Low Dimensional Trajectory Hypothesis is True: DNNs can be Trained in Tiny Subspaces

Tao Li, Lei Tan, Zhehao Huang, Qinghua Tao, Yipeng Liu, *Senior Member, IEEE*
Xiaolin Huang, *Senior Member, IEEE*

Abstract—Deep neural networks (DNNs) usually contain massive parameters, but there is redundancy such that it is guessed that they could be trained in low-dimensional subspaces. In this paper, we propose a Dynamic Linear Dimensionality Reduction (DLDR) based on the low-dimensional properties of the training trajectory. The reduction method is efficient, supported by comprehensive experiments: optimizing DNNs in 40-dimensional spaces can achieve comparable performance as regular training over thousands or even millions of parameters. Since there are only a few variables to optimize, we develop an efficient quasi-Newton-based algorithm, obtain robustness to label noise, and improve the performance of well-trained models, which are three follow-up experiments that can show the advantages of finding such low-dimensional subspaces.

Index Terms—deep neural network, training, low dimensionality, gradient descent, subspace

1 Introduction

Deep neural networks (DNNs) have achieved unprecedented success in various fields [1], [2]. In DNNs, the number of parameters is usually very large, e.g., 28.5M in VGG11 [3], 3.3M in MobileNet [4], and 21.0M in Xception [5]. However, simply regarding each parameter of DNNs as an independent variable is too rough. In fact, the parameters have strong mutual relationships. For example, the gradient is propagated from deep layers to shallow ones, and hence the gradients of parameters between different layers are closely related. The parameters in the same layers also have synergy correlations. Therefore, the number of *independent* optimization variables may not be as many as we think. In other words, it seems that DNNs can be well trained in a relatively low-dimensional subspace, as first raised by [6].

In a nutshell, the dependence and redundancy of DNNs' parameters in training could be formally described as the following hypothesis.

The Low-dimensional Trajectory Hypothesis.
For a neural network with n parameters, the parameters' trajectory over training can be approximately covered by a d-dimensional space with $d \ll n$.

If this hypothesis holds, there will be significant benefits in learning for both practical and theoretical aspects. The golden criterion is whether optimization in such low-dimensional spaces could achieve the same or similar performance as optimizing all parameters in the original space. In the pioneering work [7], the authors set 90% accuracy of the SGD training over full parameters as the criterion and find that the intrinsic dimension needed is much smaller than the number of parameters. For example, on CIFAR-10 [8], LeNet [9] with 62006 parameters could be optimized in 2900-dimensional subspaces and the obtained accuracy is 90% of regular training. Albeit that method, in which subspaces are extracted by random projections, is preliminary, the performance is very promising. Later, [10] considers different parts of the network and re-draws the random bases at every step, further reducing the required dimension to hundreds, but the accuracy downgrading is still similar.

The existing works partially but do not fully support the low-dimensional trajectory hypothesis since there is still a significant gap to regular training on full parameters. In this paper, we propose to extract the subspaces through analyzing the dynamic trajectory, instead of random projection in [7], [10]. Via the proposed method, many standard neural network architectures could be well trained by only 40 independent variables, and the performance is almost the same as regular training over full parameters, showing that

DNNs can be trained in low-dimensional subspaces

and we indeed can effectively find such subspaces.

To intuitively show the hypothesis and our aim, one can consider a DNN $f(\mathbf{x}, \mathbf{w})$ with parameters $\mathbf{w} \in \mathbb{R}^n$. Its training sequence, namely training trajectory, can be denoted as $\{\mathbf{w}_i\}_{i=0,\dots,t}$, where $\mathbf{w}_i$ refers to the value of $\mathbf{w}$ at the training step i. This hypothesis means that we can find a subspace (actually, it is an affine set, but we will do centralization later; hence we do not strictly distinguish these two concepts in this paper) to approximately cover the optimization trajectory $\{\mathbf{w}_i\}_{i=0,\dots,t}$. This phenomenon is related to the low-rank properties of Neural Tangent Kernel (NTK) [11], [12] and concentrated gradient in the top subspace [6], which will be discussed in Section 3.1.

Notice that extracting the subspace, i.e., finding independent variables, is different from selecting parameters as in model reduction methods, see, e.g., [13], [14]. Consider a toy example in Fig. 1, which contains three variables to be optimized. As shown, the optimization trajectory is in

• *T. Li, L. Tan, Z. Huang, and X. Huang are with the Department of Automation, Shanghai Jiao Tong Univeristy, Shanghai, 200240, China, e-mail: {li.tao, astray, kinght_h, xiaolinhuang}@sjtu.edu.cn*
Q. Tao is with Department of Automation, Tsinghua University, Beijing, 100084, China, e-mail: taoqh14@tsinghua.org.cn
Y. Liu is with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, e-mail: yipengliu@uestc.edu.cn
corresponding author: Xiaolin Huang

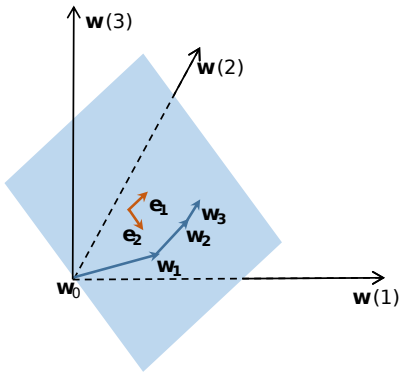


Fig. 1: There are three parameters $\mathbf{w}(1), \mathbf{w}(2), \mathbf{w}(3)$ to optimize. But the training trajectory $\{\mathbf{w}_i\}_{i=0,\ldots,t}$ could be in a two-dimensional subspace spanned by $\mathbf{e}_1$ and $\mathbf{e}_2$. If so, training in the low-dimensional space can have comparable performance as training in the high-dimensional space.

a subspace spanned by $\mathbf{e}_1$ and $\mathbf{e}_2$, i.e., the dimensionality of the optimization landscape is 2, but there is no single parameter that could be reduced. This simple example shows our focus: we aim to find a suitable combination of parameters to construct independent variables which are in the low-dimensional subspaces.

To find the subspace covering the training trajectory, one should care about the training dynamics. Therefore, we name the proposed subspace extraction method as Dynamic Linear Dimensionality Reduction (DLDR). With independent variables obtained from DLDR, we can well characterize the training dynamics in a subspace spanned by only a few bases (independent variables). In Section 5, extensive numerical experiments will show that many standard DNN architectures can be well trained by only 40 independent variables. Meanwhile, the test accuracy can be maintained almost the same as regular training on full parameters in the original space.

