BSFA: Leveraging the Subspace Dichotomy to Accelerate Neural Network Training

Wenjie Zhou[1,2*], **Bohan Wang**[3*], **Wei Chen**[1,2†], **Xueqi Cheng**[1,2]
[1]State Key Laboratory of AI Safety, Institute of Computing Technology,
Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
[3]Independent researcher
zj4323005@gmail.com, bhwangfy@gmail.com, {chenwei2022, cxq}@ict.ac.cn

Abstract

Recent studies (Gur-Ari et al., 2018; Song et al., 2024; Wen et al., 2024) highlight a fundamental dichotomy in deep learning optimization: Although parameter updates along the top eigendirections of the loss Hessian (Dom-space) capture most of the update magnitude, they often contribute minimally to loss reduction. In contrast, updates in the orthogonal component (Bulk-space) have smaller magnitudes but drive most learning progress. In this work, we further advance the understanding of this phenomenon and introduce the **Bulk-Space-Filtration-Accelerator (BSFA)**, a novel plug-and-play framework. BSFA accelerates training by differentially scaling update components projected onto these distinct subspaces, simultaneously enhancing stability by moderating updates in the dominant subspace and boosting convergence speed by amplifying those in the bulk-space. To ensure BSFA is both practical and scalable for contemporary large models, we introduce two key innovations: an efficient estimator using Principal Component Analysis (PCA) on historical updates for fast subspace estimation, and a block-wise strategy that applies this estimation on a per-parameter-block basis. These designs make BSFA computationally tractable and highly effective. We demonstrate BSFA's acceleration across various tasks, notably achieving approximately 2× speedup when pre-training LLaMA-72M on WikiText-103 and LLaMA-134M on OpenWebText compared to vanilla AdamW.

1 Introduction

Deep learning has revolutionized artificial intelligence, achieving remarkable breakthroughs across domains such as computer vision (He et al., 2016; Dosovitskiy et al., 2020; Liu et al., 2021), natural language processing (especially LLMs) (Vaswani et al., 2017; Devlin et al., 2019; Achiam et al., 2023), and healthcare (Miotto et al., 2018; Liu et al., 2020; Esteva et al., 2019). At the heart of this success lies optimization - the indispensable engine driving the training of complex neural networks. By iteratively minimizing loss functions through algorithms like stochastic gradient descent (SGD) (Robbins and Monro, 1951) and its adaptive variants (Duchi et al., 2011; tie, 2012; Kingma and Ba, 2014), optimization enables models to uncover intricate patterns in high-dimensional data. However, the ever-increasing scale of data and model complexity (Kaplan et al., 2020; Hoffmann et al., 2022; Molybog et al., 2023) has led to escalating training costs, motivating researchers to continuously innovate toward methods that can achieve high computational efficiency with reduced resource demands. This relentless pursuit of scalable and adaptive optimization techniques underscores their critical role in advancing the frontiers of deep learning.

Recent research has revealed that optimization landscapes in deep learning exhibit unique characteristics distinct from traditional machine learning objectives. For instance, Zhang et al. (2020) demonstrated that neural network training processes often exhibit dramatic variations in local smoothness positively correlated with gradient magnitudes. Wu et al. (2018); Cohen et al. (2021) identified the "edge of stability" phenomenon, where Hessian matrices implicitly adapt to optimization hyperparameters until oscillation emerges. Zhang et al. (2024a) further discovered "Block Heterogeneity" in Transformer training, characterized by significant disparities in Hessian spectra across parameter blocks. These observations not only explain the superior performance of adaptive optimizers in certain architectures but also inspire specialized optimization algorithm designs (Roulet et al., 2024; Zhang et al., 2024b).

In overparameterized neural networks, the loss landscape is highly anisotropic. Spectral analyses of the Hessian reveal two distinct components.

[*]Equal contribution
[†]Corresponding author

The bulk of its eigenvalues lies near zero and corresponds to flat directions in parameter space, while a small set of large eigenvalues identifies sharp directions (Hochreiter and Schmidhuber, 1997; Sagun et al., 2017). Extensive research (Ghorbani et al., 2019; Papyan, 2018) into the Hessian's prevalent low-rank structure further quantifies this: for k–class classification problems, there are typically around k such dominant eigenvalues, while for Large Language Models (Zhang et al., 2024a; Liu et al., 2023), the largest few tens of eigenvalues are generally significantly more prominent and are sufficient to define the dominant subspace. As a result, at each point in parameter space, the tangent space splits into a high-dimensional flat subspace (*bulk subspace*) and a low-dimensional sharp subspace(*dominant subspace*).

Building on this geometric picture, most recently, a line of works (Gur-Ari et al., 2018; Song et al., 2024; Wen et al., 2024) show that optimization dynamics in the dominant and bulk subspaces follow a clear dichotomy. Most update norm falls into the dominant subspace but contributes little to loss reduction. In contrast, the relatively small update projected onto the bulk subspace drives most of the loss descent (see Section 3 for details). This discovery has been leveraged to explain the effectiveness of modern learning rate schedulers like Warm-Stable-Decay (WSD) (Hu et al., 2024).

Building upon these observations, this work moves beyond phenomenological explanations to algorithmic innovation. Specifically, we address the fundamental question:

Can we exploit the subspace structure in update to accelerate optimization convergence?

Our contributions are threefold:

1. We first extend the understanding of the subspace dichotomy in deep learning optimization. Our empirical analysis reveals that independently modulating update components within the dominant and bulk subspaces—derived from the Hessian Eigenspectrum—yields distinct effects: controlling update magnitudes in the dominant subspace primarily affects training stability, while updates within the bulk subspace predominantly influence convergence speed.

2. Based on this insight, we propose the **Bulk-Space-Filtration-Accelerator (BSFA)**, a plug-and-play framework that differentially scales updates in these subspaces. Initial validation with exact Hessian information shows up to $4\times$ acceleration on training ResNet18 on CIFAR10 and DenseNet121 on CIFAR100. Then we introduce two key algorithmic enhancements to make BSFA practical and scalable: an efficient PCA-based estimator using historical updates for rapid dominant subspace approximation and a block-wise strategy applying this per parameter block. These render BSFA computationally efficient for large models.

3. We validate the practical BSFA framework on large-scale Transformer models, demonstrating significant training acceleration. Notably, BSFA achieves approximately $2\times$ speedup in pre-training LLaMA-72M on WikiText-103 and LLaMA-134M on OpenWebText, and in training ViT-Small on ImageNet-1k, compared to AdamW.

2 Related work

In this section, we review existing literature on optimization behaviors unique to deep learning and acceleration techniques in this domain.

Optimization behaviors specific to deep learning. While classical machine learning tasks (Platt, 1998; Freund and Schapire, 1996) typically feature convex, smooth landscapes with static local properties, deep learning optimization landscapes are inherently more complex. These landscapes are often characterized by non-convexity (Li et al., 2018) and non-smoothness (Zhang et al., 2020), creating seemingly chaotic optimization dynamics. Recent studies, however, have revealed intriguing structural patterns within this complexity. For instance, (Zhang et al., 2020) observed in NLP tasks that the spectral norm of the Hessian matrix exhibits a positive correlation with gradient norms, providing theoretical justification for the effectiveness of gradient clipping techniques. In transformer-based architectures, (Zhang et al., 2024a) identified heavy-tailed distributions in parameter updates, while (Zhu et al., 2024) demonstrated that large learning rates induce oscillations in subtle classification rule learning while preserving deterministic feature acquisition. A particularly influential observation across multiple studies (Jastrzębski et al., 2018; Wu et al., 2018; Jastrzębski et al., 2020; Cohen et al., 2021) is the "edge of stability" phenomenon, where loss sharpness increases until reaching an oscillation threshold determined by optimization hyperparameters.

Most relevant to our work are the findings of (Song et al., 2024; Wen et al., 2024), who demonstrated that parameter updates in deep learning can be decomposed into two distinct subspaces: one accounting for the majority of update magnitude but minimal loss reduction, and another comprising smaller updates that drive most of the loss descent. This dichotomy forms the foundation for our proposed acceleration methodology.

