon, our methods logically follow from either the particular form or certain assumptions about the trained networks.

On the other hand, in contrast to the mentioned theoretical studies, we are interested in "general-purpose" low-loss connection methods that are, in principle, applicable to any pair of endpoints Θ^A, Θ^B, any network size, and any training data. To clarify this point, consider the most trivial connection performed by a straight line segment: $t \mapsto \Theta(t) = (1-t)\Theta^A + t\Theta^B$. The performance of this method is quite poor (the loss can grow significantly for intermediate t), but the method is universally applicable, given by an explicit analytic formula, and the geometry of the path is essentially independent of the values Θ^A, Θ^B. The papers (Freeman & Bruna, 2016; Nguyen, 2019) represent another extreme case, where Θ^A and Θ^B are connected using a complex path and meticulous adjustment of individual neuron parameters (and only under rather restrictive assumptions on the model), but achieving a perfect solution of task (1). In the present paper, we are interested in the intermediate setting: possibly generally applicable, endpoint-insensitive methods with possibly simple paths, yet improving performance of the trivial straight-line connection.

2. Our contribution and the structure of the paper

We start by considering networks with a single hidden layer (Section 3). Our connection methods are largely motivated by the "macroscopic" view of the network as a sample from some distribution in a suitable state space of neurons; we recall this picture in Section 3.1.

- In Section 3.2 we describe the idea of connections preserving the neuron distribution, and specifically describe "Arc Connection" which is an analytic method essentially as simple as the trivial segment connection, but preserving the variance of the neuron distribution. The Arc Connection is a perfect solution of the connection problem in the limit of infinitely wide networks if the neurons are normally distributed.

- In Section 3.3 we generalize Arc Connection to non-Gaussian distributions of neurons. To this end, we introduce "learnable methods" aimed to learn the neuron distribution in a typical local minimum of the loss function. In this way, we construct a "global connection model" that can be used subsequently to connect any two new local minima.

- In Section 3.4 we describe "Optimal Transportation" methods in which the connecting path consists of two stages. In the first stage the distribution of neurons in one local minimum is optimally transported to the distribution in another minimum, and in the second stage the neurons are permuted to be in the required order.

- Finally, in Section 3.5 we describe what we call "Weight Adjustment" methods in which the first layer weights are connected by a simple analytic prescription while the second layer weights (on which the network output depends linearly) are adjusted appropriately, by solving suitable linear systems.

In Section 4 we extend these methods to multi-layer networks (by a suitable layer-wise reduction) and convnets.

In Section 5 we perform an experimental comparison of these connection methods.

Finally, in Section 6 we discuss one potential practical application of the connection task: one can use low-loss connections between different low-loss weight vectors to form an "ensemble" of networks with an accuracy slightly better than that of the individual networks. In contrast to conventional ensembles, this can be achieved with only a small computational overhead on inference, by reusing initial computation of one of the networks.

3. One Hidden Layer

3.1. Reduction to Distributions

The theory of networks with a single hidden layer can be relatively easily translated into the language of distributions, so that the network output, the loss function, and the gradient descent are described in terms of the weight distributions rather than individual values. We sketch the main ideas, referring the reader to the papers (Mei et al., 2018; Rotskoff & Vanden-Eijnden, 2018; Sirignano & Spiliopoulos, 2018; Chizat & Bach, 2018) for details and precise statements.

Consider the predictive model of the form

$$\widehat{\mathbf{y}}_n(\mathbf{x}; \Theta) = \frac{1}{n}\sum_{i=1}^{n} \sigma(\mathbf{x}; \theta_i), \qquad (2)$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input, $\Theta = \{\theta_i\}_{i\in[n]}$ is the collection of weights $\theta_i \in \mathbb{R}^D$, and $\sigma : \mathbb{R}^d \times \mathbb{R}^D \to \mathbb{R}^m$ is some map. In particular, we obtain the standard fully-connected neural network with a single hidden layer by setting $\theta_i =$

$(b_i, \mathbf{l}_i, \mathbf{c}_i) \in \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^m$ and $\sigma(x; \theta_i) = \mathbf{c}_i \phi(\langle \mathbf{l}_i, x_i \rangle + b_i)$ with a scalar activation function ϕ. Each term in the sum then corresponds to a hidden neuron, see Fig. 1.

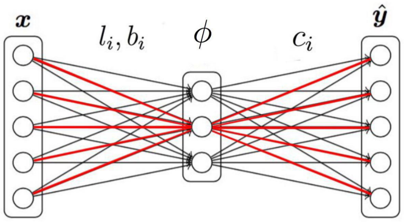

Figure 1: A graphical representation of a one hidden layer network, as in Eq. (2). Weights of one particle $\theta_i = (b_i, \mathbf{l}_i, \mathbf{c}_i)$, colored in red, resemble a butterfly.

Let us now write predictive model (2) in the form

$$\widehat{\mathbf{y}}(\mathbf{x}; p) = \int \sigma(\mathbf{x}; \theta) p(d\theta), \tag{3}$$

where p is the normalized counting measure on the space $\mathbb{R}^D$ concentrated at the weights θ_i: $p = \frac{1}{n}\sum_{i=1}^n \delta_{\theta_i}$. One important advantage of this new representation in terms of p is that we get rid of the excessive degree of freedom associated with permutations of neurons. Another advantage is that formula (3) naturally generalizes to any measure p on $\mathbb{R}^D$. Finally, while the representation (2) is, in general, not linear in the weights θ_i, the representation (3) is linear in p, so that if the loss function is convex in $\widehat{\mathbf{y}}$, it is also convex in p. In the sequel, we will assume this convexity of the loss.