In theory, reducing millions of parameters to only a few independent variables could explain the good generalization performance of DNNs even when the training set is not very big. In practice, as only a few independent variables are needed to be optimized, we can apply second-order methods rather than first-order optimization methods, e.g., SGD [15], to overcome some inherent drawbacks, such as scale-sensitivity and slow convergence. In the existing works, part of second-order information like momentum has been introduced and has significantly improved DNNs' performance, leading to the current popular adaptive strategies like Adam [16], RMSprop [17], etc. Thanks to the low-dimensional subspace found by DLDR, quasi-Newton methods, such as DFP and BFGS [18], [19], become applicable to the training of DNNs. In this paper, a BFGS algorithm on the projected subspace, hence called *P-BFGS*, is proposed and obtains about 30% time-saving from SGD.

Another follow-up application of the low-dimension trajectory hypothesis is to improve the model's robustness against label noise. Since DNNs are working in the over-parameterization regime, they could easily fit any label, even incorrect or meaningless ones [20]. Therefore, when the training labels are corrupted by noise, DNNs could be easily destroyed. Now since we have found the very low-dimensional subspace, training DNNs in such subspace is expected to be more robust to label noise, as the degree of freedom for training is significantly reduced. In Section 5.4, we will find that without any other robustness enhancement techniques, training in low-dimensional subspaces can attain over 50% test accuracy on CIFAR-10, even when 90% of the training labels are set randomly.

The most important contribution of this paper is to verify the low-dimensional trajectory hypothesis, which shows that optimizing DNNs in low-dimensional subspaces could achieve similar performance as training over all parameters. Detailed contributions include a subspace extracting methods and three follow-up advantages of extracing low-dimensional subspaces:

- A Dynamic Linear Dimensionality Reduction technique to efficiently find the low-dimensional subspace;
- Training time saving by a quasi-Newton-based algorithm in the low-dimensional subspace;
- Robustness to label noise by training in the low-dimensional subspace;
- Performance improvement from well-trained models by subspace training.

The rest of this paper is organized as follows. We first review the related works in Section 2. Then DLDR algorithm for the low-dimensional trajectory hypothesis is proposed and verified in Section 3. Based on DLDR, we design a quasi-Newton algorithm in Section 4. We then evaluate dimensionality reduction performance in Section 5. Section 6 ends the paper with a brief discussion. The code is released[1].

2 Related Works

Analyzing and understanding the landscape of DNNs' optimization objective is of great importance. For example, Li *et al.* [21] visualize the loss landscape of DNNs using a range of visualization methods. He *et al.* [22] observe that there are many asymmetric directions at a local optimum along which the loss increased sharply on one side and slowly on the other. One important aspect is to measure the intrinsic dimensionality of DNNs' landscape. In the pioneering work [7], it is found that with random projection, optimization in a reduced subspace can reach 90% performance of regular SGD training. Based on that, it is claimed that the intrinsic dimensionality is much smaller than the number of parameters. The following work [10] improves the random bases training performance by considering different parts of the network and re-drawing the random bases at every step. Different from previous works, we extract the subspace via analyzing the DNNs' training dynamics and then get significant improvement: the intrinsic dimensionality is reduced by an order of magnitude, and the accuracy is improved to be almost the same as regular training.

Verifying the low-dimensional trajectory hypothesis and finding the tiny subspaces in which DNNs could be well trained are very important, not only for understanding the DNNs' learning but also for designing powerful optimization methods. It coincides with the discovery in [6] that

1. https://github.com/nblt/DLDR

after a short period of training, the gradients of DNNs can converge to a very small subspace spanned by a few top eigenvectors of the Hessian matrix. In practice, the low-dimensional trajectory hypothesis may inspire more powerful optimization methods and bring more potentials to overcome some existing barriers in learning. Since in the tiny subspace, the number of optimization variables is largely reduced, higher-order information can be utilized in a relatively easy way. In [23], [24], etc., delicate methods are designed to use curvature information while keeping computation efficiency, which is, however, an incompatible contradiction unless the number of optimization variables can be effectively reduced. Another closely related direction is low-rank training [25], [26] or updating [27], [28]. For example, LoRA [28] improves the efficiency of fine-tuning large language models by learning low-rank parameter matrices to update the pre-trained models. While these methods focus on the static low-rank structure of parameter matrices or their adaption, we exploit the low-rank properties of training dynamics and are complementary to these approaches, which can bring further possible efficiency.

3 Dynamic Linear Dimensionality Reduction

3.1 Motivation

The main target of this paper is to point out and verify that the degree of freedom of the parameters in DNNs is quite low. This can be concisely represented as a hypothesis that the parameters' trajectory is in a low-dimensional space.

In a pioneering work [6], the authors find that training in a low-dimensional *randomly selected* space can still result in a meaningful neural network. Specifically, the parameters in the original space $\mathbf{w} \in \mathbb{R}^n$ is updated in a d-dimensional space as by $\mathbf{v} \in \mathbb{R}^d$ as the following,

$$\mathbf{w}_{t+1} = \mathbf{w}_t + P\mathbf{v}_t, \tag{1}$$

where $P \in \mathbb{R}^{n\times d}$ is a random inverse projection matrix. It is obvious that there is only d degree-of-freedom in (1). As numerically verified in [23], even when $d \ll n$, (1) can output meaningful results. Here, we say "meaningful" means that the performance is much better than initialization but has downgraded from well-trained neural networks. The performance is improved in [10] by replacing the random projection with a heuristic. The performance of training in those low-dimensional spaces could be observed in Table 1, which partially but not fully supports the hypothesis that DNNs can be trained in a low-dimensional space.

To investigate the training trajectory and the low-dimensional landscape, we can make use of Neural Tangent Kernel (NTK), which ideally formulates the gradient flow of a single-output neural network as follows,

$$\dot{\mathbf{w}}_t = -\nabla_{\mathbf{w}} f(\mathcal{X}, \mathbf{w}_0)^\top \nabla_{f(\mathcal{X},\mathbf{w}_t)} \mathcal{L}, \tag{2}$$

where $\mathcal{X}$ stands for the training set of size m, $\mathcal{L}$ refers to the loss function, $\nabla_{\mathbf{w}} f(\mathcal{X}, \mathbf{w}_t) \in \mathbb{R}^{m\times n}$ denotes the gradients at moment t. In the view of NTK, the trajectory being low-dimensional equivalently means that the NTK is low-rank. Let us consider the Singular Value Decomposition (SVD) for $\nabla_{\mathbf{w}} f(\mathcal{X}, \mathbf{w}_0)$, i.e., the gradients at the initialization $\nabla_{\mathbf{w}} f(\mathcal{X}, \mathbf{w}_0) = U_0 \Sigma_0 V_0^\top$, with $U_0 \in \mathbb{R}^{m\times m}$, $V_0 \in \mathbb{R}^{n\times n}$ and $\Sigma_0 \in \mathbb{R}^{m\times n}$. Thus, the NTK at the initialization can be formulated as below:

$$\Theta_0 \triangleq \nabla_{\mathbf{w}} f(\mathcal{X}, \mathbf{w}_0) \nabla_{\mathbf{w}} f(\mathcal{X}, \mathbf{w}_0)^\top = U_0 \Sigma_0 \Sigma_0^\top U_0^\top. \tag{3}$$