Acceleration techniques in deep learning. Since the introduction of stochastic gradient descent (SGD) (Robbins and Monro, 1951), researchers have persistently sought improvements to optimization efficiency. Early innovations like momentum (Polyak, 1964) enhanced convergence by incorporating historical gradient information, effectively dampening oscillations through velocity-based updates. The advent of adaptive learning rate methods marked a pivotal advancement, with algorithms like AdaGrad (Duchi et al., 2011), RMSProp (tie, 2012), and Adam (Kingma and Ba, 2014) addressing scale variations across parameters. Notably, Adam's dominance in modern practice stems from its synthesis of momentum and adaptive step-size mechanisms. Recent efforts have explored more radical departures, including sign-based updates (Chen et al., 2023), matrix-structured optimization (Jordan et al., 2024), and second-order approximations (Liu et al., 2023), each targeting different aspects of the optimization geometry.

In contrast to these approaches, our methodology draws inspiration from the intrinsic structure of parameter updates. By explicitly leveraging the subspace decomposition phenomenon observed in (Song et al., 2024; Wen et al., 2024), we propose a novel acceleration framework that strategically prioritizes critical update components.

3 Subspace Dichotomy in Training Dynamics

In this section, we first briefly introduce the subspace dichotomy phenomenon (Gur-Ari et al., 2018; Song et al., 2024; Wen et al., 2024) and then present our main findings.

Let $L(\theta) : \mathbb{R}^p \to \mathbb{R}$ be the loss function of the neural network parameterized by θ. For $\theta \in \mathbb{R}^p$, let $H(\theta) = \nabla^2 L(\theta) \in \mathbb{R}^{p\times p}$, with eigenvalues $\lambda_1(\theta) \geq \cdots \geq \lambda_p(\theta)$ and corresponding orthonormal eigenvectors $u_1(\theta), \ldots, u_p(\theta)$. With these notations, (Gur-Ari et al., 2018; Song et al., 2024) defines the top-k **dominant subspace** as $S_k(\theta) = \text{span}\{u_1(\theta), \ldots, u_k(\theta)\}$, and the **bulk subspace** as its orthogonal complement $S_k^\perp(\theta)$. The corresponding projectors to the top-k dominant subspace and the bulk subspace are respectively denoted as $P_k(\theta)$ and $P_k^\perp(\theta)$.

The key finding of Song et al. (2024) is *learning happens in the Bulk Subspace:* Consider a SGD training process $\theta_{t+1} = \theta_t - \eta g_t$ (where g_t denotes gradient at step t). If each update is projected onto the dominant subspace, i.e. $\theta_{t+1} = \theta_t - \eta P_k(\theta_t) g_t$, it fails to decrease the training loss further. Conversely, projecting each update onto the bulk subspace, $\theta_{t+1} = \theta_t - \eta P_k^\perp(\theta_t) g_t$, is still capable of driving down the training loss. We validate this observation in Figure 1, where we train a 2-layer Transformer (2M parameters) on the SST2 dataset using SGD; details are provided in Appendix A.1.

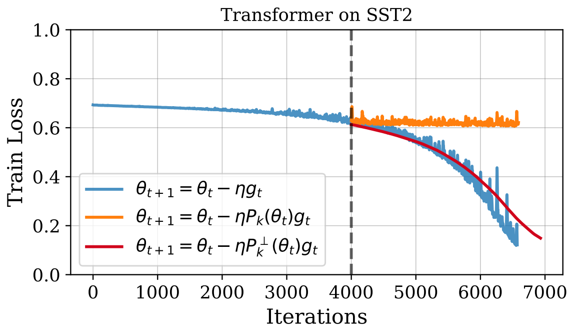
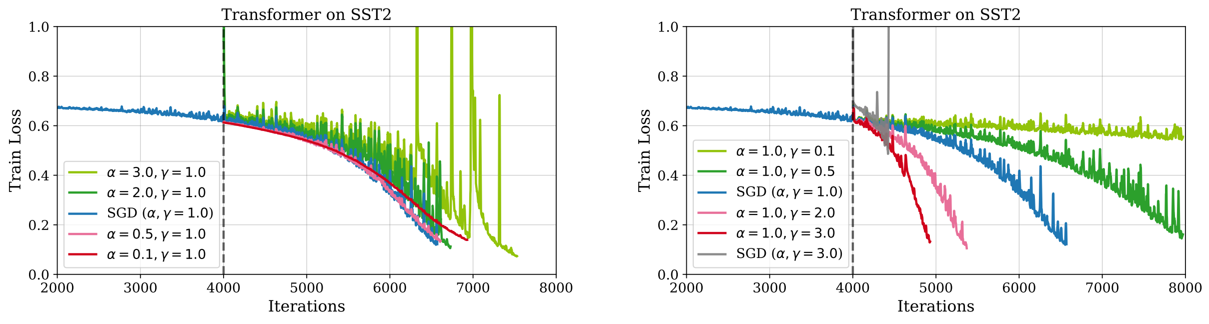


Figure 1: SGD in dominant and bulk subspace ($k = 2$).

Building on this observation, a natural question arises: how do updates projected onto these two subspaces affect the training process? To tease apart these two effects, we introduce a simple, two-parameter projector $\mathcal{P}_{\alpha,\gamma}(\theta)$ that independently scales the updates in the dominant and bulk subspaces. Specifically, let

$$\mathcal{P}_{\alpha,\gamma}(\theta) = \alpha P_k(\theta) + \gamma P_k^\perp(\theta),$$

where $P_k(\theta)$ is the projector onto the dominant subspace and $P_k^\perp(\theta) = I - P_k(\theta)$, as defined above. Then, we conduct experiments using this projector to modify the training iteration: $\theta_{t+1} = \theta_t - \eta \mathcal{P}_{\alpha,\gamma}(\theta_t) g_t$, scaling the dominant and bulk components of the update by α and γ. Following the setup of Figure 1, we sweep $\alpha \in \{0.1, 0.5, 1, 2, 3\}$ and $\gamma \in \{0.1, 0.5, 1, 2, 3\}$ at step 4000, we then compare these projected training runs with a well-tuned SGD baseline, and the results are presented



(a) Varying the dominant subspace update magnitude. (b) Varying the bulk subspace update magnitude.

Figure 2: **Subspace-specific Update Scaling.** We follow the setting of Figure 1(SGD), and after 4000 steps, we introduce two scaling factors, α and γ, to independently modulate the update magnitudes in the dominant and bulk subspaces. Training is terminated once the model's training accuracy reaches 0.99.

in Figure 2. We summarize two notable observations as follows:

- **Dom-Update and Training Stability:** In Figure 2a, with the bulk subspace learning rate held constant, varying the dominant subspace scaling factor α does not significantly alter the overall convergence rate. However, larger values of α (green curves) induce more frequent and pronounced loss spikes during training and induce instability, whereas smaller values of α (red curves) mitigate these spikes and promote smoother convergence.

- **Bulk-Update and Training Speed:** In Figure 2b, with the dominant subspace learning rate fixed, varying the bulk subspace scaling factor γ has a marked impact on convergence speed. When chosen appropriately, higher values of $\gamma = 3$ (red curves) can substantially accelerate training; however, simply increasing the overall update by 3 times (which is equal to setting $\alpha, \gamma = 3$) (grey curve) would lead to divergence.

These findings highlight a functional dichotomy: updates in the dominant subspace predominantly govern training stability, with smaller magnitudes promoting smoother convergence. Conversely, updates in the bulk subspace are primary drivers of convergence speed, where appropriately scaled larger magnitudes can yield substantial acceleration. Such distinct behaviors strongly motivate a decoupled control of update components within these respective subspaces to enhance overall training performance.

4 BSFA: Accelerating Training through Amplify Bulk Update

This section details our methodology for leveraging the observed subspace dichotomy to accelerate neural network training. We first introduce the Bulk-Space-Filtration-Accelerator (BSFA) framework in Section 4.1. Subsequently, Section 4.2 presents an efficient PCA-based estimator for approximating dominant eigendirections. Finally, Section 4.3 describes a block-wise BSFA strategy to enhance scalability for large models by exploiting the Hessian's approximate block-diagonal structure.