The gradient descent for the model (2) can also be described in terms of p, by a suitable integro-differential equation, and the dependence of the GD trajectory on the initial distribution is sufficiently regular. This suggests the following approach to the connection task (1). Suppose that the weight vectors $\Theta^A, \Theta^B \in \mathbb{R}^D$ have been obtained by optimizing the loss function starting from two different random initializations Θ_0^A, Θ_0^B obtained by sampling the weights independently from the same initial distribution $p = p_0$ (for example, by sampling the weights as i.i.d. normal variables, as is the usual practice). Then, by the law of large numbers, for a large network we expect the initial distributions $p_0^A = \frac{1}{n}\sum_{i=1}^n \delta_{\theta_{0,i}^A}, p_0^B = \frac{1}{n}\sum_{i=1}^n \delta_{\theta_{0,i}^B}$ to be close to p_0, and hence expect their whole gradient descent trajectories to be close in the space of distributions. Accordingly, the final optimized weight vectors Θ^A, Θ^B should also be described by approximately the same distribution p. Then, to connect the points Θ^A and Θ^B by a low-loss path $\psi : t \in [0,1] \mapsto \Theta(t) \in \mathbb{R}^D$, we want to choose it in such a way that the weight distribution for $\Theta(t)$ is also approximately equal to p for all t. If we manage to do so, the output of neural networks along this path will be approximately the same:

$$\widehat{\mathbf{y}}_n(\mathbf{x}; \psi(t)) \approx \widehat{\mathbf{y}}_n(\mathbf{x}; p). \tag{4}$$

3.2. Distribution Preserving Methods

The above arguments suggest reducing the connection task (1) to constructing a "distribution-preserving" deformation. Specifically, let X and Y be two independent random vectors of length D sampled from an unknown distribution p on $\mathbb{R}^D$. We want to construct a continuous path $\psi : [0,1] \to \mathbb{R}^D$ such that $\psi(0) = X, \psi(1) = Y$, and the distribution of the random vector $\psi(t)$ is p for any $t \in [0,1]$. Once we have such a method, we can apply it to connect the network weight vectors Θ^A, Θ^B in a component-wise way:

$$\Theta(t) = (\psi_i(t))_{i=1}^n,$$

where ψ_i connects $X = \theta_i^A$ to $Y = \theta_i^B$.

Now we will consider several particular methods of connecting X to Y, and we start from the trivial baseline:

Linear Connection is the basic most naive way to connect two weight vectors:

$$\psi(t) = (1-t)X + tY. \tag{5}$$

Note that this method is not measure-preserving, in general: if $X, Y \sim p$, then $\psi(t) \not\sim p$ for $t \in (0,1)$. This can be seen, for example, by considering the covariance matrix $\Sigma_{\psi(t)}$ of $\psi(t)$, which is equal to $(1-t)^2\Sigma_X + t^2\Sigma_Y = ((1-t)^2 + t^2)\Sigma_p \neq \Sigma_p$ (so the Linear Connection "squeezes" the distribution p). This explains why performance of the Linear Connection is typically rather poor.

The following proposition suggests how to modify formula (5) to make the connection measure preserving in the case of a multivariate Gaussian distribution p (see Fig. 2).

Proposition 1 *If X, Y are i.i.d. vectors with the same centered multivariate Gaussian distribution p, then for any $t \in \mathbb{R}$, $\psi(t) = \cos(\frac{\pi}{2}t)X + \sin(\frac{\pi}{2}t)Y$ has the same distribution p, and also $\psi(0) = X, \psi(1) = Y$.*

One can easily prove this known fact by using characteristic function of multivariate normal distribution. We can then give our first improvement of Linear Connection.

Arc Connection is the method that assumes that X, Y are already Gaussian with the same covariance matrix and center: $\mu = \mathbb{E}X = \mathbb{E}Y$. Then, we set:

$$\psi(t) = \mu + \cos(\tfrac{\pi}{2}t)(X - \mu) + \sin(\tfrac{\pi}{2}t)(Y - \mu). \tag{6}$$

However, the assumed normality is a severe restriction: the distribution of weights is non-normal in general. We can

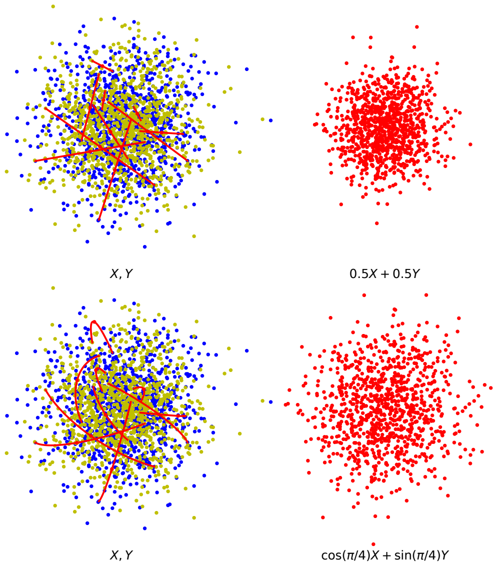

Figure 2: Connection of two samples $X, Y \sim \mathcal{N}(0, \mathbf{1}_{2\times 2})$. **Top:** Linear, Eq.(5), squeezes the distribution. **Bottom:** Arc, Eq.(6), preserves the distribution.

generalize the Arc Connection to non-normal X, Y by considering a general transformation ν making X, Y normal:

$$\psi(t) = \nu^{-1}[\cos(\tfrac{\pi}{2}t)\nu(X) + \sin(\tfrac{\pi}{2}t)\nu(Y)] \qquad (7)$$

(see Fig. 3). In practice, we don't know the map ν, but we can try to learn a suitable map from the data. This leads us to what we refer to as learnable connection methods.

