Symmetry Teleportation for Accelerated Optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

Existing gradient-based optimization methods update the parameters locally, in a direction that minimizes the loss function. We study a different approach, symmetry teleportation, that allows the parameters to travel a large distance on the loss level set, in order to improve the convergence speed in subsequent steps. Teleportation exploits parameter space symmetries of the optimization problem and transforms parameters while keeping the loss invariant. We derive the loss-invariant group actions for test functions and multi-layer neural networks, and prove a necessary condition of when teleportation improves convergence rate. We also show that our algorithm is closely related to second order methods. Experimentally, we show that teleportation improves the convergence speed of gradient descent and AdaGrad for several optimization problems including test functions, multi-layer regressions, and MNIST classification.

1 Introduction

Consider the optimization problem of finding $\text{argmin}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$, where $\mathcal{L}$ is a loss function and $\boldsymbol{w}$ are the parameters. While finding global minima of $\mathcal{L}(\boldsymbol{w})$ is hard for non-convex problems, we can use gradient descent (GD) to find local minima. In GD we apply the following update rule at every step:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \nabla \mathcal{L}, \tag{1}$$

where η is the learning rate. Gradient-descent is a first-order method that uses only gradient information. It is easy to compute but suffers from slow convergence. Second-order methods such as Newton's method use the second derivative to account for the landscape geometry. These methods enjoy faster convergence, but calculating the second derivative (Hessian) can be computationally expensive in high dimensions Hazan (2019).

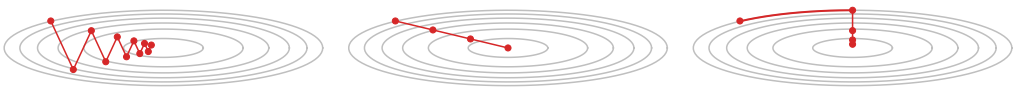

Figure 1: Left to right: gradient descent, second-order methods, proposed method.

In this paper, we propose a new optimization method based on parameter space symmetries. Our algorithm takes advantage of higher-order landscape geometry but uses only gradient information in most steps and thus avoids the computational cost of second-order methods.

Specifically, we look beyond local optimization and ask: *what if we allow the parameters to make a big jump once in a while?* As shown in Figure 1, during optimization, we teleport the parameters to another point with steeper gradients using a loss-invariant symmetry transformation. After teleportation, the loss stays the same, but the gradient and hence the rate of loss decay may change.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

These transformations move the parameters to a better trajectory under gradient descent, which improves the convergence speed of subsequent steps.

The locality of gradient descent is reflected in its formulation in terms of a proximal term; our method circumvents this locality by teleporting to new locations with the same loss. For GD, an equivalent problem uses the following proximal mapping Combettes and Pesquet (2011). Let $\langle \cdot, \cdot \rangle$ denote the inner product, we have

$$\boldsymbol{w}_{t+1} = \text{argmin}_{\boldsymbol{w}} \left\{ \eta \langle (\nabla \mathcal{L})|_{\boldsymbol{w}_t}, \boldsymbol{w} \rangle + \frac{1}{2} \|\boldsymbol{w} - \boldsymbol{w}_t\|_2^2 \right\}. \tag{2}$$

The term $\frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2$ is the proximal term that keeps $\boldsymbol{w}_{t+1}$ close to $\boldsymbol{w}_t$. Adaptive gradient methods, such as Adagrad, define the proximal term using the Mahalanobis distance $\|\boldsymbol{w} - \boldsymbol{w}_t\|^2_{G^{-1/2}}$ (Duchi et al., 2011) to account for landscape geometry. Our proposed teleportation technique relaxes the requirement from the proximal term. We teleport to points on the same level set of $\mathcal{L}(\boldsymbol{w}_t)$, but allow $\boldsymbol{w}$ to be far from $\boldsymbol{w}_t$ in Euclidean distance.

In summary, our main contributions are:

- We propose *symmetry teleportation*, an accelerated gradient-based optimization algorithm that exploits symmetries in the loss landscape.
- We derive loss-invariant group actions of test functions and multi-layer neural networks.
- We provide necessary conditions and examples of when teleportation improves convergence rate of the current and subsequent steps.
- We show that our algorithm is closely related to second order methods.
- Experimentally, we show that teleportation improves the convergence speed of gradient descent and AdaGrad in different optimization problems.

2 Related Work

Continuous parameter space symmetries have been identified in neural networks with homogeneous (Badrinarayanan et al., 2015; Du et al., 2018) and radial activation functions (Ganev and Walters, 2021). The effect of symmetry transformations on gradients has been examined for translation, scale, and rescale symmetries in Kunin et al. (2021). We contribute to this line of work by deriving loss-invariant group actions in multi-layer neural networks with invertible activation functions.

Several works exploit symmetry to facilitate optimization. For example, Path-SGD (Neyshabur et al., 2015) improves optimization in ReLU networks by path regularization. $\mathcal{G}$-SGD (Meng et al., 2019) performs weight updates in a scale-invariant space, which also exploits the symmetry in ReLU networks. Van Laarhoven (2017) analyzes the effect of L2 regularization in batch normalization which gives scale-invariant functions. Bamler and Mandt (2018) transforms parameters in the symmetry orbits to address slow movement along directions of weakly broken symmetry and find the optimal $g \in G$ that minimizes $\mathcal{L}(g \cdot \boldsymbol{w})$. In comparison, we search within G-orbits for a point that has the largest gradient, i.e., we find the optimal g to maximize $d\mathcal{L}(g \cdot \boldsymbol{w})/dt$.

Natural gradient Amari (1998), adaptive gradient methods Duchi et al. (2011); Kingma and Ba (2014), and their approximations Martens and Grosse (2018); Gupta et al. (2018) improve the direction of parameter updates instead of directly transforming parameters. If the group acts transitively on the level set, and we teleport to the point with maximum gradient, then our update direction is the same as that in Newton's method. We prove that our algorithm is connected to second-order optimization methods and show that teleportation can be used to improve these methods empirically.

The concept of neural teleportation was first explored using quiver representation theory Armenta and Jodoin (2021); Armenta et al. (2020). These works provide a way to explore level curves of the loss of neural networks and show that random teleportation speeds up gradient descent experimentally and theoretically. Our algorithm improves neural teleportation by searching for teleportation destinies that leads to the largest improvement in the magnitude of gradient.

Several works study how parameter initialization affects convergence rate Saxe et al. (2014); Tarmoun et al. (2021); Min et al. (2021). If we apply a group transformation only at initialization, our method is similar to that of Tarmoun et al. (2021). We do not guarantee that the transformed parameters lead

to faster convergence rate throughout the entire training. However, we accelerate convergence at least for a short time after initialization. Additionally, our method is not restricted to initialization and can be applied at any time during training.

Most contemporary neural networks are overparametrized. While this has been shown to improve generalization (Belkin et al., 2019), an important question is how overparametrization affects optimization. A series of works starting from Arora et al. (2018) show that overparametrization resulting from the depth of a neural network accelerates convergence. Another view is that the symmetry created by overparametrization poses constraints on trajectories in the form of conserved quantities (Głuch and Urbanke, 2021). Additionally, the symmetry generates additional trajectories. When the trajectories created by overparametrization are equivalent, model compression by removing symmetry reduces training time (Ganev and Walters, 2021). However, when trajectories are not equivalent, gradient flow on some paths are faster than others. We search for the better paths created by overparametrization.

