Neural Lifting

Abstract

In recent years, machine learning models have grown rapidly in size, with highly overparameterized architectures becoming the norm. While large models dominate in terms of their performance on a range of tasks, their high resource demands make training impractical in many settings. This, along with the advantages of interpretability, faster inference, and easier deployment, underscores the continued relevance of small models. As we argue, empirical results in model compression and recent advances in learning theory suggest that substantial reductions in model size can be achieved while maintaining strong test-time performance. However, we also argue that improving the performance of small models may require moving beyond the currently-dominant paradigm of stochastic local search techniques. We introduce *neural lifting*, a novel training methodology that enables small models to benefit from a temporary form of mild overparameterization during training. We demonstrate the efficacy of neural lifting on benchmark and synthetic datasets, obtaining notable improvements over standard optimization methods.

1 INTRODUCTION

With the emergence of foundation models, much of the effort in recent machine learning research has been directed towards developing large, highly-overparameterized deep neural networks [Bommasani et al., 2021]. While the prevailing view favors ever-larger models, this perspective is not without critique. An opposing standpoint raises concerns about unsustainable compute and memory demands, neglect of important applications in fields like health and climate science, and the concentration of power in AI development [Varoquaux et al., 2024, Chen and Varoquaux, 2024].

These drawbacks have prompted research into achieving similar test-time performance with smaller models. In particular, research on model compression techniques (such as pruning, distillation, and quantization [Frankle and Carbin, 2018, Li et al., 2020, Hsieh et al., 2023, Zhu et al., 2024]) has demonstrated the following phenomenon.

Phenomenon 1 (Expressibility)
The solutions learned when training highly-overparameterized models can often be well-approximated by much smaller models.

Although model compression is an important means of reducing model size, these approaches require first training a large model. Thus, they provide only inference-time savings. In this paper, we are instead focused on *directly* training smaller models on task-specific losses, without assuming access to a larger model. Hence, our aims are complementary to those of model compression, and the two directions may be able to benefit from each other's insights.

Although quite important in its own right, Phenomenon 1 does not guarantee that there is a computationally tractable approach to *find* a good-performing small model without first training a large model. Indeed, stochastic gradient-based (SG-based) local search techniques, such as SGD and Adam, do not generally find the same solutions for small models as those found when compressing a large model. In recent years, the field of ML has developed a better understanding of this phenomenon:

Phenomenon 2 (Optimizability)
The optimization landscapes of highly-overparameterized models are highly amenable to stochastic gradient-based local search, whereas the optimization landscapes of small models are not.

Taking into account both the expressibility and optimizability phenomena, we are led to consider new optimization techniques and to put forth the following hypothesis.

Submitted to the 41st Conference on Uncertainty in Artificial Intelligence (UAI 2025). **To be used for reviewing only**.

Small model optimization (SMO) hypothesis:
Using novel training methods, one can achieve good test-time performance for a small model. Furthermore, such training methods are also necessary *for achieving such performance.*

The SMO hypothesis emphasizes that existing training methods may be insufficient to achieve our aims. In particular, the optimization landscapes of small models may exhibit a significant number of spurious local minima, in which case SG-based local search techniques are likely to get stuck.

Inspired by Phenomenon 2 and classical optimization techniques, we propose *neural lifting* to temporarily leverage the advantages of overparameterization when training a small model. During training, when a small model gets stuck at a local minimum, we temporarily *lift* the model into a moderately higher-dimensional space, then *project* back to a new point in the original (lower-dimensional) space. Viewed from the lower-dimensional space, this procedure "jumps" to a new point, allowing the optimization to converge to a potentially different (and hopefully, better) local optimum.

The remainder of the paper is organized as follows:

- In Section 2, we review the field's current understanding of the expressibility and optimizability phenomena.
- In Section 3, we introduce neural lifting as a general approach to small-model optimization.
- In Section 4, we elaborate upon the intuition behind neural lifting from the perspective of function spaces.
- In Section 5, we demonstrate the efficacy of our approach in two carefully-designed settings.
- In Section 6, we discuss future directions for neural lifting and for small-model optimization in general.

2 THE SMO HYPOTHESIS

In proposing the Small Model Optimization (SMO) hypothesis, we pointed to two key observations, which we called the *expressibility* and *optimizability* phenomena. Both phenomena have been under intense study in recent years, and the field's understanding of these topics is still evolving.

In this section, we do not seek to give a comprehensive review of these topics. Rather, we seek to demonstrate that these observations are well-supported by the field's current understanding, and to lay out some desiderata for approaches to small-model optimization.

2.1 THE EXPRESSIBILITY PHENOMENON

Two fields provide concordant support for the expressibility phenomenon: first, research on model compression provide strong empirical support, and second, results from learning theory provide strong theoretical support.

Empirical support from model compression In model compression, one begins with a pre-trained large (often highly overparameterized) model and aims to compress it into a smaller model, without a significant decrease in test-time performance [Li et al., 2020]. This can be achieved in various ways. For example, in *pruning*-based approaches, model size is decreased by setting a significant number of parameters to zero [Frankle and Carbin, 2018, Sun et al., 2024], and in *quantization*-based approaches, model size is reduced by using lower-precision data types to store some or all model parameters [Yao et al., 2022, Han et al., 2016]. In both of these approaches, the core model architecture (width and depth) is retained. Meanwhile, in *distillation*-based approaches, a large model is used to train a smaller model, which is generally shallower and/or less wide [Hinton et al., 2014, Hsieh et al., 2023, Gu et al., 2024].

Here, we are not directly interested in the specific details of these approaches. Rather, we are focused on a lesson taught to us by their success: the functions implemented by large models are often (approximately) *expressible* by a much smaller model, e.g. in computer vision, these approaches have reduced models from 552MB of memory to 11.3MB with no significant change in accuracy [Han et al., 2016].

Theoretical support from learning theory From another angle, the expressibility phenomenon helps to explain the generalization capabilities of highly overparameterized models. Based on intuitions from traditional statistical learning theory, it is surprising that such models achieve small generalization error without any explicit regularization [Zhang et al., 2021]. In recent years, it has become increasingly clear that a key factor in explaining generalization is *implicit* regularization, i.e., the bias towards simple, generalizable solutions that emerges from the interactions between more basic aspects of the learning algorithm. One of the most well-studied such interactions is the one between overparameterization and SG-based local search techniques [Poggio et al., 2017, Dziugaite and Roy, 2017, Arora et al., 2019, Huh et al., 2023]. In particular, the solutions learned when training highly overparameterized models exhibit a *simplicity bias*, often implying that these solutions can be expressed by much smaller models [Arora et al., Lotfi et al., 2022]. Thus, both empirically and theoretically, Phenomenon 1 appears to be robust, leading to a natural question: *"Do we really need model compression?"* [Gordon, 2020].

2.2 THE OPTIMIZABILITY PHENOMENON

The expressibility phenomenon suggests the existence of what we will refer to as *high-quality solutions* in small models, i.e., solutions with good test-time performance. To put the next point succinctly, the existence of such solutions is **necessary, but not sufficient** for practical purposes: we must also be able to efficiently find these solutions.

Table 1: **Possible approaches to small-model optimization.** As discussed in Section 2.3, our method is designed to be *general-purpose* across a range of architectures, to be *resource-efficient*, and to offer a *size-based reduction* of overparameterization; existing approaches do not satisfy all three of these desiderata.

