mathematics

MDPI

Article

Neural Teleportation

Marco Armenta [1,2,*,†], Thierry Judge [1,†], Nathan Painchaud [1], Youssef Skandarani [3], Carl Lemaire [1], Gabriel Gibeau Sanchez [1], Philippe Spino [1] and Pierre-Marc Jodoin [1]

[1] Department of Computer Science, Université de Sherbrooke, Sherbrooke, QC J1K 2R1, Canada
[2] Department of Mathematics, Université de Sherbrooke, Sherbrooke, QC J1K 2R1, Canada
[3] Department of Computer Science, Université de Bourgogne Franche-Comte, 21000 Dijon, France
* Correspondence: marco.armenta@usherbrooke.ca
† These authors contributed equally to this work.

Abstract: In this paper, we explore a process called neural teleportation, a mathematical consequence of applying quiver representation theory to neural networks. Neural teleportation *teleports* a network to a new position in the weight space and preserves its function. This phenomenon comes directly from the definitions of representation theory applied to neural networks and it turns out to be a very simple operation that has remarkable properties. We shed light on the surprising and counter-intuitive consequences neural teleportation has on the loss landscape. In particular, we show that teleportation can be used to explore loss level curves, that it changes the local loss landscape, sharpens global minima and boosts back-propagated gradients at any moment during the learning process.

Keywords: quiver representations; neural networks; teleportation

MSC: 16G20; 68T07

check for updates

Citation: Armenta, M.; Judge, T.; Painchaud, N.; Skandarani, Y.; Lemaire, C.; Gibeau Sanchez, G.; Spino, P.; Jodoin, P.-M. Neural Teleportation. *Mathematics* **2023**, *11*, 480. https://doi.org/10.3390/math11020480

Academic Editor: Xiang Li

Received: 8 December 2022
Revised: 4 January 2023
Accepted: 6 January 2023
Published: 16 January 2023

1. Introduction

Despite years of research, our theoretical understanding of neural networks, their loss landscape and their behavior during training and testing is still limited. A recent novel theoretical analysis of neural networks, using quiver representation theory given by [1], has been introduced, where the algebraic and combinatorial nature of neural networks is exposed. Among other things, the authors present a by-product of representing neural networks through the lens of quiver representation theory, i.e., the notion of *neural teleportation*.

This is the mathematical foundation that explains why practitioners of deep learning have observed the property of scale invariance for the very particular case of neural networks with positive scale invariant activation functions, see, for example, the work of [2]. Nevertheless, this type of invariance has not been studied or observed in its full generality, for example, across batch norm layers, residual connections, concatenation layers or activation functions other than ReLU. Even more, it has been claimed, for example, by [3], that there is no scale invariance across residual connections. This is because it is not until a neural network is drawn as a quiver that the invariance of the network function under isomorphisms across any architecture becomes obvious (see Appendix A for an illustration of this). Positive scale invariance also restricts the type of scaling factor, reducing them to positive real numbers and positive scale invariant activation functions, while neural teleportation allows to teleport any neural architecture with any non-zero scaling factor.

In this work, we intend to exhibit empirical evidence that using quiver representation theoretic concepts produces measurable changes on the behaviours of deep neural networks. Due to its mathematical nature, neural teleportation is an intrinsic property of every neural network, and so it is independent of the architecture, the activation functions and even the task at hand or the data. Here, we perform extensive experiments on classification tasks with feedforward neural networks with different activation functions and scaling factor sampling.

As will be explained later, neural teleportation is the mathematical consequence of applying the concept of *isomorphisms of quiver representations* to neural networks. This process has the unique property of changing the weights and the activation functions of a network while, at the same time, preserving its function, i.e., a teleported network makes the same predictions for the same input values as the original, non-teleported network.

Isomorphisms of quiver representations have already been used on neural networks, often unbeknownst to the authors, through the concept of *positive scale invariance* (also called *positive homogeneity*) of some activation functions, see [2–5]. However, representation theory lays down the mathematical foundations of this phenomenon and explains why this has been observed only on networks with positive scale invariant activation functions. Following the quiver representation theoretic approach to neural networks, it becomes clear that any neural network can be teleported with an isomorphism, as opposed to what is remarked in the literature. Namely, it has been claimed that there is no scale invariance across residual connections [3] or on the parameters beta and gamma for batch norm layers [6]. We explain (see Appendix A for illustrations on residual connections and batchnorm layers) how the quiver approach is essential to apply teleportation to any architecture.

The concept of positive scale invariance derives from the fact that one can choose a *positive* number c for each hidden neuron of a *ReLU* network, multiply every incoming weight to that neuron by c and divide every outgoing weight by c and still have the same network function. Note that models such as maxout networks [7], leaky rectifiers [8] and deep linear networks [9] are also positive scale-invariant. This concept in previous works is always restricted to positive scale-invariant activation functions and to positive scaling factors. However, the generality in which quiver representation theory describes neural networks allows to teleport any architecture, with any activation function and any non-zero scaling factor.

In this paper, we show that neural teleportation is more than a trick, as it has concrete consequences on the loss landscape and the network optimization. We account for various theoretical and empirical consequences that teleportation has on neural networks.

Our findings can be summarized as follows:

1. Neural teleportation can be used to explore loss level curves;
2. *Micro-teleportation vectors* have the property of being perpendicular to back-propagated gradients computed with any kind and amount of labeled data, even random data with random labels;
3. Neural teleportation changes the flatness of the loss landscape;
4. The back-propagated gradients of a teleported network scale with respect to the scaling factor;
5. Randomly teleporting a network before training speeds up gradient descent (with and without momentum). Actually, we also found that *one* teleportation can accelerate training even when used at the middle of the training.

2. Neural Teleportation

In this section, we explain what neural teleportation is and the implications it has on the loss landscape and the back-propagated gradients. For more details on the theoretical interpretation of neural networks according to quiver representation theory, please refer to the work of [1].

2.1. Isomorphisms and Change of Basis (CoB)

Neural networks are often pictured as oriented graphs. [1] show that neural network graphs are a specific kind of quiver with a loop at every hidden node. They call these graphs *network quivers* (c.f. Figure 1a). They also mention that neural networks, as they are generally defined, are network quivers with a weight assigned to every edge and an activation function at every loop (c.f. Figure 1b).

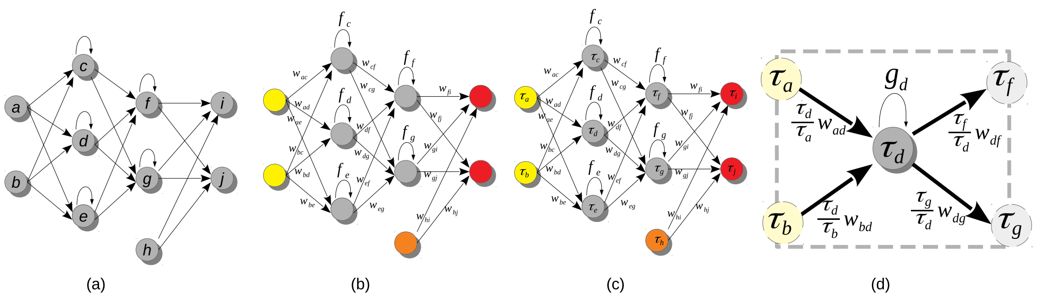

Figure 1. (**a**) Network quiver Q as introduced by [1]. (**b**) Neural network based on Q with weights W and activation functions f. Input, hidden, bias, and output neurons are in yellow, gray, orange and red, respectively. (**c**) Same neural network but with a change of basis (CoB) τ_ϵ at each neuron ϵ. (**d**) Neural teleportation of the weights attached to neuron d. The resulting activation function is $g_d(x) = \tau_d f_d(x/\tau_d)$.

According to representation theory, two quiver representations are equivalent if an *isomorphism* exists between the two. Mathematically, neural networks are quiver representations with activation functions, and this allows to apply isomorphisms of quiver representations directly to every neural network independently of the architecture or the task at hand.

Isomorphisms are given by sets of non-zero real numbers subject to some conditions. One such set of non-zero numbers is called a *change of basis* (CoB), see, for example, [10], where each node of the quiver is assigned a non-zero number. In order to apply an isomorphism to a neural network, each neuron ϵ must be assigned a CoB represented by $\tau_\epsilon \in \mathbb{R}^{\neq 0}$ in Figure 1c.

However, isomorphisms of quiver representations are very general and they may break implementations of neural network layers. For example, applying an isomorphism to a convolutional layer may not result in a convolutional layer. However, in any case, we will always obtain a network with the exact same network function. It was proved by [1] that these isomorphisms can be restricted to preserve the implementation of neural network layers for any architecture. Neural teleportation is the process of applying one of such particular isomorphisms of quiver representations to neural networks.

The following is not an exhaustive list of all the cases for teleportation. It seems case-by-case because we break it down for the most common layers. It is precisely here where the quiver approach to neural networks is required to define a general teleportation for any neural network [1]. The conditions over the CoB (to be defined latter) to produce a teleportation of a neural network are the following.

1. The CoB of every input, output and bias neuron must be equal to 1 (i.e., $\tau_a, \tau_b, \tau_h, \tau_i, \tau_j = 1$ in Figure 1c).
2. Neurons k, l connected by a residual connection should have the same CoB: $\tau_k = \tau_l$. See Appendix A for details.
3. For convolutional layers, the neurons of a given feature map should share the same CoB.
4. For batch norm layers, a CoB must be assigned to parameters β and γ, but not to the mean and variance. See Appendix A for details.
5. The CoBs of neurons connected by a dense connection are obtained by concatenating those in its input layers.

Condition 1 is the *isomorphism condition*. Any two networks related by a CoB satisfying this condition are said to be *isomorphic*. Applying the notion of isomorphism of quiver representation to neural networks leads to the following theorem:

Theorem 1 ([1]). *Isomorphic neural networks have the same network function.*

Said otherwise, despite having different weights and different activation functions, isomorphic neural networks return the *same predictions for the same inputs*. It also means that they have *exactly* the same loss values (c.f. [1] for the proof).

Conditions 2 to 5 have to do with the architecture of the network. They ensure that the produced isomorphic networks share the same architecture (again c.f. [1] for more details). These conditions ensure that the teleportation of a residual connection remains a residual connection (condition 2), the teleportation of a conv layer remains a conv layer (condition 3), the teleportation of a batch norm remains a batch norm (condition 4) and the teleportation of a dense layer remains a dense layer (condition 5). Please note that [3] introduced a concept similar to condition 3.

2.2. Teleporting a Neural Network

Neural teleportation is a process by which the weights W and the activation functions f of a neural network are converted to a new set of weights V, and a new set of activation functions g. From a practical standpoint, this process is illustrated in Figure 1d.

Considering $w_{ab} \in \mathbb{R}$, the weight of the connection from neuron a to neuron b, and $\tau_a \in \mathbb{R}^{\neq 0}$ (resp. $\tau_b \in \mathbb{R}^{\neq 0}$) the CoB of neuron a (resp. b). The teleportation of that weight is simply:

$$v_{ab} = \frac{\tau_b}{\tau_a} w_{ab}. \tag{1}$$

To teleport an entire network, this operation is carried out for every weight of the network. Note that positive homogeneity, see [2–5], implies a similar operation but with the restriction of positive scaling factors. In the case of batch norm layers, the parameter γ is treated like a weight between two hidden neurons and β as a weight starting from a bias neuron [1]. As such, γ and β are teleported like any other weight using Equation (1).

Neural teleportation also applies to activation functions. If f_d is the activation function of neuron d in Figure 1d, then the teleported activation is

$$g_d(x) = \tau_d \cdot f_d\left(\frac{x}{\tau_d}\right). \tag{2}$$

This is a critical operation to make sure the pre- and post-teleported networks have the same function. We can see that if $\tau_d > 0$ and f_d is positive scale invariant (such as ReLU) then $g_d(x) = \tau_d f_d(x/\tau_d) = \tau_d/\tau_d f_d(x) = f_d(x)$, and so positive scale invariance is a consequence of neural teleportation.

3. Previous Work

It was shown that positive scale invariance affects training by inducing symmetries on the weight space. As such, many methods have tried to take advantage of it, for example, [2,4,11,12]. Our notion of teleportation gives a different perspective as (i) it allows any non-zero real-value CoB to be used as scaling parameters, (ii) it acts on any kind of activation functions, and (iii) our approach does not impose any constraints on the structure of the network nor the data it processes. Moreover, neural teleportation does not require new update rules as it only has to be applied *once* during training to produce an impact.

Refs. [2,3] accounted for the fact that ReLU networks can be represented by the values of “basis paths” connecting input neurons to output neurons. Ref. [3] made clear that these paths are interdependent and proposed an optimization based on it. They designed a space that manages these dependencies, proposing an update rule for the values of the paths that are later converted back into network weights. Unfortunately, their method only works for neural nets having the same number of neurons in each hidden layer. Furthermore, they guarantee no invariance across residual connections. This is unlike neural teleportation, which works for any network architecture, including residual networks.

Scale-invariance in networks with batch normalization has been observed by [6,13], but not in the sense of quiver representations.

Positive scale invariance of ReLU networks has also been used to prove that common flatness measures can be manipulated by the re-scaling factors [5]. Here, we experimentally show that the loss landscape changes when teleporting a network with positive or negative CoB, regardless of its architecture and activation functions. Note that the proofs of [5] are for two layer neural networks while we teleport deeper and much more sophisticated architectures, and provide empirical evidence of how teleportation sharpens the local loss landscape.