3 Symmetry Teleportation

We propose *symmetry teleportation*, an accelerated gradient-based optimization algorithm that exploits symmetries in the loss landscape. Symmetry teleportation searches for the best gradient descent trajectory by teleporting parameters to a point with larger gradients using a group action. The resulting algorithm requires only gradient computations but is able to account for the global landscape geometry, leading to faster loss decay.

Let the group G be a set of symmetries which leave the loss function $\mathcal{L}$ invariant: $\mathcal{L}(g \cdot (\boldsymbol{w}, X)) = \mathcal{L}(\boldsymbol{w}, X)$, where $g \in G$. We perform gradient descent for T steps. We define an index set $K \subseteq \{0, 1, 2, ..., T\}$ as a teleportation schedule. At epochs that are in the schedule, we transform parameters using group element $g \in G$ to the location where the gradient is largest, then continue with gradient descent steps. Algorithm 1 describes the details of this procedure. Note that the loss does not change after teleportation (Line 2-5) since $\mathcal{L}$ is G-invariant.

Algorithm 1: Symmetry Teleportation

Input: Loss function $\mathcal{L}(\boldsymbol{w})$, learning rate η, number of epochs T, initialized parameters $\boldsymbol{w}_0$, symmetry group G, teleportation schedule K.

Output: $\boldsymbol{w}_T$.

1 **for** $t \leftarrow 0$ **to** $T-1$ **do**
2 **if** $t \in K$ **then**
3   $g \leftarrow \text{argmax}_{g \in G} \|(\nabla \mathcal{L})|_{g \cdot \boldsymbol{w}_t}\|^2$
4   $\boldsymbol{w}_t \leftarrow g \cdot \boldsymbol{w}_t$
5 **end if**
6 $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta (\nabla \mathcal{L})|_{\boldsymbol{w}_t}$
7 **end for**
8 **return** $\boldsymbol{w}_T$

Algorithm 1 can be generalized to apply teleportation to stochastic gradient descent (Appendix A.1). We discuss some design choices below and provide detailed analysis in the next two sections.

When the action of G is continuous, teleportation can be implemented by parameterizing and performing gradient ascent on g. For example, the $\mathrm{SO}(2)$ group can be parameterized by the rotation angle θ. Small transformations $g \in \mathrm{GL}_d(\mathbb{R})$ (general linear group) can be parameterized as $g \approx I + \varepsilon T$ where $\varepsilon \ll 1$ and T are arbitrary $d \times d$ matrices. For discrete groups, search algorithms or random sampling can be used to find a group element that improves the magnitude of the gradient.

Although $g \cdot (\boldsymbol{w}, X)$ does not change X in the cases we consider in this paper, Algorithm 1 can be extended to allow transformations on both parameters and data. The group actions on data during teleportations can be precomposed and applied to the input data at inference time. The algorithm extension that allows transformation on the input can be found in Appendix A.2.

The teleportation schedule K is a hyperparameter that determines when to perform teleportation. We define K as a set to allow flexible teleportation schedules, such as with non-fixed frequencies or teleporting only at the earlier epochs.

4 Symmetry Groups of Certain Optimization Problems

We give a few practical examples to demonstrate how teleportation can be used to accelerate optimization. Specifically, we first consider two test functions which are often used to evaluate optimization algorithms Back (1996). We then derive the symmetries of multi-layer neural networks.

4.1 Test functions

Rosenbrock function. The Rosenbrock function originally introduced by Rosenbrock (1960) has a characteristic global minimum that is inside a long, narrow, parabolicly-shaped flat valley. Finding the valley is easy but reaching the minimum is difficult. On a 2-dimensional space, the Rosenbrock function has the following form:

$$\mathcal{L}_r(x_1, x_2) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2. \tag{3}$$

Booth function. The Booth function Jamil and Yang (2013) is also defined on $\mathbb{R}^2$ and has one global minimum at $(1, 3)$ where the function evaluates to 0:

$$\mathcal{L}_b(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2. \tag{4}$$

The following proposition identifies the symmetry of these two test functions.

Proposition 4.1. *The Rosenbrock and Booth functions have rotational symmetry. In other words, there exist action maps $a_r, a_b : \mathrm{SO}(2) \times \mathbb{R}^2 \to \mathbb{R}^2$, such that for all $g \in \mathrm{SO}(2)$,*

$$\mathcal{L}_r(x_1, x_2) = \mathcal{L}_r(a_r(g, [x_1, x_2])) \quad \textit{and} \quad \mathcal{L}_b(x_1, x_2) = \mathcal{L}_b(a_b(g, [x_1, x_2])).$$

The exact form of the action maps are deferred to Appendix B.1.2 and B.1.3. During the teleportation step, our goal is to maximize the gradient within a level set of the loss: $\max_{g \in \mathrm{SO}(2)} \| \frac{d\mathcal{L}(g \cdot (x_1, x_2))}{dt} \|$.

4.2 Multi-layer Neural Networks

Next, we consider feed-forward neural networks. Denote the output of the mth layer by $h_m \in \mathbb{R}^{d_m \times n}$, where d_m is the hidden dimension and n is the number of samples. Denote the input by $h_0 = X \in \mathbb{R}^{d_0 \times n}$. Each layer first performs a linear transformation $\tilde{h}_m = W_m h_{m-1}$ where $W_m \in \mathbb{R}^{d_m \times d_{m-1}}$ (we absorb biases into W_m by adding an extra row of ones in $\tilde{h}_{m-1}$). The output h_m is

$$h_m = \sigma(\tilde{h}_m) = \sigma\left(W_m h_{m-1}\right). \tag{5}$$

We assume the activation functions $\sigma : \mathbb{R} \to \mathbb{R}$ are bijections. For instance, Leaky-ReLU is bijective. For other activation, such Sigmoid or Tanh, we analytically extend them to bijective functions (e.g. $\tanh(x) + e^{x-N} - e^{-x+N}$ for $N \gg 1$). For linear activations, we have linear symmetries:

Proposition 4.2. *A linear network is invariant under all groups $G_m \equiv \mathrm{GL}_{d_m}$ acting as*

$$g \cdot (W_m, W_{m-1}) = (W_m g^{-1}, g W_{m-1}), \qquad g \cdot W_k = W_k, \quad \forall k \notin \{m, m-1\}. \tag{6}$$

Similar, in the nonlinear case, we want to find $g \cdot W_m$ that keep the outputs h_k for $k \neq m-1$ invariant. With nonlinearity, $\tilde{h}_m = W_m \sigma(W_{m-1} h_{m-2})$. This network has a $\mathrm{GL}_{d_{m-1}}$ symmetry given by

$$g \cdot (W_m, h_{m-1}) = (W_m g^{-1}, g h_{m-1}), \quad g \cdot h_m = W_m g^{-1} g h_{m-1} = h_m. \tag{7}$$

Thus, we have the following proposition regarding symmetries of nonlinear networks:

Proposition 4.3. *A multi-layer network with bijective activation has a $\mathrm{GL}_{d_{m-1}}$ symmetry. For $g_m \in G_m = \mathrm{GL}_{d_{m-1}}(\mathbb{R})$ the following group action keeps h_p with $p \geq m$ invariant*

$$g_m \cdot W_k = \begin{cases} W_m g_m^{-1} & k = m \\ \sigma^{-1}\left(g \sigma\left(W_{m-1} h_{m-2}\right)\right) h_{m-2}^{+} & k = m-1 \\ W_k & k \notin \{m, m-1\} \end{cases} \tag{8}$$

where h_{m-2}^{+} is the right pseudo-inverse for h_{m-2} (assuming $n \ll d_{m-2}$ and that h_{m-2} is full rank).