When Θ_0 is low-rank, Σ_0 has a low-rank approximation, i.e., a low-rank approximation $\Sigma_0 \approx \tilde{U}_0 \tilde{\Sigma}_0 \tilde{V}_0^\top$. Then, the simplified training dynamics under gradient flow in neural tangent regime can be formulated as,

$$\begin{aligned} \dot{\mathbf{w}}_t &= -\nabla_{\mathbf{w}} f(\mathcal{X}, \mathbf{w}_0)^\top \nabla_{f(\mathcal{X},\mathbf{w}_t)} \mathcal{L} \\ &\approx -\underbrace{V_0 \tilde{V}_0}_{\text{Variables}} \underbrace{(\tilde{\Sigma}_0 \tilde{U}_0^\top U_0^\top \nabla_{f(\mathcal{X},\mathbf{w}_t)} \mathcal{L})}_{\text{Projected Gradient}}, \end{aligned} \tag{4}$$

where $\tilde{U}_0 \in \mathbb{R}^{m\times d}$, $\tilde{V}_0 \in \mathbb{R}^{n\times d}$ and $\tilde{\Sigma}_0 \in \mathbb{R}^{d\times d}$. It shows that, ideally, training dynamics $\dot{\mathbf{w}}_t$ of dimension n could be well embedded in a d-dimensional space. $V_0\tilde{V}_0$, of which the columns represent the independent variables, actually establishes a projection from the d-dimensional variable subspace back to the n-dimensional parameter space. The second part in the right-side term of (4) plays a role of gradient in the projected low-dimensional subspace. The approximation error in (4) is determined by the eigen decay of NTK. Analytically, Fan *et al.* [29] show this eigen decay in an asymptotic regime; Empirically, Geifman *et al.* [30] reveal a polynomial decay of the spectrum of NTK at initialization.

Although NTK provides a promising tool to analyze training dynamics and is very inspiring, we need to be aware of the gap between theory and practice. Strict preconditions, like the infinite-width assumption, do not match the setting of practical neural networks. Another crucial limitation of the above discussion is that the NTK is not statistic but dynamic, i.e., the matrix varies during the training process. Therefore, to sufficiently capture the low-dimensional structure, dynamic information is needed. For that, Gur-Ari *et al.* [6] investigate the Hessian in a variety of deep learning scenarios and empirically find that gradient descent mostly happens in a subspace spanned by a few top eigenvectors of Hessian. Their results show that the obtained subspace manifests a high overlap for different training times, also indicating that training trajectory could be approximately covered during the training process.

Based on the above interesting observations, we formally present the *low-dimensional trajectory hypothesis*, given in Section 1. Mathematically, for a parameters' trajectory $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_t, \ldots$, low-dimensional trajectory hypothesis claims that there is a low-dimensional trajectory $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t, \ldots$ and a projection $P \in \mathbb{R}^{n\times d}$ such that

$$\mathbf{w}_{t+1} - \mathbf{w}_t \approx P(\mathbf{v}_{t+1} - \mathbf{v}_t).$$

This hypothesis focuses on the training of neural networks and its verification is straightforwards, i.e.,

If neural networks can be well trained in a d-dimensional space, then the low-dimensional trajectory hypothesis is true.

As discussed before, the existing works now can only partially verify the hypothesis since training on the subspace they extracted, neural networks can be trained to meaningful solutions, but the results are still significantly worse than training on the whole parameter space. In the following, we will present a novel method to find the low-dimensional space in which neural networks can be well-trained. This result verifies the low-dimensional trajectory

hypothesis, which is promising to bring a new understanding for neural networks, e.g., generalization capability, implicit regularization, efficient learning algorithm, and so on.

3.2 Methodology

The key issue of reducing the dimensionality is to find the low-dimensional subspace that approximately covers the parameter trajectory. Instead of handling the continuous trajectory, we practically use its discretization, i.e., discretely sampled points, to characterize the trajectory. The basic operations include:

- First, sample t steps of neural network parameters during the training, namely, $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_t\}$.
- Second, centralize these samples as $\overline{\mathbf{w}} = \frac{1}{t}\sum_{i=1}^{t} \mathbf{w}_i$ and $W = [\mathbf{w}_1 - \overline{\mathbf{w}}, \mathbf{w}_2 - \overline{\mathbf{w}}, \ldots, \mathbf{w}_t - \overline{\mathbf{w}}]$.
- Third, find a d-dimensional subspace spanned by $P = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d]$ to cover W. Notice that in DNNs the number of parameters n is commonly significantly larger than t and d.

The third step is to find a subspace that minimizes the sum of distances from the column vectors of W to the subspace. With the l_2 norm, it could be formulated as maximizing the variance of projection of W, i.e.,

$$\max_{P} \quad \mathrm{tr}\left(P^\top W W^\top P\right), \quad \text{s.t.} \quad P^\top P = I. \tag{5}$$

This is a standard PCA problem that can be solved by performing spectral decomposition on $WW^\top$. The eigenvectors corresponding to the largest d eigenvalues are orthonormal bases, or equivalently, the independent variables that we want for learning.

However, $WW^\top$ is an $n \times n$ matrix that has difficulties in storing, not to mention the high cost of its spectral decomposition. Notice that $WW^\top$ is low-rank since n is far greater than t. We alternatively consider the SVD of W:

$$W = U\Sigma V^\top, \tag{6}$$

where $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n]$, $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_t)$, and $V = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t]$. The first d columns of U are the independent variables that we want herein. Since W and $W^\top$ essentially share the same SVD decomposition, we can first compute $\mathbf{v}_i, i = 1, \ldots, d$ by the spectral decomposition of $W^\top W$, which is only a squared matrix of $t \times t$, so that the vectors $\mathbf{u}_i$ can be computed as

$$W\mathbf{v}_i = \sigma_i \mathbf{u}_i, \ i = 1, \ldots, d. \tag{7}$$

In summary, our dimensionality reduction algorithm, i.e., DLDR, is given in Algorithm 1.