4. Neural Teleportation and the Loss Landscape

Despite its apparent simplicity, neural teleportation has surprising consequences on the loss landscape. In this section, we underline such consequences and lay empirical evidence for it.

4.1. Inter- and Intra-Loss Landscape Teleportation

As mentioned before, teleporting a positive scale invariant activation function f_d with a positive τ_d results in $g_d = f_d$. This means that the teleported network ends up inside the same loss landscape but with weights V at a different location than W (c.f., Figure 2a). In other cases (for τ_d negative or non-positive scale invariant activation functions f_d), neural teleportation changes the activation function and, thus, the overall loss landscape. For example, with $\tau_d < 0$ and f_d a ReLU function, the teleportation of f_d becomes: $g_d(x) = \tau_d \mathbf{max}(0, x/\tau_d) = \mathbf{min}(0, \tau_d x/\tau_d) = \mathbf{min}(0, x)$.

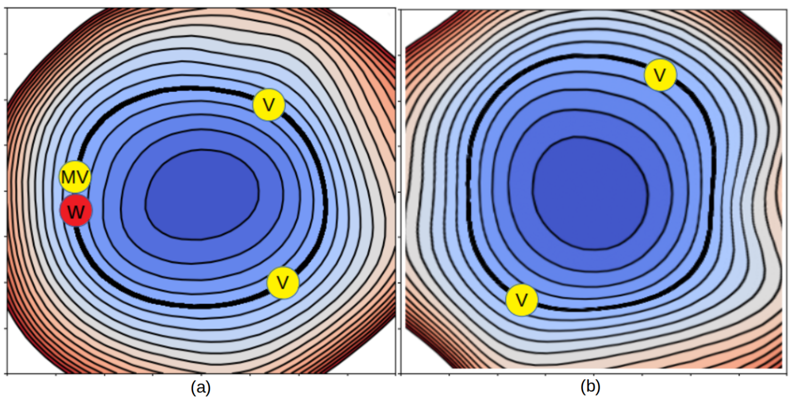

Figure 2. (**a**) 2D slice of a loss landscape with a W dot as the location of a given network. The V dots are two teleported versions of W inside the same landscape while the MV dot stands for a micro-teleportation of W. (**b**) The dots are teleported versions of W in a different loss landscape. Since neural teleportation preserves the network function, the six networks have the same loss value. In the case of ReLU, for example, the teleported networks with intra-landscape teleportation lie in the same landscape.

Thus, the way CoB values are chosen has a concrete effect on how a network is teleported. A trivial case is when $\tau_\epsilon = 1$ for every hidden neuron ϵ, which leads to no transformation at all: $V = W$ and $g = f$. For our method, we choose the CoB values by randomly sampling either of two distributions. The first one is a uniform distribution centered on 1: $\tau_\epsilon \in [1-\sigma, 1+\sigma]$ with $0 < \sigma < 1$. We call σ the *CoB-range*; the larger σ is, the more different the teleported network weights V will be from W. In addition, when this sampling operation is combined with positive scale invariant activation function (like ReLU), the teleported activation functions stay unchanged ($g = f$) and, thus, the new weights V are guaranteed to stay within the same landscape as the original set of weights W (as in Figure 2a). We, thus, call this operation an *intra-landscape* neural teleportation.

The other distribution is a mixture of two equal-sized uniform distributions: one centered at $+1$ and the other at -1: $\tau_\epsilon \in [1-\sigma, 1+\sigma] \cup [-1-\sigma, -1+\sigma]$. With high probability, a network teleported with this sampling will end up in a new loss landscape, as illustrated in Figure 2b. We, thus, call this operation an *inter-landscape* neural teleportation.

4.2. Loss Level Curves

Since τ_ϵ can be assigned any non-zero real values, a network can be teleported an infinite amount of times to an infinite amount of locations within the same landscape or across different landscapes. This is illustrated in Figure 2, where W is teleported to 5.

We validated this assertion by teleporting an MLP 100 times with an inter-landscape sampling and a CoB-range of $\sigma = 0.9$. While the mean average difference between the original weights W and the teleported weights V is of 0.08 (a large value considering that the magnitude of W is of 0.18), the average loss difference was in the order of 10^{-10}, i.e., no more than a floating point error.

4.3. Micro Teleportation

One can easily see that the gradient of a function is always perpendicular to its local level curve (c.f. [14] chap. 2). However, back-propagation computes noisy gradients that depend on the amount of data used, that is, the batch-size. Thus, the noisy back-propagated gradients do not a priori have to be strictly perpendicular to the local loss level curves.

This concern can be (partly) answered via the notion of *micro-teleportation*, which derives from the previous subsection. Let us consider the intra-landscape teleportation of a network with positive scale invariant activation functions (such as ReLU) with a CoB-range σ close to zero. In that case, $\tau_\epsilon \approx 1$ for every neuron ϵ and the teleported weights V (computed following Equation (1)) end up being very close to W. We call this a *micro-teleportation* and illustrate it in Figure 1a (the MV dot illustrates the micro teleportation of W).

Because V and W are isomorphic, they both lie on the same loss level curve. Thus, if σ is small enough, the vector $\overrightarrow{WV}$ between W and V is locally co-linear to the local loss level curve. We call $\overrightarrow{WV}$ a *micro-teleportation vector*.

A rather counter-intuitive empirical property of micro-teleportations is that $\overrightarrow{WV}$ is perpendicular to *any back-propagated gradient* across different batch sizes, because we know that the back-propagated gradient depends on the batch size but the teleportation does not. Moreover, if one uses L2 regularization the loss is not the same anymore before and after teleportation, so it does not make sense that micro teleportation is perpendicular to the gradient as we were using L2 regularization in our experiments. This surprising observation leads to the following conjecture.

Conjecture 1: *For any neural network, any dataset and any loss function, there exists a sufficiently small CoB-range σ so that every micro-teleportation produced with it is perpendicular to the back-propagated gradient with any batch size.*

We empirically assessed this conjecture by computing the angle between micro-teleportation vectors and back-propagated gradients of four models (MLP, VGG, ResNet and DenseNet) on three datasets (CIFAR-10, CIFAR-100 and random data) 2 different batch sizes (8 and 64) with a CoB-range $\sigma = 0.001$. A cross-entropy loss was used for all models. Figure 3 shows angular histograms for VGG on CIFAR-10 and an MLP (with one hidden layer of 128 neurons) on random data (results for other configurations are in the Appendix A). The first row shows the angular histogram between micro-teleportation vectors and gradients. We used batch sizes of 8 for MLP and 64 for VGG. As can be seen, both histograms exhibit a clear Dirac delta on the 90° angle. As a mean of comparison, we report angular histograms between micro-teleportation vectors and random vectors, between back-propagated gradients and random vectors, and between random vectors. While random vectors in a high-dimensional space are known to be quasi-orthogonal (c.f work of [15]), by no means are they exactly orthogonal, as shown by the green histograms. The Dirac delta of the first row versus the wider green distributions is a clear illustration of our conjecture.

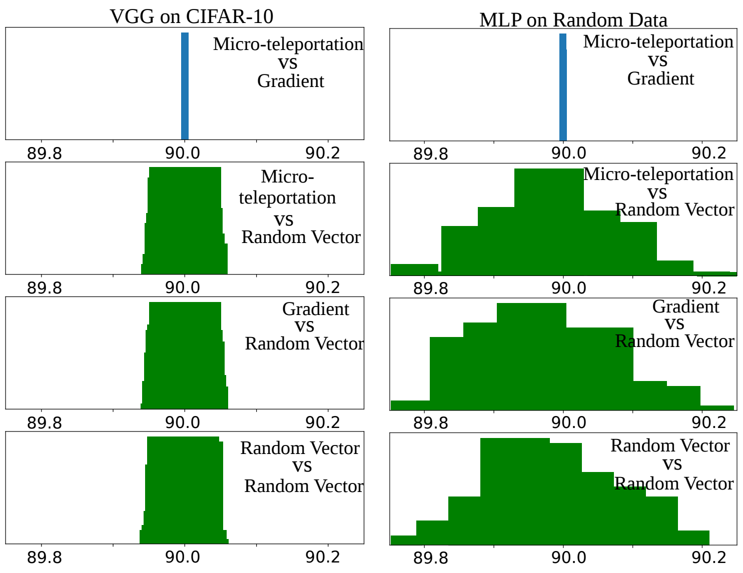

Figure 3. (Top) Angle histograms between micro-teleportation vectors and back-prop. gradients for VGGnet on CIFAR-10 data and MLP on random data. The other rows are angle histograms between micro-teleportation vectors and random vectors, between gradient and random vectors and between random vectors.

These empirical findings suggest that although back-propagation computes noisy gradients, their orientation in the weight space does not entirely depend on the data nor the loss function. In other words, back-propagation computes gradients that point to directions perpendicular to micro-teleportation vectors, but micro-teleportation vectors do not depend on the loss nor the data. Finally, we note that this conjecture is false if one adds a regularization term that does not depend directly on the output of the network, for example, $l2$ regularization adds a term that depends on the weights but not on the output of the network.

4.4. Teleportation and Landscape Flatness

It has been shown by [5] for positive scale invariant activation functions, that one can find a CoB (called *reparametrization* in their paper) so that the most commonly used measures for flatness can be manipulated to obtain a sharper minimum with the exact same network function. We empirically show that neural teleportation systematically sharpens the found minima, independently of the architecture or the activation functions.

A commonly used method to compare the flatness of two trained networks is to plot the 1D loss/accuracy curves on the interpolation between the two sets of weights. It is also well known that small batch sizes produce flatter minima than bigger batch sizes. We trained on CIFAR-10 a 5 hidden-layer MLP two times, first with a batch size of 8 (network A) than with a batch size of 1024 (network B). Then, as performed by [16], we plotted the 1D loss/accuracy curves on the interpolation between the two weight vectors of the networks (c.f. Figure 4a). We then performed the same interpolation but between the teleportation of A and B with CoB-ranges σ of 0.6, 0.9, and 0.99. As can be seen from Figure 4, the landscape becomes sharper as the CoB-range increases. Said otherwise, a larger teleportation leads to a locally-sharper landscape. More experiments with other models can be found in Appendix A.

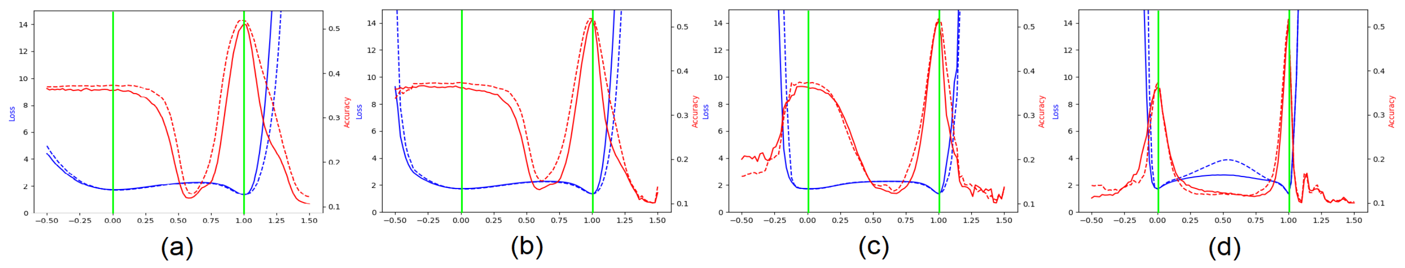

Figure 4. (**a**) Loss/accuracy profiles obtained by linearly interpolating between two optimized MLP A and B. Network A is for $x = 0$ and B for $x = 1$ (green vertical lines). Dotted lines are for training and solid lines for validation. Remaining plots are similar interpolations but between teleported versions of A and B with CoB range σ of (**b**) 0.6, (**c**) 0.9, and (**d**) 0.99.

5. Optimization

In the previous section, we showed that teleportation moves a network along a loss level curve and sharpens the local loss landscape. In this section, we show that teleportation increases the magnitude of the local normalized gradient and accelerates the convergence rate when used during training, even only *once*. Moreover, we empirically show that this impact is controlled by the CoB-range factor σ.

We remark that once a neural network is initialized, it is assigned a specific network function and, in principle, a teleported network of this initialized network (which will have the same network function) does not have to train better than the original network and, nevertheless, the teleported network trains better.

In all of our experiments we used L2 regularization and learning rate decay at epoch 80.

5.1. Back-Propagated Gradients of a Teleportation

It has already been noticed by [2,5], that under positive scaling of ReLU networks, the gradient scales inversely than the weights with respect to the CoB. Here, we show that the back-propagated gradient of a teleported network has the same property, regardless of the architecture of the network, the data, the loss function and the activation functions.

Let (W, f) be a neural network with a set of weights W and activation functions f. We denote $W^{[\ell]}$ the weight tensor of the ℓ-th layer of the network, and analogously for the teleportation $V^{[\ell]}$. The CoB τ at layer ℓ is denoted $\tau^{[\ell]}$, which is a column vector of non-zero real numbers. Let us also consider a data sample $(\mathbf{x}, t)$ with $\mathbf{x}$ the input data and t the target value and dW, dV the gradient of the networks (W, f) and (V, g) with respect to $(\mathbf{x}, t)$ the gradient of the loss with respect to W and V at (x,t). Following Equation (1), we have that

$$V^{[\ell]} = \frac{1}{\tau^{[\ell-1]}} \cdot W^{[\ell]} \cdot \tau^{[\ell]}, \tag{3}$$

where the operation $\frac{1}{\tau^{[\ell-1]}} \cdot W^{[\ell]}$ multiplies the columns of the matrix $W^{[\ell]}$ by the coordinate values of vector $\frac{1}{\tau^{[\ell-1]}}$, while the operation $W^{[\ell]} \cdot \tau^{[\ell]}$ multiplies the rows of matrix $W^{[\ell]}$ by the coordinate values of the vector $\tau^{[\ell]}$.