Proofs can be found in Appendix B.2. Note that this group action depends on the input to the network as well as the current weights of all the lower layers. Yet, since this action keeps the output of upper and lower layers invariant, multiple G_m for different m applied at the same time still keep the network output invariant. That is, (8) gives a well-defined non-linear group action of $G = GL_{d_1} \times \ldots G_{d_{p-1}}$ on $(X, W_1, \ldots, W_p)$ which depends on but fixes X and keeps the loss function invariant.

5 Theoretical Analysis

5.1 What symmetries help accelerate optimization

We now discuss the conditions that need to be satisfied for teleportation to accelerate GD. For brevity, we denote all trainable parameters (e.g. $\{W_1, \cdots W_p\}$ for the p-layer neural network) by a single flattened vector $\boldsymbol{w} \in \mathbb{R}^n$. Consider a symmetry G of the loss function $\mathcal{L}(\boldsymbol{w})$, meaning $\forall g \in G$, $\mathcal{L}(\boldsymbol{w}) = \mathcal{L}(g \cdot \boldsymbol{w})$. We quantify how teleportation by G changes the rate of loss decay, given by

$$\frac{d\mathcal{L}(\boldsymbol{w})}{dt} = \left\langle \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}, \frac{d\boldsymbol{w}}{dt} \right\rangle = -\left[\nabla \mathcal{L}\right]^T \eta \nabla \mathcal{L} = -\|\nabla \mathcal{L}\|_\eta^2 \tag{9}$$

where η is the matrix of learning rates (which must be positive semi-definite) and $\|v\|_\eta^2 = v^T \eta v$ is the Mahalanobis norm.

The following proposition states the condition a symmetry needs to satisfy to accelerate optimization. (Proofs can be found in Appendix C.1.)

Proposition 5.1. *Let $\boldsymbol{w}' = g \cdot \boldsymbol{w}$ be a point we teleport to. Let $J = \partial \boldsymbol{w}' / \partial \boldsymbol{w}$ be the Jacobian. Symmetry teleportation using g accelerates the rate of decay in $\mathcal{L}$ if it satisfies*

$$\left\| \left[J^{-1}\right]^T \nabla \mathcal{L}(\boldsymbol{w}) \right\|_\eta^2 > \|\nabla \mathcal{L}(\boldsymbol{w})\|_\eta^2 \tag{10}$$

If the action of the symmetry group $G \subset GL_n$ is linear we have $\boldsymbol{w}' = g \cdot \boldsymbol{w} = g\boldsymbol{w}$ and $J = g$. It follows that if G is a subgroup of the certain orthogonal groups, $d\mathcal{L}/dt$ will be invariant:

Corollary 5.2. *Let O_η denote the orthogonal group of invariances of the inverse of the learning rate, η^{-1}, meaning for $g \in O_\eta$, $g^T \eta^{-1} g = \eta^{-1}$. Then*

$$\forall g \in O_\eta, \qquad \frac{d\mathcal{L}(g \cdot \boldsymbol{w})}{dt} = \frac{\mathcal{L}(\boldsymbol{w})}{dt} \tag{11}$$

In the simple case where the learning rate is a constant, $O_\eta = O(n)$ becomes the classic orthogonal group (e.g. rotations) with $g^T g = I$. In general, when g preserves the norm of the gradient, symmetry teleportation has no effect on the convergence rate.

From Theorem 3.2 in Hazan (2019), assuming that $\mathcal{L}$ is β-smooth and is bounded by $|\mathcal{L}| \leq M$, the gradient norm in gradient descent converges as $\frac{1}{2\beta} \sum_{t=1}^{T} \|(\nabla \mathcal{L})|_{\boldsymbol{w}_t}\| \leq 2M$. Teleportation increases $(\nabla \mathcal{L})|_{\boldsymbol{w}_t}$, therefore requiring less time to reach convergence.

5.2 Improvement of subsequent steps

Since teleportation moves the parameters to a point with a larger gradient, and subsequent GD steps are local, we would expect that teleportation improves the magnitude of gradient for a few future steps as well. The following results formalize this intuition (proofs in Appendix C.2.).

Assumption 5.3 (Lipschitz Continuity)**.** *The l2 norm of the gradient is Lipschitz continuous with constant $L \in \mathbb{R}^{\geq 0}$, which is*

$$\left| \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_1} \right\|_2 - \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_2} \right\|_2 \right| \leq L \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|_2 \tag{12}$$

where $\boldsymbol{w}_1, \boldsymbol{w}_2$ are two points in the parameter space and L is the Lipschitz constant.

Proposition 5.4. *Consider the gradient descent with a G-invariant loss $\mathcal{L}(\boldsymbol{w})$ and learning rate $\eta \in \mathbb{R}^+$. Let $\boldsymbol{w}_t$ be the parameter at time t and $\boldsymbol{w}'_t = g \cdot \boldsymbol{w}_t$ the parameter teleported by $g \in G$. Let $\boldsymbol{w}_{t+T}$ and $\boldsymbol{w}'_{t+T}$ be the parameters after T more steps of gradient descent from $\boldsymbol{w}_t$ and $\boldsymbol{w}'_t$ respectively. Under Assumption 5.3, if $\eta L < 1$, and*

$$\frac{\|\partial \mathcal{L}/\partial \boldsymbol{w}'_t\|_2}{\|\partial \mathcal{L}/\partial \boldsymbol{w}_t\|_2} \geq \frac{(1+\eta L)^T}{(1-\eta L)^T} \tag{13}$$

then

$$\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}'_{t+T}} \right\|_2 \geq \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_{t+T}} \right\|_2 \tag{14}$$

The proposition provides a sufficient condition for teleportation to improve T future steps. The condition is met when L is small, η is small, or the initial improvement, $\left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t'}\right\|_2 / \left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t}\right\|_2$, is large.

5.3 Relation to second-order optimization methods

Symmetry teleportation is closely related to second-order optimization methods. We show that at the target point to teleport to, gradient descent becomes equivalent to Newton's method. Full proofs of the results in this section can be found in Appendix C.3 and C.4.

Lemma 5.5. *If $\partial_{\boldsymbol{v}} \left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}\right\|_2^2 = 0$ for all unit vector $\boldsymbol{v}$ that is orthogonal to $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$, then $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ is an eigenvector of the Hessian of $\mathcal{L}$.*

The following proposition is a direct consequence of Lemma 5.5.