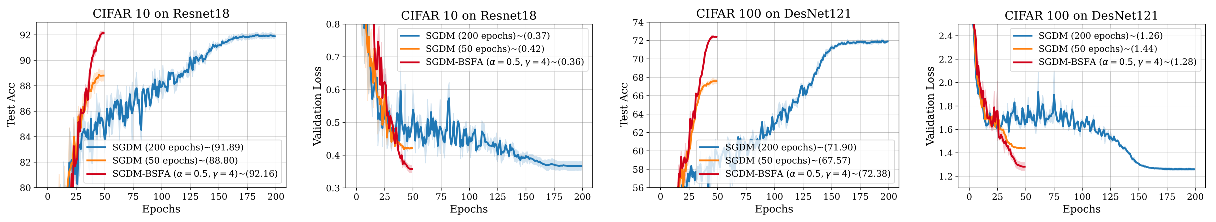
4.1 BSFA Framework

Recalling Figure 2, updates in the dominant and bulk subspaces contribute differently to the training dynamics. Building on this insight, we propose Bulk-Space-Filtration-Accelerator (BSFA), a plug-and-play acceleration framework that can be integrated with any optimizer. Concretely, given a generic update $\theta_{t+1} = \theta_t + v_t$, BSFA modifies it to

$$\theta_{t+1} = \theta_t + \eta\, \mathcal{P}_{\alpha,\gamma}(\theta_t)\, v_t,$$

where $\mathcal{P}_{\alpha,\gamma}(\theta_t)$ denote the projector at iteration t, it scales the base optimizer's update in the dominant subspace by α and in the bulk subspace by γ. In practice, as illustrated in Figure 2, we typically choose $\alpha < 1$ to enhance training stability and setting $\gamma > 1$ generally promotes a faster decrease in the loss. We recompute $\mathcal{P}_{\alpha,\gamma}(\theta_t)$ every T steps, with $T = 10$ by default. We summarize the BSFA framework in Algorithm 1.

By definition, the projector $\mathcal{P}\alpha, \gamma$ needs to be constructed by approximating the top k Hessian eigenvectors $u_i i = 1^k$. To obtain these, a common approach is the Lanczos method (see Appendix B.1



(a) CIFAR10 Accuracy (b) CIFAR10 Validation Loss (c) CIFAR100 Accuracy (d) CIFAR100 Validation Loss

Figure 3: **SGDM with BSFA (**$\alpha = 0.5, \gamma = 4$**) achieves a** $4\times$ **acceleration compared to tuned SGDM.** In both experiments, we employ a cosine learning-rate schedule with different total epoch counts. In each case, BSFA consistently achieves lower validation loss at identical epoch checkpoints and matches the baseline's 200-epoch test accuracy in just 50 epochs. We provide training results for three random seeds.

Algorithm 1 Bulk-Space-Filtration-Accelerator (**BSFA**)

```
1: Input: Current iteration step $t$, Update $\boldsymbol{v}_t \in \mathbb{R}^p$ of base optimizer at step $t$, Dominant subspace's rank $k$, Interval $T$, Projection matrix estimator, Domspace scaler $\alpha$, Bulkspace scaler $\gamma$.
2: BSFA Projector $\mathcal{P}_{\alpha,\gamma} \leftarrow \boldsymbol{I}_{k\times d}$
3: if $t \mod T = 0$ and $t > 0$ then
4:     $\mathcal{P}_{\alpha,\gamma} \leftarrow$ Estimator$(\alpha, \gamma, k)$
5: end if
6: $\boldsymbol{v}'_t \leftarrow \mathcal{P}_{\alpha,\gamma}\boldsymbol{v}_t$
7: Return $\boldsymbol{v}'_t$
```

for details). Leveraging this off-the-shelf routine, we define our **Lanczos-based Projector Estimator (LPE)** as a direct application of Lanczos to construct the following projector:

$$\mathcal{P}_{\alpha,\gamma} = \alpha \sum_{i=1}^{k} u_i u_i^\top + \gamma \Big(I - \sum_{i=1}^{k} u_i u_i^\top \Big).$$

LPE is applied in Figures 1 and 2, yielding highly accurate estimates. Pseudocode for LPE is given in Appendix B.1, Algorithm 4.

To validate the BSFA framework with its Lanczos-based Projector Estimator (LPE), we conducted experiments on ResNet18/CIFAR10 and DenseNet121/CIFAR100 (Figure 3) by integrating BSFA (using $\alpha = 0.5, \gamma = 4$) with well-tuned SGDM (further experiment details are provided in Appendix A.2). These experiments demonstrated that BSFA enables SGDM to achieve higher terminal test accuracy within the same number of training epochs. Furthermore, it provides a significant $4\times$ acceleration, allowing SGDM to reach the baseline's 200-epoch test accuracy and validation loss in just 50 epochs.

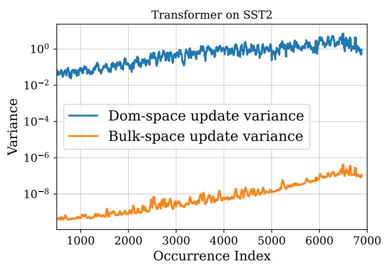
4.2 Fast Dom-Subspace Estimation via Principal Component Analysis

The previous section validated BSFA's substantial acceleration benefits. However, a key challenge of LPE estimator is the high computational cost of estimating the top eigenvectors using the Lanczos method, which involves multiple forward and backward propagation steps, rendering the top-k eigenvector computation both prohibitively expensive and very time-consuming. Therefore, we next explore how this computational time can be significantly reduced by efficiently approximating the dominant eigendirections during training.

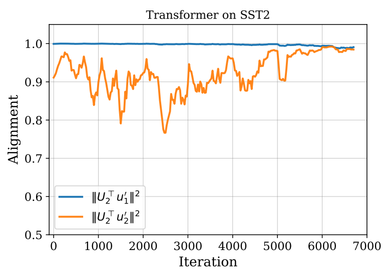
Key insight: Oscillatory Dynamics in Dominant Subspace. To address this, we first draw intuition from the oscillatory dynamics of historical updates in the dominant subspaces. We present the following proposition to provide insight that PCA can capture these top eigenvectors over historical updates.

Proposition 1 (Top eigenspace recovery via PCA)**.** *Let $H \in \mathbb{R}^{p\times p}$ be symmetric positive–semidefinite with simple eigenvalues $\lambda_1 > \cdots > \lambda_k > \lambda_{k+1}$ and suppose all trailing eigenvalues are equal, i.e. $\lambda_{k+1} = \cdots = \lambda_d := \lambda_{\text{tail}} \geq 0$. Pick a stepsize $\eta > 0$ such that $\eta\lambda_k > 1$, $0 < \eta\lambda_{\text{tail}} < 1$, and $\eta(\lambda_k + \lambda_{\text{tail}}) > 2$. Run gradient descent $\theta_{s+1} = \theta_s - \eta H \theta_s$ from any x_0 whose first k eigencoordinates are non-zero, write gradient $g_s = H\theta_s$, and form $G_t = [g_t, \ldots, g_{t+l-1}]$ with window length $l \geq k$ and $l > 1$. Then, as $t \to \infty$, the k-dimensional principal subspace identified by PCA of the gradient matrix G_t converges to the target eigenspace $S_k = \mathrm{span}\{v_1, \ldots, v_k\}$.*

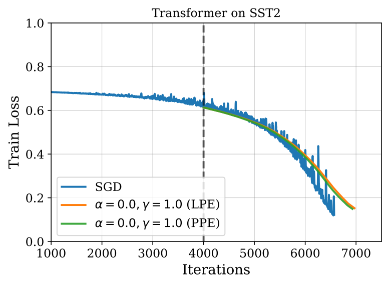
Proposition 1 indicate that the top-k eigenvectors of loss Hessian can be recovered by applying PCA to a list of recent gradients: directions with larger eigenvalues oscillate more strongly and thus



(a) Variance of updates in two subspaces.



