NysAct: A Scalable Preconditioned Gradient Descent using Nyström-Based Approximation

Anonymized for blind review

***Abstract*—First-order methods are computationally efficient and exhibit fast convergence rates but often have poorer generalization compared to SGD. In contrast, second-order methods enhance convergence and generalization but typically incur high computational and memory costs. In this work, we introduce NYSACT, a scalable first-order gradient preconditioning method that strikes a balance between traditional first-order and second-order optimization methods. NYSACT leverages an eigenvalue-shifted Nyström method to approximate the activation covariance matrix, which is used as a preconditioning matrix, significantly reducing time and memory complexities with minimal impact on test accuracy. Our experiments show that NYSACT not only achieves improved test accuracy compared to both first-order and second-order methods but also demands considerably less computational resources than second-order methods.**

***Index Terms*—Deep learning optimization, Gradient preconditioning, Nyström approximation**

I. INTRODUCTION

The success of deep learning models heavily depends on optimization strategies, with gradient-based methods being crucial for effective training. Gradient preconditioning has gained traction for its ability to accelerate convergence by adjusting gradients during training. First-order methods such as stochastic gradient descent with momentum (SGD) [25] and Adam(W) [16, 20] are popular for their computational efficiency, with Adam(W) using adaptive learning rates based on the second moments of gradients. However, despite their low per-iteration cost, their convergence is often slow.

Second-order methods can improve the convergence by preconditioning gradients to make them effective in navigating ill-conditioned loss landscapes [8, 10, 18]. However, their computational overhead is often prohibitive, especially in large-scale deep learning tasks. Directly leveraging the Hessian matrix [6, 29] requires double backpropagation, significantly increasing time and memory demands. To improve efficiency, methods like KFAC [21] and Shampoo [10] approximate the (empirical) Fisher information matrix (FIM), instead of the Hessian, and decompose it into the Kronecker product of smaller matrices but still result in longer training times compared to SGD. For example, in our experiments with ResNet-110 on CIFAR-100 using a single GPU, AdaHessian [32] took on average 46.33 seconds per epoch, KFAC 21.75 seconds, and Shampoo 200.63 seconds, while SGD took only 8.88 seconds.

In this work, we propose a novel preconditioned optimizer, called NYSACT, whose performance is as good as that of second-order methods while having significantly less computation and memory requirement. KFAC approximates the FIM $\mathbf{F} \in \mathbb{R}^{mn \times mn}$ of a layer with the Kronecker product of two matrices: $\mathbf{F} = \mathbf{P} \otimes \mathbf{A}$, where $\mathbf{P} \in \mathbb{R}^{m \times m}$ and $\mathbf{A} \in \mathbb{R}^{n \times n}$ are covariance matrix of pre-activation gradients and activations, respectively. A recent work [1] empirically found that the term $\mathbf{P}$ makes no contribution to the high performanace of KFAC and proposed an optimizer, called FOOF, that replaces $\mathbf{P}$ with an identity matrix $\mathbf{I}$. Based on this observation, NYSACT chooses to use activation covariance matrix as a preconditioner. While only computing and maintaining $\mathbf{A}$ saves both computation and memory space, FOOF still suffers from high complexity of matrix operations on $\mathbf{A}$, rendering it impractical for large neural networks.

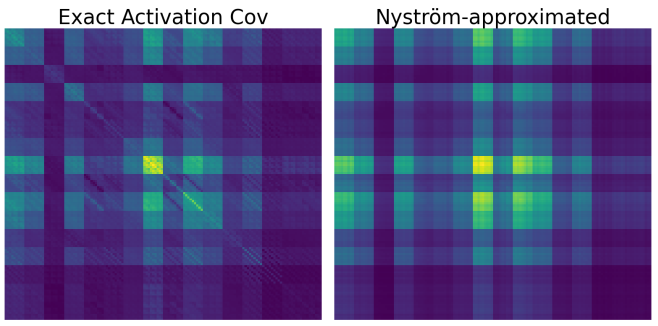
To further improve scalability, NysAct approximates the activation covariance matrix $\mathbf{A}$ using the eigenvalue-shifted Nyström method [24, 30]. Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and a fixed rank $r \ll n$, the Nyström method requires only $\mathcal{O}(r^3 + nr^2)$ time and $\mathcal{O}(nr)$ memory to compute the approximation of $\mathbf{A}^{-1}$. In contrast, low-rank approximations that use singular value decomposition (SVD) have time and memory complexities of $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively. The eigenvalue-shifted Nyström method due to Tropp et al. [30] is specifically adapted for positive semi-definite matrices and offers a sharp approximation error bound, making it an effective choice for scalable gradient preconditioning in deep learning. We also make an important observation. There are two commonly used sampling methods for Nystöm approximation, uniform column sampling without replacement and Gaussian sketching. In our experiments, we compare the efficacy of these two sampling methods in the context of curvature matrix approximation and found that both methods perform similarly when the input dataset is small to medium scale dataset but, then the input is large-scale, the Gaussian sketching methods yielded inferior performance than the subcolumn sampling method.