3.3. Learnable Connection Methods

We want to learn a suitable transformation ν in Eq. (7). For this purpose we propose to use neural network architectures that support inverse transformation (note that Eq. (7) requires us to compute both ν and its inverse). Such architectures are often used in normalizing flows methods, which aim to transform simple known probability distribution (e.g. Gaussian) into a complicated multi-modal one and still be able to compute the probability of the point. The training and fast inference of the models are achieved by using transformations whose Jacobian determinants are easy to compute.

Our learnable methods are characterized by two elements: the network architecture used to compute ν, and the optimization algorithm used to train the network. We utilize two architectures as ν. The first one is the **RealNVP** model as described in (Dinh et al., 2016), the second one is the Inverse Autoregressive Flow (**IAF**) model as described in (Kingma et al., 2016). To emphasize that now transformation ν has parameters, we write it as ν_W, where W are the weights of one of the above two networks.

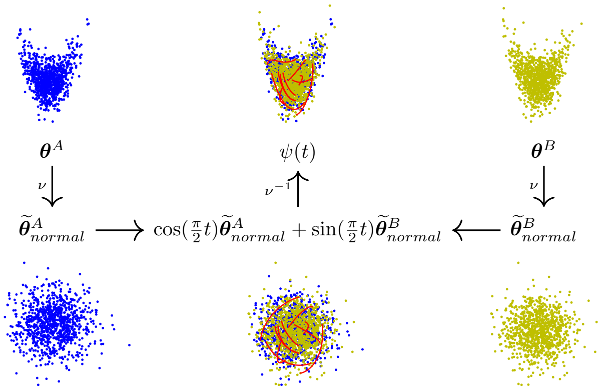
$\boldsymbol{\theta}^A$
$\psi(t)$
$\boldsymbol{\theta}^B$
ν
ν^{-1}
ν
$\widetilde{\boldsymbol{\theta}}^A_{normal}$
$\cos(\frac{\pi}{2}t)\widetilde{\boldsymbol{\theta}}^A_{normal} + \sin(\frac{\pi}{2}t)\widetilde{\boldsymbol{\theta}}^B_{normal}$
$\widetilde{\boldsymbol{\theta}}^B_{normal}$

Figure 3: General distribution-preserving path, Eq.(7), maps target distributions $\boldsymbol{\theta}^A, \boldsymbol{\theta}^B$ to standard normal, mixes them, and maps the mix back.

We consider two optimizing procedures, **Flow** and **Bijection**. In **Flow**, we follow the algorithm proposed in (Dinh et al., 2016) and maximize the likelihood $\mathbb{E}_{\mathbf{x}\sim p} \log \left[\eta(\nu(\mathbf{x}))|\det \frac{\partial \nu(\mathbf{x})}{\partial \mathbf{x}}|\right]$, where η is the standard normal probability density function. After training is done, we can use Eq. (7) to generate samples along the path. Note that the transformation ν should map samples from the target distribution to the standard Gaussian if the training procedure is successful.

We also propose a new training procedure which we call **Bijection**. Assume we have a dataset $V = \{\Theta\}$ of low loss weight vectors for a One Hidden layer network. We can easily create such V by training models that minimize any user-specified loss $L(\Theta)$. Now we want to have low loss for any two models in V and any point t on the curve (7) that is convenient to rewrite as

$$\psi_W(t, \Theta^A, \Theta^B)$$
$$= \nu_W^{-1}[\cos(\tfrac{\pi}{2}t)\nu_W(\Theta^A) + \sin(\tfrac{\pi}{2}t)\nu_W(\Theta^B)].$$

Similarly to (Garipov et al., 2018), in order to achieve this we propose to optimise the computationally tractable loss

$$l(W) = \mathbb{E}_* L(\psi_W(t, \Theta^A, \Theta^B)), \qquad (8)$$

where expectation $\mathbb{E}_*$ is w.r.t. $t \sim U(0,1), \Theta^A \sim U(V), \Theta^B \sim U(V \setminus \Theta^A)$. To minimize Eq. (8), at each iteration we sample $\widehat{t}$ from the uniform distribution $U(0,1)$, $\widehat{\Theta}^A, \widehat{\Theta}^B$ are drawn from V uniformly in a way that $\widehat{\Theta}^A \neq \widehat{\Theta}^B$, then we make a gradient step for W with respect to the loss $L(\psi_W(\widehat{t}, \widehat{\Theta}^A, \widehat{\Theta}^B))$. We repeat these updates until convergence.