	General Purpose	Resource Efficient	Size-based Reduction	Example Approaches
Neural teleportation	✗	✓	✓	Armenta et al. [2023], Zhao et al. [2022]
Brute force	✓	✗	✓	Picard [2021]
Sparse training	✓	✓	✗	Dettmers and Zettlemoyer [2019], Mostafa and Wang [2019] Evci et al. [2020]
Quantized training	✓	✓	✗	Wang et al. [2023], Hao et al. [2024]
Neural lifting	✓	✓	✓	This paper

It has become increasingly well-recognized that the success of large models results not just from their expressiveness, but also from their *optimizability*. Although large models may have optimization landscapes which are highly nonconvex, this nonconvexity takes a relatively benign form: the optimization landscape often contains few if any spurious local minima, both theoretically and empirically [Soltanolkotabi et al., 2018, Du et al., 2018, Kawaguchi, 2016, Draxler et al., 2018]. Relatedly, in overparameterized models, there is a high probability of initializing at a point from which there is a descent path to a global optimum [Safran and Shamir, 2016, Shin and Karniadakis, 2020].

Conversely, Shin and Karniadakis [2020] suggest that overparameterization is actually *necessary* for SG-based local search to converge to a global optimum, and recent works such as Jentzen and Riekert [2024] have begun to formalize such necessary conditions in the form of non-convergence results for SGD and Adam. Such results suggest two paths forward. We might consider only training overparameterized models, effectively giving up on the prospect of training small models from scratch. Alternatively, we may work to develop new methods explicitly tailored toward training small models and informed by the recent progress discussed here. In this work, we advocate for the latter option.

2.3 DESIDERATA FOR NEW APPROACHES

Since the design space of new training approaches is vast, we outline four desiderata to guide the development of new approaches, noting that the relative priority of each point will vary according to application.

Desideratum 1: General-purpose effectiveness. We desire an approach which is applicable and effective across a range of architectures. Hence, we do not consider approaches which require manual, architecture-specific derivations, e.g. neural teleportation, which uses architecture-specific parameter symmetries [Armenta et al., 2023].

Desideratum 2: Resource efficiency. We desire an approach which is memory-efficient and computationally efficient. Hence, we do not consider brute-force approaches like large random seed search [Picard, 2021].

Desideratum 3: Size-based reduction. We desire an approach which reduces overparameterization by reducing the "most basic" aspects of model size: width and depth. Hence, we do not consider sparse training or quantized training methods [Dettmers and Zettlemoyer, 2019, Hao et al., 2024], which reduce overparameterization and memory footprint by taking advantage of different dimensions.

Our three desiderata are summarized in Table 1. In short, we desire methods which are "plug-and-play", i.e., methods which can easily be integrated into existing pipelines and which are directly interoperable with other approaches; our method satisfies all of these criteria.

3 NEURAL LIFTING

Here, we focus on a standard supervised learning task with input space $\mathcal{X}$ and output space $\mathcal{Y}$, though our method is directly applicable to other settings. We view a deep learning model as divided into distinct modules. We refer to the model to be trained as the `baseModel`, e.g. a deep neural network with convolutional layers followed by fully connected layers. This base model can be decomposed into a `trunk`: $\mathcal{X} \to \mathcal{Z}$, which maps the input to an intermediate latent representation in some space $\mathcal{Z}$, and a `head`: $\mathcal{Z} \to \mathcal{Y}$, which maps the latent representation to the output. We do not restrict the head to comprise only the final layer of the network; this division can be made at any intermediate layer. When expressed as function composition, a forward pass through the base model can be represented as `baseModel` $=$ `head` $\circ$ `trunk`, as in Algorithm 1.

Our method performs several iterations of three phases. In **Phase 1**, we simply train the `baseModel` to minimize a task-specific loss $\mathcal{L}_{\text{task}}$ using a standard optimizer, until we meet some stopping criterion, e.g. some number of batches with no improvement to the loss, as described in

Algorithm 1 VANILLA NEURAL LIFTING

Input: Modules and losses:

1. Modules `head`, `trunk`, and `lifter`
2. Task-specific loss $\mathcal{L}_{\text{task}}$
3. Matching loss $\mathcal{L}_{\text{match}}$

Set `baseModel` $\leftarrow$ `head` $\circ$ `trunk` and initialize
while Not converged **do**
*# **Phase 1:** Standard training for base model*
    Optimize $\mathcal{L}_{\text{task}}$(`baseModel`) until stopping criterion is met
*# **Phase 2:** Lifted training*
    Initialize `lifter` near identity
    Set `liftedModel` $\leftarrow$ `head` $\circ$ `lifter` $\circ$ `trunk`
    Optimize $\mathcal{L}_{\text{task}}$(`liftedModel`) until stopping criterion is met
*# **Phase 3**: Projection*
    Let $\mathcal{D}_{\text{distill}} \leftarrow \{(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^{n}$, where $\mathbf{z}^{(i)} = (\texttt{lifter} \circ \texttt{trunk})(\mathbf{x}^{(i)})$
    Set `baseModel` $\leftarrow$ `head` $\circ$ `trunk`
    Optimize $\mathcal{L}_{\text{match}}$(`baseModel`, $\mathcal{D}_{\text{distill}}$) until stopping criterion is met
Return `baseModel`

Algorithm 1.

3.1 LIFTING AND PROJECTION

In **Phase 2**, we insert a `lifter` module between the head and trunk of the current `baseModel` to obtain a lifted model, and then train in this lifted parameter space. First, in Algorithm 1, we initialize the `lifter` module at or near the identity function.[1] Then, in Algorithm 1, the `lifter` module is inserted between the head and the trunk to form the `liftedModel`, which is trained using the same loss function $\mathcal{L}_{\text{task}}$, as described in Algorithm 1.

In **Phase 3**, we want to project from the parameter space of the lifted model down to the parameter space of the base model. To carry out this projection, we borrow ideas from knowledge distillation. In Algorithm 1, we create a *distillation dataset* $\mathcal{D}_{\text{distill}}$, which pairs each input $\mathbf{x}^{(i)}$ with its intermediate representation in the lifted model, denoted $\mathbf{z}^{(i)} \in \mathcal{Z}$. In Algorithm 1, we remove the `lifter` module, bringing us back to the parameter space of the base model. Since we will attempt to match the latent representation of the lifted model, it is natural to retain the same parameters for the `head`. For simplicity, we also retain the parameters for the `trunk`. Finally, in Algorithm 1, we train `baseModel` using a user-specified *matching loss*, $\mathcal{L}_{\text{match}}$. Here, $\mathcal{L}_{\text{match}}$ may simply be the mean squared error if the $\mathbf{z}^{(i)}$ are intermediate latents, or it may be KL-divergence between $\text{softmax}(\mathbf{z})$ and $\text{softmax}(\mathbf{z}')$ if $\mathbf{z}$ and $\mathbf{z}$' are the logits in a classification task, i.e., if `head` is simply the final softmax layer.

[1]For example, if the `lifter` module is a ResNet, i.e., $\texttt{lifter}(\mathbf{z}) = \mathbf{z} + \texttt{residual}(\mathbf{z})$, we can initialize `residual` near zero (using a hyperparameter to control the sparsity and/or magnitude of the weights of the `residual` module).