DLDR involves a spectral decomposition of a $t \times t$ matrix and two matrix productions. The total time complexity is $\mathcal{O}(t^3 + t^2 n + t^2 n)$ and the matrix operations involved in DLDR can be greatly sped up by GPUs. Generally, The time consumption is negligible compared to DNNs training.

3.3 Training Performance

The proposed DLDR can reduce the dimensionality of optimization space from n to d, based on the hypothesis that the optimization trajectory approximately lies in a low-dimensional subspace. To verify the hypothesis, we optimize DNNs in such low-dimensional subspaces, checking whether the performance could be similar to training over all parameters in the original space.

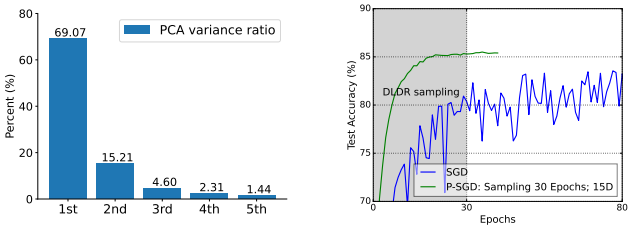
Algorithm 1 Dynamic Linear Dimensionality Reduction (DLDR)

Input: Number of samplings t, dimensionality of the subspace d
Output: Orthonormal bases $[\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d]$

1: Sample parameter trajectory $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_t\}$ along the training;
2: $\overline{\mathbf{w}} = \frac{1}{t}\sum_{i=1}^{t} \mathbf{w}_i$;
3: $W = [\mathbf{w}_1 - \overline{\mathbf{w}}, \mathbf{w}_2 - \overline{\mathbf{w}}, \ldots, \mathbf{w}_t - \overline{\mathbf{w}}]$;
4: Perform spectral decomposition on $W^\top W$ and obtain the largest d eigenvalues $[\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2]$ with the corresponding eigenvectors $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d]$;
5: $\mathbf{u}_i = \frac{1}{\sigma_i} W\mathbf{v}_i$;
6: Return $[\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d]$ as the orthonormal bases.

Firstly, we conduct experiments on training ResNet8 [31] for CIFAR-10, which is also considered by the pioneering work [10]. As stated, the criterion to verify the low-dimensional trajectory hypothesis is to compare the performance of training in subspace and in the full parameter spaces, for which we use SGD to train 78330 parameters of ResNet8. The detailed setting is: learning rate is 0.1, batch size is 128. Notice that in the experiments of this paper, the SGD always contains momentum term and here the momentum parameter is 0.9. With 3 trials, SGD gets averagely 83.84% test accuracy.

Then, we apply DLDR to extract the low-dimensional subspace from the parameters by sampling the trajectory over 30 epochs of SGD training. The detailed sampling strategy is that we sample the model parameters after every epoch training. In Fig. 2a, the variance ratios of the top 5 projected components are plotted, showing that over 90% of the total variance is owned to these five components. This observation coincides with our hypothesis on the existence of such a low-dimensional subspace that can approximately cover the optimization trajectory.



(a) (b)

Fig. 2: Experiments for ResNet8 trained on CIFAR-10. (a) the PCA ratio of the first 30 epochs training trajectory in the principal directions. (b) the training performance of SGD and P-SGD (in 15 dimensional subspace).

Next, in the subspace extracting by DLDR, we train the neural networks from *scratch*. The dimension we set here is 15 and we use the SGD optimizer in the projected subspace, named as *P-SGD*. To avoid other effects for fairness, we use the same hyper-parameter setting as regular SGD and

start from the same initialization. From Fig. 2b, it can be seen that P-SGD quickly surpasses the performance when the DLDR sampling stops (therefore, the good performance of P-SGD is not from the passable solution given by the DLDR sampling stage) and reaches an accuracy similar to or even better than regular SGD. Here, the superiority over SGD may come from the de-noising (variance reduction) effect of low-dimensional subspace and could be further investigated. Yet, at least, it shows that we can effectively train ResNet8 for CIFAR-10 in a subspace with a significantly lower dimensionality, i.e., 15, which strongly supports our hypothesis that the optimization trajectory can approximately lie in a low-dimensional subspace.

In Table 1, we report the dimension used for optimization and the test accuracy. In [7], 7982 dimensions were used to achieve averagely 58.35% accuracy, which was improved to 70.26 in [10]. Now DLDR can find a subspace with much fewer dimensions and achieve significantly better accuracy. In Section 5, we will consider more complicated DNN architectures and more complicated tasks to further verify the low-dimensional trajectory hypothesis.

TABLE 1: Performance of training ResNet8 in low-dimensional subspaces for CIFAR-10 (optimizer: SGD)

Dimension reduction method	Dimension	Test accuracy (%)
—	78330	83.84±0.34
Li *et al.* [7]	7982	58.35±0.04
Gressmann *et al.* [10]	7982	70.26±0.02
DLDR (ours)	**15**	**85.18±0.20**

4 DLDR-based Quasi-Newton Algorithm

Since DNNs generally have massive parameters to optimize, first-order methods, i.e., gradient-descent-based methods, are the dominating methodology. However, there are some fundamental limitations in first-order methods, such as slow convergence around optima and high sensitivity to the learning rate. For these problems, second-order methods are the remedy, but due to the high computational burden, there are great difficulties in applying them to train DNNs involving massive parameters. Instead, part of second-order information, like momentum and accumulation information, has been used, resulting in many popular training algorithms, like Adam [16], RMSprop [17], and AdaGrad [32], [33]. Now with the proposed DLDR, it becomes possible to find only a few (dozens of) independent variables to optimize, which makes it applicable to use second-order methods in training complex DNNs. Following this idea, we develop a quasi-Newton method based on the framework of BFGS [18], [19]. Analogously, the main steps include Hessian matrix approximation based on historical gradients, quasi-Newton update, and backtracking line search, of which the details are given in the following subsections.

4.1 Hessian Matrix Approximation

In Newton method, the descent direction is given as

$$\mathbf{q} = -H^{-1}\mathbf{g},$$

where $\mathbf{g}$ is the gradient and $H \in \mathbb{R}^{n\times n}$ is the Hessian matrix. When n is large, it is computationally intractable to calculate the inverse of H, unless we can well approximate it as

$$H \approx PH_0P^\top,$$

where H_0 is the Hessian matrix in the subspace with orthonormal bases $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d$, i.e., $H_0 \in \mathbb{R}^{d\times d}$. With a small d, its inverse could be efficiently calculated, and then the Newton direction becomes

$$\mathbf{q} = -H^{-1}\mathbf{g} \approx -(PH_0P^\top)^\dagger\mathbf{g} = -PH_0^{-1}P^\top\mathbf{g}, \quad (8)$$

where † denotes the pseudo-inverse operator.