We have found experimentally that it is usually sufficient to optimize the model only in the middle point $t = 0.5$, as the model tends to always have the highest loss there:

$$l(W) = \mathbb{E}_{\Theta^A \sim U(V), \Theta^B \sim U(V\setminus\Theta^A)} L(\psi_W(0.5, \Theta^A, \Theta^B)).$$

Strictly speaking, **Bijection**-based connection methods are not constructed as distribution-preserving along the path, but they are expected to generate low-loss paths between any similarly trained models. However, let us note that one possible solution for **Bijection** procedure is to learn a map to centered Gaussian distribution.

We name learnable methods in the following manner: the first part of the name is a network architecture name, and the second is a training procedure name. Combining various architecture and training procedure, we get four connection methods: **RNVP Flow**, **IAF Flow**, **RNVP Bijection** and **IAF Bijection**. However, training a network with **Bijection** requires a fast computation of ν_W and ν_W^{-1}. For this reason, in this case we use only **RealNVP** networks, not **IAF**.

Note that the approach proposed in this section can be described as "training a global connection model". We perform a single initial training of this model, but once done, we can connect any pair of unseen samples from the distribution p using our learned transformation ν_W. In terms of connecting network weights, this means that we can use this global model to connect any pair of weight vectors, assuming they have been trained in the way similar to the one used to generate the training data for the global connection model.

3.4. Optimal Transportation Methods

We consider now an alternative approach (referred to as **OT** in the sequel) that is also based on the idea of connecting two distributions, but attempts to do it by taking into account the whole set of "butterflies". Specifically, we use a version of Optimal Transportation (OT) in the neuron state space $\mathbb{R}^D$ to connect the sample of hidden neurons of the network A to that of B. If the number of neurons is large, then we can find a bijective map between the neurons of A and B that maps each neuron of A to a nearby neuron of B. In this way, we can transform the network A to a network isomorphic to B (namely, different from B only by the order of hidden neurons) by a short linear segment in the full weight space $\mathbb{R}^{nD}$, so that the distribution of neurons remains approximately constant on this segment. We use the POT library (Flamary & Courty, 2017) for the solution of this OT problem.

Note, however, that the OT transformation alone does not solve our connection task, since this task requires us to connect each neuron of A to a particular target neuron of B, i.e., keep the prescribed order of neurons. Therefore, we supplement the above OT-stage of the path by the "permutation" stage. This second stage can be implemented by a continuous piecewise linear curve adjusting the neurons one-by-one. In each step, a pair of neurons is swapped placing one of them at the required position. The swap can be implemented by a singe linear transformation. Since the contribution of each hidden neuron to the network output is $O(1/n)$ and completed swaps do not change the network output, this path maintains low values of the loss function.

3.5. Joint Weight Adjustment

For completeness, we also consider a connection method that goes beyond the distributional picture and uses a direct analytic weight adjustment for the given pair of weight vectors Θ^A, Θ^B. Let us write a network with a single hidden layer in the standard form

$$\widehat{\mathbf{y}} = W_2\phi(W_1\mathbf{x}),$$

where W_1, W_2 are matrices (of size $d_1 \times d_0, d_2 \times d_1$, respectively), and the activation function ϕ is meant to act separately on each component of the vector $W_1\mathbf{x}$. For simplicity, we do not include the bias terms in this formula (the bias can be introduced in the first layer by assuming that $\mathbf{x}$ has an additional component with a constant value).

Let $\Theta^A = (W_1^A, W_2^A)$ and $\Theta^B = (W_1^B, W_2^B)$ be two weight vectors for which the network has close outputs. This condition can be written as follows. Let $\mathbf{X}$ be the $d_0 \times N$ matrix of the set $S = \{\mathbf{x}_q\}_{q=1}^N \in \mathbb{R}^{d_0}$ of N input vectors on which we consider the action of the network. On this set S, the network output can be written as

$$\widehat{\mathbf{Y}} = W_2\phi(W_1\mathbf{X}).$$

Let $\widehat{\mathbf{Y}}^A$, $\widehat{\mathbf{Y}}^B$ be the outputs with the weight values Θ^A, Θ^B; we then assume that $\widehat{\mathbf{Y}}^A \approx \widehat{\mathbf{Y}}^B$.

We choose now a path $\Theta = \Theta(t) = (W_1(t), W_2(t)), t \in [0, 1]$, that connects Θ^A to Θ^B and approximately preserves the output $\widehat{\mathbf{Y}}$. To this end, we first connect W_1^A to W_1^B in a more or less arbitrary way, for example using the basic linear connection $W_1(t) = (1-t)W_1^A + tW_1^B$. Then, we adjust the weights in the second layer, which essentially means that we need to solve the linear system

$$W_2(t)\phi(W_1(t)\mathbf{X}) \approx \widehat{\mathbf{Y}}^A \tag{9}$$

for $W_2(t)$, at each $t \in [0, 1]$. A solution can be written as

$$W_2(t) = \widehat{\mathbf{Y}}^A \Big[\phi(W_1(t)\mathbf{X})\Big]^+, \tag{10}$$

where $[\cdot]^+$ denotes the pseudo-inverse matrix. In general, this solution may be discontinuous in t (at the points where the rank of $\phi(W_1(t)\mathbf{X})$ changes), and the boundary values $W_2(0), W_2(1)$ may be different from W_2^A, W_2^B. The last issue is related to the degeneracy of the system (9) and, by linearity, can be resolved simply by adding to the path (10) two extra legs linearly connecting W_2^A to $W_2(0)$ and $W_2(1)$ to W_2^B. As for discontinuity, we resolve this issue by applying Eq. (10) only at finitely many values of t, and