To demonstrate the effectiveness of the Nyström method in approximating the activation covariance matrix, we trained ResNet-32 model [11] on CIFAR-100 [17] dataset for 100 epochs using SGD with a mini-batch size of 128. Figure 1 shows the heatmaps of actual and Nyström approximated covariance matrices. As shown, the Nyström method has an ability to recover the whole covariance matrix from the randomly sampled subset of r columns.

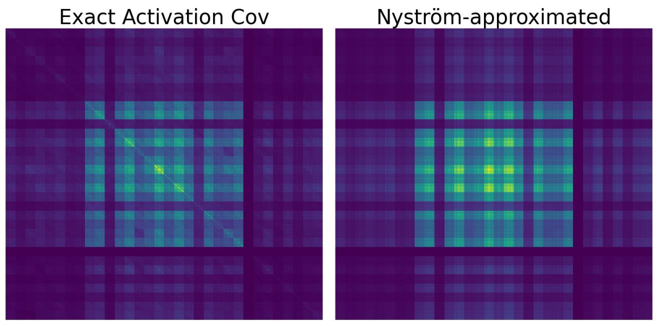
A. Contributions

The key contributions of this paper are summarized as follows.

Scalable Gradient Preconditioning. We introduce NYSACT, a scalable gradient preconditioning method that significantly reduces computational costs while maintaining a minimal



(a) Layer1.2.Conv1



(b) Layer2.1.Conv1

Fig. 1: Comparison of exact activation covariance and Nyström-approximated activation covariance in ResNet-32 architecture trained on CIFAR-100 dataset

compromise in performance. By integrating the Nyström approximation with activation covariance, our method strikes a balance between efficiency and accuracy, making it well-suited for large-scale deep learning tasks.

Extensive Experimental Validation. We provide extensive experimental results that demonstrate the effectiveness of NysAct in image classification tasks. Our experiments, conducted across various network architectures on CIFAR and ImageNet datasets, show that NysAct not only achieves higher test accuracy than both traditional first-order and second-order gradient preconditioning methods but also requires less time and memory resources compared to second-order methods.

II. Related Work

SketchySGD [6] utilizes the Nyström method [31] to approximate the Hessian matrix with a low-rank representation, relying on mini-batch Hessian-vector products. However, SketchySGD requires double backpropagation to compute these approximations, leading to substantial memory and time overhead. In our experiments training ResNet-110 on the CIFAR-100 dataset, SketchySGD averaged 58.14 seconds per epoch and consumed 21,890 MB of memory with a mini-batch size of 128 on a single GPU. In contrast, NysAct took only 11.32 seconds and 1,142 MB of memory. While SketchySGD is most closely related to NysAct, we excluded it from our baselines due to resource constraints. Other notable methods based on low-rank approximation on preconditioner includes MFAC [7], SKFAC [27], and Eva [35]. MFAC introduced rank-1 approximations for estimating inverse-Hessian vector products, employing iterative conjugate gradient solvers. However, this method necessitates multiple forward and backward passes, which significantly raises both computational and memory requirements. SKFAC proposed a low-rank formulation for the inverse of FIM using the Sherman-Morrison-Woodbury formula. It stores both the activation covariance and pre-activation gradient covariance matrices as used in KFAC, requiring the inversion of both matrices, whereas NysAct stores only sketched activation covariance matrices. Among KFAC's low-rank approximation variants, Eva is the most efficient method that preserves KFAC's original performance. Eva computes and stores batch-averaged activation and pre-activation gradient vectors, and updates the inverse of the approximated FIM using the Sherman-Morrison formula. However, as demonstrated in [1], KFAC's effectiveness as a second-order method is primarily driven by the activation term, rather than the pre-activation gradient term.