(b) Squared projection norms.



(c) Training loss comparison.

Figure 4: (a) The Variance of updates in Dom-space is much higher than Bulk-space. (b) Alignment of Dom-space estimated by PPE and LPE is close to 1. (c) PPE and LPE exhibit comparable efficacy.

dominate the principal components (proof is in Appendix C). In Figure 4a, we validate this oscillatory behavior on a Transformer trained on SST2 (same setup as Figure 2) and, for each update, project the most recent 30 gradient vectors onto the estimated dominant subspace and its orthogonal complement. We then compute the sample variance of these projections and observe that the Variance in the dominant subspace far exceeds that in the bulk subspace, confirming that PCA on gradient histories effectively isolates the Hessian's leading eigenspace.

Motivated by Proposition 1 and the variance separation phenomenon illustrated in Figure 4a, we introduce our **PCA-based projector estimator (PPE)** in Algorithm 2. The estimator maintains a fixed-length queue storing the most recent l updates(with $l > k$) and performs PCA on the queued updates, and retains the top k principal components.

Algorithm 2 PCA-based Projector Estimator (PPE)

```
1: Input: Historical updates matrix $\boldsymbol{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_l]$, number of principal components $k$, domain scaler $\alpha$, bulk scaler $\gamma$
2: $\boldsymbol{U}_k \leftarrow \mathrm{PCA}(\boldsymbol{V}, k)$
3: $\mathcal{P}_{\alpha,\gamma} \leftarrow \alpha\, \boldsymbol{U}_k \boldsymbol{U}_k^\top + \gamma\, (\boldsymbol{I} - \boldsymbol{U}_k \boldsymbol{U}_k^\top)$
4: Return: $\mathcal{P}_{\alpha,\gamma}$
```

To validate the efficacy of PPE, we train a Transformer on SST2 ($k = 2$ for this binary classification task) following the setup in Figure 1, and compare the two-dimensional dominant subspaces estimated by PPE ($U = (u_1, u_2)$) and LPE ($U' = (u_1', u_2')$) in Figure 4b. We compute the squared projection norms $\|U^\top u_i'\|^2$ for $i = 1, 2$ at multiple checkpoints. As shown in Figure 4b, the leading eigenvector attains a projection norm virtually equal to 1, and the second eigenvector remains close to 1 on average. Moreover, Figure 4c demonstrates that BSFA with either estimator yields nearly identical training trajectories and final accuracies. Table 1 reports the average per-update time on an RTX4090, showing a 99.84% speed-up with PPE over LPE. Additional runtime comparisons are provided in Appendix B.2.

Estimator	Time
LPE	10.28s
PPE	0.12s ($\downarrow$ $\mathbf{99.84\%}$)

Table 1: Wall-clock time on 1 RTX4090.

4.3 Block-wise Strategy for Enhancing BSFA Scalability

In the preceding sections, we introduced the BSFA framework and presented our PCA estimator, which delivers substantial acceleration in approximating the dominant subspace and thus renders BSFA far more practical. However, directly applying this PCA estimator to large-scale models' entire, high-dimensional parameter vector presents a significant performance bottleneck: the essential SVD operation becomes extremely memory-intensive and computationally slow when processing such massive, monolithic parameter vectors.

Leveraging Block-Diagonal Hessian Structure. To overcome the scalability challenges posed by extremely high-dimensional parameter vectors, we leverage the intuition that the loss Hessian in deep neural networks is approximately block-diagonal. Extensive empirical studies have demonstrated that, across architectures—including modern transformers—the Hessian naturally decomposes into independent blocks, each with its eigenvalue spectrum (Collobert, 2004; Roux et al., 2007; Martens and Grosse, 2015; Zhang et al., 2024a).

Motivated by this structure, we introduce the **Block-wise PCA-based Projector Estimator (BPPE)**, detailed in Algorithm 3. BPPE constructs a block-diagonal projector by partitioning the full parameter vector into B disjoint subvectors (typically via PyTorch's default block-wise segmentation). For each block, our PCA estimator is independently applied to its historical parameter updates to extract the local dominant subspace and form a block-specific projector. In the context of large language models, BPPE strategically excludes the input and output (embedding) layers from these subspace operations—applying only the base optimizer to these large, sparse components—a practice consistent with optimizer designs like Adam-Mini (Zhang et al., 2024b) and Muon (Liu et al., 2025; Jordan et al., 2024).

Algorithm 3 Block-wise PCA-based Projector Estimator (**BPPE**)

```
1: Input: Historical updates matrix $\boldsymbol{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_l] \in \mathbb{R}^{p \times l}$, parameter blocks $B$, principal components $k$, domspace scaler $\alpha$, bulkspace scaler $\gamma$
2: Partition the parameter indices into $\{I_b\}_{b=1}^{B}$ via PyTorch default partition
3: for $b$ in $B$ do
4:     if LLM & $b \in \{\text{embedding}, \text{output}\}$ then
5:         continue
6:     end if
7:     $P_{\alpha,\gamma}^{(b)} \leftarrow$ PPE$\big(\boldsymbol{V}[I_b, :], k, \alpha, \gamma\big)$
8: end for
9: $\mathcal{P}_{\alpha,\gamma} \leftarrow \text{blockdiag}\big(P_{\alpha,\gamma}^{(1)}, \dots, P_{\alpha,\gamma}^{(B)}\big)$
10: Return: $\mathcal{P}_{\alpha,\gamma}$
```

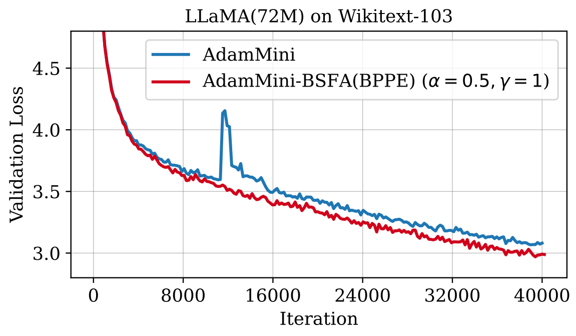


Figure 6: Testing BPPE on LLaMA.

This block-wise application via BPPE inherently reduces the memory footprint, improves cache efficiency, and enhances numerical stability and parallelism compared to a dense projector. These advantages collectively accelerate the overall projector estimation and critically improve BSFA's scalability. We evaluate BPPE in Figure 6 by training LLaMA (72M) (Touvron et al., 2023) on WikiText-103 (detailed settings in Appendix A.2). Integrated into AdamMini-BSFA with dominant subspace updates moderated by $\alpha = 0.5$, BPPE effectively mitigates training loss spikes and enhances stability, demonstrating its reliability in leveraging block-specific dominant subspace information for improved training dynamics.

5 Experiments

5.1 Experiment settings

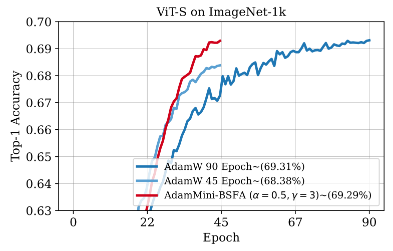
For language tasks, we examine the performance of BSFA in two experiments: LLaMA-72M on WikiText-103 and LLaMA-134M on OpenWebText (Gokaslan and Cohen, 2019). For vision tasks, we choose ViT-Small (Dosovitskiy et al., 2020) on ImageNet-1k (Deng et al., 2009).

For all experiments, we use the default AdamW as the baseline optimizer, we follow the training protocols of nanoGPT and LLaMA for language tasks, with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay $\lambda = 0.1$. The learning rate is linearly warmed to `lr_max` and decayed via a cosine scheduler. For each task, we tune `lr_max` to optimize AdamW; further details are provided in Appendix A.3.