Theorem 2. *Let (V, g) be a teleportation of the neural network (W, f) with respect to the CoB τ. Then*

$$dV^{[\ell]} = \tau^{[\ell-1]} \cdot dW^{[\ell]} \cdot \frac{1}{\tau^{[\ell]}}$$

for every layer ℓ of the network (proof in the Appendix A).

If we look at the magnitude of the teleported gradient, we have that

$$\|dV\| = \sqrt{\sum_{i,j} \left(dW_{i,j} \frac{\tau_j}{\tau_i} \right)^2}.$$

We can see that the ratio τ_a^2/τ_b^2 appears multiplying the squared non-teleported gradient dW^2. For an *intra*-landscape teleportation, τ_a is randomly sampled from a uniform distribution $[1-\sigma, 1+\sigma]$ for $0 < \sigma < 1$. Since τ_a, τ_b are independent random variables, the mathematical expectation of this squared ratio is:

$$\begin{aligned}
\mathbb{E}\left[\tau_a^2/\tau_b^2\right] &= \mathbb{E}\left[\tau_a^2\right] \cdot \mathbb{E}\left[1/\tau_b^2\right] \\
&= \int_{1-\sigma}^{1+\sigma} \tau_a^2 P(\tau_a) d\tau_a \cdot \int_{1-\sigma}^{1+\sigma} P(\tau_b)/\tau_b^2 d\tau_b \\
&= \frac{\sigma^2+3}{3(1-\sigma^2)}.
\end{aligned}$$

Thus, when $\sigma \to 0$ (i.e., no teleportation as described in Section 4.1), then $\mathbb{E}\left[\tau_a^2/\tau_b^2\right] \to 1$, which means that on average the gradients are multiplied by 1 and thus remain unchanged. However, when $\sigma \to 1$, then $\mathbb{E}\left[\tau_a^2/\tau_b^2\right] \to \infty$, which means that the gradients magnitude is multiplied by an increasingly large factor.

We empirically validate this proposition with four different networks in Figure 5 (plots for the other models and datasets can be found in the Appendix A). There we put the CoB-range σ on the x-axis versus 20 different teleportations for which we computed the absolute difference of normalized gradient magnitude: $|\|dW\|/\|W\| - \|dV\|/\|V\||$. We can see that a larger CoB-range leads to a larger difference between the normalized gradient magnitudes.

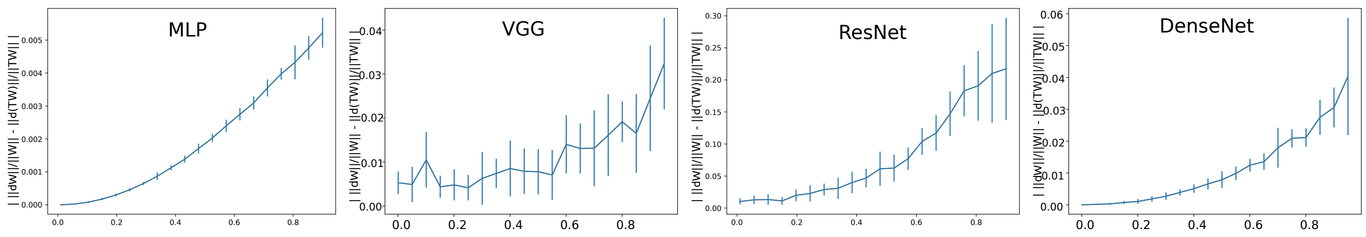

Figure 5. Mean absolute difference (±std-dev) between the back-propagated gradients' magnitudes of teleported networks and their original non-teleported network. Larger CoB generate larger gradients. Results are for CIFAR-10. Plots for these same models on CIFAR-100 can be found in Appendix A.

This shows that randomly teleporting a neural network increases the sharpness of the surrounding landscape and, thus, the magnitude of the local gradient. This holds true for any network architecture and any dataset. This analysis also holds true for *inter*-landscape CoB sampling since the CoB-range σ appears squared always.

5.2. Gradient Descent and Teleportation

From the previous subsection we obtain that the CoB-range controls the difference on the normalized gradients. Here we empirically show that the CoB-range also has an influence on the training when teleportation is applied *once* during training.

In order to assess this, we trained four models: an MLP with five hidden layers with 500 neurons each, a VGGnet, a ResNet18, and a DenseNet, all with ReLU activation. Training was conducted on two datasets (CIFAR-10 and CIFAR-100) with two optimizers (vanilla SGD and SGD with momentum), and three different learning rates (0.01, 0.001 and 0.0001) for a total of 72 configurations. For each configuration, we trained the network with and without neural teleportation right after initialization. Training was performed five times for 100 epochs. The chosen CoB range σ is 0.9 with an *inter*-landscape sampling for all these experiments. The teleported and non-teleported networks were initialized with the same randomized operator following the "Kaiming" method.

This resulted into a total of 720 training curves that we averaged across the learning rates (±std-dev shades), see Figure 6 (plots for the other models and datasets can be found in Appendix A). As can be seen, neural teleportation accelerates gradient descent (with and without momentum) for every model on every dataset.

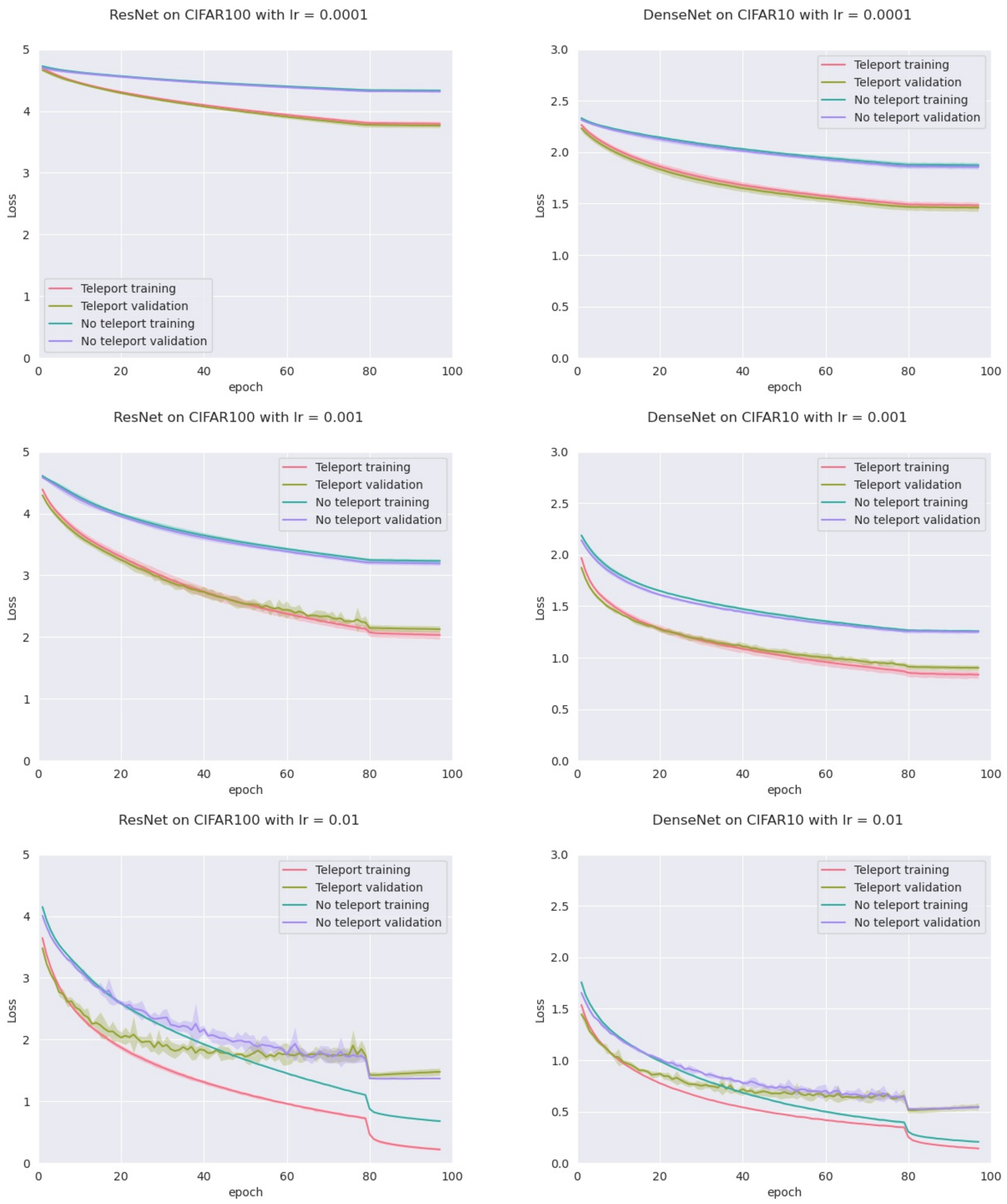

Figure 6. Average loss of ResNet on CIFAR100 (**left**) and DenseNet on CIFAR10 (**right**) over 5 experiments, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

To make sure that these results are not unique to ReLU networks, we trained the MLP on CIFAR-10 and CIFAR-100 with three different activation functions: LeakyReLU, Tanh and ELU, again with and without neural teleportation after the uniform initialization, which is the default PyTorch init mode for fully connected layers. As can be seen from Figure 7 (plots for the other models and datasets can be found in Appendix A), here again, *one* teleportation accelerates gradient descent (with and without momentum) across datasets and models.

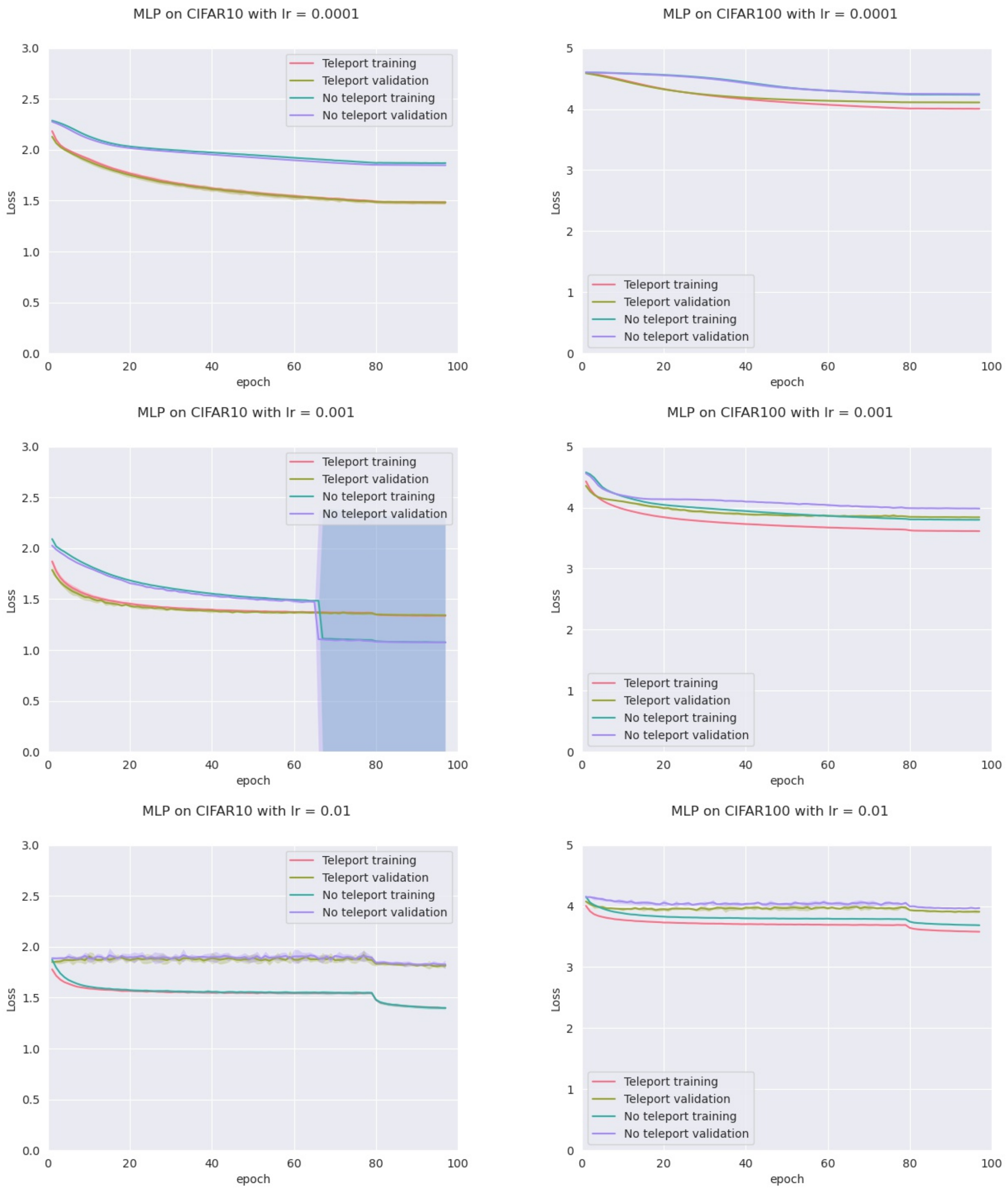

Figure 7. Average loss of MLPs with ELU activation function on CIFAR10 (**left**) and Tanh on CIFAR100 (**right**) over 5 experiments for each, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

In order to measure the impact of the initialization procedure, we ran a similar experiment with a basic Gaussian initialization as well as an Xavier initialization. We obtained the same pattern for the Gaussian case; for the Xavier initialization, teleportation helps the training of the VGG net, while for the others the difference is less prominent, see Figure 8 (plots for the other models and datasets can be found in Appendix A).