3.2 DESIGN CONSIDERATIONS

Our choice to initialize the lifter at or near the identity function is to ensure a smooth transition to the lifted space, allowing the model to *"continue where it left off"*, rather than training the lifted model from scratch. We take inspiration from neural teleportation [Armenta et al., 2023, Zhao et al., 2022], which optimizes over an architecture-specific parameterization of the loss function level sets to "teleport" the model to a different point in the *same* parameter space with the same loss. Meanwhile, our neural lifting approach lifts the model to a *different* parameter space altogether, without requiring any architecture-specific invariances as input.

The efficacy of Algorithm 1 relies on how well the trunk of the base model matches the lifted output during Phase 3. If $\mathcal{L}_{\text{match}}$ is not reduced to zero in practice, we can optimize a combined objective which balances between matching the lifted model and task-specific performance:

$$\mathcal{L}_{\text{combined}} = \lambda \mathcal{L}_{\text{match}} + (1 - \lambda)\mathcal{L}_{\text{task}}, \tag{1}$$

where we may fix λ, or vary it from $\lambda = 1$ (fully prioritizing matching loss) to $\lambda = 0$ (fully prioritizing the task-specific loss). We summarize this approach in Appendix A. Finally, since each phase uses a different model-loss combination, there is also scope for improvement by the learning rates and optimizers between phases.

4 FUNCTION SPACE PERSPECTIVE

Taking a different perspective, neural lifting can be understood as a way of switching between a smaller and a larger

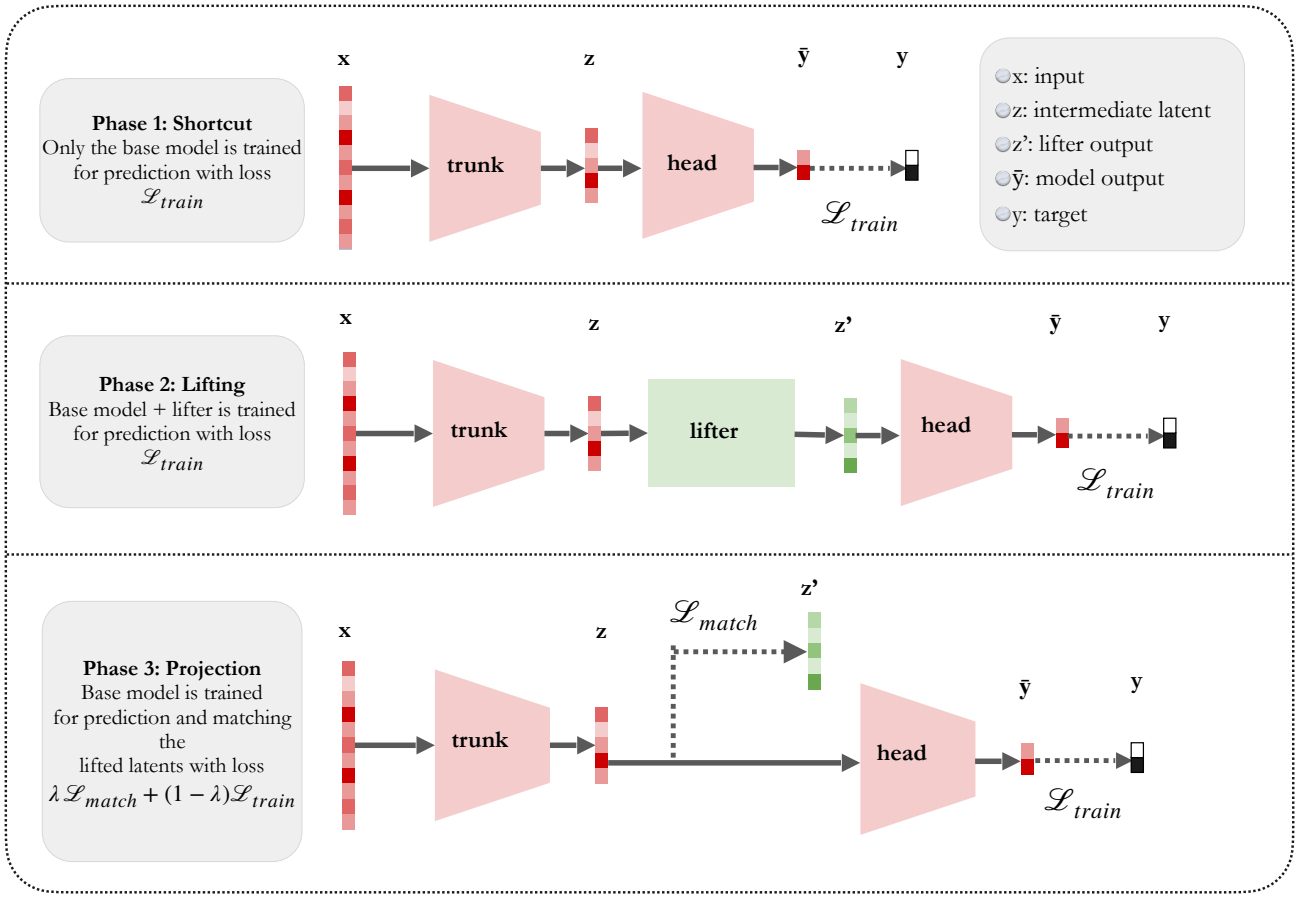

Figure 1: **Training via neural lifting comprises three phases: shortcut training, lifted training, and projection**.

function space, with the overall progress in each iteration captured by a sequence of functions in these two spaces.

4.1 COMPOSITION SPACES AND COMPLEXITY

Given two sets $\mathcal{A}$ and $\mathcal{B}$, we let $\mathcal{B}^{\mathcal{A}} := \{f : \mathcal{A} \to \mathcal{B}\}$ denote the set of all functions from $\mathcal{A}$ to $\mathcal{B}$. Given two functions spaces $\mathcal{F} \subseteq \mathcal{B}^{\mathcal{A}}$ and $\mathcal{G} \subseteq \mathcal{C}^{\mathcal{B}}$ with compatible input and output spaces, we let $\mathcal{G} \circ \mathcal{F} := \{g \circ f \mid f \in \mathcal{F}, g \in \mathcal{G}\}$ denote the *composition space* of $\mathcal{G}$ with $\mathcal{F}$.

Let $\mathcal{T} \subseteq \mathcal{Z}^{\mathcal{X}}$, $\mathcal{E} \subseteq \mathcal{Z}^{\mathcal{Z}}$, and $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{Z}}$ denote the function spaces for the `trunk`, `lifter`, and `head` modules, respectively, For example, if the `trunk` module is a neural network $t_\theta(\cdot)$ with parameters $\theta \in \mathbb{R}^d$, then $\mathcal{T} = \{t_\theta \mid \theta \in \mathbb{R}^d\}$. We define the composition spaces

$$\begin{aligned} \widetilde{\mathcal{T}} &:= \mathcal{E} \circ \mathcal{T} \subseteq \mathcal{Z}^{\mathcal{X}}, \\ \mathcal{M} &:= \mathcal{H} \circ \mathcal{T} \subseteq \mathcal{Y}^{\mathcal{X}}, \text{ and} \\ \widetilde{\mathcal{M}} &:= \mathcal{H} \circ \mathcal{E} \circ \mathcal{T} = \mathcal{H} \circ \widetilde{\mathcal{T}} \subseteq \mathcal{Y}^{\mathcal{X}}. \end{aligned} \tag{2}$$

We call $\widetilde{\mathcal{T}}$ the *lifted trunk space*, we call $\mathcal{M}$ the *base function space*, and we call $\widetilde{\mathcal{M}}$ the *lifted function space*. By design, we ensure that $\mathrm{id}_{\mathcal{Z}} \in \mathcal{E}$; thus, $\mathcal{M} \subseteq \widetilde{\mathcal{M}}$.