For DNNs, the low-dimensional trajectory hypothesis indicates that we can do optimization in a tiny subspace with dimension d, in which the Newton direction is given by (8). Specifically, the procedure is given as follows:

1) $P^\top\mathbf{g}$: project the gradient of parameters onto the independent variable space;
2) $-H_0^{-1}(P^\top\mathbf{g})$: calculate the Newton direction in the independent variable space;
3) $P(-H_0^{-1}(P^\top\mathbf{g}))$: back project the Newton direction to the original parameter space, while the projection matrix is kept the same during the training.

4.2 Quasi-Newton Update

Although we can find only a few independent variables to optimize, their gradients are calculated by projecting gradients of the original parameters in the current DNN framework. Therefore, directly calculating the second-order gradient is still impractical. Alternatively, we adopt a quasi-Newton method to approximate the Hessian matrix and its inverse. In this way, the standard BFGS algorithm [18], [19] is used with the rank-two correction update as follows,

$$\begin{aligned} B_{k+1} &= V_k^\top B_k V_k + \rho_k\tilde{\mathbf{s}}_k\tilde{\mathbf{s}}_k^\top, \\ \mathbf{y}_k &= \tilde{\mathbf{g}}_{k+1} - \tilde{\mathbf{g}}_k, \\ \rho_k &= (\mathbf{y}_k^\top\tilde{\mathbf{s}}_k)^{-1}, \\ V_k &= I - \rho_k\mathbf{y}_k\tilde{\mathbf{s}}_k^\top, \end{aligned} \quad (9)$$

where B_k is the inverse Hessian approximation matrix in the k-th step, $\tilde{\mathbf{g}}_k$ is the projected gradient by $P^\top\mathbf{g}_k$, and $\tilde{\mathbf{s}}_k = P^\top(\mathbf{w}_{k+1} - \mathbf{w}_k)$ is the projected difference between the parameters in the successive two steps. Here, we begin with initializing the inverse Hessian approximation as $B_0 = I$.

Notice that the above BFGS algorithm is working in the subspace, so the matrices and vectors involved are all related to the dimension d instead of n. Therefore, the computational complexity is very low, and it is totally different from the existing BFGS-based training methods [34], [35], [36], [37], [38], [39] that are working in the original n-dimensional space.

4.3 Backtracking Line Search

DNNs are highly non-convex, and the Hessian matrix is not always positive semi-definite thereby. To ensure the loss descent, BFGS requires $\mathbf{y}_k^\top\tilde{\mathbf{s}}_k > 0$, guaranteeing the positiveness of the Hessian approximation B_{k+1}. In the

proposed method, we adopt the backtracking line search to satisfy the Armijo condition [40],

$$\mathcal{L}_{\mathcal{B}_k}(\mathbf{w}_k - \alpha_k P B_k \tilde{\mathbf{g}}_k) \leq \mathcal{L}_{\mathcal{B}_k}(\mathbf{w}_k) - c\alpha_k \tilde{\mathbf{g}}_k^\top B_k \tilde{\mathbf{g}}_k, \quad (10)$$

where $\mathcal{L}_{\mathcal{B}_k}$ is the loss with the mini-batch $\mathcal{B}_k$, and c is a positive constant. We start from $\alpha_k = 1$ and repeatedly multiply it by a constant factor $\beta \in (0, 1)$ until (10) holds true. In this paper, we adopt $c = 0.4$ and $\beta = 0.55$, which empirically shows good performance and is of course not necessarily optimal for all scenarios.

4.4 Algorithm Summary

We now summarize the developed quasi-Newton algorithm based on DLDR in Algorithm 2. Since it is essentially a BFGS in the projected subspace, we name it as *P-BFGS*.

Algorithm 2 P-BFGS

Input: Dimensionality of the subspace d, network architecture parameterized by $\mathbf{w}$
Output: Trained model $\mathbf{w}_k$

Obtain $P = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d]$ by DLDR in Algorithm 1;
Initialize $k \leftarrow 0$ and $B_k \leftarrow I_{d\times d}$;
while not converging **do**
  Sample the mini-batch data $\mathcal{B}_k$;
  Compute the gradient $\mathbf{g}_k$ on $\mathcal{B}_k$;
  Perform the projection $\tilde{\mathbf{g}}_k \leftarrow P^\top \mathbf{g}_k$;
  if $k > 0$ **then**
    Do Quasi-Newton update on B_k with (9);
  end if
  Compute α_k using backtracking line search;
  $\tilde{\mathbf{s}}_k \leftarrow -\alpha_k B_k \tilde{\mathbf{g}}_k$;
  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + P\tilde{\mathbf{s}}_k$; {Update the parameters}
  $k \leftarrow k + 1$;
end while

5 Numerical Experiments

After introducing the experiment setup, we in this section will numerically evaluate the following aspects. (1) We apply P-SGD to train DNNs in the subspaces, of which the dimension is fixed to **40**, extracted by DLDR, to verify the low-dimensional trajectory hypothesis on different neural network architectures. (2) We evaluate the performance of the proposed P-BFGS algorithm and show its potential in speeding up the training. (3) We conduct experiments with label noise to demonstrate the inherent robustness obtained from subspace training. (4) We apply DLDR on well-trained networks to further improve their performance.

5.1 Experiments Setup

The datasets used in our experiments include CIFAR-10, CIFAR-100 [8], and ImageNet [41]. For CIFAR, all images are normalized by channel-wise mean and variance. Data augmentations [31] are also performed: horizontal image flipping with probability 0.5, 4-pixel padding, and cropping. We test on ResNet20 and ResNet32 [31], and also other 11 DNN architectures. The numbers of full parameters in these networks are from 0.27M to 28.5M, but we always choose only **40** independent variables in this paper. We train the DNNs using SGD optimizer, for which the weight decay is set as 1e-4, momentum parameter as 0.9, and batch size as 128. The default initial learning rate is set as 0.1. For CIFAR-10, we train the DNNs for 150 epochs and divide the learning rate by 10 at 100 epochs, while for CIFAR-100, we train for 200 epochs and divide at 150 epochs. For ImageNet, our code is modified from the official PyTorch example[2]. The experiments are performed on Nvidia Geforce GTX 2080 TI. We use one GPU for CIFAR and four for ImageNet. Mean and standard deviation are for 5 independent trials. More detailed analysis of DLDR and settings for experiments can be founded in Appendix.