III. Preliminaries

A. Notations

The set $1, 2, \ldots, N$ is represented by $[N]$. The vectorization of a matrix $\mathbf{M} \in \mathbb{R}^{p\times q}$, denoted as $\mathrm{vec}(\mathbf{M})$, converts $\mathbf{M}$ into a vector $\mathrm{vec}(\mathbf{M}) \in \mathbb{R}^{pq}$, arranged as $\mathrm{vec}(\mathbf{M}) = \begin{bmatrix}\mathbf{M}_{*,1}^{\intercal} & \mathbf{M}_{*,2}^{\intercal} & \cdots & \mathbf{M}_{*,n}^{\intercal}\end{bmatrix}^{\intercal}$, where $\mathbf{M}_{*,j}$ denotes the j^{th} column of matrix $\mathbf{M}$. For any matrix $\mathbf{M}$, we denote the set of its eigenvalues by $\lambda(\mathbf{M})$ and the set of its singular values by $\sigma(\mathbf{M})$, both assumed to be sorted in descending order. We denote the Kronecker product by $\otimes$.

B. Setup for Architecture and Training

Consider a network $f(\mathbf{x};\boldsymbol{\theta})$ composed of L layers, trained on a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$. For each layer $l \in [L]$, let $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ represents the weight matrix, and $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$ represents the bias vector. The forward propagation of f is defined as follows:

$$\begin{aligned}
&\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \in \mathbb{R}^{d_l}\,,\\
&\mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)}) \in \mathbb{R}^{d_l}\,, \qquad \mathbf{a}^{(0)} = \mathbf{x}\,,\\
&\boldsymbol{\theta}^{(l)} = [\mathrm{vec}(\mathbf{W}^{(l)}), \mathbf{b}^{\intercal}]^{\intercal} \in \mathbb{R}^{d_l(d_{l-1}+1)}\,,\\
&\boldsymbol{\theta} \in [(\boldsymbol{\theta}^{(1)})^{\intercal}, \ldots, (\boldsymbol{\theta}^{(L)})^{\intercal}]^{\intercal} \in \mathbb{R}^{p}\,,
\end{aligned}$$

where $\mathbf{z}$ denotes the pre-activations, $\mathbf{a}$ represents the activations, and ϕ is the activation function. For convolutional layers, similar to KFAC, we employed patch extraction to unfold the input into patches, transforming the additional axes into a format compatible with matrix operations.

We consider training a deep neural network that takes an input $\mathbf{x}$ and produces an output $f(\boldsymbol{\theta};\mathbf{x})$, where $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable and possibly nonconvex in $\boldsymbol{\theta}$. Given training examples $\mathcal{D}$, we aim to learn the network parameters $\boldsymbol{\theta}$ by minimizing the empirical loss $\mathcal{L}$ over the training set:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}) := \frac{1}{n}\sum_{i=1}^{n} \ell\left(f(\mathbf{x}_i;\boldsymbol{\theta}), y_i\right), \tag{1}$$

where ℓ is a loss function.

C. FIM-based Gradient Preconditioning

To solve the problem (1), KFAC approximates the FIM with a Kronecker product of smaller matrices as $(\mathbf{F})_{i,j} = \mathbf{A}_{i-1,j-1} \otimes \mathbf{P}_{i,j}$, where $\mathbf{A}_{i,j} = \mathbb{E}\left[\mathbf{a}^{(i)}(\mathbf{a}^{(j)})^\intercal\right]$ denotes the activation covariance from layer i and j, and $\mathbf{P}_{i,j} = \mathbb{E}\left[\frac{\partial\mathcal{L}}{\partial\mathbf{z}^{(i)}}\frac{\partial\mathcal{L}}{\partial\mathbf{z}^{(j)}}^\intercal\right]$ represents the pre-activation gradient covariance between layer i and j. Assuming the independence between layer i and j for $i \neq j$, KFAC computes the diagonal blocks of FIM only, denoted by $\mathbf{A}^{(l-1)} \otimes \mathbf{P}^{(l)} = \mathbf{A}_{l-1,l-1} \otimes \mathbf{P}_{l,l}$, which results in the following update rule for layer l at iteration k.