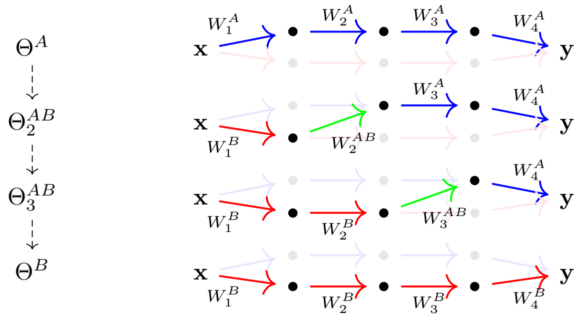

Figure 4: Intermediate points on the path from a four-layer network A to a network B. Starting from the first layer, the weights of model A are gradually replaced with weights of model B.

forming the full path as the piecewise linear curve with these breakpoints.

Let us introduce the following name convention for variants of this weight adjustment method. The first part of the name refers to the distribution interpolation method that we use to connect the weight vectors W^A_1 and W^B_1. We connect them by considering the lines of the matrix W_1 as sampled from an unknown distribution p. The second part of the name emphasizes that we use the weight adjustment in the second layer. In particular, we make experiments with the methods **Linear + Weight Adjustment**, **Arc + Weight Adjustment** and **OT + Weight Adjustment**.

4. Extensions to more complex networks

4.1. Multi-layer networks

Let us introduce some additional notations: $\mathbf{X}_k = \phi(W_k\phi(\ldots\phi(W_1\mathbf{X})\ldots))$ is the input for the layer k, $\mathbf{X}_0 = \mathbf{X}$ is the initial input for the network, $W^{AB}_{k+1} = W^A_{k+1}\mathbf{X}^A_k\left[\mathbf{X}^B_k\right]^+$ is the weight adjustments of the k'th layer of network A to the k'th layer of network B (as in Eq. (10)), and $\Theta^{AB}_k = \{W^B_1, ..., W^B_{k-1}, W^{AB}_k, W^A_{k+1}, W^A_{k+2}, ..., W^A_n\}$ are intermediate points that we cross on the way from the weights Θ^A to Θ^B.

We propose to connect two weight vectors, Θ^A and Θ^B, of a multi layer dense net with the following intermediate points: $\Theta^A \to \Theta^{AB}_2 \to \Theta^{AB}_3 \to, ..., \to \Theta^{AB}_n \to \Theta^B$ (see Fig. 4). The output of the network at any intermediate point Θ^{AB}_k is approximately equal to $\widehat{\mathbf{Y}}^A$ as we have appropriately adjusted the weights W^{AB}_k in the layer k.

To connect $\Theta^{AB}_n \to \Theta^B$ we need to change only the last layer. We can use any of our methods to do so. Note that it is sufficient to use the simple linear interpolation if the loss function is convex with respect to the last layer.

To connect any intermediate points $\Theta^{AB}_k \to \Theta^{AB}_{k+1}$ or $\Theta^A \to \Theta^{AB}_2$ note that Θ^{AB}_k and Θ^{AB}_{k+1} differ only in layers k and $k+1$. So we can consider these two layers in Θ^{AB}_k and Θ^{AB}_{k+1} as One Hidden layer subnetworks. The inputs of these subnetworks are identical, and the outputs are approximately the same thanks to the weight adjustment. This means we can use any method we describe in Section 3 to connect the weights of these subnetworks.

The name convention is similar to the one we use for One Hidden layer network. We refer to the method as **Linear + Butterfly**, **Arc + Butterfly** or **OT + Butterfly** if we connect One Hidden layer subnetworks of intermediate points using the Butterfly weight representation and one of our distributional method. Alternatively, in the methods **Linear + Weight Adjustment**, **Arc + Weight Adjustment** and **OT + Weight Adjustment** we consider the rows in the weight matrix of the first subnetwork layer as samples, connect them with one of our methods, and perform weight adjustment on the second layer.

Let us also note that in case of **Butterfly**–methods we can skip the Θ^{AB}_n intermediate point from the proposed path, so it becomes $\Theta^A \to \Theta^{AB}_2 \to \Theta^{AB}_3 \to, ..., \to \Theta^{AB}_{n-1} \to \Theta^B$.

4.2. CNNs and networks with skip connections

The connection methods described above can be naturally generalized to convnets. In this case, the analog of the distribution of neurons would be the distribution of filters (since different filters can be viewed as independent, permutable entities). Of course, the distributional point of view should be more efficient if the number of filters is large. Our experiments below include connection of convnets such as VGG16.

In the present paper we do not consider connection for networks with skip connections such as ResNets, mainly because the implementation in this case is relatively complex. We remark, however, that it is rather clear how that can be done by generalizing the stepwise procedure of Section 4.1: proceed layer-by-layer; in each layer, connect directly the weights sitting on all the incoming edges from earlier layers, and then adjust accordingly all the outgoing edges.

5. Experiments

In this section, we test experimentally the proposed connection methods on the datasets CIFAR10 and MNIST. For each method, we measure the worst accuracy that the method provides along the path.[1] For both datasets, we use the standard train–test split.

All considered models were trained using the cross-entropy

[1] We release source code at `https://github.com/avecplezir/distribution-based-connectivity`

loss with the SGD optimizer, for 400 epochs and 30 epochs on CIFAR10 and MNIST, respectively, with learning rate 0.01 and batch size 128. For CIFAR10 we use the same standard data augmentation as (Huang et al., 2017). For MNIST we do not use any augmentation. The activation function in all the networks is ReLU.