DLDR needs to sample the optimization trajectory. For CIFAR, we adopt the simplest sampling strategy: the model parameters are sampled after every epoch of training. For ImageNet, the parameters are uniformly sampled 3 times in each epoch of training. A more delicate sampling strategy may improve the performance.

For P-SGD, we adopt the same batch size and momentum factor as SGD. We set the initial learning rate as 1, training epoch as 40, and divide the learning rate by 10 at 30 epochs. For P-BFGS, we set batch size as 1024 for CIFAR and 256 for ImageNet. Notice that this second-order method does not need a learning rate schedule.

5.2 Verification on Various Architectures

In subsection 3.3, experiments are conducted on CIFAR-10, and here we verify the low-dimensional trajectory hypothesis on CIFAR-100. We will train DNNs by SGD on all the parameters and in the reduced subspace (the latter actually is the proposed P-SGD), respectively. For different neural network architectures, we always choose 40 independent variables. If SGD and P-SGD give a comparable performance, it supports our hypothesis and meanwhile verifies the effectiveness of the proposed DLDR. This experiment contains 11 popular DNNs, including VGG11 [3], DenseNet121 [42], Inception [43], NasNet [44], etc., and the number of parameters varies from 780K to 28.5M.

In TABLE 2, we report the test accuracy using SGD with 50/100/200 epochs, and the test accuracy after 200 epochs serves as the baseline. We then apply P-SGD in 40D subspaces, which DLDR extracts from 50 or 100 epochs sampling, and conduct 40 epochs training from the initialization. See the results in the last column in TABLE 2. It clearly shows that P-SGD with 40 independent variables could reach competitive performance of SGD over full parameters. This competitive performance is for all these architectures and strongly supports our low-dimensional trajectory hypothesis. A minor finding is that generally, the performance would be better if the subspaces are better extracted.

5.3 Performance of P-BFGS Algorithm

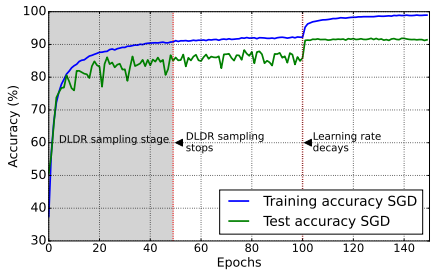
After empirically demonstrating the low-dimensional trajectory hypothesis, we now try a second-order algorithm, namely P-BFGS. We firstly consider ResNet20 [31] on CIFAR-10. In Fig. 3a, the training and test accuracy curves of SGD are plotted. The gray region indicates where we

2. https://github.com/pytorch/examples/tree/master/imagenet

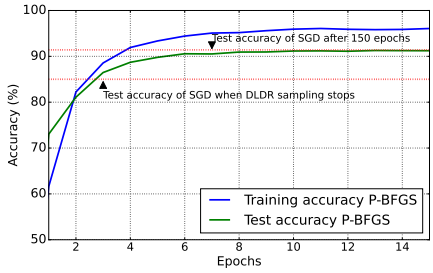
TABLE 2: Test accuracy on CIFAR-100 for regular training and optimization in 40D subspaces

Models	#Param.	SGD #training epochs 50	100	200	P-SGD #sampling epochs 50	100
VGG11_bn [3]	28.5M	58.38	59.90	68.87	68.72	70.18
EfficientNet-B0 [45]	4.14M	62.53	63.68	72.94	71.68	72.64
MobileNet [4]	3.3M	57.15	58.67	67.94	66.86	68.00
DenseNet121 [42]	7.0M	65.39	64.57	76.76	74.25	76.34
Inceptionv3 [43]	22.3M	61.68	64.00	76.25	75.15	76.83
Xception [5]	21.0M	64.57	65.81	75.47	75.68	75.56
GoogLeNet [46]	6.2M	62.32	66.32	76.88	75.66	77.27
ShuffleNetv2 [47]	1.3M	62.90	63.15	72.06	71.34	72.29
SequeezeNet [48]	0.78M	59.52	58.56	70.29	69.89	70.60
SEResNet18 [49]	11.4M	64.74	64.68	74.95	74.33	75.09
NasNet [44]	5.2M	63.73	66.80	77.34	77.19	77.03

get samples for DLDR and then extract the independent variables. After obtaining the 40 independent variables, we use P-BFGS starting from the same initialization and plot the training curves in Fig. 3b. After only 2 epochs, P-BFGS attains better performance than SGD with 50 epochs, i.e., the samplings for DLDR, and within 10 epochs, P-BFGS arrives at the performance of SGD with 150 epochs, which preliminarily demonstrates the advantages of applying second-order methods in efficiency.



(a)



(b)

Fig. 3: The accuracy of ResNet20 on CIFAR-10 in different epochs. (a) SGD on 0.27M parameters: DLDR samples the first 50 epochs and there is learning rate decay after 100 epochs. (b) P-BFGS on 40 variables: trains from the initial, P-BFGS could surpass the DLDR sampling stage very quickly and achieve similar performance as (a).

Next, we report quantitative comparisons between SGD and P-BFGS on both accuracy and computational time. The tasks include CIFAR-10, CIFAR-100, and ImageNet. For CIFAR-10 and CIFAR-100, we extract 40D subspaces from 80 and 100 epochs of SGD, respectively. ImageNet is a more challenging task and requires more independent variables, for which 120 independent variables are obtained from 60 epochs of SGD. In TABLE 3, we report the test accuracy comparisons. Generally, the accuracy of P-BFGS is similar to that of SGD, showing again it is sufficient to optimize a DNN in a very low-dimensional space. Thanks to the fact that the number of optimization variables is small now, it is possible to use second-order methods, which may have benefits on optimization, e.g., fast convergence and getting riding of manually tuning learning rate.

TABLE 3: Performance on different datasets (#epochs)

Dataset	Model	Method	#Sampling	#Training	Accuracy (%)
CIFAR-10	ResNet20	SGD	–	150	91.55±0.23
		P-BFGS	80	20	91.72±0.10
CIFAR-100	ResNet32	SGD	–	200	68.40±0.45
		P-BFGS	100	20	69.90±0.48
ImageNet	ResNet18	SGD	–	90	69.794
		P-BFGS	60	4	69.720

The detailed wall-clock time comparisons are presented in Fig. 4, where we normalize the time according to the total training time of SGD. For P-BFGS, the time consumption includes two parts: i) DLDR sampling; ii) optimization in the subspaces. As expected, applying second-order methods significantly improves the convergence speed: the epochs that P-BFGS requires are quite a few. Overall, P-BFGS saves around 30% time from SGD. Notably, over 80% of the total time is used in DLDR sampling. In the future, more sophisticated techniques of identifying the independent variables are promising to further speed up the training.