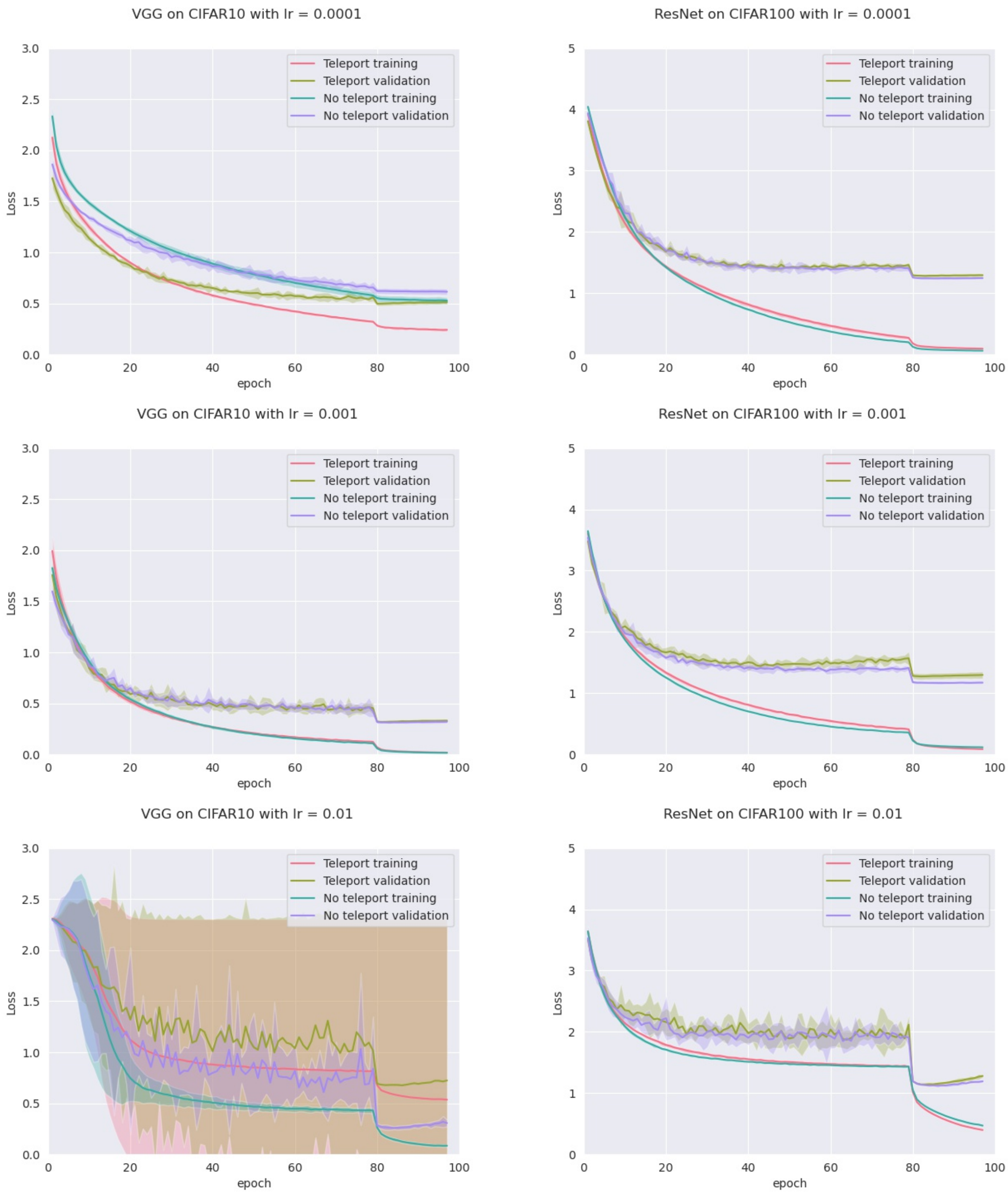

Figure 8. Average loss of VGG with normal initialization on CIFAR10 (**left**) and ResNet with xavier initialization on CIFAR100 (**right**) both traing with SGD with momentum over 5 experiments for each, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

Given a neural network W, we produced another one, V, by first teleporting W to τW and then sampling V from the sphere with center W and radius vector $W - \tau W$. We called this a *pseudo-teleportation*. We then trained the four models with the same regime, comparing pseudo-teleportation to no teleportation. Results in Figure 9 reveal that pseudo-teleportation does not improve gradient descent training (plots for the other models and datasets can be found in Appendix A).

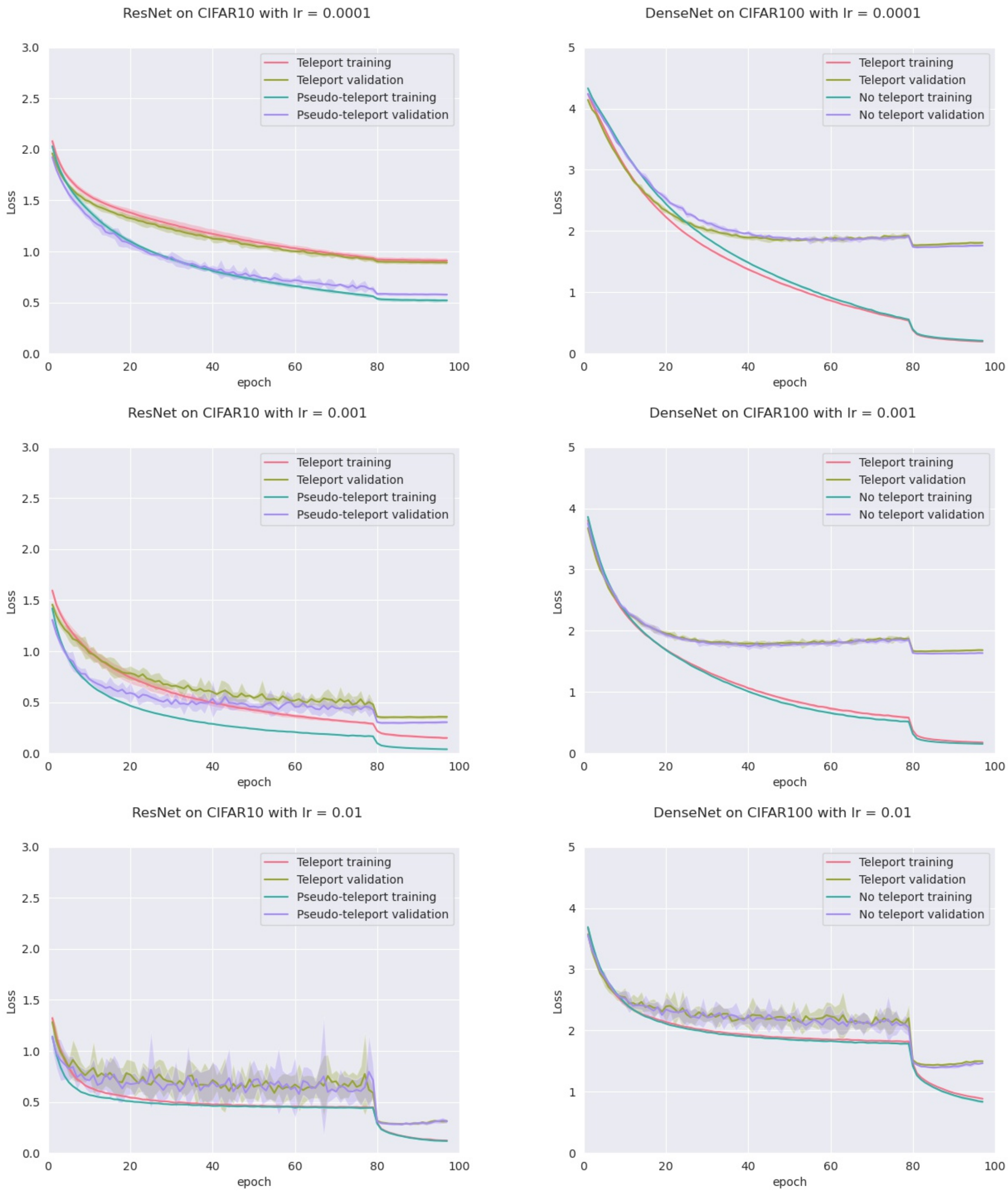

Figure 9. Average loss of ResNet on CIFAR10 (**left**) and DenseNet on CIFAR100 (**right**) both training with SGD over 5 experiments for each, with and without pseudo-teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

We show in Figure 10 the weight histograms of an MLP network initialized with a uniform distribution (PyTorch's default) before (left) and after (right) teleportation.

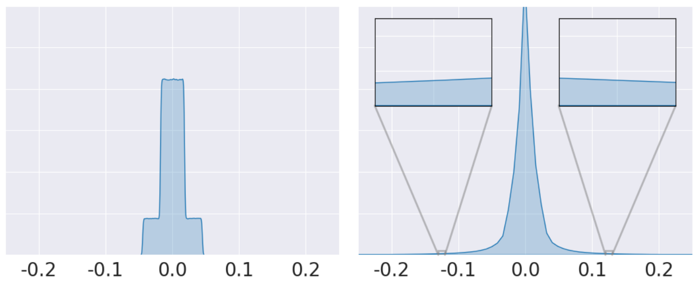

Figure 10. Weight histogram of an MLP before teleportation (**left**) and after teleportation (**right**) with CoB-range of 0.9. Initialization is the standard on PyTorch which is uniform and contains values between -0.05 and 0.05 while the teleported weights are more broadly distributed as shown in the zooms.

For all the previous experiments, we teleported the neural networks at the beginning of the training. Finally, we performed another set of experiments by teleporting the neural network once at epoch 30. We can clearly see the jump in accuracy right after the teleportation in Figure 11. In Figure 12, we compare the training curve of a model with a training where we teleport once at epoch 30 with different CoB-ranges of 0.7, 0.8 and 0.9. We can see how the bigger the Cob-range, the higher the impact on validation accuracy. Experiments shown in Figures 11 and 12 were run with a learning rate of 0.0001 on CIFAR-10.

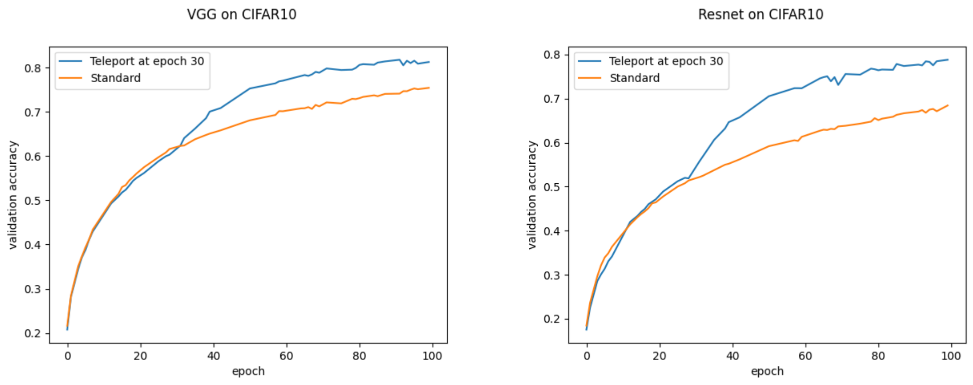

Figure 11. Validation accuracy for VGG (**left**) and ResNet (**right**). We plot two curves for each model, the orange curves are the usual SGD training and the blue curves are SGD with one teleportation at epoch 30.

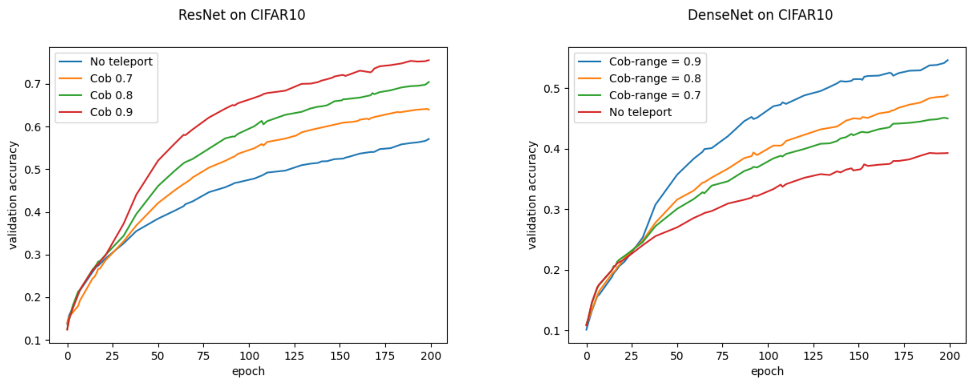

Figure 12. Validation accuracy for ResNet (**left**) and DenseNet (**right**). We plot four curves for each model each one corresponding to applying one teleportation at epoch 30 with different CoB-ranges.