In light of the expressibility phenomenon, our approach is based on the assumption that there is some function $f^\star \in \mathcal{M}$ which achieves low test loss (i.e., low training loss $\mathcal{L}_{\text{task}}(f^\star)$ and low generalization error). Thus, $\mathcal{M}$ must be sufficiently expressive, or else any $f \in \mathcal{M}$ would underfit. However, the optimizability phenomenon suggests that f^* is difficult to find by SG-based local search. Thus, improving small-model optimization requires a deeper understanding of the interplay between function complexity and optimization.

Measuring function complexity. Henceforth, assume that we have two functionals measuring function complexity, $\Gamma_{\mathcal{Z}} : \mathcal{Z}^{\mathcal{X}} \to \mathbb{R}_{\geq 0}$, and $\Gamma_{\mathcal{Y}} : \ : \mathcal{Y}^{\mathcal{X}} \to \mathbb{R}_{\geq 0}$. Here, $\Gamma_{\mathcal{Z}}$ takes as input a function $\tau : \mathcal{X} \to \mathcal{Z}$, and return a measurement of its complexity, and similarly for $\Gamma_{\mathcal{Y}}$ given a function $f : \mathcal{X} \to \mathcal{Y}$. Importantly, note that $\Gamma_{\mathcal{Z}}$ and $\Gamma_{\mathcal{X}}$ measure the complexity of *functions*, not function classes. Thus, they should not be confused with notions such as Rademacher complexity, but rather as proxies for notions such as algorithmic (Kolmogorov) complexity [Zenil, 2020].

In machine learning, relevant notions of complexity include RKHS norms (in the regression setting where $\mathcal{Y} = \mathbb{R}$) and

the critical sample ratio (in the classification setting where $\mathcal{Y} = \{0, 1\}$) [Krueger et al., 2017]. In Section 5, we will approximate the complexity of ground truth and learned functions in the RKHS defined by the RBF kernel, but the present discussion is agnostic to this particular choice.

4.2 LIFTING AND PROJECTING OF FUNCTIONS

To reason about the behavior of our method in function space, we consider just the starting and ending points of each phase. At this level of granularity, the t-th iteration of our algorithm passes through the sequence of functions

$$\left(f_{1,\text{init}}^{(t)},\ f_{1,\text{term}}^{(t)},\ f_{2,\text{init}}^{(t)},\ f_{2,\text{term}}^{(t)},\ f_{3,\text{init}}^{(t)},\ f_{3,\text{term}}^{(t)}\right),$$

where $f_{a,\text{init}}^{(t)}$ denotes the function realized by the model at the start of Phase a, and $f_{a,\text{term}}^{(t)}$ denotes the function realized by the model at the end of Phase a.

Since $\mathcal{M}$ and $\widetilde{\mathcal{M}}$ are both composition spaces, all of these functions are decomposable, but we only need one of these decompositions to describe our method:

$$f_{2,\text{term}}^{(t)} = h_{2,\text{term}}^{(t)} \circ e_{2,\text{term}}^{(t)} \circ \tau_{2,\text{term}}^{(t)}, \tag{3}$$

for $h_{2,\text{term}}^{(t)} \in \mathcal{H}$, $e_{2,\text{term}}^{(t)} \in \mathcal{E}$, and $\tau_{2,\text{term}}^{(t)} \in \mathcal{T}$. At the start of our method, we randomly initialize at a function $f_{\text{rand}} \in \mathcal{M}$. For notational convenience, we let $f_{3,\text{term}}^{(0)} := f_{\text{rand}}$.

Finally, to describe our methods at this more macroscopic level, we abstract away the inner-loop optimization by letting $\mathbb{Q}(\cdot \mid f\,; \mathcal{L})$ denote the distribution over terminal functions when using a stochastic optimizer to minimize the $\mathcal{L}$, starting from the function f. In this notation, we can write Phase 1 or our method as

$$\begin{aligned} f_{1,\text{init}}^{(t)} &\leftarrow f_{3,\text{term}}^{(t-1)} \\ f_{1,\text{term}}^{(t)} &\sim \mathbb{Q}\left(\cdot \mid f_{1,\text{init}}^{(t)}\,;\ \mathcal{L}_{\text{task}}\right) \end{aligned} \tag{4}$$

Similarly, we can write Phase 2 as

$$\begin{aligned} f_{2,\text{init}}^{(t)} &\leftarrow f_{1,\text{term}}^{(t)} + \text{perturbation} \\ f_{2,\text{term}}^{(t)} &\sim \mathbb{Q}\left(\cdot \mid f_{2,\text{init}}^{(t)}\,;\ \mathcal{L}_{\text{task}}\right), \end{aligned} \tag{5}$$

where the perturbation term is included to informally account for for the fact that we may choose to initialize the lifter module near, but not precisely at, the identity function. Finally, in Phase 3, we make use of our decomposition in Equation (3), writing the phase as

$$\begin{aligned} f_{3,\text{init}}^{(t)} &\leftarrow h_{2,\text{init}}^{(t)} \circ \tau_{2,\text{init}}^{(t)} \\ f_{3,\text{term}}^{(t)} &\sim \mathbb{Q}\left(\cdot \mid f_{3,\text{init}}^{(t)}\,;\ \mathcal{L}_{\text{match}}\right) \end{aligned} \tag{6}$$

Equipped with this notation, we now provide further intuition on the three phases of neural lifting.

Phase 1 (Standard training). We expect that the loss does not increase, and that function complexity does not decrease.

$$\begin{aligned} \mathcal{L}_{\text{task}}\left(f_{1,\text{term}}^{(t)}\right) &\leq \mathcal{L}_{\text{task}}\left(f_{3,\text{term}}^{(t-1)}\right) \\ \Gamma_{\mathcal{Y}}\left(f_{1,\text{term}}^{(t)}\right) &\geq \Gamma_{\mathcal{Y}}\left(f_{3,\text{term}}^{(t-1)}\right) \end{aligned} \tag{7}$$

Put simply, we expect that training behaves reasonably in that it lowers the loss, and that such an improvement can't be achieved by lowering the function complexity.

Phase 2 (Lifted training). We expect that the loss decreases by at least $a(t)$, and that the lifted model is more complex than the unlifted model by a margin of $c(t)$.

$$\begin{aligned} \mathcal{L}_{\text{task}}\left(f_{2,\text{term}}^{(t)}\right) &\leq \mathcal{L}_{\text{task}}\left(f_{1,\text{term}}^{(t)}\right) - a(t) \\ \Gamma_{\mathcal{Y}}\left(f_{2,\text{term}}^{(t)}\right) &\geq \Gamma_{\mathcal{Y}}\left(f_{1,\text{term}}^{(t)}\right) + c(t) \end{aligned} \tag{8}$$

Put simply, we expect that we can reach a better function in the lifted space, and such an improvement is achieved by increasing function complexity.