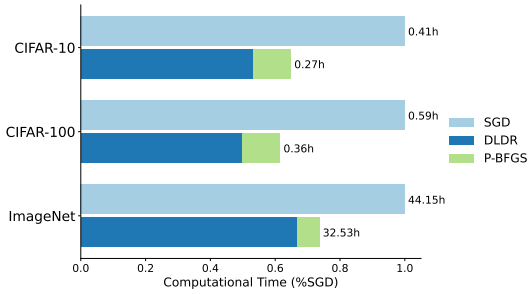


Fig. 4: Wall-clock time consumption comparisons.

5.4 Robustness to Label Noise

Due to the interpolation essence of DNNs, they are sensitive to label noise, i.e., when labels are incorrect, DNNs will follow these incorrect and meaningless labels. Even worse, there is no apparent difference to distinguish whether DNNs learn correct or incorrect labels [20]. Currently, early stopping can be used [50], but how to choose the best stopping is still challenging, since even validation data can also be corrupted. After verifying that DNNs can be trained in low-dimensional subspaces, we expect that the low-dimensional properties could naturally bring robustness to DNNs against label noise.

To examine the performance under label noise, we consider CIFAR-10 and assign random labels to a fraction c of

the training data (which are randomly selected and fixed for different methods). We then train a ResNet20 model with the corrupted data. With label noise, the full training performance of SGD is significantly dropped, as plotted by a blue curve in Fig. 5. Early stopping (red curve) can indeed help but have worse test accuracy on clean data ($c = 0$). Training in low-dimensional subspaces (green curve) can consistently outperform the early stopping with a large margin while keeping the same performance on clean data as regular training. Note that here we obtain robustness without any enhancement techniques, e.g., modifications on the loss function [51], [52], and thus the results are promising to be further improved via these techniques.

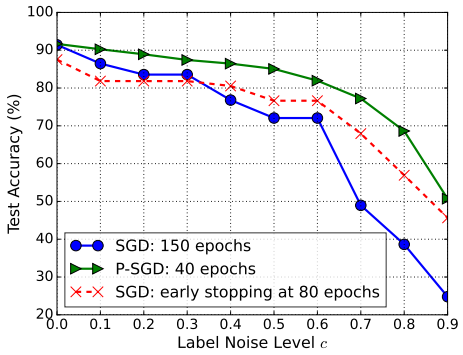


Fig. 5: The performance under different levels of label noise.

The robustness of P-SGD against label noise comes from the low-dimensional properties of training subspaces, or in other words, the degree of freedom for training is quite small. To further investigate the effect of independent variables' number, we vary d from 10 to 40 and report the test accuracy in TABLE 4. With different label-noise levels, P-SGD with full training (i.e., we do not select early stopping) can always have better accuracy than SGD with full training. We also provide the best accuracy obtained by SGD, i.e., we monitor the test accuracy during the training and select the best test accuracy that occurred. Of course, the SGD (best) cannot be reached in practice, but it can serve as a reference showing that, training DNNs in the low-dimensional subspace is robust to label noise.

TABLE 4: The effects of independent variables' number

Noise Level	P-SGD Final				SGD Final	SGD Best
	$d = 10$	$d = 20$	$d = 30$	$d = 40$		
$c = 0.7$	68.92	77.00	77.01	**77.22**	49.0	73.4
$c = 0.8$	64.95	**68.63**	68.51	68.57	38.6	63.6
$c = 0.9$	50.63	50.49	**51.00**	50.38	24.8	49.3

5.5 Improving Well-trained Models

In this subsection, we investigate the subspace extracted from a well-training stage. We start from ResNet18/50 well-trained on ImageNet [41] (from `torchvision.models`). Their accuracy is shown as "well-trained" in TABLE 5. Here, "well-trained" means that continuning orignial SGD training cannot improve the performance. Now we extract subspaces with 30 dimensions (sampled from 5/10/15 epochs of SGD with a learning rate of 0.005) and apply P-BFGS. In this stage, the solutions are near the optima and the landscape is flatter. Hence, the sampling epochs and dimensions could be less than in previous experiments. Since several solutions in the well-trained stage are obtained, stochastic weight averaging (SWA, [53]), an effective and commonly adopted method for generalization improvement, is applicable and we include its performance as a comparison. It could be observed that well-trained models could be further improved by optimization in subspace and the performance is better than SWA.

TABLE 5: P-BFGS on well-training stage on ImageNet.

Model	Methods	Top-1 Accuracy (%) well-trained	5 epochs	10 epochs	15 epochs
ResNet18	SWA	69.758	70.084	70.240	70.378
	P-BFGS		**70.278**	**70.378**	**70.450**
ResNet50	SWA	76.138	76.674	76.760	76.870
	P-BFGS		**76.770**	**76.870**	**76.920**

6 Conclusions and Further Works

The main claim of this paper is the low-dimensional trajectory hypothesis. Based on training dynamics, we design an efficient dimension reduction method called DLDR. In comprehensive experiments, optimizing a few (e.g., dozens of) independent variables extracted by DLDR can attain similar performance as regular training over full parameters. Both optimization performance and reduced dimension have been dramatically improved from prior works [7], [10], strongly supporting the hypothesis and showing that DNNs can be well trained in a tiny subspace.

From the new perspective of DNNs' training, three follow-up applications are tried out to further support our hypothesis and obtain great benefits: 1) as the dimensions substantially reduce, second-order methods become applicable, from which we design a P-BFGS algorithm and bring great efficiency to training; 2) training in low-dimensional subspace naturally brings robustness to label noise; 3) we also show subspace training can significantly improve the performance of well-trained models. These applications further support the low-dimensional trajectory hypothesis. Although they are quite straightforward, e.g., using the original BFGS framework and without any enhancement techniques, their performance implies that finding such low-dimensional subspace could benefit theoretical and practical learning. Possible directions for further works include applying subspace training to fine-tune tasks [27], [28], combining with other low-rank training methods [25], [26], to understand overfitting, over-parameterization [54] and to investigate few-shot learning [55], meta-learning [56], etc.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments.

The research leading to these results has received funding from National Key Research and Development Project (2018AAA0100702), National Natural Science Foundation of China (61977046, 62171088), and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[5] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1251–1258.

[6] G. Gur-Ari, D. A. Roberts, and E. Dyer, "Gradient descent happens in a tiny subspace," *arXiv preprint arXiv:1812.04754*, 2018.