6. Conclusions

In this paper, we provided empirical evidence that neural teleportation can project the weights (and the activation functions) of a network to an infinite amount of places in the weight space while always preserving the network function. We show that: (1) It can be used to explore loss level curves; (2) Micro-teleportation vectors are always perpendicular to back-propagated gradients (Figure 3); (3) teleportation reduces the landscape flatness (Figure 2) and increases the magnitude of the normalized local gradient (Figures 5 and 12) proportionally to the CoB range σ used, and (4) applying *one* teleportation during training accelerates the convergence of gradient descent with and without momentum (Figures 6–8 and 11).

Although teleportation looks only like a trick to preserve the network function by just rescaling the weights and the activation functions, it is more than a trick as (i) it comes from the foundational definitions and constructions of representation theory, which neural networks exactly satisfy, and (ii) it has unexpected properties on neural network training when applied *once*. The latter underlines the fact that we really do not know the landscape of neural networks and that more tools (or new mathematics) are needed for further understanding. Moreover, in principle, a neural network and its teleportation should train very similarly, since they both have the same network function, and we have demonstrated that this is not the case.

We conclude that neural teleportation, which is a very simple operation, has remarkable and unexpected properties. We expect these phenomena to motivate the study of the link between neural networks and quiver representations.

Finally, we acknowledge that several different types of change of basis sampling can be performed instead of the uniform distribution around 1 and -1 that we have chosen. In any case, if there are other types of change of basis sampling for which the properties of interest do not hold, then we ask: what makes uniform sampling different?

Author Contributions: Methodology, T.J.; Software, N.P., Y.S., C.L., G.G.S. and P.S.; Investigation, M.A.; Supervision, P.-M.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Our results can be reproduced with the code available here: https://github.com/vitalab/neuralteleportation, accessed on 24 December 2022.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Appendix A.1. Residual Connections and Batchnorm

Here, we present an illustration of how teleportation acts on both residual connections and batchnorm as performed by ([1], chap. 5).

Consider a simple neural network with 5 neurons where the second and fourth neurons are connected by a residual connection, i.e., its underlying graph (quiver) is of the form

$$\begin{tikzcd}
a \arrow[r, "\alpha"] & b \arrow[loop above] \arrow[r, "\beta"] \arrow[rr, bend right, "\epsilon"'] & c \arrow[loop above] \arrow[r, "\gamma"] & d \arrow[loop above] \arrow[r, "\delta"] & e
\end{tikzcd}$$

and the neural network looks like

$$\begin{tikzcd}
\mathbb{R} \arrow[r, "W_\alpha"] & \mathbb{R} \arrow[loop above, "f_b"] \arrow[r, "W_\beta"] \arrow[rr, bend right, "1"'] & \mathbb{R} \arrow[loop above, "f_c"] \arrow[r, "W_\gamma"] & \mathbb{R} \arrow[loop above, "f_d"] \arrow[r, "W_\delta"] & \mathbb{R}.
\end{tikzcd}$$

where f_b, f_c and f_d are activation functions. A CoB of this neural newtork is of the form $\tau = (1, \tau_b, \tau_c, \tau_d, 1)$, so that the teleportation with respect to τ is the following neural newtork

$$\begin{tikzcd}
\mathbb{R} \arrow[r, "W_\alpha \tau_b"] & \mathbb{R} \arrow["\tau_b \cdot f_b", loop above] \arrow[r, "W_\beta/\tau_b"] \arrow[rr, bend right, "\tau_b/\tau_d"'] & \mathbb{R} \arrow["\tau_c \cdot f_c", loop above] \arrow[r, "W_\gamma \tau_d"] & \mathbb{R} \arrow["\tau_d \cdot f_d", loop above] \arrow[r, "W_\delta/\tau_d"] & \mathbb{R}.
\end{tikzcd}$$

where $\tau_b \cdot f_b(x) = \tau_b f_b(x/\tau_b)$. So if $\tau_b = \tau_d$ then the the teleported network looks as follows

$$\begin{tikzcd}
\mathbb{R} \arrow[r, "W_\alpha \tau_b"] & \mathbb{R} \arrow["\tau_b \cdot f_b", loop above] \arrow[r, "W_\beta/\tau_b"] \arrow[rr, bend right, "1"'] & \mathbb{R} \arrow["\tau_c \cdot f_c", loop above] \arrow[r, "W_\gamma \tau_b"] & \mathbb{R} \arrow["\tau_b \cdot f_d", loop above] \arrow[r, "W_\delta/\tau_b"] & \mathbb{R}.
\end{tikzcd}$$

Therefore, if the CoBs of layers connected with residual connections are equal then the teleportation produced with that CoB will have a residual connection in the same place as the original network.

Consider now the following graph (quiver)

$$\begin{tikzcd}
& a \arrow[loop above] \arrow[rr] & & c \arrow[loop above] \\
b \arrow[ur] & & d \arrow[ur] &
\end{tikzcd}$$

We can describe the batchnorm operation on this graph if we consider a layer with only one neuron as follows

$$\begin{tikzcd}
& \mathbb{R} \arrow["1", loop above] \arrow[rr, "\gamma/\sigma^2"'] & & \mathbb{R} \arrow["1", loop above] \\
\mathbb{R} \arrow[ur, "-\mu"'] & & \mathbb{R} \arrow[ur, "\beta"'] &
\end{tikzcd}$$

where the bottom neurons are considered as bias neurons. It is easy to see that if we feed an input x through this network (i.e., in the top left neuron) we obtain the desired operation $x \mapsto x - \mu \mapsto (x - \mu)(\gamma/\sigma^2) + \beta$. A CoB for this simple layer is of the form $\tau = (\tau_a, \tau_b, \tau_c, \tau_d)$ where $\tau_b = \tau_d = 1$ because the bottom neurons are considered as bias neurons. So, a teleportation of the previous batchnorm layer with a CoB $\tau = (\tau_a, 1, \tau_c, 1)$ is of the form

$$\begin{tikzcd}
& \mathbb{R} \arrow["1", loop above] \arrow[rr, "\gamma\tau_c/\sigma^2\tau_a"'] & & \mathbb{R} \arrow["1", loop above] \\
\mathbb{R} \arrow[ur, "-\mu\tau_a"'] & & \mathbb{R} \arrow[ur, "\beta\tau_c"'] &
\end{tikzcd}$$

and so if $\tau_a = 1$ then we obtain the layer

$$\begin{tikzcd}
& \mathbb{R} \arrow["1", loop above] \arrow[rr, "\gamma\tau_c/\sigma^2"'] & & \mathbb{R} \arrow["1", loop above] \\
\mathbb{R} \arrow[ur, "-\mu"'] & & \mathbb{R} \arrow[ur, "\beta\tau_c"'] &
\end{tikzcd}$$

which is a batchnorm layer since the mean and variance μ and σ^2 remain the same after teleportation.

Appendix A.2. Micro-Teleportations

Following Section 4.3 on micro-teleportations, we provide more angular histograms between back-propagated gradients and random micro-teleportations of the network. We present here some more histograms for models MLP, VGG, ResNet and DenseNet on datasets CIFAR-10, CIFAR-100 and random data. Each histogram in Figures A1–A3 was computed with 100 random micro-teleportations with a CoB-range $\sigma = 0.001$.

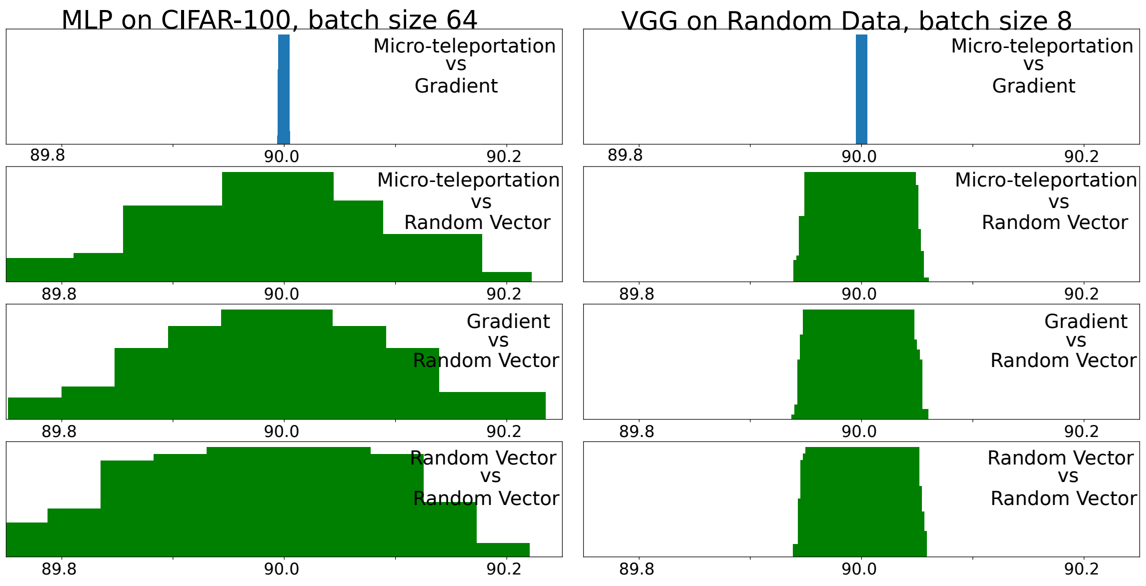

Figure A1. Micro teleportation experiments for MLP on CIFAR-100 and VGG on random data.

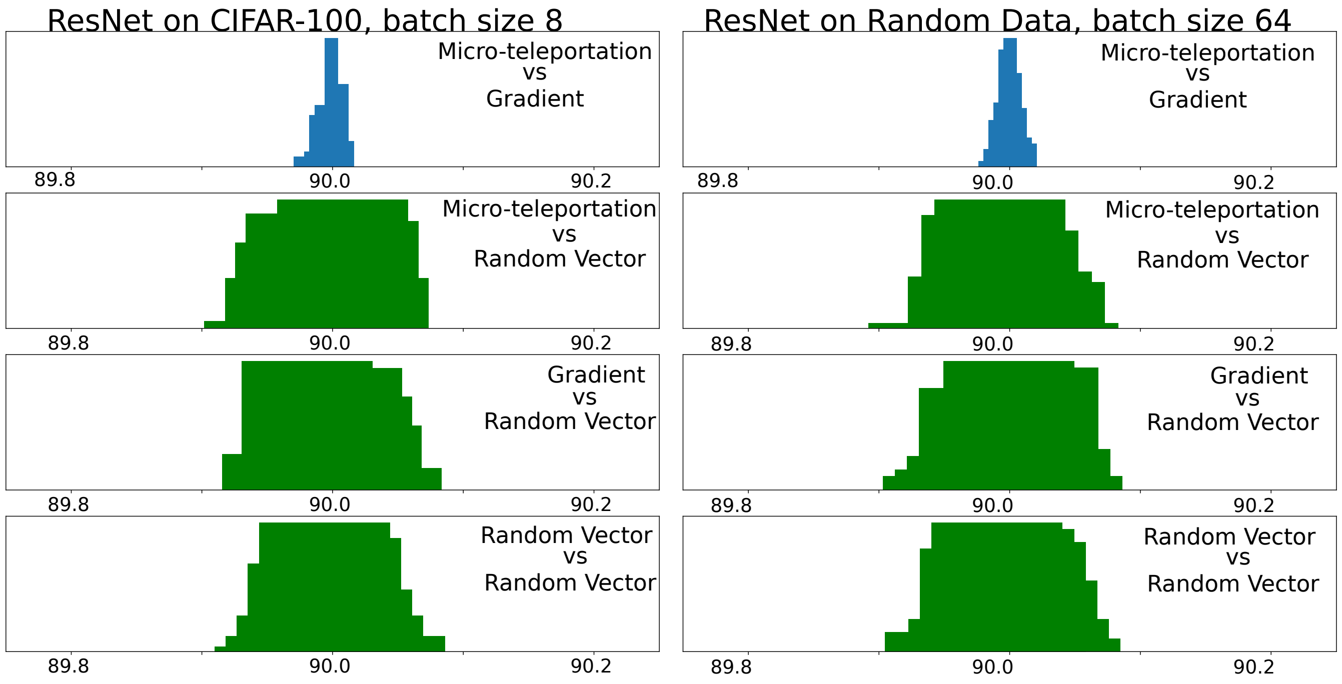

Figure A2. Micro teleportation experiments for ResNet on CIFAR-100 and random data.

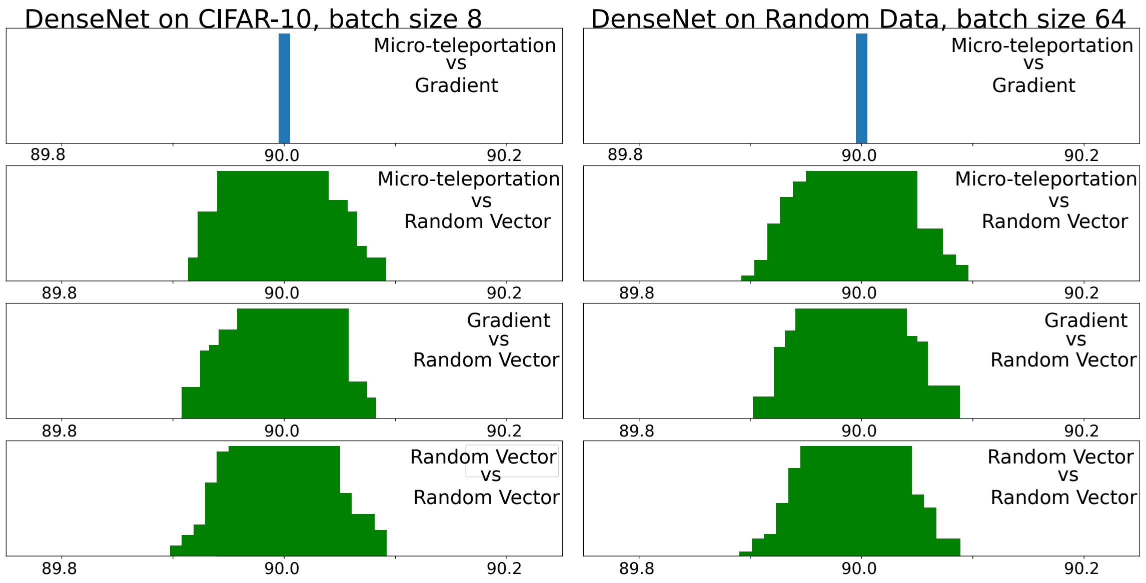

Figure A3. Micro teleportations experiments for DenseNet on CIFAR-10 and random data.