Phase 3 (Projection). We expect that the loss increases by at most $b(t)$, and that the new unlifted model is less complex than the lifted model by at most $d(t)$.

$$\begin{aligned} \mathcal{L}_{\text{task}}\left(f_{3,\text{term}}^{(t)}\right) &\leq \mathcal{L}_{\text{task}}\left(f_{2,\text{term}}^{(t)}\right) + b(t) \\ \Gamma_{\mathcal{Y}}\left(f_{3,\text{term}}^{(t)}\right) &\geq \Gamma_{\mathcal{Y}}\left(f_{2,\text{term}}^{(t)}\right) - d(t) \end{aligned} \tag{9}$$

Put simply, we expect that distillation is reasonably effective: although we anticipate some drop in performance and a decrease in complexity when going back to the smaller function space, we expect these changes to be bounded.

Overall change. From chaining these inequalities, we expect that at iteration t, we have

$$\begin{aligned} \mathcal{L}_{\text{task}}\left(f_{3,\text{term}}^{(t)}\right) &\leq \mathcal{L}_{\text{task}}\left(f_{3,\text{term}}^{(t-1)}\right) - (a(t) - b(t)) \\ \Gamma_{\mathcal{Y}}\left(f_{3,\text{term}}^{(t)}\right) &\geq \Gamma_{\mathcal{Y}}\left(f_{3,\text{term}}^{(t-1)}\right) + (c(t) - d(t)) \end{aligned} \tag{10}$$

In particular, if the loss decrease $a(t)$ in Phase 2 is always larger than the loss increase $b(t)$ in Phase 3, then the sequence of function $(f_{1,\text{term}}^3, f_{2,\text{term}}^3, \ldots)$ has decreasing loss. Moreover, if the complexity increase $c(t)$ in Phase 2 is always larger than the complexity decrease $d(t)$ in Phase, then the sequence of function $(f_{1,\text{term}}^3, f_{2,\text{term}}^3, \ldots)$ is increasingly complex. Under these idealized conditions, neural lifting may also be interpreted as a method for accessing increasingly complex regions of the function space $\mathcal{M}$, which are inaccessible during standard training.

5 EXPERIMENTS

We perform two experiments. The first experiment is a classification task using the CIFAR-10 benchmark dataset

[Krizhevsky [2009]]. The second is a set of regression tasks on synthetic datasets to how neural lifting enables small models to learn more complex functions. In both experiments, we perform one cycle of training the neural lifting model (completion of an iteration of all three phases). In Phase 3 of all experiments, we use the combined loss function given in Equation (1). The code for replicating our experiments can be found at this link.

5.1 CIFAR-10 DATASET

Baselines. We use a ResNet-20 model with $272,474$ parameters as the large model baseline, similar to that implemented in [Mirzadeh et al., 2019], and a LeNet-5 [LeCun et al., 1998] architecture with $70,346$ parameters as the small model baseline. The LeNet-5 architecture also serves as the `baseModel` on which we will perform neural lifting during training. From hereon, we shall refer to the small baseline model as the LeNet-SGD model, and the model obtained via neural lifting as LeNet-NL model.

Network Architectures. We divide the LeNet `baseModel` into a `trunk` encoder and a classification `head` consisting of a single linear layer, which maps the latent encoding of the trunk to a distribution over the 10 output classes via softmax. The `lifter` consists of two linear layers followed by a residual connection that adds the input back to the transformed output. For more details on the model architecture, see Appendix C. We initialize its parameters using Xavier initialization [Glorot and Bengio, 2010], with a binary mask, and set biases to zero, to ensure initialization close to the identity mapping. The binary mask is controlled by a Bernoulli parameter $p = 1 - \texttt{sparsity}$. In our experiments, the lifter has a single hidden layer of size 1000, adding $169,084$ parameters to the `baseModel`, resulting in a `liftedModel` with $239,430$ parameters.

Hyperparameter Tuning. We train the large ResNet-20 baseline model in a manner similar to Mirzadeh et al. [2019]. For the LeNet-SGD and LeNet-NL models, we initialize the `baseModel` components using Xavier initialization. Both baseline models are trained for 180 epochs each. For LeNet-SGD, we used `Optuna` [Akiba et al., 2019] to select hyperparameters for standard regularizers such as weight decay and gradient clipping. For LeNet-NL, we keep the selected hyperparameters, and use `Optuna` to select additional hyperparameters, see Appendix C. We limited the search to 20 trials to identify a viable set of hyperparameters. The LeNet-NL model uses different learning rates across its three training phases. To switch to the next phase in LeNet-NL, we use a validation set and a patience hyperparameter ρ, switching to the next phase if the validation loss does not improve within ρ number of epochs. We use the standard CIFAR-10 test set of $10,000$ samples, while the remaining $50,000$ instances are split $80-20$ for training and validation. This data split is applied for all models.

Model Name	Parameter Count	Accuracy (%)
ResNet-20	$272,474$	84.87
LeNet-SGD	$70,346$	67.37
LeNet-NL	$70,346$	69.68

Table 2: **Neural lifting improves the performance of a LeNet-5 model on CIFAR-10**. The accuracy reported is on the test dataset.

5-block	1-block		1-block-NL	
MSE	MSE	Ratio	MSE	Ratio
$2.20e{-3}$	$4.12e{-3}$	1.87	$3.21e{-3}$	1.46
$3.86e{-3}$	$6.49e{-3}$	1.68	$4.33e{-3}$	1.12
$9.53e{-3}$	$15.50e{-3}$	1.63	$12.80e{-3}$	1.34

Table 3: **1-block-NL better matches the large model, than its counterpart.** Ratio columns show MSE relative to 5-block model.

Results on CIFAR-10. The experimental results with test accuracies are summarized in Table 2, and the training dynamics are report in Figure 2. One cycle of training LeNet-NL took 249 epochs, with 68 in Phase 1, 117 epochs in Phase 2, and 64 epochs in Phase 3. Although in Phase 2, the number of parameters being trained in LeNet-NL (`liftedModel`) is comparable with those in ResNet-20, the `lifter` is only used for $\approx \frac{1}{2}$ the epochs, with only the `baseModel` being trained for the remaining epochs.

Given the method's large design space, further optimization could yield additional improvements.

5.2 SYNTHETIC DATASET

To investigate the behavior of our method from the function-space perspective described in Section 4, we created three synthetic datasets of varying complexity. To achieve this property, we varied the number of ResNet blocks as a proxy for complexity. Here, we focus on regression instead of classification to show the another application of neural lifting.

Synthetic Function Generation. Our synthetic experiment follows the *teacher-student* setup common in deep learning theory [Goldt et al., 2019]. We attempt to learn the *teacher* function $f^*\colon \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^5$ and $\mathcal{Y} = \mathbb{R}$. In particular, f^* will be randomly sampled from a known *teacher* function space $\mathcal{F}_{\text{teach}}$, and the complexity of f^* depends on how it is sampled. We set $\mathcal{F}_{\text{teach}}$ as the function space of an L-layer ResNet, and f^* is randomly sampled from $\mathcal{F}_{\text{teach}}$ using Kaiming initialization. To get a range of complexities from the teacher function, we perform experiments for $L = 7, 8, 9$. The inputs to the ResNet models are drawn