$$\begin{aligned}\boldsymbol{\theta}_{k+1}^{(l)} &= \boldsymbol{\theta}_k^{(l)} - \eta_k(\mathbf{A}_k^{(l-1)} \otimes \mathbf{P}_k^{(l)})^{-1}\mathbf{g}_k^{(l)} \\ &= \boldsymbol{\theta}_k^{(l)} - \eta_k \operatorname{vec}\left((\mathbf{P}_k^{(l)})^{-1}\mathbf{G}_k^{(l)}(\mathbf{A}_k^{(l-1)})^{-1}\right), \end{aligned} \tag{2}$$

where $\mathbf{G}$ represents the gradient of the loss with respect to the parameters, and $\mathbf{g}$ denotes the gradient in vectorized form. A notable scalable KFAC variant recently proposed is Eva which has following update rule:

$$\boldsymbol{\theta}_{k+1}^{(l)} = \boldsymbol{\theta}_k^{(l)} - \eta_k \operatorname{vec}\left((\tilde{\mathbf{P}}_k^{(l)})^{-1}\mathbf{G}_k^{(l)}(\tilde{\mathbf{A}}_k^{(l-1)})^{-1}\right),$$

where $\tilde{\mathbf{A}} = \mathbb{E}\left[\mathbf{a}^{(i)}\right]\mathbb{E}\left[\mathbf{a}^{(j)}\right]^\intercal$ and $\tilde{\mathbf{P}} = \mathbb{E}\left[\frac{\partial\mathcal{L}}{\partial\mathbf{z}^{(i)}}\right]\mathbb{E}\left[\frac{\partial\mathcal{L}}{\partial\mathbf{z}^{(j)}}\right]^\intercal$. The update rule for FOOF is given by substituting $\mathbf{P}$ in (2) into the identity matrix $\mathbf{I}$:

$$\boldsymbol{\theta}_{k+1}^{(l)} = \boldsymbol{\theta}_k^{(l)} - \eta_k \operatorname{vec}\left(\mathbf{G}_k^{(l)}(\mathbf{A}_k^{(l-1)})^{-1}\right).$$

D. Nyström Method

The Nyström method [31] is a well-established technique for constructing low-rank approximations of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ by selecting a subset of its columns. Specifically, let $\mathbf{S} \in \mathbb{R}^{n\times r}$ be a matrix that randomly samples $r \ll n$ columns of $\mathbf{A}$, where each column of $\mathbf{S}$ is a vector having one entry equal to 1 and all other entries are 0. Then $\mathbf{AS} \in \mathbb{R}^{n\times r}$ corresponds to the submatrix of $\mathbf{A}$ formed by r randomly sampled columns of $\mathbf{A}$. The Nyström approximation $\mathbf{A}_{\text{nys}}$ of $\mathbf{A}$ is given by

$$\mathbf{A} \approx \mathbf{A}_{\text{nys}} = \mathbf{AS}(\mathbf{S}^\intercal\mathbf{AS})^\dagger\mathbf{S}^\intercal\mathbf{A},$$

where $\mathbf{X}^\dagger$ denotes the Moore-Penrose pseudoinverse. Notice that $\mathbf{A}_{\text{nys}}$ can be obtained by storing $\mathbf{S}^\intercal\mathbf{AS} \in \mathbb{R}^{r\times r}$ and $\mathbf{AS} \in \mathbb{R}^{n\times r}$, which only takes $\mathcal{O}(nr)$ memory space.

There are alternative ways of constructing the sampling matrix $\mathbf{S}$. One way is to sample each entry of $\mathbf{S}$ from the standard Guassian distribution $\mathcal{N}(0,1)$, which is called Gaussian sketching. Instead of sampling columns, the Gaussian sketching randomly projects the points in $\mathbf{A}$ onto a lower dimensional space.