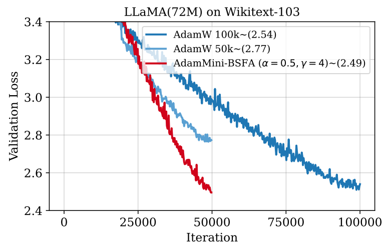
BSFA Implementation We integrate BSFA into AdamMini under the same settings as above. AdamMini is chosen primarily for two reasons: its 2x memory reduction compared to AdamW accommodates BPPE's overhead, and its minimal block-wise second-moment statistics ensure a distinct operational mechanism that does not overlap with BSFA's subspace adjustments, facilitating a clean integration. Note that AdamW's baseline performance is comparable to or slightly better than AdamMini's (Zhang et al., 2024b). This context allows our setup with AdamMini to effectively showcase BSFA's acceleration capabilities. Moreover, because we use BPPE as the estimator, our method's average per-step time is similar to AdamW; Table 2 shows the time comparison when the update interval is $T = 10$.

5.2 Main Results

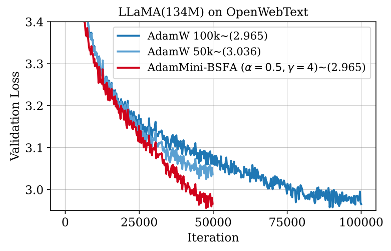
In Figure 5, we train ViT-S on ImageNet-1k and compare a well-tuned AdamW with AdamW-BSFA ($\alpha = 0.5, \gamma = 3$), and we evaluate LLaMA-72M on WikiText-103 and LLaMA-134M on OpenWebText, comparing well-tuned AdaW



(a) ViT-Small on Imagenet-1k



(b) LLaMA(72M) on WikiText-103



(c) LLaMA(134M) OpenWebText

Figure 5: AdamMini–BSFA outperform AdamW baseline in ViT and LLaMA training.

Experiment	AdamW (/step)	BSFA (/step)
ViT-S	0.868s	1.02s
LLaMA(72M)	0.386s	0.422s
LLaMA(134M)	1.86s	2.12s

Table 2: Comparison of average per-step wall-clock time (in seconds) on four RTX4090 GPUs. "BSFA" indicates the AdamMini-BSFA.

with AdamMini-BSFA ($\alpha = 0.5, \gamma = 4$) and AdamMini-BSFA ($\alpha = 0.5, \gamma = 2$). Across all settings, for the same total number of training steps, BSFA consistently achieves lower training loss (and higher top-1 accuracy on ImageNet-1k) than vanilla AdamW or AdamMini. Moreover, BSFA reaches the performance of vanilla AdamW or AdamMini trained for twice as many steps, corresponding to a 2× acceleration.

5.3 Ablation Study

In the above experiments, BSFA with BPPE was applied across all major architectural blocks, categorized as **Norm**, **Attention**, and **MLP** blocks. Here, we conduct experiments to investigate the contribution of applying BPPE to different blocks towards acceleration and training stability. All ablations adhere to the BSFA configuration from Figure 5b, with ($\alpha = 0.5, \gamma = 4$). We apply BSFA solely to the selected blocks, while the remaining blocks continue with vanilla AdamW. As shown in Figure 7, we derive the following two conclusions:

Result is shown in Figure 7. We derive the following two conclusions:

- Norm layers play a crucial role in maintaining training stability. In Figure 7 (Top), applying BSFA to the norm layers does not accelerate training. However, compared to vanilla AdamW, no significant loss spike is observed. When BSFA is not applied to the norm layers, training diverges.
- Both Attention and MLP blocks contribute to training acceleration. In Figure 7(Bottom), we compare the acceleration effects of applying BSFA to (Attention, Norm) and (MLP, Norm) blocks (with Norm layers ensuring training stability). Both configurations accelerate training relative to vanilla AdamW, achieving similar acceleration slightly inferior to block-wise BSFA applied to all blocks (Attention, MLP, Norm), indicating that both Attention and MLP blocks contribute comparably to accelerating the model.

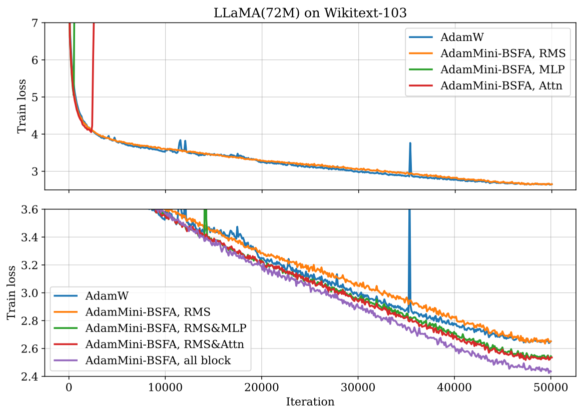


Figure 7: (Top) Diverge if not applying BSFA to Norm blocks. (Bottom)Apply BSFA to either attention or MLP blocks alongside norm layers.

5.4 Memory Footprint and Efficiency of BSFA

BSFA explicitly caches the top-k principal directions to build its projector, incurring an additional memory overhead equivalent to storing k full gradients. Because memory is a primary bottleneck in training, we examine: (i) whether the additional memory indeed translates into faster training, and (ii) can low-bit quantization reduce BSFA's memory usage without degrading performance?

The Memory-Performance Trade-off of BSFA To assess whether the additional memory consumed by BSFA is justified by its performance

Optimizer	micro_BS	Memory	max_iter	avg_per_step_time	total_time	final_val_loss
AdamW	30	10.2GB	100k	386ms	~10.72h	2.54
AdamW	60	17.6GB	100k	335ms	~9.32h	2.54
BSFA	**30**	**18.2GB**	**50k**	**422ms**	**~5.86h**	**2.49**

Table 3: Comparison with a memory budget similar to AdamW with a doubled micro-batch size. "Memory" indicates peak memory usage per GPU

benefits, We test whether BSFA's extra memory is better spent on BSFA itself or on larger micro-batches. On LLaMA-72M, BSFA adds 8 GB/GPU (storing 30× gradient-sized dominant directions), while increasing the micro-batch from 30 to 60 costs 7.4 GB/GPU in activation memory. We compare their time overheads in Table 3. This result demonstrates that in pre-training LLaMA-72M, allocating extra memory to BSFA is a more effective strategy for acceleration. A key insight is that while additional memory can be used to enlarge the micro-batch size, GPU utilization is already near saturation. Consequently, any performance gains from further increasing the micro-batch size become marginal.

Potential for Memory Reduction: 4-bit BSFA Although the memory-performance trade-off ostensibly favors BSFA, its auxiliary memory requirements may become prohibitive as model sizes continue to scale. To explore reducing BSFA's memory footprint, we evaluate whether quantizing BSFA can lower memory usage without degrading performance. Following the exact protocol of Figure 5b, we implement a 4-bit variant that linearly quantizes BSFA's historical gradients and projection matrix. Specifically, we apply 4-bit linear quantization to the historical gradients and the projection matrix required by BSFA. The per-step updates and the dominant directions are quantized using a `group_size` of 64 and stored compactly via nibble-packing. For each group, the scale and zero-point are stored in the FP8 (E4M3) format. These values are only dequantized during the parameter update step, which significantly conserves memory. We perform a comparative study following the exact experimental setup of Figure 5b. The table below presents the validation loss for each optimizer at various training steps.

The results below show that 4-bit BSFA achieves performance comparable to the full-precision version while reducing the additional memory overhead from 8.0 GB to just 1.3 GB, approximately

Optimizer	Loss@50k	Peak Memory
AdamW	2.77	10.2GB
BSFA	2.49	18.2GB
4bit-BSFA	**2.54**	**11.5GB**

Table 4: Comparison of peak memory usage and final loss on four RTX4090 GPUs. "BSFA" indicates the AdamMini with BSFA, "4bit-BSFA" indicates the AdamMini with 4bit-BSFA, "Peak Memory" indicates peak memory usage per GPU.

$1/6$ of its initial value. This result shows that quantizing BSFA to 4 bits is an effective memory-reduction strategy that preserves training quality, providing a practical path to further improve BSFA's memory efficiency. We emphasize that the core contribution of this work is the insightful acceleration strategy derived from our analysis of the loss landscape, leaving the development of more practical, lightweight variants for future work.