We compare our methods with connection curves numerically found in (Garipov et al., 2018). In Table 1, **Garipov (3)** refers to the polygon with two segments, **Garipov (5)** refers to the polygon with four segments between the end points. Each Garipov's curve was optimized for 200 and 60 epochs for CIFAR10 and MNIST datasets, respectively, with batch size 128 as described in the original paper.

5.1. One Hidden Layer

Table 1 shows results for One Hidden Layer networks with 2000 hidden neurons, on MNIST and CIFAR10.

As explained in Section 3.3, learnable methods (IAF flow, RealNVP bijection) require us to first collect a set V of low-loss weight vectors Θ, to be used for learning the connection methods. We created such a set of 16 models using the same training procedure but different random weight initializations and dataset augmentations.

The "train" and "test" columns in Table 1 refer to the respective subsets of MNIST and CIFAR10. In the case of learnable connection methods, learning only used the training part of the dataset; moreover, both "train" and "test" results were computed for endpoints Θ^A, Θ^B not belonging to the model set V used to learn the connection method. IAF Flow failed to converge on CIFAR10. In the methods involving Weight Adjustment (Section 3.5), adjustment of the second layer was also performed using only the training part of the dataset.

In the supplementary materials (Section A) we analyze how the considered methods perform for other network widths.

5.2. Three Layer Network

In the column FC3 of Table 2 we report results for three-layer networks with 6144 neurons in the first hidden layer and 2000 neurons in the second hidden layer.

In this table, **Linear** and **Arc** are the simplest baseline methods that do not involve any layer-wise stages. We simply simultaneously connect the respective rows of the weight matrices (W_1, W_2, W_3, in the case of three-layer networks) independently of each other, by considering these rows as sampled from some distributions (p_1, p_2, p_3, respectively), and using either linear segments as in Eq.(5) or arcs as in Eq.(6) for connection.

In the supplementary materials (Section A) we show how performance of the methods changes as we vary network depth.

Note that we make weight adjustment using the train dataset and report performance on the train and test datasets. However, it is clear from Table 2 that we do not observe the identical output along the path even on the train dataset. Let us point out why this can happen. Denote $\phi(W_2(0.5)\mathbf{X}_1^B)$ by $\mathbf{X}_2^{AB}$. Then the output of the network at the worst point of the connecting path on the dataset $\mathbf{X}$ is $\widehat{\mathbf{Y}}^A \approx W_3(0.5)\phi(W_2(0.5)\phi(W_1^B\mathbf{X})) = W_3^A\mathbf{X}_2^A[\mathbf{X}_2^{AB}]^+\mathbf{X}_2^{AB}$. The approximate equality becomes exact if $[\mathbf{X}_2^{AB}]^+\mathbf{X}_2^{AB} = I$. This happens when the network is overparameterized and all data points in $\mathbf{X}_2^{AB}$ are independent of each other, see the left side of Fig. 5. On the other hand, if we have more data points than neurons in the hidden layer, then we only have the approximate inequality $[\mathbf{X}_2^{AB}]^+\mathbf{X}_2^{AB} \approx I$. Moreover, the more points we have compared to the number of neurons in the hidden layer, the more approximate this equality becomes. The underparameterized case is shown on the right side of Fig. 5. The drop of performance is clearly more drastic for the weight adjustment in the second hidden layer, which has only 2000 hidden units (the interval $[1, 2]$ in the plots), compared to the weight adjustment in the first hidden layer, which has 6144 hidden units (the interval $[0, 1]$).

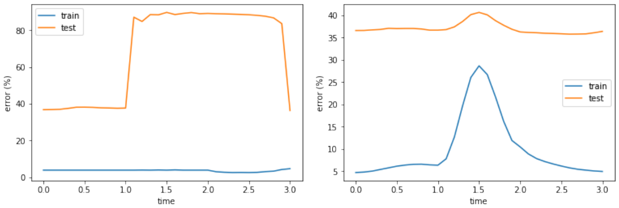

Figure 5: Train and test error rates on a **Arc + Weight Adjustment** path connecting two local minima of a three-layer network. The intervals $[0, 1]$, $[1, 2]$, $[2, 3]$ correspond to sub-paths $\Theta^A \to \Theta_2^{AB}, \Theta_2^{AB} \to \Theta_3^{AB}, \Theta_3^{AB} \to \Theta^B$, respectively. **Left:** The train dataset is reduced to have a small size equal to the minimum hidden layer width of the network, 2000. The reduced dataset is used both to perform Weight Adjustment and measure the accuracy of the method. **Right:** Results with the full train dataset.

5.3. Convolution Networks

In columns Conv2FC1 and VGG16 of Table 2 we report results for the respective convolutional networks. Conv2FC1 is a simple network having 32 and 64 channels in the convolution layers (with kernel size 5), and 3136 neurons in the fully connected layer. VGG16 is used without batch normalization. The results show that methods without the WA procedure fail to construct low-loss paths for VGG16. Otherwise, the trends in the performance of different methods are similar to those observed for dense multi-layer networks. See Section C in Supplementary materials for more results

Table 1: Train and test accuracy (%) of different methods for networks with a single hidden layer. End Point values show accuracy at the ends of the path. WA is short for Weight Adjustment. We show mean and one standard deviation of the worst point along the path.