Proposition 5.6. *Let $S_{\mathcal{L}_0} = \{\boldsymbol{w} : \mathcal{L}(\boldsymbol{w}) = \mathcal{L}_0\}$ be a level set of $\mathcal{L}$. If at a particular $\boldsymbol{w} \in S_{\mathcal{L}_0}$ we have $\|\nabla \mathcal{L}(\boldsymbol{w})\|_2 \geq \|\nabla \mathcal{L}(\boldsymbol{w}')\|_2$ for all $\boldsymbol{w}'$ in a small neighborhood of $\boldsymbol{w}$ in $S_{\mathcal{L}_0}$, then the gradient $\nabla \mathcal{L}(\boldsymbol{w})$ has the same direction as the Newton's direction $H^{-1}\nabla \mathcal{L}(\boldsymbol{w})$.*

Proposition 5.6 provides an alternative way to interpret teleportation. Instead of computing the Newton's direction, we search within the loss level set for a point where the gradient has the same direction as Newton's direction. However, symmetry teleportation does not require computing the full Hessian matrix. The second derivative required for optimizing continuous groups are obtained by taking derivatives with respect to parameters and then with respect to the group element, as opposed to taking derivatives with respect to parameters twice. This makes the computation significantly more feasible than Newton's method on neural networks with large number of parameters.

In practice, however, the group action used for teleportation is usually not transitive. Additionally, we do not teleport using the optimal group element since it can be unbounded. We also do not apply teleportation in every gradient descent step. Therefore, Proposition 5.6 serves as an intuition instead of an exact formulation of how teleportation works. We provide empirical evidence in Section 6 that these approximations do not erase the benefits of teleportation completely, and leave theoretical investigations of the connection to second-order methods under approximations as future work.

6 Experiments

6.1 Acceleration through symmetry teleportation

We examine the effect of symmetry teleportation on optimization. We illustrate teleportation in the parameter space on two test functions and show the speedup in regression and classification problems using multilayer neural networks. For test functions, we compared with GD for illustration purposes. For multi-layer neural networks, we also include AdaGrad as a more competitive baseline.

Rosenbrock function. We apply symmetry teleportation to optimize the 2-variable Rosenbrock function (3). The parameters x_1, x_2 are initialized to $(-1, -1)$. Each algorithm is run 1000 steps with learning rate 10^{-3}. We teleport the parameters every 100 steps. The group elements are found by gradient ascent on θ, the parameter for the $\mathrm{SO}(2)$ group, for 10 steps with learning rate 10^{-1}.

The trajectory of parameters and the loss level sets are plotted in Figure 2a,b. The blue star denotes the final position of parameters, the green star denotes the target, and orange dots are the positions from which symmetry transforms start. While gradient descent is not able to reach the target in 1000 steps, teleportation allows large steps and reaches the target much earlier. Figure 2d shows that teleportation improves the norm of gradients in the following step and 2c shows its effect on the loss value. Teleportations clearly reduces the overall convergence time.

Booth function. We also test symmetry teleportation on the Booth function defined in Eqn. (4). We initialize the parameters x_1, x_2 to $(5, -5)$. Each algorithm is run 10 steps with learning rate 0.08. We perform symmetry teleportation on the parameters once, before epoch 5. The group elements are found by gradient ascent on θ for 10 steps with learning rate 0.001. θ is initialized uniformly at random over $[0, \pi)$. Similar to the Rosenbrock function, teleportation moves the parameters to a trajectory with a larger convergence rate (Figure 2 bottom row).

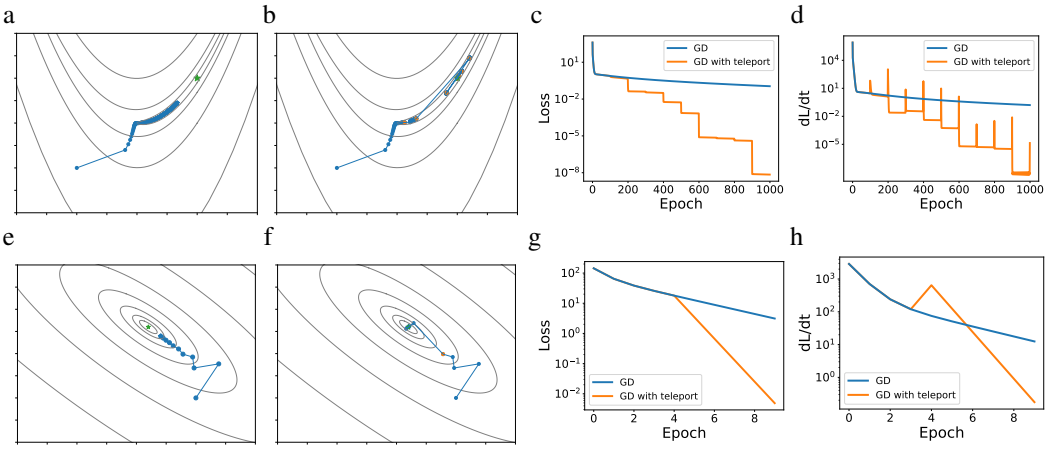

Figure 2: Optimization of the Rosenbrock function (top row) and Booth function (bottom row). using (a) gradient descent and (b) the proposed algorithm. Contours represent the level sets of the loss function. Loss $\mathcal{L}$ and convergence rate $d\mathcal{L}/dt$ are shown in (c) and (d). Teleportation helps move the parameters towards the target.

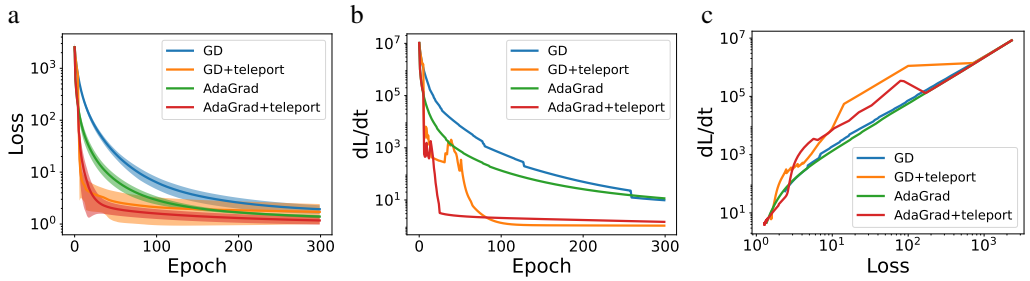

Figure 3: Multi-layer network optimization using gradient descent with and without teleportation. Loss $\mathcal{L}$ and convergence rate $d\mathcal{L}/dt$ are shown in (a) and (b). Teleportation improves the overall convergence rate. The gradient stays large after teleportation.

Multilayer neural network regression. We further evaluated our method on a three-layer neural network with a regression loss $\min_{W_1,W_2,W_3} \|Y - W_3\sigma(W_2\sigma(W_1X))\|_2$. The dimension of weight matrices are $W_3 \in \mathbb{R}^{8\times 7}$, $W_2 \in \mathbb{R}^{7\times 6}$, and $W_1 \in \mathbb{R}^{6\times 5}$. $X \in \mathbb{R}^{5\times 4}$ is the data, $Y \in \mathbb{R}^{8\times 4}$ is the target, and σ is the LeakyReLU activation with slope coefficient 0.1. Discussion of the choice of learning rates and additional hyperparameter details can be found in Appendix D.2.

Teleportation of the $\mathrm{GL}(\mathbb{R})$ group can be performed by finding an element x in its Lie algebra, such that transforming $\boldsymbol{w}$ by the group element $g = \exp(x)$ improves the gradient $\frac{d\mathcal{L}}{dt}$. We use the first order approximation of the exponential map. Between each pair of weight matrices, we replace g_m by $I + T$ and g_m^{-1} by $I - T$ in (8), where $T \in \mathbb{R}^{d_m \times d_m}$ is initialized to 0. Then we perform gradient ascent steps on T with objective defined in Line 3 of Algorithm 1 and update the pair of weights.