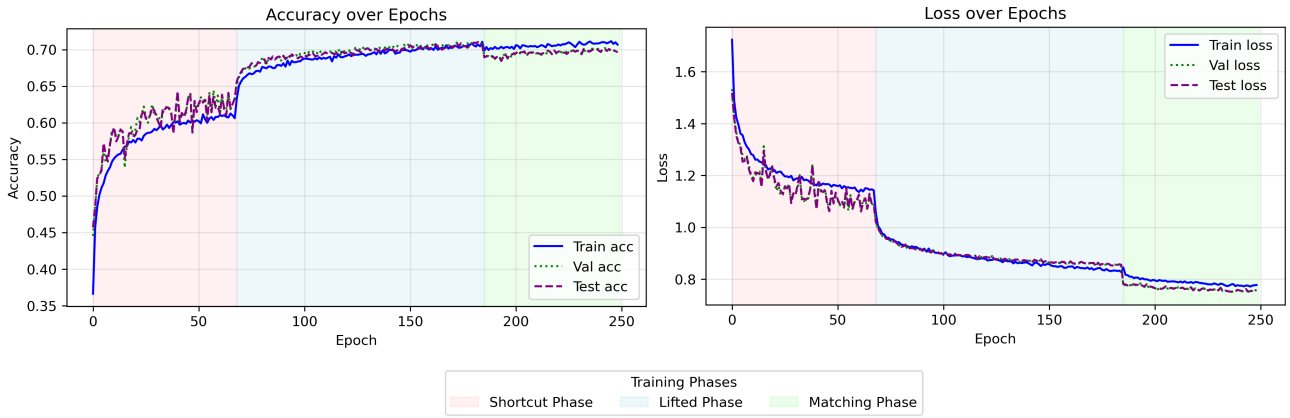

Figure 2: **The empirical training dynamics of neural lifting reinforce our theoretical motivations**. At the end of Phase 1 (shortcut phase), a lack of improvement in validation loss triggers a switch to Phase 2. In Phase 2 (lifted phase), the model initially makes rapid improvements, then gradually reaches another plateau, triggering Phase 3. In Phase 3 (matching phase), the base model has an initial dip in performance but nearly recovers the performance of its lifted counterpart.

from $\mathrm{Uniform}[0,1]^5$, with the model outputs as regression targets. These input-output pairs form our regression dataset. For a visual representation of the dataset, see Appendix C.

Network Architectures. For our experiments, we consider models with the same ResNet-block architecture as that used in the teacher functions, but with fewer blocks. We select a model with 5 ResNet blocks (**5-block**) as the large baseline, and a model with a single ResNet block (**1-block**) as the small baseline. We define **1-block-NL** as a single block ResNet model, with the `lifter` as a ResNet block. We describe the model architecture in detail in Appendix C.

Implementation. We tuned the learning rate hyperparameters using `Optuna` to determine the learning rates for each model. We also tune λ_{match} and initialization of `lifter` for the **1-block-NL** model. The hyperparameter selection was done using 5-Fold cross validation. We train each model for 75 epochs, on a dataset of size 10000 with a 70-10-20 train-validation-test split.

Results on synthetic datasets. The experimental results are summarized in Table 3. We see that **1-block-NL** consistently outperforms the simple **1-block** model and is able to more closely match the output of the **5-block** model.

6 DISCUSSION

In this paper, we put forth the **Small Model Optimization (SMO) hypothesis**, which posits that good test-time performance is achievable with small models, *if and only if* we use training methods from outside the dominant SG-based local search paradigm. This hypothesis was motivated by the *expressibility* and *optimizability* phenomena, for which we provided a review of current empirical and theoretical evidence. Motivated by this need, we described three desiderata for new approaches to small-model optimization, and proposed our own approach, which we call *neural lifting*.

In our experiments, we showed that neural lifting is a promising approach to small-model optimization, allowing small models to perform better at inference time than they would with regular training.

Future Directions This work is an initial foray into a vast space of methods for small-model optimization. We summarize future research directions suggested by our work.

- **Theoretical analysis of neural lifting:** Our high-level theoretical sketch in Section 4 gives rise to several conjectures regarding the sequence of local minima found throughout our approach, and it would be interesting to provide concrete conditions under which these intuitions are correct.
- **Improving "vanilla" neural lifting:** The neural lifting framework has a large design space, especially considering the architectural choices for the `lifter` module, initialization choices for each phase, and optimization choices how we distill in Phase 3. Our method can be considered a "vanilla" variant of neural lifting, from which several improvements ought to be possible.
- **Other approaches to small-model optimization:** Our neural lifting framework was motivated by neural teleportation and lifting techniques in the traditional optimization literature. However, our motivation is broader: we have attempted to make a general case for rethinking optimization when it comes to training small models, and we expect that many ideas from classical optimization can be adapted to the neural setting.
- **Architecture-specific methods:** We have only focused on architecture-agnostic methods to small-model opti-

mization. Often the small models used by practitioners have quite specific architectures, e.g. in the case of neurosymbolic AI [Garcez and Lamb, 2023]. Such models may be particularly hard to optimize, as they are often designed as continuous relaxations of difficult discrete optimization problems. In such cases, it may be reasonable to invest additional effort into architecture-specific methods.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Marco Armenta, Thierry Judge, Nathan Painchaud, Youssef Skandarani, Carl Lemaire, Gabriel Gibeau Sanchez, Philippe Spino, and Pierre-Marc Jodoin. Neural teleportation. *Mathematics*, 2023.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. *Advances in Neural Information Processing Systems*, 2019.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Lihu Chen and Gaël Varoquaux. What is the role of small models in the LLM era: A survey. *arXiv preprint arXiv:2409.06857*, 2024.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

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

Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *Uncertainty in Artificial Intelligence*, 2017.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, 2023.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Sebastian Goldt, Madhu Advani, Andrew M Saxe, Florent Krzakala, and Lenka Zdeborová. Dynamics of stochastic gradient descent for two-layer neural networks in the teacher-student setup. *Advances in Neural Information Processing Systems*, 2019.

Mitchell A. Gordon. Do we really need model compression?, 2020. URL http://mitchgordon.me/machine/learning/2020/01/13/do-we-really-need-model-compression.html.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations*, 2024.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2016.

Yongchang Hao, Yanshuai Cao, and Lili Mou. NeuZip: Memory-efficient training and inference with dynamic compression of neural networks. *arXiv preprint arXiv:2410.20650*, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *Neural Information and Processing Systems 2014 Deep Learning Workshop*, 2014.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics*, 2023.

Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *Transactions on Machine Learning Research*, 2023.

Arnulf Jentzen and Adrian Riekert. Non-convergence to global minimizers for Adam and stochastic gradient descent optimization and constructions of local minimizers in the training of artificial neural networks. *arXiv preprint arXiv:2402.05155*, 2024.

Kenji Kawaguchi. Deep learning without poor local minima. *Advances in Neural Information Processing Systems*, 2016.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, and Aaron Courville. Deep nets don't learn via memorization. 2017.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on Machine Learning*, 2020.

Sanae Lotfi, Marc Finzi, Sanyam Kapoor, Andres Potapczynski, Micah Goldblum, and Andrew G Wilson. PAC-Bayes compression bounds so tight that they can explain generalization. *Advances in Neural Information Processing Systems*, 2022.

Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant, 2019.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, 2019.

David Picard. Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*, 2021.

Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep — but not shallow —networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

Itay Safran and Ohad Shamir. On the quality of the initial basin in overspecified neural networks. In *International Conference on Machine Learning*, 2016.

Yeonjong Shin and George Em Karniadakis. Trainability of ReLU networks and data-dependent initialization. *Journal of Machine Learning for Modeling and Computing*, 2020.

Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 2018.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, 2024.