Algorithm 1 NysAct

```
Require: Learning rate $\eta_k$, Momentum $\beta_1$, EMA $\beta_2$, Damping $\rho$, Covariance update frequency $\tau_{cov}$, Inverse update frequency $\tau_{inv}$, Rank $r$
Initialize: Parameter $\boldsymbol{\theta}_0$, Momentum $\mathbf{m}_0 = \mathbf{0}$, Sketching $\mathbf{C}_0 = \mathbf{O}$, Preconditioner $\mathbf{A}_0^{-1} = \mathbf{I}$
1: for $k = 1, 2, 3, \ldots$ do
2:     if Sketch == "Gaussian" then
3:         $\mathbf{S}^{(l)} \in \mathbb{R}^{d_{l-1}\times r} \sim \mathcal{N}(\mathbf{0}, \frac{1}{p}\mathbf{I})$
4:     else if Sketch == "Subcolumn" then
5:         Indices = RandPerm$(d_{l-1})[:r]$
6:         $\mathbf{S}^{(l)} = \mathbf{I}_{[\text{Indices}]} \in \mathbb{R}^{d_{l-1}\times r}$, where $\mathbf{I} \in \mathbb{R}^{d_{l-1}\times d_{l-1}}$
7:     if $(k \mod \tau_{cov}) = 0$ then
8:         $\mathbf{C}_k^{(l)} = \beta_2 \cdot \mathbf{C}_{k-1}^{(l)} + (1-\beta_2)\cdot\mathbf{A}_k^{(l)}\mathbf{S}^{(l)} \in \mathbb{R}^{d_{l-1}\times r}$
9:     if $(k \mod \tau_{inv}) = 0$ then
10:        $\widehat{\mathbf{C}}_k^{(l)} = \mathbf{C}_k^{(l)}/(1-\beta_2^{(k \mod \tau_{cov})})$
11:        $\widehat{\mathbf{C}}_{k,\text{damped}}^{(l)} = \widehat{\mathbf{C}}_k^{(l)} + \rho\cdot\mathbf{S}^{(l)}$
12:        $\mathbf{W}^{(l)} = (\mathbf{S}^{(l)})^\intercal\widehat{\mathbf{C}}_{k,\text{damped}}^{(l)} \in \mathbb{R}^{r\times r}$
13:        Eigendecomposition: $\mathbf{W}^{(l)} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\intercal$
14:        $\mathbf{W}_{\text{shifted}}^{(l)} = \mathbf{Q}\left[\boldsymbol{\Lambda} + \left(|\lambda_{\min}(\mathbf{W}^{(l)})| + \rho\right)\mathbf{I}\right]\mathbf{Q}^\intercal$
15:        Cholesky decomposition: $\mathbf{W}_{\text{shifted}}^{(l)} = \mathbf{L}\mathbf{L}^\intercal$
16:        $\mathbf{X}^{(l)} = \widehat{\mathbf{C}}_{k,\text{damped}}^{(l)}\mathbf{L}^{-1} \in \mathbb{R}^{d_{l-1}\times r}$
17:        Singular value decomposition: $\mathbf{X}^{(l)} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\intercal$
18:        $\widetilde{\boldsymbol{\Sigma}} = \operatorname{diag}\left(\max\left(\boldsymbol{\sigma}^2 - (|\lambda_{\min}(\mathbf{W}^{(l)})| + \rho)\mathbf{1}_r, \mathbf{0}_r\right)\right) \in \mathbb{R}^{r\times r}$
19:        $(\mathbf{A}_k^{(l)})^{-1} = \mathbf{U}_{[:r]}\widetilde{\boldsymbol{\Sigma}}^{-1}\mathbf{U}_{[:r]}^\intercal + \frac{1}{\rho}(\mathbf{I} - \mathbf{U}_{[:r]}\mathbf{U}_{[:r]}^\intercal) \in \mathbb{R}^{d_{l-1}\times d_{l-1}}$
20:    $\mathbf{G}_k = \nabla\mathcal{L}(\boldsymbol{\theta}_k) \in \mathbb{R}^{d_l\times d_{l-1}}$
21:    $\mathbf{m}_k^{(l)} = \beta_1\mathbf{m}_{k-1}^{(l)} - \eta_k \operatorname{vec}\left(\mathbf{G}_k^{(l)}(\mathbf{A}_k^{(l)})^{-1}\right)$
22:    $\boldsymbol{\theta}_k^{(l)} = \boldsymbol{\theta}_{k-1}^{(l)} + \mathbf{m}_k^{(l)}$
```

IV. Algorithm

In this section, we present NysAct as an optimizer to solve the problem (1) and describe each step in the algorithm in detail. The pseudocode of NysAct is provided in Algorithm 1.