Figure 3a shows the training curves, where shaded area denotes one standard deviation from 5 runs. Since GD and AdaGrad use different learning rates, they are not directly comparable. However, the addition of teleportation clearly improves both algorithms. Figure 3b and 3c shows the squared norm of gradient from a single run. Teleportation increases the magnitude of gradient, and the trajectory with teleportation has a larger $d\mathcal{L}/dt$ value at the same loss values, which demonstrates that teleportation finds a better trajectory.

MNIST classification. We apply symmetry teleportation on the MNIST classification task Deng (2012). We split the training set into 48,000 for training and 12,000 for validation. The input data has dimension 28×28 and is flattened into a vector. The output of the neural network has dimension 10 corresponding to the 10 digit classes. We used a three-layer neural network with hidden dimension

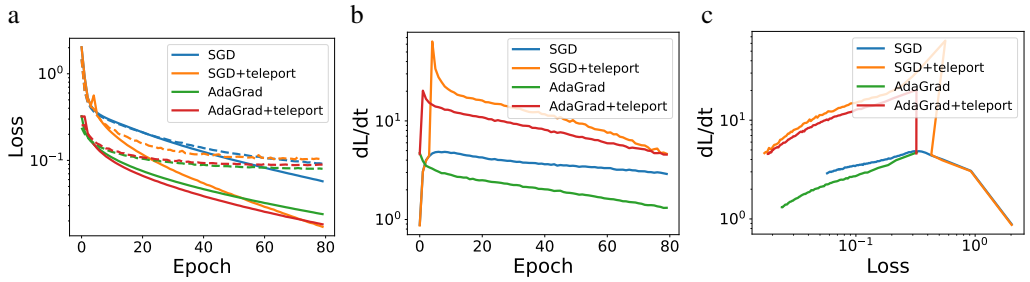

Figure 4: MNIST classification using gradient descent with and without teleportation. Solid lines are training loss and dashed lines are validation loss.

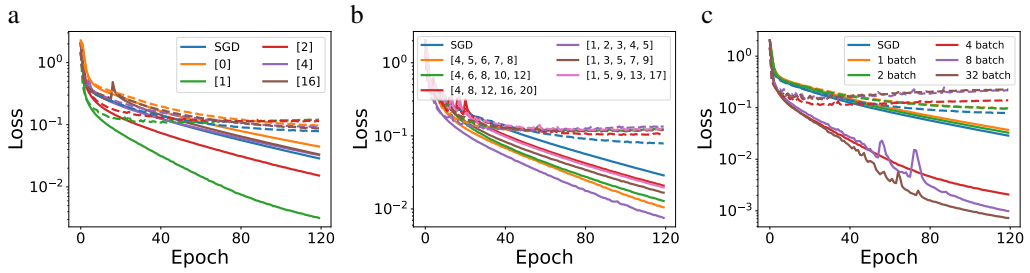

Figure 5: Teleportation (a) once at different epoch, (b) 5 times with different teleportation schedules, and (c) using different number of mini-batches. The lists in the legend of (a) and (c) denote the epoch numbers in teleportation schedule where teleportation happens.

$[512, 512]$, LeakyReLU activations, and cross-entropy loss. Learning rate is 2×10^{-3}, and learning rate for teleportation is 10^{-3}. Each optimization algorithm is run 80 epochs with batch size of 20. Immediately after the first epoch, we apply teleportation using data from one mini-batch, and repeat for 4 different mini-batches. For each mini-batch, 10 gradient ascent steps are used to optimize g_m.

Figure 4a shows the effect of teleportation on training and validation loss, and Figure 4b,c shows the norm of the gradient in training. While teleportation significantly accelerates the decrease of the training loss, its effect on the validation loss is limited for SGD and detrimental for AdaGrad. Therefore, teleportation on MNIST makes training faster at the beginning but leads to earlier overfit and slightly worse validation accuracy. A possible reason is that regions with large gradients have sharp minimum that do not generalize well.

6.2 Teleportation schedule

The effect of teleportation varies depending on its time and frequency. Figure 5 shows the result of teleportation on MNIST with different hyperparameters. In all experiments, we use SGD with batch size 20, learning rate 2×10^{-3}, and 10 gradient ascent steps for each teleportation.

In Figure 5a, we randomly select 4 different nimi-batches and apply teleportation on each of them individually, but at different epochs. Teleportation before training has the worst performance. After epoch 0, the effect of teleportation is stronger when it is applied earlier. In Figure 5b, we again apply teleportations on 4 randomly selected mini-batches, but repeat this with 5 different teleportation schedules (hyperparameter K in Algorithm 2) as shown in the legend. All schedules has the same number of teleportations. Smaller intervals between teleportation accelerate convergence more significantly. In Figure 5c, we apply teleportation immediately after the first epoch, but use different numbers of mini-batches (hyperparameter B in Algorithm 2) and teleport using each of them individually. Using more mini-batches to teleport leads to faster decrease in training loss but is also more prone to overfitting.

6.3 Runtime analysis

The additional amount of computation introduced by teleportation depends on the implementation of Line 2-5 of Algorithm 1. We discuss our implementation for teleporting multi-layer neural networks

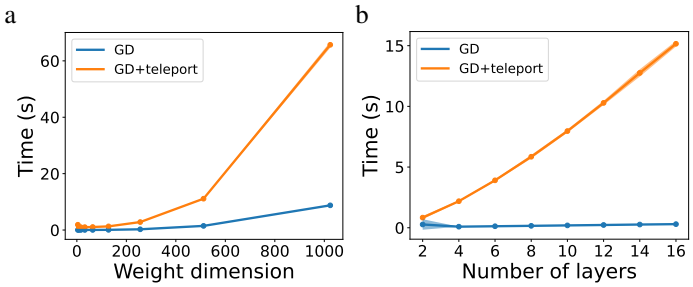

Figure 6: Gradient descent training time on a Leaky-ReLU neural network for 300 epochs, with a 10-step teleportation every 10 epochs. (a) 3-layer network using square weights and data matrices, with different dimensions. (b) Weights and data are all 128 by 128 matrices, but the network has different number of layers.

as an example. Teleportating each pair of adjacent weight matrices requires computing the inverse of the output from the previous layer. Denote the batch size as n, the largest dimension of all layers as d_{max}, and the number of layers as l. Assume that $d_{max} > n$. Computing the pseudoinverse of the output of each layer has complexity $O(d_{max}^2 n)$, and computing the pseudoinverse for all layers has complexity $O(d_{max}^2 nl)$. Note that all matrices we invert has dimensions at most $d_{max} \times n$.

For one gradient ascent step on g, the forward and backward pass both have complexity $O(d_{max}^2 nl)$. This is the same as the forward and backward pass of gradient descents on $\boldsymbol{w}$ because the architecture is the same except with approximately twice as many layers. We perform t gradient ascent steps on g. Therefore, the computation cost for one teleportation is $O(d_{max}^2 nlt)$.