Appendix A.3. Teleportation and Landscape Flatness

Following Section 4.4, we trained a VGGnet with two different batch sizes and then produced a 1D loss plot by interpolating the two models. We then re-produced 1D loss plots by teleporting the models. As shown in Figure A4, neural teleportation has the effect of sharpening the loss landscape.

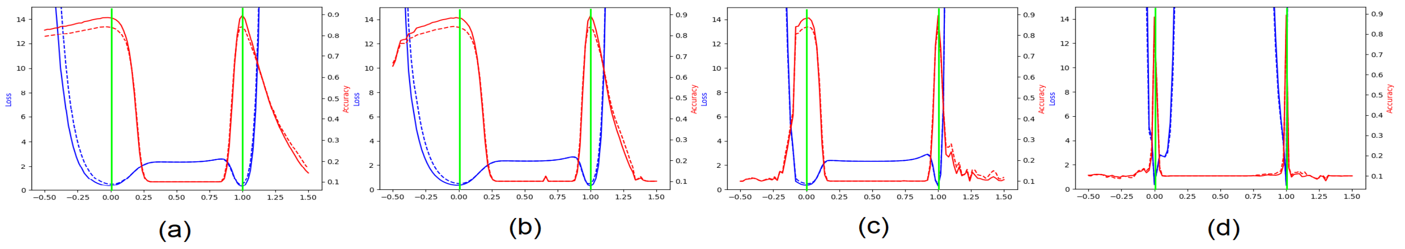

Figure A4. (**a**) Loss/accuracy profiles obtained by linearly interpolating between two optimized VGG A and B. Network A is for $x = 0$ and B for $x = 1$ (green vertical lines). Dotted lines are for training and solid lines for validation. Remaining plots are similar interpolations but between teleported versions of A and B with CoB range σ of (**b**) 0.6, (**c**) 0.9, and (**d**) 0.99.

Appendix A.4. Back-Propagated Gradients of a Teleportation

Here, we give the proof of Theorem 5.1 and reproduce the experiment shown in Figure 5 for the CIFAR-100 dataset. Results are shown in Figure A5.

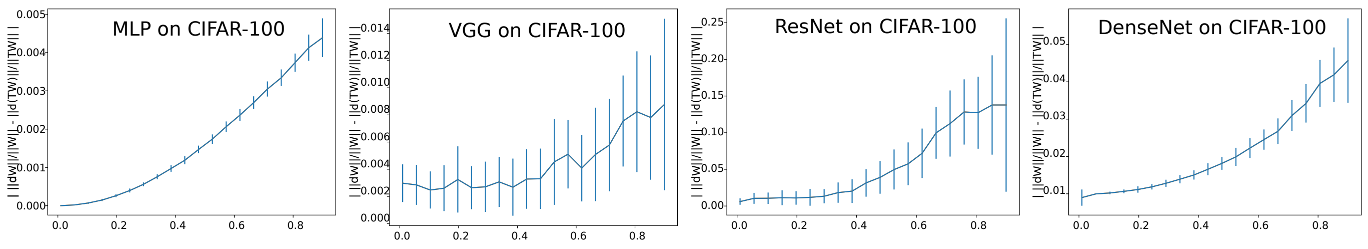

Figure A5. Mean absolute difference (±std-dev) between the back-propagated gradients' magnitudes of teleported networks and their original (non-teleported) network. Larger CoB generate larger gradients.

Appendix A.5. Proof of Theorem 5.1

We will introduce the notation for a forward pass on a neural network (W, f) with L hidden layers without bias vertices for clarity. For the forward pass of (W, f) we first fix a data sample (x, t) and define the vector of activation outputs of neurons at layer ℓ by $a_W^{[\ell]}$, and the vector of pre-activations at layer ℓ by $z_W^{[\ell]}$. We will denote by $f^{[\ell]}$ the vector of activation functions at layer ℓ. For the input layer we have $a_W^{[0]} = z_W^{[0]} = x$. Next, we define inductively

$$z_W^{[\ell]} = W^{[\ell]} a_W^{[\ell-1]} \text{ and } a_W^{[\ell]} = f^{[\ell]}\left(z_W^{[\ell]}\right) \tag{A1}$$

for every $\ell = 1, \dots, L+1$.

In the case of the backward pass, we denote the vector of derivatives of activations in layer ℓ by df^{ℓ}, and the vector of derivatives with respect to the activation outputs of neurons on layer ℓ by $da_W^{[\ell]}$. On the output layer we have

$$da_W^{[L+1]} = \frac{\partial \mathcal{L}}{\partial a_W^{[L+1]}}(a_W^{[L+1]}, y) \tag{A2}$$

where $\mathcal{L}$ is the loss function. If we denote by $\odot$ the point-wise (Hadamard) product of vectors of the same size, then inductively from layer $L+1$ down to the input layer we have

$$dW^{[\ell]} = \left(da_W^{[\ell]} \odot df^{[\ell]}\left(z_W^{[\ell]}\right)\right) a_W^{[\ell-1]^T}, \tag{A3}$$

and also that

$$da_W^{[\ell-1]} = \left(W^{[\ell]}\right)^T \left(da_W^{[\ell]} \odot df^{[\ell]}\left(z_W^{[\ell]}\right)\right). \tag{A4}$$

If $\tau : (W, f) \to (V, g)$ is an isomorphism of neural networks, given by a choice of change of basis in every hidden neuron and we denote by $\tau^{[\ell]}$ the vector of change of basis for layer ℓ, then

$$V^{[\ell]} = \frac{1}{\tau^{[\ell-1]}} \cdot W^{[\ell]} \cdot \tau^{[\ell]}. \tag{A5}$$

and for the activation functions we have

$$g^{[\ell]}(x) = f^{[\ell]}\left(x \cdot \frac{1}{\tau^{[\ell]}}\right) \cdot \tau^{[\ell]}, \tag{A6}$$

where the operation $\frac{1}{\tau^{[\ell-1]}} \cdot -$ on the left of a matrix, multiplies its columns by the coordinate values of vector $\frac{1}{\tau^{[\ell-1]}}$. While the operation $- \cdot \tau^{[\ell]}$ on the right of a matrix, multiplies its rows by the coordinate values of the vector $\tau^{[\ell]}$. By transposing Equation (A5), we obtain

$$\left(V^{[\ell]}\right)^T = \tau^{[\ell]} \cdot \left(W^{[\ell]}\right)^T \cdot \frac{1}{\tau^{[\ell-1]}}. \tag{A7}$$

By the chain rule and Equation (A6) we can see that

$$dg^{[\ell]}(x) = df^{[\ell]}\left(x \cdot \frac{1}{\tau^{[\ell]}}\right). \tag{A8}$$

In addition, from the proof of Theorem 4.9 [1],

$$z_V^{[\ell]} = z_W^{[\ell]} \cdot \tau^{[\ell]} \quad \text{and} \quad a_V^{[\ell]} = a_W^{[\ell]} \cdot \tau^{[\ell]}. \tag{A9}$$

Theorem A1. *Let (V, g) be a teleportation of the neural network (W, f) with respect to the CoB τ. Then*

$$dV^{[\ell]} = \tau^{[\ell-1]} \cdot dW^{[\ell]} \cdot \frac{1}{\tau^{[\ell]}}$$

for every layer ℓ of the network.

Proof. We proceed by induction on the steps of the backward pass of the network (V, g). By Theorem 4.9 of [1], both networks (W, f) and (V, g) give the same output $a_W^{[L+1]} = a_V^{[L+1]}$, so by Equation (A2) we obtain that $da_W^{[L+1]} = da_V^{[L+1]}$. We can substitute this together with Equation (A9) into Equation (A3) applied to the neural network (V, g) to obtain

$$\begin{aligned} dV^{[L+1]} &= da_V^{[L+1]} a_V^{[L]^T} \\ &= da_W^{[L+1]} \left(a_W^{[L]} \cdot \tau^{[L]}\right)^T \\ &= dW^{[L+1]} \cdot \tfrac{1}{\tau^{[L]}}. \end{aligned}$$

Now, the derivative of the loss function with respect to the activation outputs of layer L in (V, g) can be written analogously to Equation (A4), in which we substitute Equation (A7) taking into account that the CoB for the output layer is given by 1's, so we obtain

$$\begin{aligned} da_V^{[L]} &= \left(V^{[L+1]}\right)^T da_V^{[L+1]} \\ &= \left(W^{[L+1]}\right)^T \cdot \tfrac{1}{\tau^{[L]}} da_W^{[L+1]} \\ &= da_W^{[L]} \cdot \tfrac{1}{\tau^{[L]}}. \end{aligned}$$

At layer L we obtain

$$\begin{aligned} & dV^{[L]} = \left(da_V^{[L]} \odot dg^{[L]}\left(z_V^{[L]}\right)\right) a_V^{[L-1]^T} \\ & = \left(da_W^{[L]} \cdot \tfrac{1}{\tau^{[L]}}\right) \odot df^{[L]}\left(z_W^{[L]} \cdot \tau^{[L]} \cdot \tfrac{1}{\tau^{[L]}}\right) a_V^{[L-1]^T} \\ & = \left(da_W^{[L]} \odot df^{[L]}\left(z_W^{[L]}\right) \cdot \tfrac{1}{\tau^{[L]}}\right)\left(a_W^{[L-1]} \cdot \tau^{[L-1]}\right)^T \\ & = \left(da_W^{[L]} \odot df^{[L]}\left(z_W^{[L]}\right) \cdot \tfrac{1}{\tau^{[L]}}\right)\left(\tau^{[L-1]} \cdot a_W^{[L-1]^T}\right) \\ & = \tau^{[L-1]} \cdot \left(da_W^{[L]} \odot df^{[L]}\left(z_W^{[L]}\right) a_W^{[L-1]^T}\right) \cdot \tfrac{1}{\tau^{[L]}} \\ & = \tau^{[L-1]} \cdot dW^{[L]} \cdot \tfrac{1}{\tau^{[L]}}. \end{aligned}$$

We then observe that

$$\begin{aligned} & da_V^{[L-1]} = \left(V^{[L]}\right)^T da_V^{[L]} \\ & = \left(\tau^{[L]} \cdot \left(W^{[L]}\right)^T \cdot \tfrac{1}{\tau^{[L-1]}}\right)\left(da_W^{[L]} \cdot \tfrac{1}{\tau^{[L]}}\right) \\ & = \left(W^{[L]}\right)^T da_W^{[L]} \cdot \tfrac{1}{\tau^{[L-1]}} \\ & = da_W^{[L-1]} \cdot \tfrac{1}{\tau^{[L-1]}}. \end{aligned}$$

For the inductive step, we assume the result to be true for layers $L+1, \ldots, \ell+1$. We will prove the result holds for layer ℓ. Indeed,

$$\begin{aligned} & dV^{[\ell]} = \left(da_V^{[\ell]} \odot dg^{[\ell]}\left(z_V^{[\ell]}\right)\right) a_V^{[\ell-1]^T} \\ & = \left(da_W^{[\ell]} \cdot \tfrac{1}{\tau^{[\ell]}}\right) \odot df^{[\ell]}\left(z_W^{[\ell]} \cdot \tau^{[\ell]} \cdot \tfrac{1}{\tau^{[\ell]}}\right) a_V^{[\ell-1]^T} \\ & = \left(da_W^{[\ell]} \odot df^{[\ell]}\left(z_W^{[\ell]}\right) \cdot \tfrac{1}{\tau^{[\ell]}}\right)\left(a_W^{[\ell-1]} \cdot \tau^{[\ell-1]}\right)^T \\ & = \left(da_W^{[\ell]} \odot df^{[\ell]}\left(z_W^{[\ell]}\right)\right)\left(\tau^{[\ell-1]} \cdot a_W^{[\ell-1]^T}\right) \cdot \tfrac{1}{\tau^{[\ell]}} \\ & = \tau^{[\ell-1]} \cdot \left(da_W^{[\ell]} \odot df^{[\ell]}\left(z_W^{[\ell]}\right)\right) a_W^{[\ell-1]^T} \cdot \tfrac{1}{\tau^{[\ell]}} \\ & = \tau^{[\ell-1]} \cdot dW^{[\ell]} \cdot \tfrac{1}{\tau^{[\ell]}}. \end{aligned}$$

It can be appreciated that back-propagation on (V, g) computes a re-scaling of the gradient of (W, f) by the CoB, just as claimed in the statement of the theorem. □

Appendix A.6. Gradient Descent Speed Up

Following Section 5.2, we present the training curves for the remaining models and datasets of our training experiments on Figures A6–A11.