6 Conclusion

In this paper, we introduced the Bulk-Space-Filtration-Accelerator (BSFA), a novel framework that exploits the distinct roles of dominant and bulk subspaces in neural network training. By differentially scaling updates within these subspaces—moderating the dominant for stability and amplifying the bulk for speed—and leveraging an efficient PCA-based estimator with a block-wise strategy for scalability, BSFA achieves significant training acceleration, notably an approximate 2× speedup for large Transformer models like LLaMA and ViT compared to AdamW.

Limitations

Although BSFA demonstrates promising acceleration capabilities, it has several areas for potential improvement. First, the PCA estimator BPPE demands significant GPU memory, and reducing its memory requirements remains challenging. Furthermore, estimating fewer dominant directions to save memory could compromise the accuracy of the

Projector, consequently diminishing the acceleration benefits. Second, our current approach assigns an equal number of dominant directions to each block, which may not optimally accommodate the heterogeneous properties of different blocks. Future work could focus on mitigating these memory constraints and developing more adaptive strategies for block-specific control. In this work, we use up to 4 RTX4090 to train for less than 500 hours, including hyperparameter tuning. We wish this experience could boost the understanding of efficient training in the community.

Ethical Considerations

This paper seeks to promote the development of deep learning by focusing on understanding and improving the training processes of neural networks, with particular emphasis on large language models (LLMs). While our work has the potential for broad societal impact, we do not, at present, identify any specific societal implications that warrant special attention.

Acknowledgements

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB0680101), CAS Project for Young Scientists in Basic Research under Grant No. YSBR-034.

References

2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. 2022. Better plain vit baselines for imagenet-1k. *Preprint*, arXiv:2205.01580.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. 2023. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*.

Jeremy Cohen, Simran Kaur, Yiding Jiang, Zico Kolter, and Ameet Talwalkar. 2021. Gradient descent on neural networks typically occurs at the edge of stability. *arXiv preprint arXiv:2103.00065*.

Ronan Collobert. 2004. Large scale machine learning.

Alex Damian, Eshaan Nichani, and Jason D Lee. 2022. Self-stabilization: The implicit bias of gradient descent at the edge of stability. *arXiv preprint arXiv:2209.15594*.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and 1 others. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Andre Esteva and 1 others. 2019. A guide to deep learning in healthcare. *Nature Medicine*, 25:24–29.

Yoav Freund and Robert E. Schapire. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. 2019. An investigation into neural net optimization via hessian eigenvalue density. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2232–2241. PMLR.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. 2018. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Flat minima. *Neural computation*, 9(1):1–42.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and 1 others. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, and 1 others. 2024. Minicpm: Unveiling the potential of small language models with warmup-stable-decay learning rate scheduler. *arXiv preprint arXiv:2404.06395*.

Stanisław Jastrzębski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. 2018. Width of minima reached by stochastic gradient descent is influenced by learning rate to batch size ratio. In *Artificial Neural Networks and Machine Learning - ICANN 2018*, pages 392–402. Springer International Publishing.

Stanisław Jastrzębski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. 2020. The break-even point on optimization trajectories of deep neural networks. In *International Conference on Learning Representations*.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. 2024. Muon: An optimizer for hidden layers in neural networks.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

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

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. 2023. Sophia: A scalable stochastic second-order optimizer for language model pretraining. *arXiv preprint arXiv:2305.14342*.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, and 9 others. 2025. Muon is scalable for llm training. *Preprint*, arXiv:2502.16982.

Yu Liu and 1 others. 2020. A deep learning system for differential diagnosis of skin diseases. *Nature Medicine*, 26:900–908.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022.

James Martens and Roger Grosse. 2015. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR.

Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in Bioinformatics*, 19(6):1236–1246.

Igor Molybog, Peter Albert, Moya Chen, Zachary DeVito, David Esiobu, Naman Goyal, Punit Singh Koura, Sharan Narang, Andrew Poulton, Ruan Silva, Binh Tang, Puxin Xu, Yuchen Zhang, Melanie Kambadur, Stephen Roller, and Susan Zhang. 2023. A theory on adam instability in large-scale machine learning. *arXiv preprint arXiv:2304.09871*.

Vardan Papyan. 2018. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv: Learning*.

Barak Pearlmutter. 1994. Fast exact multiplication by the hessian. *Neural Computation*, 6:147–160.

John C. Platt. 1998. Sequential minimal optimization: A fast algorithm for training support vector machines. In *Advances in Neural Information Processing Systems (NIPS)*.

Boris T Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17.

Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407.

Vincent Roulet, Atish Agarwala, Jean-Bastien Grill, Grzegorz Swirszcz, Mathieu Blondel, and Fabian Pedregosa. 2024. Stepping on the edge: Curvature aware learning rate tuners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Nicolas Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. 2007. Topmoumoute online natural gradient algorithm. *Advances in neural information processing systems*, 20.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. 2017. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Minhak Song, Kwangjun Ahn, and Chulhee Yun. 2024. Does sgd really happen in tiny subspaces? *arXiv preprint arXiv:2405.16002*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and 1 others. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. 2024. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*.

Lei Wu, Chao Ma, and Weinan E. 2018. How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective. In *Advances in Neural Information Processing Systems*, pages 8279–8288.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. 2020. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations (ICLR)*.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhiquan Luo. 2024a. Why transformers need adam: A hessian perspective. In *Advances in Neural Information Processing Systems*.

Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo. 2024b. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*.

Libin Zhu, Chaoyue Liu, Adityanarayanan Radhakrishnan, and Mikhail Belkin. 2024. Catapults in SGD: spikes in the training loss and their impact on generalization through feature learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62476–62509. PMLR.

A Experiment Details

A.1 Experiment Details in Section 3

Figures 1 & 2: Transformer on SST-2. We conduct illustrative experiments on a two-layer Transformer (hidden dimension 64, 8 attention heads) following Damian et al. (2022) and Song et al. (2024), training on SST2 (Socher et al., 2013) for binary sentiment classification with cross-entropy loss. All runs use SGD with constant learning rate $\eta = 0.03$ and batch size 200, and stop when training accuracy reaches 0.99. Training is performed on a single NVIDIA RTX 4090 GPU.

A.2 Experiment Details in Section 4

Figures 3a & 3c: CNNs on CIFAR-10/100. We build on the official PyTorch tutorial, applying random horizontal flips and 32×32 crops with 4-pixel reflection padding, and normalize inputs to mean 0.5. For SGDM we use initial learning rate $\alpha_0 = 0.1$, momentum 0.9, weight decay 5×10^{-4}, batch size 1024, and train for 200 or 50 epochs with cosine decay. For BSFA we adopt LPE with $k = 10$ (CIFAR-10) or $k = 100$ (CIFAR-100), historical updates length $l = k + 10$, $T = 10$, train for 50 epochs, and tune $\alpha \in \{1, 0.5, 0.2\}$, $\gamma \in \{2, 4, 6\}$ via grid search. We avoid selecting α values approaching zero, since imprecise projector estimates would unduly attenuate updates in other directions. All experiments run on a single NVIDIA RTX 4090 GPU.

Figures 4a & 4: Transformer on SST-2. We follow exactly the setup of Section A.1.

Figure 6: LLaMA(72M) on Wikitext-103. We evaluate BSFA on LLaMA (Touvron et al., 2023), a decode-only Transformer with RoPE (Su et al., 2024), SwiGLU (Shazeer, 2020), and RMSNorm (Zhang and Sennrich, 2019), pre-trained on Wikitext-103 (103M tokens, 28K articles). The 72M-parameter model has 16 layers, 10 heads, hidden size 410, sequence length 150, batch size 240. We use AdamMini (Zhang et al., 2024b) with $(\beta_1, \beta_2, \lambda) = (0.9, 0.95, 0.1)$, learning rate $\eta = 5 \times 10^{-4}$, total steps 40 000, 500-step warmup to `lr_max` then cosine decay to `lr_max`/20, and gradient clipping 1.0. AdamMini and AdamMini-BSFA share this schedule. For AdamMini-BSFA,

BPPE estimates a rank-30 subspace using $l = 40$, $T = 10$, and we set $\alpha = 0.5, \gamma = 1.0$ to mitigate loss spikes. All LLaMA experiments run on four NVIDIA RTX 4090 GPUs.