We show empirically that the runtime for teleportation scales polynomially with matrix dimensions and linearly with the number of layers. We record the runtime of gradient descent on a Leaky-ReLU neural network for 300 epochs, with a 10-step teleportation every 10 epochs. Each pair of adjacent weight matrices is transformed by a group action during teleportation. Figure 6(a) shows the runtime for a 3-layer network using square weights and data matrices with different dimensions. Figure 6(b) shows the runtime for 128-by-128 weight and data matrices with different number of layers.

Although the teleportation step has the same complexity as a gradient descent step, the runtime is dominated by teleportation due to larger constants in the complexity analysis. The trade-off between teleportation time and convergence rate depends on specific problems. In our experiments, the convergence rate is improved by a small number of teleportation steps which does not add significant computational overhead.

7 Discussion and Conclusions

We proposed an accelerated optimization method that exploits the symmetry in the parameter space of the loss function to improve convergence. It is interesting to note that optimizing $d\mathcal{L}/dt$ locally sometimes leads to an improved global path. One example is the ellipse function (Figure 1), where teleporting once ensures that $d\mathcal{L}/dt$ is optimal at every $\mathcal{L}$ value along the trajectory. Another example is the matrix factorization problem $\mathcal{L}(U, V) = \|Y - UV\|_2^2$. Tarmoun et al. (2021) shows that the convergence rate increases with the imbalance $U^T U - V^T V$. Consider the transformation $U, V \to Ug, g^{-1}V$. To optimize $d\mathcal{L}(U, V)/dt$ locally, we would need a large $g \in GL_n$, but a large g also leads to a large $U^T U - V^T V$ which is positively correlated with the overall convergence rate of the entire trajectory. Hence teleportation is guaranteed to produce a better trajectory.

A potential future direction is to derive the exact expression for how teleportation affects the loss value at a later time in gradient flow, which may lead to a closed-form solution of the optimal teleportation destination. Additionally, inspired by the landscape view of parameter space symmetry Şimşek et al. (2021), teleportation using discrete (permutation) symmetries may allow us to reach a better minima. Finally, additional theory can be developed to explain the relationship between teleportation and second-order methods under the approximations we introduced, especially to quantify the improvement on the overall convergence rate, and to derive its effect on generalization bounds. Integrating teleportation with other advanced optimizers such as Adam, RMSprop would be another interesting future step.

References

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Marco Armenta and Pierre-Marc Jodoin. The representation theory of neural networks. *Mathematics*, 9(24), 2021. ISSN 2227-7390.

Marco Armenta, Thierry Judge, Nathan Painchaud, Youssef Skandarani, Carl Lemaire, Gabriel Gibeau Sanchez, Philippe Spino, and Pierre-Marc Jodoin. Neural teleportation. *arXiv preprint arXiv:2012.01118*, 2020.

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*, pages 244–253. PMLR, 2018.

Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

Vijay Badrinarayanan, Bamdev Mishra, and Roberto Cipolla. Symmetry-invariant optimization in deep networks. *arXiv preprint arXiv:1511.01754*, 2015.

Robert Bamler and Stephan Mandt. Improving optimization for models with continuous symmetry breaking. In *International Conference on Machine Learning*, pages 423–432. PMLR, 2018.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Simon S Du, Wei Hu, and Jason D Lee. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. *Neural Information Processing Systems*, 2018.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Iordan Ganev and Robin Walters. The QR decomposition for radial neural networks. *arXiv preprint arXiv:2107.02550*, 2021.

Grzegorz Głuch and Rüdiger Urbanke. Noether: The more things change, the more stay the same. *arXiv preprint arXiv:2104.05508*, 2021.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *35th International Conference on Machine Learning*, 2018.

Elad Hazan. Lecture notes: Optimization for machine learning. *arXiv preprint arXiv:1909.03550*, 2019.

Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2): 150–194, 2013.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Daniel Kunin, Javier Sagastuy-Brena, Surya Ganguli, Daniel LK Yamins, and Hidenori Tanaka. Neural mechanics: Symmetry and broken conservation laws in deep learning dynamics. In *International Conference on Learning Representations*, 2021.

James Martens and Roger B Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on International Conference on Machine Learning*. PMLR, 2018.

Qi Meng, Shuxin Zheng, Huishuai Zhang, Wei Chen, Zhi-Ming Ma, and Tie-Yan Liu. $\mathcal{G}$-SGD: Optimizing relu neural networks in its positively scale-invariant space. *International Conference on Learning Representations*, 2019.

Hancheng Min, Salma Tarmoun, René Vidal, and Enrique Mallada. On the explicit role of initialization on the convergence and implicit bias of overparametrized linear networks. In *International Conference on Machine Learning*. PMLR, 2021.

Behnam Neyshabur, Russ R Salakhutdinov, and Nati Srebro. Path-SGD: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, 2015.

HoHo Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120v3*, 2014.

Berfin Şimşek, François Ged, Arthur Jacot, Francesco Spadaro, Clément Hongler, Wulfram Gerstner, and Johanni Brea. Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. In *International Conference on Machine Learning*, pages 9722–9732. PMLR, 2021.

Salma Tarmoun, Guilherme Franca, Benjamin D Haeffele, and Rene Vidal. Understanding the dynamics of gradient flow in overparameterized linear models. In *International Conference on Machine Learning*, pages 10153–10161. PMLR, 2021.

Twan Van Laarhoven. L2 regularization versus batch and weight normalization. *Advances in Neural Information Processing Systems*, 2017.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]

(b) Did you include complete proofs of all theoretical results? [Yes]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Adaptations of Algorithm 1 for different problems

A.1 Stochastic gradient descent

We extend Algorithm 1 to stochastic gradient descent (SGD). We apply the group actions using data from a mini-batch X_i, and repeat for B mini-batches each time. The gradient we optimize, $\tilde{\nabla}\mathcal{L}(X_i, g \cdot \boldsymbol{w}_t)$, also uses single mini-batches. Algorithm 2 provides the framework for teleportation in SGD.

Algorithm 2: Symmetry Teleportation (SGD)

Input: Loss function $\mathcal{L}(\boldsymbol{w})$, learning rate η, number of epochs T, initialized parameters $\boldsymbol{w}_0$, symmetry group G, teleportation schedule K, number of mini-batches used to teleport B.
Output: $\boldsymbol{w}_T$.

```
for t ← 0 to T − 1 do
    if t ∈ K then
        for X_i in the first B mini-batches do
            g ← argmax_{g∈G} ‖∇̃L(X_i, g · w_t)‖^2
            w_t ← g · w_t
            w_t ← w_t − η∇̃L(X_i, w_t)
        end for
        for X_i in the rest mini-batches do
            w_t ← w_t − η∇̃L(X_i, w_t)
        end for
    else
        for all mini-batches X_i do
            w_t ← w_t − η∇̃L(X_i, w_t)
        end for
    end if
    w_{t+1} ← w_t
end for
return w_T
```

A.2 Data transformation

Algorithm 3 here modifies Algorithm 1 to allow transformations on both parameters and data. Denote g_X as the group action on data only. The group actions on data at all teleportation steps can be precomposed as a function f and applied to the input data at inference time.