Let $\mathbf{G}_k^{(l)}$ be the gradient of $\mathcal{L}$ w.r.t. the parameters of layer l. Withtout the momentum, NysAct performs the following update for each layer $l \in [L]$:

$$\boldsymbol{\theta}_{k+1}^{(l)} = \boldsymbol{\theta}_k^{(l)} - \eta_k(\mathbf{C}_k^{(l)} \otimes \mathbf{I}_m)^{-1}\operatorname{vec}(\mathbf{G}_l^{(l)}),$$

where $\mathbf{C}_k^{(l)}$ is the exponential moving average (EMA) of covariance matrix of activations of $l-1$:

$$\mathbf{C}_k^{(l)} = \beta_2\mathbf{C}_{k-1}^{(l)} + (1-\beta_2)\mathbf{A}_k^{(l-1)}.$$

A. Eigenvalue-shifted Nyström

We initially applied the standard Nyström method [31] in NysAct, but this led to highly unstable training dynamics. Instead, we draw inspiration from the eigenvalue-shifted Nyström method [24, 30] and adapt it with slight modifications to suit deep learning settings. The key modification involves

TABLE I: Comparison of time and memory complexity for updating preconditioner(s)

Method	Time Complexity	Memory Complexity
KFAC	$\mathcal{O}(d_{\text{out}}^3) + \mathcal{O}(d_{\text{in}}^3)$	$\mathcal{O}(d_{\text{out}}^2) + \mathcal{O}(d_{\text{in}}^2)$
EVA	$\mathcal{O}(d_{\text{out}}^2) + \mathcal{O}(d_{\text{in}}^2)$	$\mathcal{O}(d_{\text{out}}) + \mathcal{O}(d_{\text{in}})$
FOOF	$\mathcal{O}(d_{\text{in}}^3)$	$\mathcal{O}(d_{\text{in}}^2)$
NYSACT	$\mathcal{O}(r^3) + \mathcal{O}(d_{\text{in}} \cdot r^2)$	$\mathcal{O}(d_{\text{in}} \cdot r)$

applying the eigenvalue-shifted Nyström method to the damped activation covariance $\mathbf{A}+\rho\mathbf{I}$, where $\rho > 0$ serves as a damping term to ensure that the resulting matrix is positive definite, rather than applying to the raw activation covariance $\mathbf{A}$. This adjustment enhances stability during the inversion process, ensuring that NYSACT is numerically robust.

B. NYSACT

Now, we detail the implementation of NYSACT for deep learning tasks. Line 3 and 6 correspond to two different methods for constructing the sampling matrix. When Gaussian sampling is employed, we refer to the method as NYSACT-G, whereas subcolumn sampling leads to the method denoted as NYSACT-S. In Line 8, we update $\mathbf{C}$, the sketch matrix of $\mathbf{A}$, using an exponential moving average (EMA) and by sampling a subset of inputs for each layer, determined by the sampling matrix $\mathbf{S}$. In Line 11, we add damping ρ to the sketch matrix, and in Line 12, we form a principal submatrix $\mathbf{W}$ of $\mathbf{A}$. Line 14 applies eigenvalue shifting using the smallest eigenvalue $|\lambda_{\min}(\mathbf{W}^{(l)})|$, with an additional ρ added to ensure the positive definiteness of the shifted $\mathbf{W}$. Line 15 through 18 detail the computations for $\mathbf{A} + \rho\mathbf{I} \approx \widehat{\mathbf{C}}_{\text{damped}}\mathbf{W}_{\text{shifted}}^{-1}\widehat{\mathbf{C}}_{\text{damped}}^{\intercal}$. We back-shift the shifted eigenvalues in Line 18. In Line 19, $\mathbf{U}_{[:r]}$ denotes the truncated matrix $\mathbf{U}$, retaining only the first r columns. The resulting preconditioner $\mathbf{A}^{-1}$ thus has a fixed rank of r.

C. Complexities

We compare the asymptotic time and memory costs of preconditioning a layer with a weight matrix of size $(d_{\text{out}} \times d_{\text{in}})$ in Table I. The most computationally intensive steps in NYSACT occur in Line 13, 15, 16, and 17. However, since the rank r is typically much smaller than the dimensions of the weight matrix, NYSACT is expected to achieve significantly lower complexity compared to KFAC and FOOF, providing a more efficient approach for utilizing second-order information in deep learning optimization tasks. Numerical results comparing time and memory costs are presented in Section VI-C.