[7] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, "Measuring the intrinsic dimension of objective landscapes." in *International Conference on Learning Representations (ICLR)*, 2018.

[8] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Technical Report*, 2009.

[9] Y. LeCun, "LeNet-5, convolutional neural networks," *url: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[10] F. Gressmann, Z. Eaton-Rosen, and C. Luschi, "Improving neural network training in low dimensional random bases," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 12 140–12 150.

[11] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 8571–8580.

[12] J. Lee, L. Xiao, S. S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington, "Wide neural networks of any depth evolve as linear models under gradient descent," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2020, no. 12, p. 124002, 2020.

[13] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *British Machine Vision Conference*, 2015.

[14] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks." in *International Conference on Learning Representations (ICLR)*, 2018.

[15] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[16] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[17] Y. N. Dauphin, H. De Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," *arXiv preprint arXiv:1502.04390*, 2015.

[18] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms 1. general considerations," *IMA Journal of Applied Mathematics*, vol. 6, no. 1, pp. 76–90, 1970.

[19] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-Newton matrices and their use in limited memory methods," *Mathematical Programming*, vol. 63, no. 1, pp. 129–156, 1994.

[20] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[21] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 6389–6399.

[22] H. He, G. Huang, and Y. Yuan, "Asymmetric valleys: Beyond sharp and flat local minima," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 2549–2560.

[23] M. Tuddenham, A. Prügel-Bennett, and J. Hare, "Quasi-Newton's method in the class gradient defined high-curvature subspace," *arXiv preprint arXiv:2012.01938*, 2020.

[24] J. Sohl-Dickstein, B. Poole, and S. Ganguli, "Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods," in *International Conference on Machine Learning*. PMLR, 2014, pp. 604–612.

[25] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 6655–6659.

[26] M. Khodak, N. Tenenholtz, L. Mackey, and N. Fusi, "Initialization and regularization of factorized neural layers," in *International Conference on Learning Representations (ICLR)*, 2021.

[27] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for NLP," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2790–2799.

[28] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations (ICLR)*, 2022.

[29] Z. Fan and Z. Wang, "Spectra of the conjugate kernel and neural tangent kernel for linear-width neural networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 7710–7721.

[30] A. Geifman, A. Yadav, Y. Kasten, M. Galun, D. Jacobs, and B. Ronen, "On the similarity between the laplace and neural tangent kernels," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1451–1461.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[32] H. B. McMahan and M. J. Streeter, "Adaptive bound optimization for online convex optimization," in *Conference on Learning Theory (COLT)*, 2010, pp. 244–256.

[33] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.

[34] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989.

[35] Y.-X. Yuan, "A modified BFGS algorithm for unconstrained optimization," *IMA Journal of Numerical Analysis*, vol. 11, no. 3, pp. 325–332, 1991.

[36] A. Mokhtari and A. Ribeiro, "RES: Regularized stochastic BFGS algorithm," *IEEE Transactions on Signal Processing*, vol. 62, no. 23, pp. 6089–6104, 2014.

[37] A. Botev, H. Ritter, and D. Barber, "Practical Gauss-Newton optimisation for deep learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 557–565.

[38] R. Bollapragada, J. Nocedal, D. Mudigere, H.-J. Shi, and P. T. P. Tang, "A progressive batching L-BFGS method for machine learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 620–629.

[39] D. Goldfarb, Y. Ren, and A. Bahamou, "Practical quasi-Newton methods for training deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[40] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[42] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4700–4708.

[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.

[44] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8697–8710.

[45] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[47] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient CNN architecture design," in *Proceedings of*

the European Conference on Computer Vision (ECCV), 2018, pp. 116–131.
[48] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.
[49] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7132–7141.
[50] M. Li, M. Soltanolkotabi, and S. Oymak, "Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 4313–4324.
[51] A. Ghosh, H. Kumar, and P. Sastry, "Robust loss functions under label noise for deep neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.
[52] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in Neural Information Processing Systems*, vol. 31, 2018, pp. 8792–8802.
[53] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization," *arXiv preprint arXiv:1803.05407*, 2018.
[54] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 242–252.
[55] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," *arXiv preprint arXiv:1703.05175*, 2017.
[56] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

Tao Li received the BS degree in control science and engineering from Shanghai Jiao Tong University, China, in 2020. He is currently pursuing the Master degree at the Institute of Image Processing and Pattern Recognition, Shanghai Jiao Tong University, China. His research interests include optimization methods and robustness in machine learning.

Lei Tan received the B.S. degree in electronic and information engineering from Shanghai Jiao Tong University, China, in 2019. He is currently pursuing the Master degree at the Institute of Image Processing and Pattern Recognition, Shanghai Jiao Tong University, China. His research interests include learning theory and optimization methods.

Zhehao Huang is now a senior student at Shanghai Jiao Tong University, Shanghai, China. His research interests are architecture design and sparsity of DNNs.

Qinghua Tao received the B.S. degree from Central South University, China, in 2014, and the Ph.D. degree from Tsinghua University, China, in 2020. She is currently a postdoctoral researcher with ESAT-STADIUS, KU Leuven, Belgium. Her research interests include piecewise linear neural networks, machine learning methods, and optimization.

Yipeng Liu (S'09-M'13-SM'17) received his B.Sc. degree in biomedical engineering and Ph.D. degree in information and communication engineering from University of Electronic Science and Technology of China (UESTC), in 2006 and 2011, respectively. He was a research staff in Huawei Technologies, Chengdu, China in 2011 and a research fellow at University of Leuven from 2011 to 2014. Since 2014, he has been with UESTC as an associate professor. His research interest is tensor computation for data processing. He has published over 70 journal and conference papers, co-authored 2 books, and edited 1 book. He has been an associate editor for IEEE Signal Processing Letters.

Xiaolin Huang (S'10-M'12-SM'18) received the B.S. degree in control science and engineering, and the B.S. degree in applied mathematics from Xi'an Jiaotong University, Xi'an, China in 2006. In 2012, he received the Ph.D. degree in control science and engineering from Tsinghua University, Beijing, China. From 2012 to 2015, he worked as a postdoctoral researcher in ESAT-STADIUS, KU Leuven, Leuven, Belgium. After that he was selected as an Alexander von Humboldt Fellow and working in Pattern Recognition Lab, the Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany. From 2016, he has been an Associate Professor at Institute of Image Processing and Pattern Recognition, Shanghai Jiao Tong University, Shanghai, China. In 2017, he was awarded by "1000-Talent Plan" (Young Program). His current research areas include machine learning and optimization.