A.3 Experiment Details in Section 5

Figure 5a: ViT-S/16 on ImageNet-1k. We evaluate BSFA on training a ViT-S/16 model on ImageNet-1k. We adopt the experimental protocol established by Beyer et al. (2022). Specifically, we adopt the original `timm` implementation and apply RandAugment and Mixup (level 10, probability 0.2). The default optimizer is AdamMini with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay $\lambda = 10^{-4}$ (Beyer et al., 2022). We set the batch size to 1024 and train for 90 epochs (or 45 epochs for AdamW-BSFA), each run including 10,000 warm-up steps. The learning-rate schedule comprises a linear warm-up to `lr_max` = 10^{-3} followed by cosine decay. For AdamW-BSFA, we retain the same configuration and train for 45 epochs. We use BPPE to estimate the dominant subspace with $k = 50$, $l = 60$, and $T = 10$, and tune $\alpha \in \{1, 0.5, 0.2\}$, $\gamma \in \{2, 4, 6\}$ via grid search. All ViT-S experiments run on a single NVIDIA RTX 4090 GPU.

Figure 5b: LLaMA(72M) on Wikitext-103. In this experiment we evaluate the acceleration of BSFA when applied to AdamMini. The model and task are identical to Appendix A.2. We optimize with AdamW using $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay $\lambda = 0.1$, and run for 50,000 or 100,000 steps. The learning-rate schedule includes a 500-step warm-up followed by cosine decay from `lr_max` to `lr_min` = `lr_max`/20, with gradient clipping at norm 1.0. We tune `lr_max` for AdamW over $\{2 \times 10^{-4}$, 4×10^{-4}, 6×10^{-4}, 8×10^{-3}, $1.0 \times 10^{-3}\}$ as shown in Figure 8.

For AdamMini we follow the partitioning of Zhang et al. (2024b); Figure 5b compares AdamMini and AdamW under identical settings, indicating similar performance with AdamMini slightly behind. AdamW and AdamW-BSFA share the same learning-rate schedule. For AdamW-BSFA, we use BPPE with $k = 50$, $l = 60$, and $T = 10$, and tune $\alpha \in \{1, 0.5, 0.2\}$, $\gamma \in \{3, 4, 6\}$. Representative hyperparameter pairs are displayed in Figure 9. All LLaMA experiments use the Huggingface LLaMA implementation on four NVIDIA RTX 4090 GPUs.

Figure 5c: LLaMA(134M) on OpenWebText. We evaluate BSFA on a 6-layer LLaMA(134M) model (16 heads per layer, hidden size 768) trained on the OpenWebText corpus, with maximum sequence length 1,024 and batch size 480 (following nanoGPT codebase[1] and Liu et al. (2023)). We optimize with AdamMini using the same hyperparameters of AdamW, where$(\beta_1, \beta_2, \lambda) = (0.9, 0.95, 0.1)$, run for 50,000 or 100,000 steps (including 1,000 warm-up steps), and apply gradient clipping (norm 1.0). The learning-rate schedule consists of a 1 000-step linear warm-up to `lr_max` followed by cosine decay to `lr_min` = `lr_max`/20; we tune `lr_max` over $\{3 \times 10^{-4}$, 6×10^{-4}, 1.2×10^{-3}, $1.8 \times 10^{-3}\}$ and identify 6×10^{-4} as optimal (The result is similar to Figure 8, so we don't show them again). AdamMini and AdamMini-BSFA share this schedule. For AdamMini-BSFA, BPPE estimates the dominant subspace with rank $k = 50$, $l = 60$ historical updates and interval $T = 10$, and we grid-search $\alpha \in \{1, 0.5, 0.2\}$ and $\gamma \in \{2, 4, 6, 8\}$ (The result is similar to Figure 9, so we don't show them again). All experiments use the Huggingface LLaMA codebase on four NVIDIA RTX 4090 GPUs.

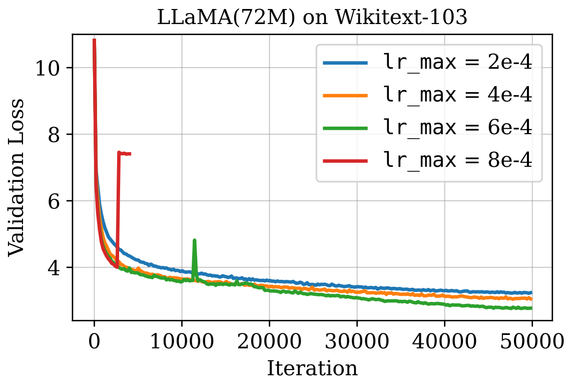


Figure 8: Tuning `lr_max` for LLaMA(72M) on Wikitext-103.

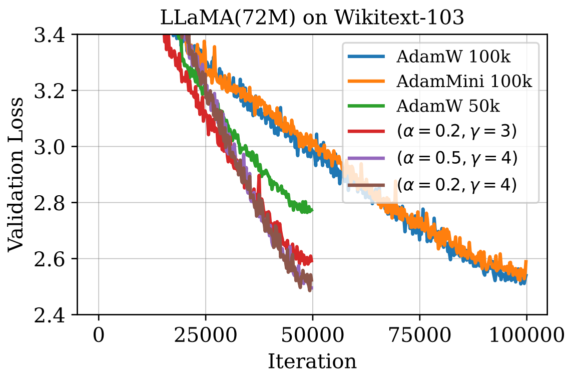


Figure 9: Tuning hyperparameter for AdamMini-BSFA.

[1]https://github.com/karpathy/nanoGPT

Network&Task	k	l	$\|\|\mathcal{D}\|\|$	LPE Time	**PPE Time**
Transformer&SST2	2	10	1000	10.28s	**0.12s** ($\downarrow$ $\mathbf{98.83\%}$)
Resnet18&CIFAR10	10	20	5000	162.1s	**0.42s** ($\downarrow$ $\mathbf{99.74\%}$)
ViT-Tiny&ImageNet-1k	20	50	5000	2002s	**3.27s** ($\downarrow$ $\mathbf{99.84\%}$)

Table 5: Average wall-clock time (in seconds) on a single RTX4090. Here, k denotes the number of eigenvectors(For both algorithms), l indicates the number of historical gradients(For PCA estimator), and $||\mathcal{D}||$ represents the size of the abridged dataset(For Lanczos estimator).

B Details on Dominant Subspace Estimation

B.1 Hessian Eigenspectrum Estimation via Lanczos Method

Lanczos method (Pearlmutter, 1994; Papyan, 2018; Ghorbani et al., 2019) is a common strategy for estimating the top-k eigenpairs of the Hessian $\boldsymbol{H}$, which we refresh every T optimizer steps to control overhead. In practice, this involves two phases:

1. **HVP construction.** On a small "abridged" dataset perform one forward pass and two backward passes to define the Hessian–vector product (HVP) operator $\boldsymbol{v} \mapsto \boldsymbol{H}\boldsymbol{v}$.

2. **Lanczos iterations.** Starting from a random unit vector q_1, apply the HVP operator and orthogonalize against previous basis vectors to build a tridiagonal projection. Specifically, for $k = 1, 2, \dots$; do:

$$\begin{aligned}
\tilde{v}_k &= \boldsymbol{H}\, q_k, \\
\alpha_k &= q_k^\top \tilde{v}_k, \\
r_k &= \tilde{v}_k - \alpha_k\, q_k - \beta_{k-1}\, q_{k-1}, \\
\beta_k &= \|r_k\|, \\
q_{k+1} &= r_k/\beta_k.
\end{aligned}$$

The resulting tridiagonal matrix can be diagonalized cheaply to yield approximations $\{\lambda_i, u_i\}_{i=1}^k$. Based on this, we propose the Lanczos-based Projector Estimator (LPE), summarized in Algorithm 4.