D. Hyperparamters

While second-order gradient preconditioning methods are often sensitive to hyperparameter choices, particularly the learning rate η, EMA coefficient β_2, and damping factor ρ, NYSACT is more robust to variations in these hyperparameters, allowing for consistent and reliable performance with less effort in hyperparameter tuning. In our experiments, we observed that NYSACT, FOOF, and other KFAC variants perform well

TABLE II: Comparison of relative wall-clock time and memory usage over SGD on CIFAR datasets

Model (# params)	ResNet-32 (0.5M)		ResNet-110 (2M)		DenseNet-121 (8M)	
	Time	Mem	Time	Mem	Time	Mem
KFAC	2.29×	1.05×	2.45×	1.09×	2.18×	1.05×
EVA	1.77×	1.00×	1.71×	1.00×	1.24×	1.00×
FOOF	1.52×	1.04×	1.55×	1.06×	1.67×	1.04×
NYSACT-G	1.33×	1.00×	1.36×	1.00×	1.22×	1.00×
NYSACT-S	1.40×	1.00×	1.31×	1.00×	1.19×	1.00×

when using the same hyperparameters as SGD, such as the learning rate, momentum coefficient β_1, and weight decay. For the remaining hyperparameters, such as β_2, ρ, inverse update frequency τ_{inv}, and rank r, we present an ablation study in Section VI-C.

V. CONVERGENCE ANALYSIS

VI. EXPERIMENTS

We assess the performance of NYSACT on a range of image classification tasks and compare it with other baseline methods. All experiments were conducted using 2 Nvidia RTX6000 GPUs.

A. CIFAR Dataset

Settings. For the CIFAR dataset, we employed ResNet-32, ResNet-110, and DenseNet-121 [14], training each model for 100 and 200 epochs. We used a mini-batch size of 128 and cosine annealing learning rate scheduling [19]. The reported metrics in this section are averaged over 5 independent runs. We compared NYSACT against state-of-the-art first- and second-order optimization methods. Specifically, we include SGD as an essential baseline and Adam and AdamW as first-order methods that precondition gradients using their second moments. We evaluate KFAC and Eva as second-order methods that precondition gradients using approximated FIM. Finally, we include FOOF and NYSACT as first-order methods that employ activation covariance-based gradient preconditioning. Detailed hyperparameter settings for each method are provided in Table VI in Appendix B.

Training Results. The training results overall indicate that NYSACT retains much of FOOF's strong performance with minimal compromise. As shown in Tables III, NYSACT outperforms most other baselines in terms of test accuracy. While Adam and AdamW exhibit faster convergence during the early stages of training, they ultimately achieve lower test accuracy compared to other methods. Second-order methods generally outperform the first-order methods, such as SGD, Adam, and AdamW. However, they show less effective performance in both convergence rate and generalization compared to the activation covariance-based preconditioning methods, FOOF and NYSACT. When comparing NYSACT to FOOF, FOOF delivered the strongest results overall, with NYSACT following closely as a strong second. Given that NYSACT is designed as a scalable alternative to FOOF, this outcome

TABLE III: Test accuracy (%) of ResNet and DenseNet on CIFAR datasets

Dataset	Model / Epoch	ResNet-32 100	ResNet-32 200	ResNet-110 100	ResNet-110 200	DenseNet-121 100	DenseNet-121 200
CIFAR-10	SGD	92.80±0.21	93.57±0.29	93.30±0.24	94.18±0.43	95.33±0.16	95.58±0.13
	ADAM	91.59±0.09	92.28±0.14	92.43±0.08	92.90±0.21	93.11±0.19	93.35±0.16
	ADAMW	90.79±0.16	91.83±0.28	92.33±0.25	93.19±0.19	94.24±0.10	94.56±0.14
	KFAC	93.16±0.17	93.78±0.13	94.35±0.13	94.64±0.10	95.23±0.16	95.57±0.07
	EVA	93.07±0.16	93.65±0.14	94.18±0.10	94.64±0.09	95.30±0.13	95.69±0.11
	FOOF	93.61±0.14†	94.05±0.17†	94.70±0.10†	95.09±0.10†	95.79±0.04†	95.95±0.08†
	NYSACT-G	93.12±0.11	93.68±0.21	94.48±0.09	94.76±0.12	95.53±0.13	95.74±0.10
	NYSACT-S	93.28±0.21‡	93.79±0.22‡	94.53±0.17‡	94.94±0.16‡	95.60±0.19‡	95.83±0.08‡
Dataset	Model / Epoch	ResNet-32 100	ResNet-32 200	ResNet-110 100	ResNet-110 200	DenseNet-121 100	DenseNet-121 200
CIFAR-100	SGD	70.47±0.38	70.67±0.49	71.44±1.90	72.48±1.36	79.63±0.15	80.32±0.24
	ADAM	67.16±0.41	67.90±0.55	70.10±0.45	71.22±0.44	73.48±0.41	73.49±0.21
	ADAMW	65.23±0.16	67.04±1.08	68.88±0.31	70.59±0.37	75.51±0.23	76.30±0.12
	KFAC	70.21±0.34	70.91±0.28	73.10±0.41	74.68±0.33	79.79±0.24	80.16±0.10
	EVA	70.32±0.31	71.11±0.50	73.55±0.33	74.13±0.34	79.32±0.08	79.89±0.27
	FOOF	71.21±0.34†	71.82±0.23†	75.13±0.26†	75.91±0.31†	80.92±0.28†	80.98±0.25†
	NYSACT-G	70.70±0.18	71.14±0.17‡	73.94±0.38‡	74.70±0.19	80.40±0.24‡	80.70±0.34‡
	NYSACT-S	70.86±0.44‡	71.12±0.34	73.76±0.45	75.01±0.16‡	80.33±0.29	80.54±0.17