	MNIST		CIFAR10	
Methods	train	test	train	test
Linear	96.54 ± 0.40	95.87 ± 0.40	32.09 ± 1.33	39.34 ± 1.52
Arc	97.89 ± 0.11	97.03 ± 0.14	49.97 ± 0.86	41.34 ± 1.39
IAF flow	96.34 ± 0.54	95.80 ± 0.45	–	–
RealNVP bijection	98.50 ± 0.09	97.53 ± 0.11	63.46 ± 0.27	53.94 ± 0.95
Linear + WA	98.76 ± 0.01	97.86 ± 0.05	52.63 ± 0.59	57.66 ± 0.26
Arc + WA	98.75 ± 0.01	97.86 ± 0.05	58.77 ± 0.32	57.88 ± 0.24
OT	98.78 ± 0.01	97.87 ± 0.04	66.19 ± 0.23	56.49 ± 0.46
OT + WA	98.92 ± 0.01	97.91 ± 0.03	67.02 ± 0.12	58.96 ± 0.21
Garipov (3)	99.10 ± 0.01	97.98 ± 0.02	68.51 ± 0.08	58.74 ± 0.23
Garipov (5)	99.03 ± 0.01	97.93 ± 0.02	67.20 ± 0.12	57.88 ± 0.32
End Points	99.14 ± 0.01	98.01 ± 0.03	70.60 ± 0.12	59.12 ± 0.26

and discussion.

6. Ensembling with Weight Adjustment

In (Izmailov et al., 2018) the authors perform averaging of several neural networks lying near each other in the weight space. Such close networks are taken from those obtained by SGD iterations. (Izmailov et al., 2018) show that this leads to a better generalization and that such averaging approximates ensembling of close models in the first order of approximation. The averaging has computational benefit compared to the usual ensemble of n models that requires n times more computation.

In this section we propose another method to perform ensembling via weight averaging, applicable to any finite set of models on the weight manifold (typically, models optimized with different randomly chosen initial weights). The method is based on the Weight Adjustment procedure described in Section 3.5.

The idea is to use the first network as a common backbone to extract features on some intermediate layer (see Fig. 6). Given a particular data set and the k'th model, let us denote the output of this intermediate layer by $\mathbf{F}_k$ and the weights of the next layer by W_k. Also, we will denote by $head_k$ the computation performed in the k'th model after the multiplication by W_k. Performing weight adjustment on the next layer, $W_k^1 = W_k \mathbf{F}_k \mathbf{F}_1^+$, we make adjusted weights to operate on the same "basis" $\mathbf{F}_1$, for every model k. Note that a net with these adjusted weights approximates the output of the k'th network. So, the average of the adjusted prediction $\frac{1}{n}\sum_{k=1}^{n} head_k(W_k^1 \mathbf{F}_1)$ approximates the true ensembling of the models, where n is the number of models in the ensemble.

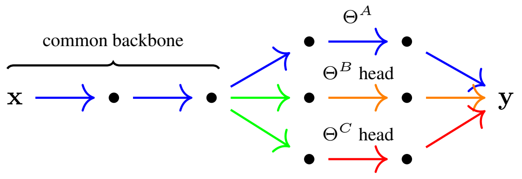

Figure 6: A WA-ensemble of three models. Models B and C are adjusted to have the same backbone as model A. A longer common backbone reduces the amount of computation and required storage.

Note that if we adjust the last layer, there is no $head_k$ subnetworks to compute and we can just average the adjusted weights in the last layer. Moreover, if the loss is convex with respect to the model output, the loss of thus averaged models does not exceed the largest of the single model losses.

In Figure 7 we compare this WA-ensemble method against the usual ensemble of independently trained networks. We see that a longer common backbone reduces the amount of computation and required storage at the cost of accuracy: the ensemble of independently trained models performs the best, followed by the WA(14) ensemble with two common layers, etc. We refer the reader to Section B in supplementary material for more results.

7. Discussion

We have described and compared a panel of generally applicable methods to connect a pair of weight vectors with a low-loss path. Our methods are inspired by the distributional picture of weights in the networks and vary in complexity and accuracy. On the whole, our experiments show that on the realistic datasets such as MNIST and CIFAR10, our

Table 2: The same as Table 1 but for complex networks on CIFAR10. B-fly is short for Butterfly.

	FC3		Conv2FC1		VGG16	
Methods	train	test	train	test	train	test
Linear	31.10 ± 0.84	27.19 ± 1.12	25.86 ± 4.62	25.41 ± 4.54	$10. \pm 0.$	$10. \pm 0.$
Arc	46.39 ± 1.03	40.17 ± 0.84	31.03 ± 2.01	30.44 ± 2.09	$10. \pm 0.$	$10. \pm 0.$
Linear + B-fly	47.81 ± 0.76	38.38 ± 0.84	44.08 ± 3.59	42.46 ± 3.43	8.41 ± 3.79	8.57 ± 3.49
Arc + B-fly	60.60 ± 0.79	49.63 ± 0.86	56.67 ± 3.93	54.56 ± 3.73	3.67 ± 4.56	4.54 ± 4.30
Linear + WA	60.93 ± 0.25	51.87 ± 0.24	71.09 ± 0.38	67.07 ± 0.49	94.16 ± 0.38	87.55 ± 0.41
Arc + WA	71.10 ± 0.23	58.86 ± 0.29	77.36 ± 0.99	73.77 ± 0.88	95.35 ± 0.239	88.56 ± 0.28
OT + B-fly	81.95 ± 0.29	59.11 ± 0.46	76.94 ± 1.41	73.66 ± 1.44	75.42 ± 18.83	68.56 ± 17.80
OT + WA	87.53 ± 0.18	61.67 ± 0.49	82.37 ± 0.44	78.11 ± 0.61	96.61 ± 0.18	89.24 ± 0.14
Garipov (3)	94.56 ± 0.08	61.38 ± 0.36	85.10 ± 0.25	80.95 ± 0.16	99.69 ± 0.03	91.25 ± 0.14
End Points	95.13 ± 0.08	63.25 ± 0.36	87.18 ± 0.14	82.61 ± 0.18	$99.99 \pm 0.$	91.67 ± 0.10