Algorithm 4 Lanczos-based Projector Estimator (**LPE**)

```
1: **Input:** Neural network function $\mathcal{F}$,Number of eigenvalues $k$, Abridged Dataset $\mathcal{D}$, Domspace scaler $\alpha$, Bulkspace scaler $\gamma$
2: $\{\lambda_i\}_{i=1}^k, \{u_i\}_{i=1}^k \leftarrow \text{Lanczos}(\mathcal{F}, \mathcal{D}, k)$
3: $\mathcal{P}_{\alpha,\gamma} \leftarrow \alpha \sum_{i=1}^k \boldsymbol{u}_i\boldsymbol{u}_i^\top + \gamma(\boldsymbol{I} - \sum_{i=1}^k \boldsymbol{u}_i\boldsymbol{u}_i^\top)$
4: **Return:** $\mathcal{P}_{\alpha,\gamma}$
```

B.2 Wall-time Comparison of PPE and LPE

We compare the runtimes of the PPE and LPE in Table 5, which shows that the PPE's computation time is negligible compared to the LPE's. This is because the PPE does not involve the forward or backward propagation of the neural network.

C Proof of Proposition 1

First, we restate our proposition for readability.

Proposition 2 (Gradient–PCA recovers the top eigenspace)**.** *Let $A \in \mathbb{R}^{d\times d}$ be symmetric and positive semi–definite with distinct eigenvalues*

$$\lambda_1 > \lambda_2 > \cdots > \lambda_k > \lambda_{k+1} = \lambda_{k+2} = \cdots = \lambda_d \ (:= \lambda_{\text{tail}} \geq 0),$$

and corresponding orthonormal eigenvectors $v_1, \dots, v_d$. Define the target subspace $E_k = \operatorname{span}\{v_1, \dots, v_k\}$.

Choose a stepsize $\eta > 0$ that satisfies

$$\eta\lambda_k > 1, \qquad 0 < \eta\lambda_{\text{tail}} < 1, \qquad \eta(\lambda_k + \lambda_{\text{tail}}) > 2. \tag{1}$$

Run gradient descent on the quadratic $f(x) = \frac{1}{2}x^\top Ax$ from an initial point $x_0 = \sum_{j=1}^d c_j v_j$ with $c_j \neq 0$ for $1 \leq j \leq k$:

$$x_{s+1} = x_s - \eta A x_s, \qquad s = 0, 1, 2, \dots$$

and denote the gradients $g_s = \nabla f(x_s) = Ax_s$. For an integer window length $l \geq k$ form the data matrix

$$G_t = [\, g_t,\ g_{t+1},\ \dots,\ g_{t+l-1} \,] \in \mathbb{R}^{d\times l}, \qquad t = 0, 1, 2, \dots$$

Then, as $t \to \infty$, the k leading left singular vectors of G_t (equivalently, the top-k eigenvectors of $G_t G_t^\top$) converge to an orthonormal basis of E_k. For simplicity, we consider this uncentered case, i.e., a direct SVD analysis of the gradient matrix; we believe this conclusion can be naturally transferred to the centered version.

Proof. **1. Closed forms.** Write $x_0 = \sum_{j=1}^d c_j v_j$. Since $Av_j = \lambda_j v_j$, one step of gradient descent gives $x_{s+1} = (I - \eta A)x_s$ then we have

$$x_s = (I - \eta A)^s x_0 = \sum_{j=1}^d c_j \mu_j^s v_j,$$

where $\mu_j := 1 - \eta\lambda_j$. This implies

$$g_s = Ax_s = \sum_{j=1}^d \alpha_j \mu_j^s v_j,$$

with $\alpha_j := c_j\lambda_j$.

2. Properties of μ_j. From the stepsize conditions (1), we have

$$\mu_1 < \mu_2 < \cdots < \mu_k < 0, \qquad 0 < \mu_{k+1} = \cdots = \mu_d < 1.$$

Since $\mu_k < 0$, we have $|\mu_k| = \eta\lambda_k - 1 > 0$. Since $\mu_{k+1} > 0$, we have $|\mu_{k+1}| = \mu_{k+1} = 1 - \eta\lambda_{\text{tail}}$.

Critically, from the third condition $\eta(\lambda_k + \lambda_{\text{tail}}) > 2$, we can derive:

$$\eta\lambda_k - 1 > 1 - \eta\lambda_{\text{tail}}$$

which means $|\mu_k| > \mu_{k+1}$. Combined with the order of μ_j values, we have:

$$|\mu_1| > |\mu_2| > \cdots > |\mu_k| > \mu_{k+1} = |\mu_{k+1}| = \cdots = |\mu_d|.$$

This ensures that $|\mu_{k+1}|/|\mu_k| < 1$, which is essential for convergence.

3. Factorization of G_t. Define $w_j \in \mathbb{R}^l$ and $w_j = (1, \mu_j, \dots, \mu_j^{l-1})^\top$. Then

$$G_t = \sum_{j=1}^d \alpha_j \mu_j^t v_j w_j^\top = V\Sigma_t W^\top,$$

where $V = [v_1, \ldots, v_d]$, $\Sigma_t = \mathrm{diag}(\alpha_j \mu_j^t)_{j=1}^d$, and $W = [w_1, \ldots, w_d]$. Consequently

$$G_t G_t^\top = V M^{(t)} V^\top, \quad \text{where} \quad M^{(t)} = \big(\alpha_i \alpha_j (\mu_i \mu_j)^t (w_i^\top w_j)\big)_{i,j=1}^d.$$

4. Analysis of $M^{(t)}$. Write $M^{(t)}$ in block form

$$M^{(t)} = \begin{pmatrix} M_{11}^{(t)} & M_{12}^{(t)} \\ M_{12}^{(t)\top} & M_{22}^{(t)} \end{pmatrix}$$

where $M_{11}^{(t)} \in \mathbb{R}^{k \times k}$. We can establish the following bounds:

$$\|M_{11}^{(t)}\|_2 \geq C_1 |\mu_k|^{2t},$$

$$\|M_{12}^{(t)}\|_2 \leq C_2 |\mu_k|^t \mu_{k+1}^t,$$

$$\|M_{22}^{(t)}\|_2 \leq C_3 \mu_{k+1}^{2t},$$

where $C_1, C_2, C_3 > 0$ are constants that depend only on $\{c_j, \lambda_j, l\}_{j=1}^d$. Specifically, C_1 depends on the minimum of $|\alpha_j|^2$ for $j \leq k$ and the inner products of the geometric progression vectors, while C_2 and C_3 depend on the maximum values of $|\alpha_i \alpha_j|$ and the corresponding inner products.

From these bounds and using eigenvalue perturbation theory, the eigen-gap between the kth and $(k+1)$th eigenvalues of $M^{(t)}$ satisfies:

$$\sigma_k(M^{(t)}) - \sigma_{k+1}(M^{(t)}) \geq C_4 |\mu_k|^{2t}$$

for some constant $C_4 > 0$, while

$$\|M_{12}^{(t)}\|_2 = O\big(|\mu_k|^t \mu_{k+1}^t\big).$$

5. Subspace Convergence via Davis–Kahan Theorem To establish the convergence of the computed subspace to E_k, we employ the Davis–Kahan $\sin\Theta$ theorem. This theorem bounds the difference between the subspace S_t (derived from $M^{(t)}$) and the target subspace E_k, using the spectral properties of $M^{(t)}$:

$$\|\sin\Theta(S_t, E_k)\|_2 \;\leq\; \frac{\|M_{12}^{(t)}\|_2}{\sigma_k(M^{(t)}) - \sigma_{k+1}(M^{(t)})} \;=\; O\Big(\big|\tfrac{\mu_{k+1}}{\mu_k}\big|^t\Big) \;\xrightarrow[t\to\infty]{}\; 0,$$

where S_t denotes the span of the top-k eigenvectors of $M^{(t)}$ (equivalently of $G_t G_t^\top$). Because V is orthogonal, the corresponding subspace in the original coordinates is generated by the first k left singular vectors of G_t. Therefore $S_t \to E_k$ as claimed. $\square$