Gaël Varoquaux, Alexandra Sasha Luccioni, and Meredith Whittaker. Hype, sustainability, and the price of the bigger-is-better paradigm in AI. *arXiv preprint arXiv:2409.14160*, 2024.

Guanchu Wang, Zirui Liu, Zhimeng Jiang, Ninghao Liu, Na Zou, and Xia Hu. DIVISION: Memory efficient training via dual activation precision. In *International Conference on Machine Learning*, 2023.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 2022.

Hector Zenil. A review of methods for estimating algorithmic complexity: Options, challenges, and new directions. *Entropy*, 2020.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 2021.

Bo Zhao, Nima Dehmamy, Robin Walters, and Rose Yu. Symmetry teleportation for accelerated optimization. *Advances in Neural Information Processing Systems*, 2022.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 2024.

Neural Lifting
(Supplementary Material)

A NEURAL LIFTING WITH INTERPOLATION

As mentioned in Section 3.2, the efficacy of "vanilla" neural lifting depends on how well the trunk of the base model can match the latent representations of the lifted model. However, the overall idea of neural level simply requires that the combination of the lifting and projecting phase allows the small model to reach a better local optimum, i.e., matching representations is not a goal in and of itself, but simply a useful heuristic. Thus, it is well within the spirit of the neural lifting framework to modify the projection step in various ways.

In Algorithm 2, we propose one simple variant of neural lifting, which we find to work well in practice. In this variant, which we call *neural lifting with interpolated losses*, we add a hyperparameter λ which mixes between task-specific loss $\mathcal{L}_{\text{task}}$ and the matching loss $\mathcal{L}_{\text{match}}$.

Algorithm 2 NEURAL LIFTING WITH INTERPOLATED LOSSES

Input: Modules, losses, and hyperparameter:

1. Modules `head`, `trunk`, and `lifter`
2. Task-specific loss $\mathcal{L}_{\text{task}}$
3. Matching loss $\mathcal{L}_{\text{match}}$
4. Interpolation hyperparameter λ

Set `baseModel` $\leftarrow$ `head` $\circ$ `trunk` and initialize
while Not converged **do**
*# **Phase 1:** Standard training for base model*
    Optimize $\mathcal{L}_{\text{task}}(\texttt{baseModel})$ until stopping criterion is met
*# **Phase 2:** Lifted training*
    Initialize `lifter` near identity
    Set `liftedModel` $\leftarrow$ `head` $\circ$ `lifter` $\circ$ `trunk`
    Optimize $\mathcal{L}_{\text{task}}(\texttt{liftedModel})$ until stopping criterion is met
*# **Phase 3**: Projection*
    Let $\mathcal{D}_{\text{distill}} \leftarrow \{(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^{n}$, where $\mathbf{z}^{(i)} = (\texttt{lifter} \circ \texttt{trunk})(\mathbf{x}^{(i)})$
    Set `baseModel` $\leftarrow$ `head` $\circ$ `trunk`
    Optimize $\mathcal{L}_{\text{combined}} = \lambda\mathcal{L}_{\text{match}}(\texttt{baseModel}, \mathcal{D}_{\text{distill}}) + (1-\lambda)\mathcal{L}_{\text{task}}(\texttt{baseModel})$ until stopping criterion is met
Return `baseModel`

B ADDITIONAL EXPERIMENTAL DETAILS

B.1 CIFAR-10 EXPERIMENTS

Hyperparameter tuning We performed fine-tuning using Optuna, an automated hyperparameter optimization framework. As we only wished to demonstrate an application of neural lifting, we conducted 20 trials, though in practice it is common to conduct around 100 trials. In all trials, we optimized for the average validation accuracy across $K = 5$ folds of K-fold cross-validation. Table 4 summarizes the search ranges for the optimization search.

Parameter	**Search Range**
Learning rate (shortcut)	[5e-3, 1e-1]
Learning rate (lifted)	[1e-3, 1e-2]
Learning rate (matching)	[1e-3, 1e-1]
λ_{match}	[0.1, 0.6]
Patience	[10, 20]
Lifter width	[800, 1300]
Lifter initialization	{zeros, kaiming, xavier}
Gradient clipping	[3.5, 5.0]

Table 4: Hyperparameter search space using Optuna

Final hyperparameters used The final hyperparameters used are summarized in Table 5.

B.2 TEACHER-STUDENT EXPERIMENTS

Synthetic data generation The inputs to the ResNet models were drawn as $X \sim \text{Uniform}[0, 1]^5$, with the model outputs being taken as regression targets. These input-output pairs form our regression dataset. For a visual understanding of the generated data, please refer to Figure 3.

Architecture Here, we describe the architecture of the networks used in the Teacher-Student experiments.

The `trunk` module comprises an input layer that maps the 5-dimensional input to a 64-dimensional representation, followed by ReLU activation. This is then followed by a single ResNet block for models **1-block** and **1-block-NL**, and 5 blocks for the **5-block** model. Each ResNet block has two linear layers with a ReLU activation in between. This is followed by an identity residual connection, after which a final ReLU is applied. The latent dimension is 64 throughout the network. The `head`module is a single output layer that maps the final 64-dimensional latent to the 1-dimensional output.

Hyperparameters used The hyperparameters used are listed out in Tables 6 to 8. The only hyperparameter tuning done was for the learning rates, λ_{match} and `lifter` initializations.

Hyperparameter	LeNet-SGD	LeNet-NL
Architecture Parameters		
Input channels	3	3
Input shape	$(3 \times 32 \times 32)$	$(3 \times 32 \times 32)$
Conv channels	[6,16]	[6,16]
Conv kernels	[5,5]	[5,5]
Strides	[1,1]	[1,1]
Padding	[2,2]	[2,2]
FC layers	[60]	[60]
Latent dimension	84	84
Output classes	10	10
Pooling size	2	2
Activation	ReLU	ReLU
Training Parameters		
Batch size	128	128
Learning rate (Phase 1)	1e-2	6e-2
Learning rate (Phase 2)	—	1.6e-3
Learning rate (Phase 3)	—	2.3e-3
Weight decay	1e-4	1e-4
Optimizer	SGD	SGD
Dropout rate	0.4	0.4
Gradient clipping	3.7	3.7
Model initialization	Xavier	Xavier
Patience	—	12
Lifter Parameters		
Lifter hidden dim	—	1000
Lifter initialization	—	Xavier
Scale	—	1.0
Sparsity	—	0.99
λ_{match}	—	0.2