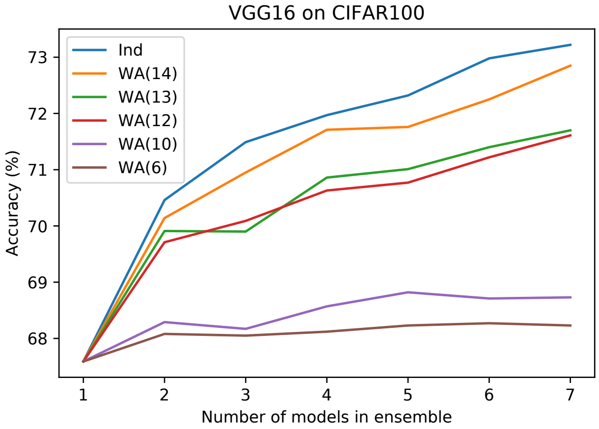

Figure 7: Test accuracy (%) of different WA-ensembles with respect to the number of models in the ensemble. Ind corresponds to the ensemble of independent networks. WA(n) is WA-Ensemble with Weight Adjustment procedure performed on the n'th layer counting from the last network layer.

connection methods are reasonably efficient, with efficiency naturally correlated with the complexity of the method.

The simplest nontrivial method – Arc Connection – is practically as simple and explicit as the baseline linear connection, but nevertheless provides a consistent improvement over the latter. The learnable methods (IAF flow, ReLNVP bijection) further improve performance, thanks to taking into account the actual distribution of neurons.

Optimal Transportantion and Weight Adjustment perform even better, approximately matching and in some cases even slightly improving the direct numerical optimization results of (Garipov et al., 2018). The key difference between them and the learnable methods is that the latter transform a neuron to the given state disregarding the states of the other neurons. In contrast, transformation of a single neuron under OT and WA takes into account the states of all neurons, which obviously creates an opportunity for a lower loss connection, at the cost of a higher computational complexity.

The observed efficiency of the Optimal Transportation confirms the distribution-based explanation of the low-loss structure of the loss surface. Note, however, that the path constructed by OT is a rather complex piecewise linear curve, with the number of pieces scaling linearly with the network size. Also, this construction depends on the initial neuron matching that requires a separate optimization for each pair of endpoints. In contrast, global learnable methods (IAF flow, RealNVP bijection), while not achieving the accuracy of OT, provide relatively simple paths that depend on the endpoints only through the explicit arc formula.

Summarizing, our results provide a relatively clear picture of connectedness of local minima in a large network. We see that natural “macroscopic” ideas lead to relatively simple low-lying paths, which can be further improved by taking into account more “microscopic” details. The resulting connection performance agrees with previously known experimental results. Moreover, we have shown that low-loss connection paths give rise to a new kind of ensembling capable of improving the accuracy of the trained model with only a moderate increase of its complexity. It would be interesting to further explore the structure of connecting paths, with the view of a further computational simplification and better guarantees of performance improvement.

8. Acknowledgment

IA thanks Denis Korzhenkov for helpful discussions and comments. DY acknowledges support of Huawei in the framework of the joint Skoltech–Huawei project “CNN expressiveness”.

References

Marco Baity-Jesi, Levent Sagun, Mario Geiger, Stefano Spigler, G Ben Arous, Chiara Cammarota, Yann LeCun, Matthieu Wyart, and Giulio Biroli. Comparing dynamics: Deep neural networks versus glassy systems. *arXiv preprint arXiv:1803.06969*, 2018.

Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in neural information processing systems*, pp. 3036–3046, 2018.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pp. 192–204, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A Hamprecht. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*, 2018.

R'emi Flamary and Nicolas Courty. Pot python optimal transport library, 2017. URL `https://github.com/rflamary/POT`.

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.

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.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pp. 4743–4751, 2016.

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. *arXiv preprint arXiv:1906.06247*, 2019.

Song Mei, Andrea Montanari, and Phan-Minh 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.

Quynh Nguyen. On connected sublevel sets in deep learning. *arXiv preprint arXiv:1901.07417*, 2019.

Grant M Rotskoff and Eric Vanden-Eijnden. Neural networks as interacting particle systems: Asymptotic convexity of the loss landscape and universal scaling of the approximation error. *arXiv preprint arXiv:1805.00915*, 2018.

Itay Safran and Ohad Shamir. Spurious local minima are common in two-layer relu neural networks. *arXiv preprint arXiv:1712.08968*, 2017.

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

Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of neural networks: A central limit theorem. *arXiv preprint arXiv:1808.09372*, 2018.