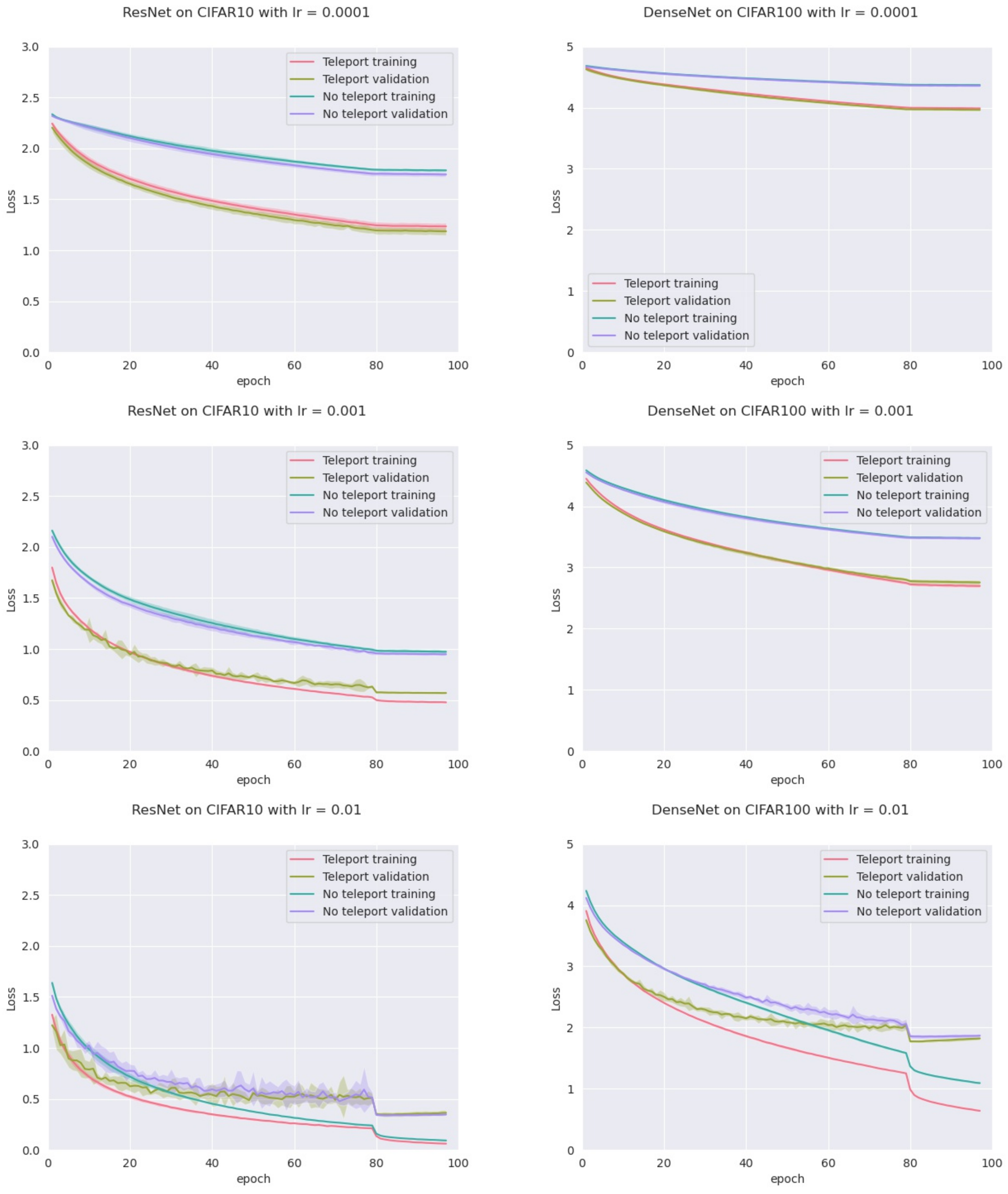

Figure A6. Average loss of ResNet on CIFAR10 (**left**) and DenseNet on CIFAR100 (**right**) over 5 experiments, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

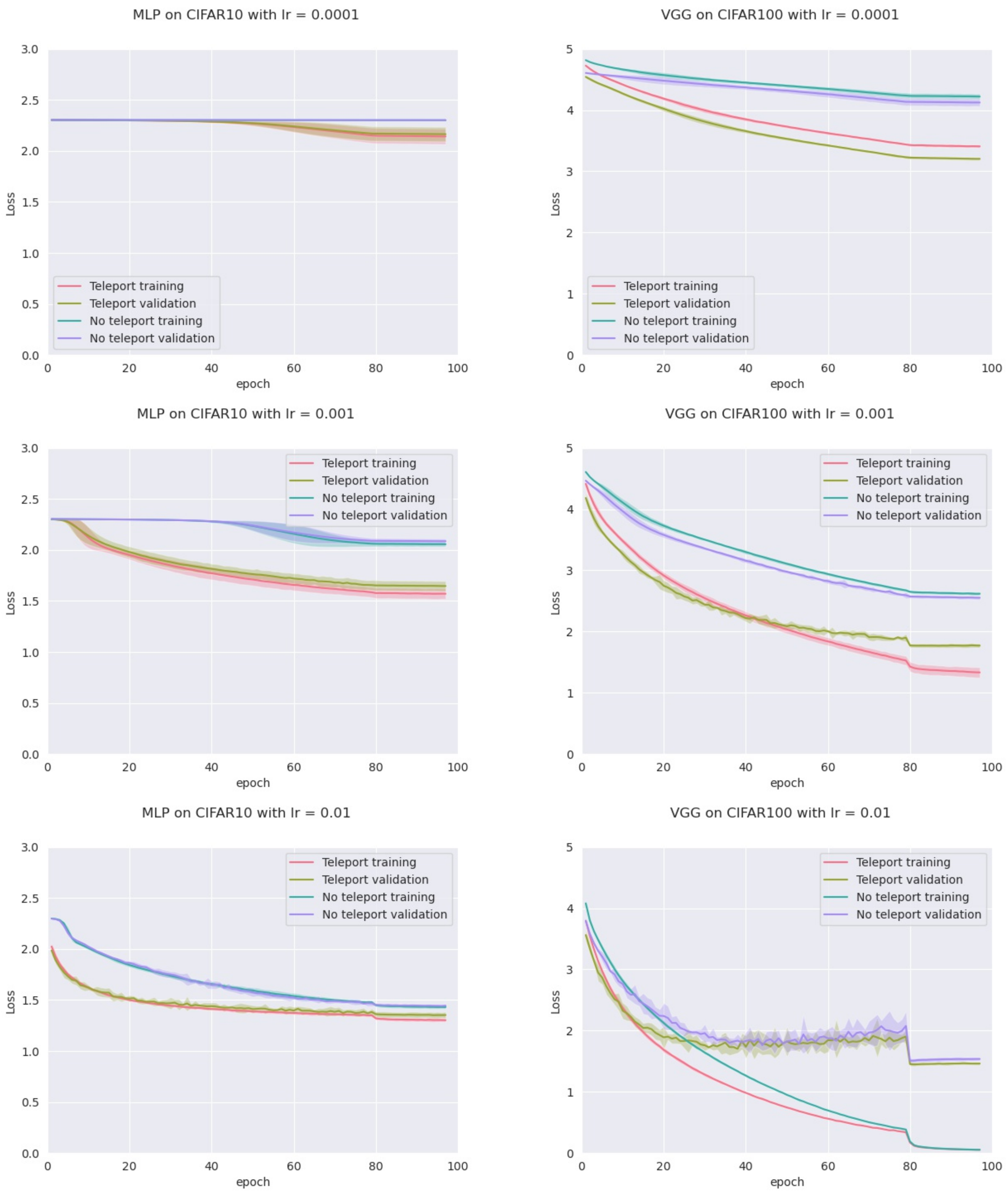

Figure A7. Average loss of MLP on CIFAR10 (**left**) and VGG on CIFAR100 (**right**) over 5 experiments, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

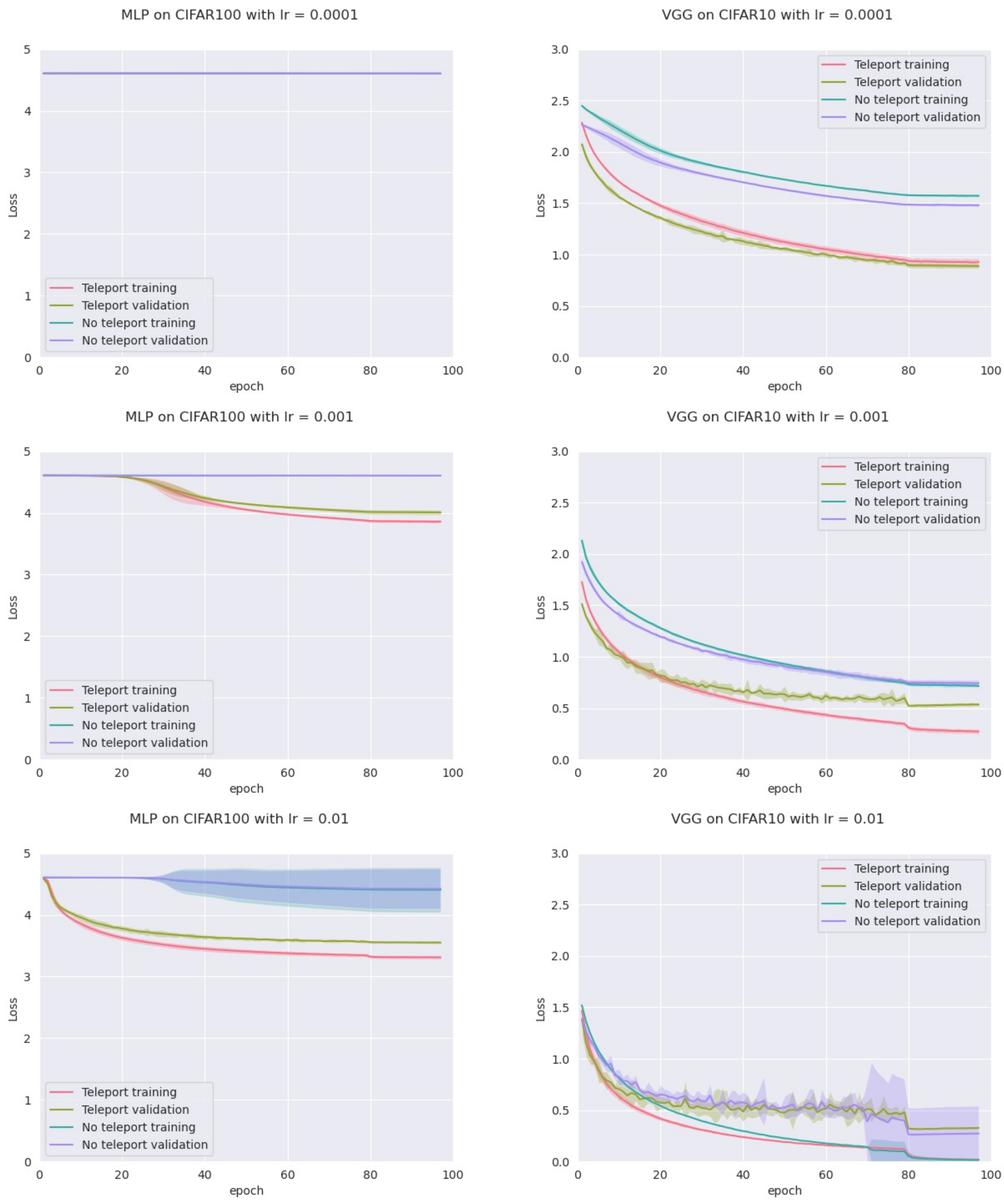

Figure A8. Average loss of MLP on CIFAR100 (**left**) and VGG on CIFAR10 (**right**) over 5 experiments, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