Algorithm 3: Symmetry Teleportation (with data transformation)

Input: Loss function $\mathcal{L}(\boldsymbol{w}, X)$, learning rate η, number of epochs T, initialized parameters $\boldsymbol{w}_0$, symmetry G, teleportation schedule K, data X.
Output: $\boldsymbol{w}_T$, data transformation f.
Initialize: $f =$ the identity function.

```
for t ← 0 to T − 1 do
    if t ∈ K then
        g ← argmax_{g∈G} ‖∇_{g·w_t} L(g · (w_t, X))‖^2
        w_t, X ← g · (w_t, X)
        f ← g_X ∘ f
    end if
    w_{t+1} ← w_t − η∇_{w_t} L
end for
return w_T, f
```

B Group actions

B.1 Continuous symmetry in test functions

B.1.1 Ellipse

Consider the following loss function with $a \in \mathbb{R}^{\geq 0}$:

$$\mathcal{L}(x_1, x_2) = x_1^2 + a x_2^2 \tag{15}$$

If we change the variables to $\mathcal{L}(u(x_1, x_2), v(x_1, x_2)) = u^2 + v^2$, 2D rotations leave $\mathcal{L}$ unchanged. Therefore $SO(2)$ is a symmetry of $\mathcal{L}(x_1, x_2)$. Let $g_\theta \in SO(2)$, and define the group action as

$$g_\theta \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1/\sqrt{a} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{a} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{16}$$

Then

$$\mathcal{L}(x_1, x_2) = \mathcal{L}(g \cdot (x_1, x_2)) \tag{17}$$

B.1.2 Rosenbrock function

Consider the Rosenbrock function with 2 variables Rosenbrock (1960):

$$\mathcal{L}(x_1, x_2) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 \tag{18}$$

Let $u = 10(x_1^2 - x_2)$ and $v = x_1 - 1$. After changing the variables from x and y to u and v, $\mathcal{L}$ has a rotational symmetry. Note that the function, $h : \mathbb{R}^2 \to \mathbb{R}^2$, that maps x_1, x_2 to u, v is bijective:

$$\begin{aligned} (u, v) &= h(x_1, x_2) = (10(x_1^2 - x_2), x_1 - 1) \\ (x_1, x_2) &= h^{-1}(u, v) = (v + 1, (v + 1)^2 - 0.1u) \\ h(x_1, x_2) &= h(y_1, y_2) \Rightarrow (x_1, x_2) = (y_1, y_2) \end{aligned} \tag{19}$$

Next, we show that $SO(2)$ is a symmetry of $\mathcal{L}(x_1, x_2)$. Let ρ be a representation of $SO(2)$ acting on $\mathbb{R}^2$. For $g \in SO(2)$, define the following group action:

$$g \cdot (x_1, x_2) = h^{-1}\left(\rho(g) h(x_1, x_2)\right) \tag{20}$$

Then

$$\mathcal{L}(x_1, x_2) = \mathcal{L}(g \cdot (x_1, x_2)) \tag{21}$$

For the Rosenbrock function with 2N parameters, we can construct a bijective function $h : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ by transforming each of the N pairs of variables as before, and $SO(2N)$ is a symmetry of $\mathcal{L}(x_1, ..., x_{2N})$. However, we will only use the 2 variable version in the experiments.

B.1.3 Booth function

Consider the Booth function Jamil and Yang (2013):

$$\mathcal{L}(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

Similar to the Rosebrock function, a change of variables reveals a rotational symmetry of $\mathcal{L}$:

$$\begin{aligned} (u, v) &= h(x_1, x_2) = (x_1 + 2x_2 - 7, 2x_1 + x_2 - 5) \\ (x_1, x_2) &= h^{-1}(u, v) = (-\frac{1}{3}u + \frac{2}{3}v + 1, \frac{2}{3}u - \frac{1}{3}v + 3) \end{aligned} \tag{22}$$

The function $h : \mathbb{R}^2 \to \mathbb{R}^2$ that maps x_1, x_2 to u, v is bijective. Let ρ be a representation of $SO(2)$ acting on $\mathbb{R}^2$. For $g \in SO(2)$, define the following group action:

$$g \cdot (x_1, x_2) = h^{-1}\left(\rho(g) h(x_1, x_2)\right) \tag{23}$$

Then $\mathcal{L}(x_1, x_2)$ admits an $SO(2)$ symmetry:

$$\mathcal{L}(x_1, x_2) = \mathcal{L}(g \cdot (x_1, x_2)) \tag{24}$$

B.2 Proof of Proposition 4.2 and Proposition 4.3

Proposition 4.2

Proof. In the linear network $h_m = W_m h_{m-1}$, hence

$$g \cdot (W_m, h_{m-1}) = (W_m g^{-1}, g h_{m-1}), \quad g \cdot h_m = W_m g^{-1} g h_{m-1} = h_m \tag{25}$$

which means a p-layer linear network is invariant under all G_m with $m \leq p$ as they keep the output h_p invariant ($\forall g \in G_m$, $g \cdot h_p = h_p$). □

Proposition 4.3

Proof. From (8), we want to convert $g \cdot h_{m-1}$ into a transformation on W_{m-1} instead of h_{m-1}. In other words, we want to find a set of transformed weights W'_m, W'_{m-1} which yields the same network output $\tilde{h}_m$:

$$\begin{aligned} \tilde{h}_m &= W'_m \sigma\left(W'_{m-1} h_{m-2}\right) = W_m g^{-1} g \sigma\left(W_{m-1} h_{m-2}\right) \\ \Rightarrow W'_m &= W_m g^{-1}, \qquad \sigma\left(W'_{m-1} h_{m-2}\right) = g \sigma\left(W_{m-1} h_{m-2}\right) \end{aligned} \tag{26}$$

To obtain $W'_{m-1} = g \cdot W_{m-1}$ we first find a right pseudo-inverse $h^+_{m-2} \in \mathbb{R}^{n \times d_{m-2}}$ for h_{m-2} meaning $h^+_{m-2} h_{m-2} = I$ (assuming $n \ll d_{m-2}$ and that h_{m-2} is full rank). Solving (26) we get

$$W'_{m-1} = \sigma^{-1}\left(g \sigma\left(W_{m-1} h_{m-2}\right)\right) h^+_{m-2} \tag{27}$$

(8) follows from (26) and (27). □

C Theoretical analysis of teleportation

C.1 Proof of Proposition 5.1 and Corollary 5.2

Proposition 5.1

Proof. Let $\boldsymbol{w}' = g \cdot \boldsymbol{w}$. Denote the Jacobian as J, where $J_{ij} = \partial w'_i / \partial w_j$. Then the inverse of J has entries $J^{-1}_{ij} = \partial w_i / \partial w'_j$.