Table 5: Hyperparameters used in CIFAR-10 experiments

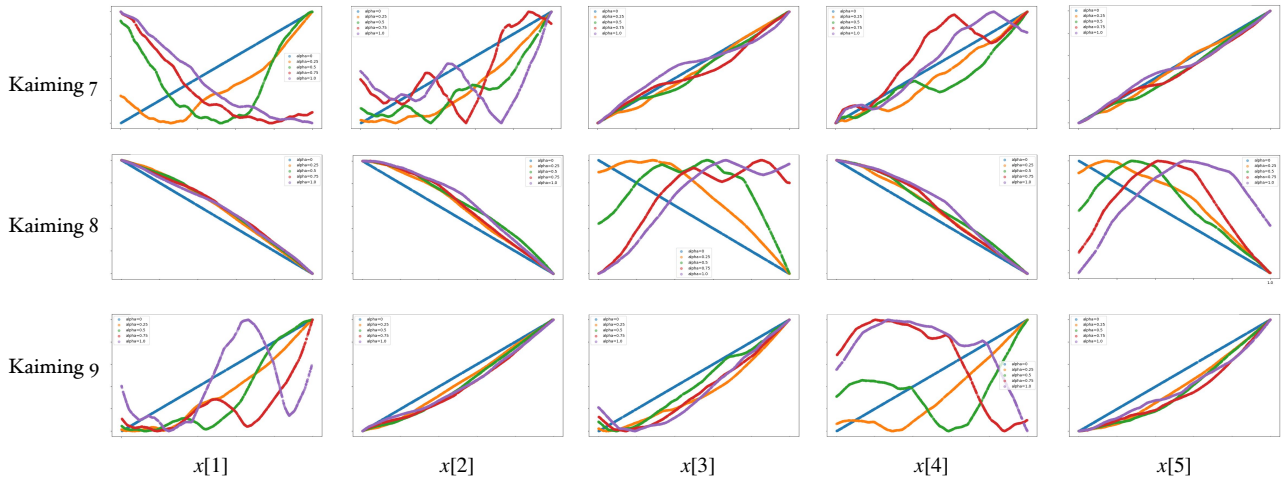

Figure 3: **Synthetic datasets generated using outputs of Kaiming-initialized ResNet models with 7, 8 and 9 ResNet blocks.** Both axes are in the $[0, 1]$ range. Since the input x is 5-dimensional, each plot in column i is generated by keeping all other $x[j], j \neq i$ fixed at α, and varying $x[i]$ from 0 to 1

.

Hyperparameter	5-block	1-block	1-block-NL
Architecture Parameters			
Input dimension	5	5	5
Hidden dimension	64	64	64
Output dimension	1	1	1
Number of blocks	5	1	1
Activation	ReLU	ReLU	ReLU
Model initialization	Xavier	Xavier	Xavier
Training Parameters			
Batch size	128	128	128
Learning rate (Phase 1)	9.90e-3	5.99e-3	1.83e-2
Learning rate (Phase 2)	—	—	5.96e-3
Learning rate (Phase 3)	—	—	7.57e-2
Weight decay	1e-4	1e-4	1e-4
Gradient clipping	5.0	5.0	5.0
Epochs	75	75	75
Patience	—	—	5
Lifter Parameters			
Lifter blocks	—	—	1
Lifter initialization	—	—	Zeros
Scale	—	—	1.0
Sparsity	—	—	None
λ_{match}	—	—	0.3

Table 6: Hyperparameters for experiments on the Kaiming 7 dataset

Hyperparameter	5-block	1-block	1-block-NL
Architecture Parameters			
Input dimension	5	5	5
Hidden dimension	64	64	64
Output dimension	1	1	1
Number of blocks	5	1	1
Activation	ReLU	ReLU	ReLU
Model initialization	Xavier	Xavier	Xavier
Training Parameters			
Batch size	128	128	128
Learning rate (shortcut)	9.90e-3	2.66e-3	1.55e-2
Learning rate (lifted)	—	—	4.72e-3
Learning rate (matching)	—	—	8.91e-3
Weight decay	1e-4	1e-4	1e-4
Gradient clipping	5.0	5.0	5.0
Epochs	75	75	200
Patience	76	76	5
Lifter Parameters			
Lifter blocks	—	—	1
Lifter initialization	—	—	Xavier
Scale	—	—	1.0
Sparsity	—	—	None
λ_{match}	—	0.3	0.3

Table 7: Hyperparameters for experiments on the Kaiming 8 dataset

Hyperparameter	**5-block**	**1-block**	**1-block-NL**
Architecture Parameters			
Input dimension	5	5	5
Hidden dimension	64	64	64
Output dimension	1	1	1
Number of blocks	5	1	1
Activation	ReLU	ReLU	ReLU
Model initialization	Xavier	Xavier	Xavier
Training Parameters			
Batch size	128	128	128
Learning rate (shortcut)	9.90e-3	6.23e-3	1.83e-2
Learning rate (lifted)	1.58e-3	—	5.96e-3
Learning rate (matching)	1.07e-2	—	7.57e-2
Weight decay	1e-4	1e-4	1e-4
Gradient clipping	5.0	5.0	5.0
Epochs	75	75	75
Patience	76	76	10
Lifter Parameters			
Lifter blocks	—	—	1
Lifter initialization	—	—	Xavier
Scale	—	—	1.0
Sparsity	—	—	None
λ_{match}	—	—	0.2

Table 8: Hyperparameters for experiments on the Kaiming-9 dataset

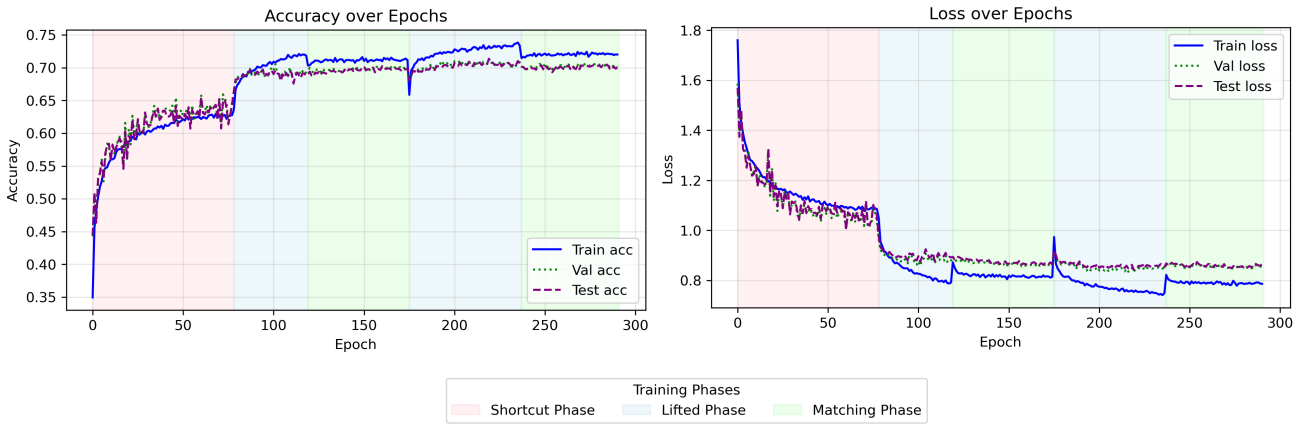

Figure 4: **Neural lifting with convolutional lifter**. Here we train the model using Algorithm 2 for 2 cycles. The lifter parameter count here is $9,264$, as opposed to the $169,084$ parameters of the fully-connected lifter in Section 5. The model performance is comparable across the two lifter settings.

C EXPERIMENTS ON CIFAR-10 WITH CONVOLUTIONAL LIFTER

We perform training of a LeNet model on the CIFAR-10 dataset with a convolutional lifter, comprising two convolutional layers, each using a 3×3 kernel with stride 1 and padding 1. The first layer has 16 input channels and 32 output channels, followed by ReLU activation, and the second layer has 32 input channels and 16 output channels, to which a residual connection is added. The lifter is applied after the final convolutional layer of the base LeNet model.

This yields a lifter with with total parameter count of $9,264$ parameters, significantly smaller than the fully-connected lifter mentioned in Section 5. We train the model using Algorithm 2 for two cycles, using learning rates of 0.05, 0.007 and 0.0016 for the shortcut, lifted and matching phases respectively. With fewer parameters, we obtain a comparable performance with an accuracy of 69.95%, showed in Figure 4.