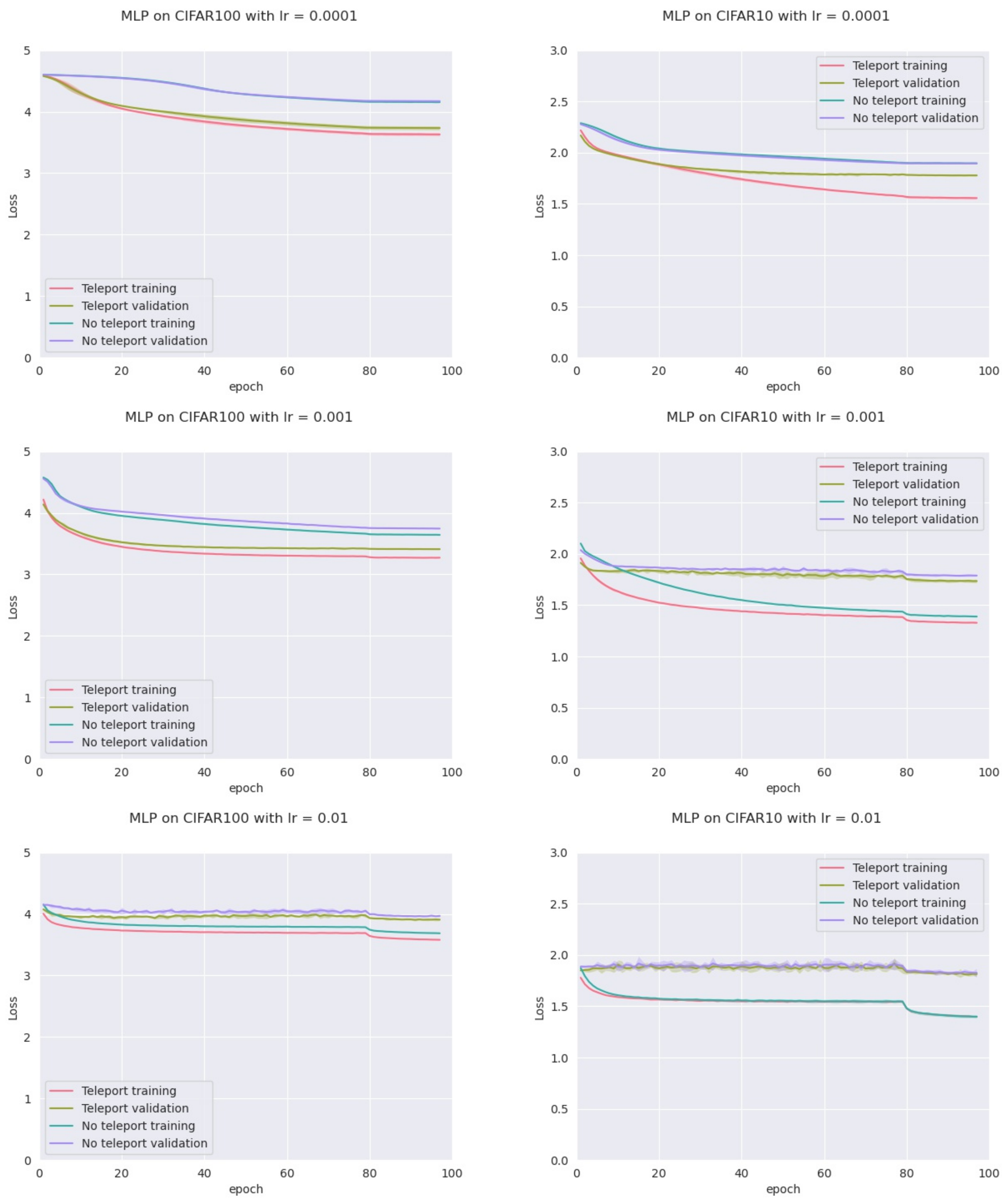

Figure A9. Average loss of MLPs with ELU activation function on CIFAR100 (**left**) and Tanh on CIFAR10 (**right**) over 5 experiments for each, with and without teleportation at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

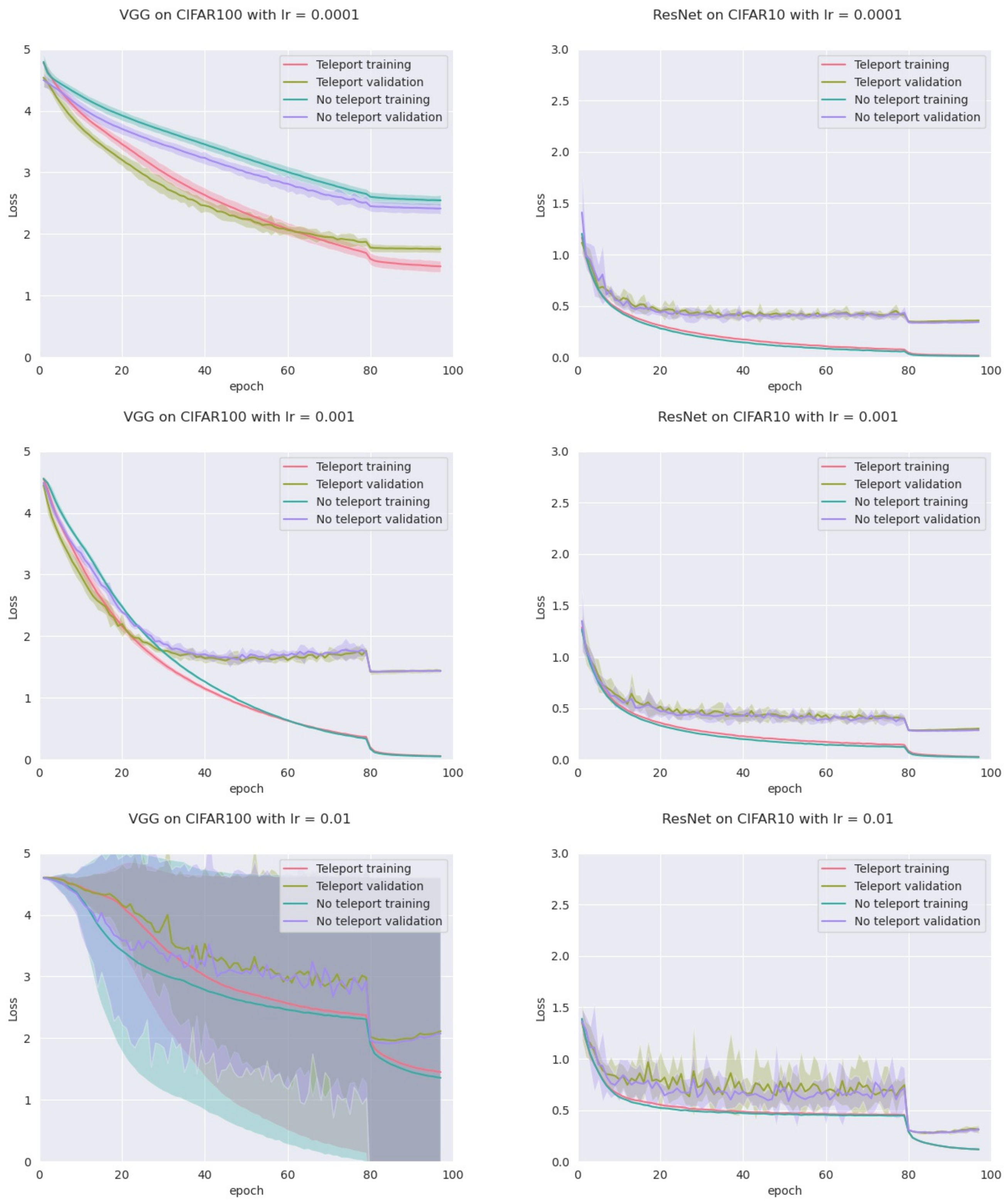

Figure A10. Average loss of VGG with normal initialization on CIFAR100 (**left**) and ResNet with Xavier initialization on CIFAR10 (**right**) both training with SGD with momentum over 5 experiments for each, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

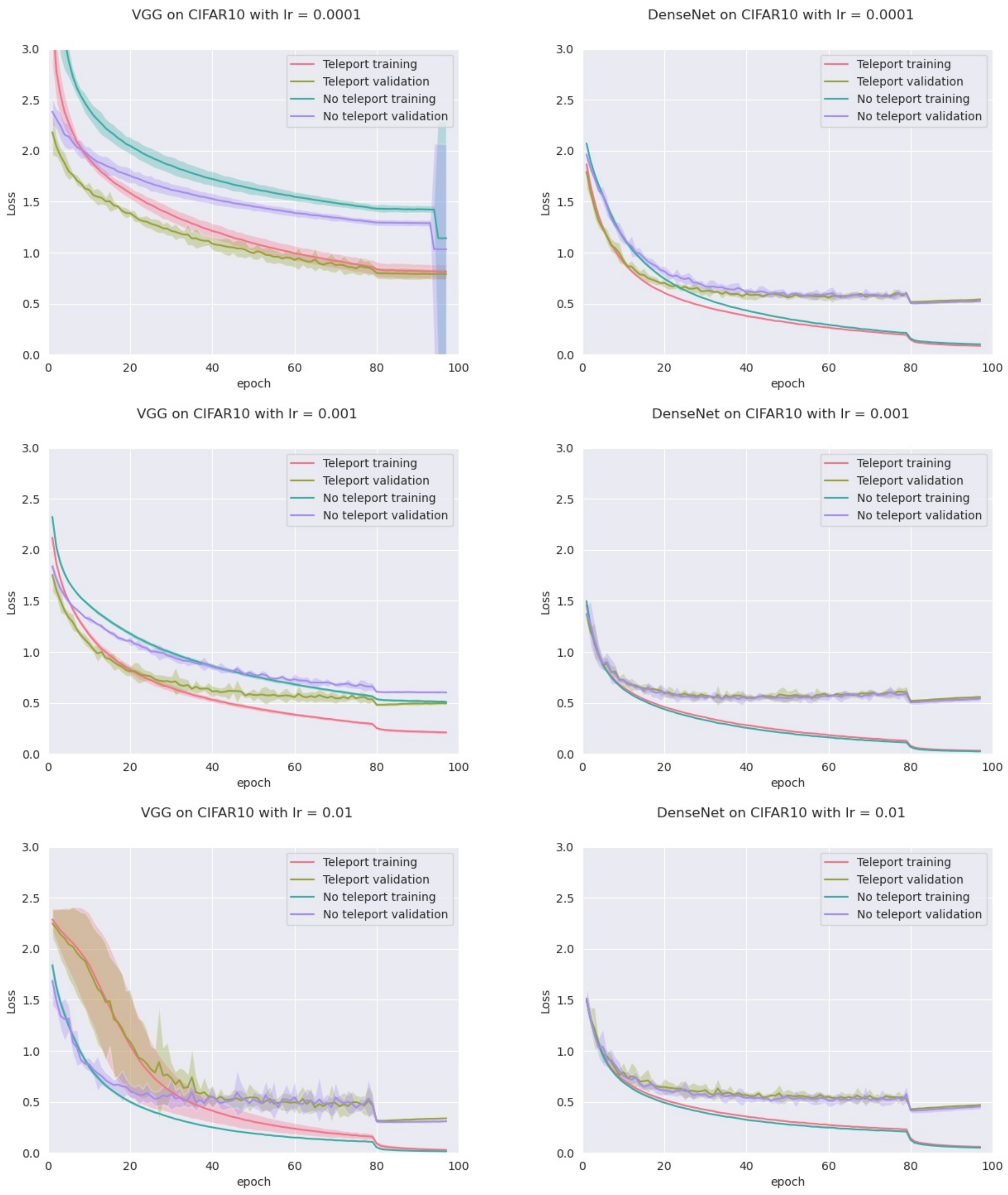

Figure A11. Average loss of MLP with normal initialization on CIFAR100 (**left**) and DenseNet with Xavier initialization on CIFAR10 (**right**) both traing with SGD over 5 experiments for each, with and without teleportation, at the beginning of the training with CoB of 0.9. Learning rate increases from top to bottom rows. We also show a shadow around the average curves at two times the standard deviation over the five experiments.

References

1. Armenta, M.A.; Jodoin, P.M. The Representation Theory of Neural Networks. *arXiv* **2020**, arXiv:2007.12213 .
2. Neyshabur, B.; Salakhutdinov, R.; Srebro, N. Path-SGD: Path-Normalized Optimization in Deep Neural Networks. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*; MIT Press: Cambridge, MA, USA, 2015; pp. 2422–2430.
3. Meng, Q.; Zheng, S.; Zhang, H.; Chen, W.; Ye, Q.; Ma, Z.; Yu, N.; Liu, T. G-SGD: Optimizing ReLU Neural Networks in its Positively Scale-Invariant Space. *arXiv* **2018**, arXiv:1802.03713.
4. Badrinarayanan, V.; Mishra, B.; Cipolla, R. Understanding Symmetries in Deep Networks. *arXiv* **2015**, arXiv:1511.01029.
5. Dinh, L.; Pascanu, R.; Bengio, S.; Bengio, Y. Sharp Minima Can Generalize for Deep Nets. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 1019–1028.
6. Arora, S.; Li, Z.; Lyu, K. Theoretical Analysis of Auto Rate-Tuning by Batch Normalization. *arXiv* **2018**, arXiv:1812.03981.
7. Goodfellow, I.J.; Warde-Farley, D.; Mirza, M.; Courville, A.; Bengio, Y. Maxout networks. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; Volume 3, pp. 1319–1327.
8. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
9. Saxe, A.M.; Mcclelland, J.L.; Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural network. *arXiv* **2013**, arXiv:1312.6120.
10. Assem, I.; Simson, D.; Skowronski, A. *Elements of the Representation Theory of Associative Algebras. Vol. 1: Techniques of Representation Theory*; London Mathematical Society Student Texts; Cambridge University Press: Cambridge, UK, 2006; Volume 65, p. ix, 458.
11. Badrinarayanan, V.; Mishra, B.; Cipolla, R. Symmetry-invariant optimization in deep networks. *arXiv* **2015**, arXiv:1511.01754.
12. Huang, L.; Liu, X.; Qin, J.; Zhu, F.; Liu, L.; Shao, L. Projection based weight normalization: Efficient method for optimization on oblique manifold in DNNs. *Pattern Recognit.* **2020**, *105*, 107317. [CrossRef]
13. Cho, M.; Lee, J. Riemannian Approach to Batch Normalization. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*; Curran Associates Inc.: Red Hook, NY, USA, 2017; pp. 5231–5241.
14. Spivak, M. *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*; Mathematics Monograph Series; Addison-Wesley Publishing Company: Boston, MA, USA, 1965.
15. Kainen, P.C.; Kůrková, V. Quasiorthogonal Dimension. In *Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy etc. Methods and Their Applications*; Kosheleva, O., Shary, S., Xiang, G., Zapatrin, R., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2020; pp. 615–629.
16. Li, H.; Xu, Z.; Taylor, G.; Studer, C.; Goldstein, T. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*; Curran Associates Inc.: Red Hook, NY, USA, 2018; pp. 6391–6401.