The gradient at $\boldsymbol{w}'$ is

$$\frac{\partial \mathcal{L}(\boldsymbol{w}')}{\partial \boldsymbol{w}'} = \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}'} = \sum_j \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}_j} \frac{\partial \boldsymbol{w}_j}{\partial \boldsymbol{w}'_i} = \sum_j \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}_j} J^{-1}_{ji} = \left(\left(\frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} \right)^T J^{-1} \right)^T = (J^{-1})^T \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} \tag{28}$$

The rate of change of $\mathcal{L}$ in gradient flow is

$$\frac{d\mathcal{L}(\boldsymbol{w}')}{dt} = \left\langle \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}'}, \frac{d\boldsymbol{w}'}{dt} \right\rangle = - \left\| (J^{-1})^T \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} \right\|^2_\eta \tag{29}$$

Thus we will have a speedup if

$$\left\| (J^{-1})^T \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} \right\|^2_\eta > \left\| \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} \right\|^2_\eta \tag{30}$$

□

Corollary 5.2

Proof. Since $J = g$, using (9) and the l.h.s. of (10) we have

$$\frac{d\mathcal{L}(g \cdot \boldsymbol{w})}{dt} = -\nabla \mathcal{L}^T g^{-1} \eta \left[g^{-1}\right]^T \nabla \mathcal{L} = \nabla \mathcal{L}^T \left[g^T \eta^{-1} g\right]^{-1} \nabla \mathcal{L} = \|\nabla \mathcal{L}\|^2_\eta \tag{31}$$

□

C.2 Proof of Proposition 5.4

Proof. From the definition of Lipschitz continuity and the update rule of gradient descent,

$$\left| \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_{t+1}} \right\|_2 - \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right\|_2 \right| \leq L \| \boldsymbol{w}_{t+1} - \boldsymbol{w}_t \|_2 = L \left\| \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right\|_2 \tag{32}$$

Equivalently,

$$(1 - \eta L) \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right\|_2 \leq \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_{t+1}} \right\|_2 \leq (1 + \eta L) \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right\|_2 \tag{33}$$

By unrolling T steps, we have

$$(1 - \eta L)^T \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right\|_2 \leq \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_{t+T}} \right\|_2 \leq (1 + \eta L)^T \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right\|_2 \tag{34}$$

Similarly, for a teleported point $\boldsymbol{w}'_t = g \cdot \boldsymbol{w}_t$,

$$(1 - \eta L)^T \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}'_t} \right\|_2 \leq \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}'_{t+T}} \right\|_2 \leq (1 + \eta L)^T \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}'_t} \right\|_2 \tag{35}$$

Therefore, if

$$(1 - \eta L)^T \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}'_t} \right\|_2 \geq (1 + \eta L)^T \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right\|_2 \tag{36}$$

then it is guaranteed that

$$\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}'_{t+T}} \right\|_2 \geq \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_{t+T}} \right\|_2 \tag{37}$$

□

C.3 Proof of Lemma 5.5

Proof. From the definition of the directional derivative,

$$\partial_{\boldsymbol{v}} \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right\|_2^2 = \boldsymbol{v} \cdot \frac{\partial}{\partial \boldsymbol{w}} \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right\|_2^2 \tag{38}$$

Writing the last term in indices,

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{w}_i} \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right\|_2^2 &= \frac{\partial}{\partial \boldsymbol{w}_i} \sum_j \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_j} \right)^2 \\ &= \sum_j \frac{\partial}{\partial \boldsymbol{w}_i} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_j} \right)^2 \\ &= \sum_j 2 \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_j} \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{w}_i \partial \boldsymbol{w}_j} \\ &= 2 \left(H \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right)_i \end{aligned} \tag{39}$$

Removing the indices,

$$\frac{\partial}{\partial \boldsymbol{w}} \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right\|_2^2 = 2H \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \tag{40}$$

Substitute back and we have

$$\partial_{\boldsymbol{v}} \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right\|_2^2 = \boldsymbol{v} \cdot \left(2H \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right) \tag{41}$$

Since $\partial_{\boldsymbol{v}} \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right\|_2^2 = 0$ for all vector $\boldsymbol{v}$ that is orthogonal to $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$, $\boldsymbol{v} \cdot \left(2H \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \right) = 0$ for all vector $\boldsymbol{v}$ that is orthogonal to $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$. In other words, $2H \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ is orthogonal to all vectors that are orthogonal to $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$. Therefore, $2H \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ has the same direction of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$, and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ is an eigenvector of the Hessian of $\mathcal{L}$. □

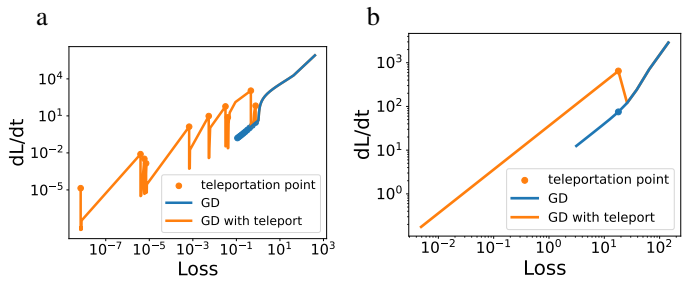

Figure 7: Gradient on the trajectory of optimizing the Rosenbrock function (left) and Booth function (right). At the same loss value, the graident is larger on the trajectory with teleportation, indicating a better descent path.

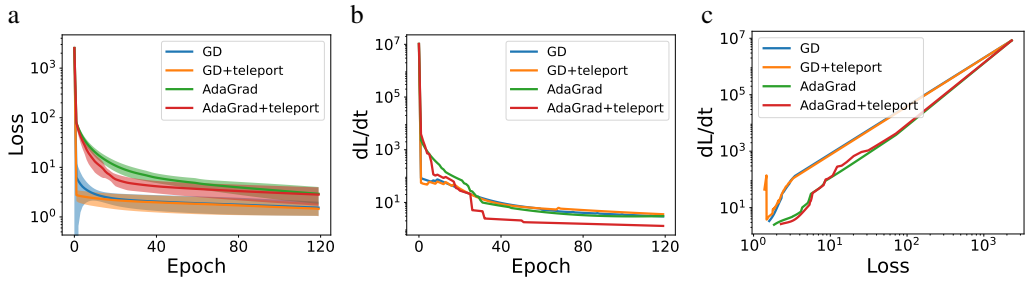

Figure 8: Multi-layer network optimization with a different learning rate. Teleportation improves both gradient descent and AdaGrad, similar to using the original learning rate.

C.4 Proof of Proposition 5.6

Proof. From Lemma 5.5, $\frac{dL}{dw}$ is an eigenvector of H. Therefore, it is also an eigenvector of H^{-1}. Hence $\frac{dL}{dw}$ has the same direction as $H^{-1}\frac{dL}{dw}$. □

D Experiment details and additional results

D.1 Test functions

We compare the gradient at different loss values for gradient descent with and without teleportation. Figure 7 shows that the trajectory with teleportation has a larger $d\mathcal{L}/dt$ value than the trajectory without teleportation at the same loss values. Therefore, the rate of change in the loss is larger in the trajectory with teleportation, which makes it favorable.

D.2 Multilayer neural network

Data X, Y and initialization of parameters W are set uniformly at random over $[0, 1]$. The group elements used for these transforms are found by gradient ascent on T for 10 times, with learning rate 10^{-7} in GD and 10^{-6} in AdaGrad. When using teleportation, we perform symmetry transform on the parameters once every 10 epochs in the first 50 epochs. Each algorithm is run 300 steps. GD uses learning rate 10^{-4} and AdaGrad uses 10^{-1}.

To confirm that teleportation works with different learning rates, we repeat the experiment using learning rate 10^{-3} for GD and 10^{0} for AdaGrad, which is 10 times higher than the learning rate used in the main paper. Increasing learning rates further prevents convergence. Since gradient descent is expected to converge faster with higher learning rates, we adjust the teleportation schedule to once every 4 epochs for the first 20 epochs and run each algorithm 120 steps. Fig. 8 shows the results for the new learning rates. Teleportation improves both GD and AdaGrad in the similar way as using the original learning rates.