† and ‡ indicate the best and second-best test accuracies, respectively.

suggests that its approximation of the activation covariance matrix is reasonably effective. In comparison to KFAC and Eva, NYSACT either matches or exceeds their performance in CIFAR-10 training, and it distinctly outperforms in CIFAR-100 training for all networks tested. Figure 2 shows the progression of training loss and test accuracy over 200 epochs on CIFAR-100 dataset. The results for ResNets clearly demonstrate that NYSACT effectively balances the fast convergence of first-order methods with the strong generalization capabilities of second-order methods. For DenseNet-121, NYSACT performs comparably to the other second-order baselines.

Time and Memory Complexities. Table II highlights the computational efficiency of NYSACT. As demonstrated, NYSACT has considerably faster execution time than both FOOF and KFAC, while also using less memory. Notably, NYSACT consistently achieves faster execution times compared to Eva, which relies solely on vector multiplications during the computation of preconditioners, across all tested architectures.

B. ImageNet Dataset

Settings. In our experiments on the ImageNet (ILSVRC 2012) [4] dataset, we trained ResNet-50, ResNet-101, and DeiT Small (DeiT-S) [28] architectures for 100 and 200 epochs. Each training session utilized a mini-batch size of 1,024 and employed a cosine annealing schedule for learning rate adjustment. We evaluated NYSACT against the same baselines used in our CIFAR experiments. For a comprehensive overview of the experimental settings, refer to Table VII and Table **??** in Appendix B.

Training Results. The experimental results on ImageNet dataset, summarized in Table IV.

Why NYSACT-G performs poorly?

TABLE IV: Top-1 accuracy (%) of ResNets and DeiT on ImageNet dataset

Model / Epoch	ResNet-50 100	ResNet-50 200	ResNet-101 100	ResNet-101 200	DeiT-S 100	DeiT-S 200
SGD	78.05	79.46	79.66	81.34	69.08	75.27
ADAMW	76.73	79.14	77.76	80.62	73.78	77.96
KFAC	78.16	79.34	79.53	81.13	69.84	✗
EVA	77.71	79.48	79.55	81.06	69.67	76.57
FOOF	78.37	79.69	79.96	81.04	65.37	
NYSACT-S	75.47				70.72	76.16

✗ indicates a training failure.

TABLE V: Comparison of relative wall-clock time and memory usage over SGD on ImageNet dataset

Model (# params)	ResNet-50 (27M) Time	ResNet-50 (27M) Mem	ResNet-101 (45M) Time	ResNet-101 (45M) Mem	DeiT-S (22M) Time	DeiT-S (22M) Mem
KFAC	1.21×	1.02×	1.25×	1.02×	1.37×	1.03×
EVA	1.02×	1.00×	1.03×	1.00×	1.01×	1.00×
FOOF	1.25×	1.01×	1.31×	1.02×	1.11×	1.02×
NYSACT-S	1.06×	1.00×	1.07×	1.00×	1.03×	1.00×

Time and Memory Complexities. Table V highlights the computational efficiency gains of NYSACT.

C. Ablation Study

We performed a hyperparameter study on NYSACT and compared its results with other gradient preconditioning methods. We selected KFAC, Eva, and FOOF as baselines because they all rely on approximations of FIM and share similar hyperparameters, making them ideal for direct comparison with NYSACT. This analysis was carried out using ResNet-32

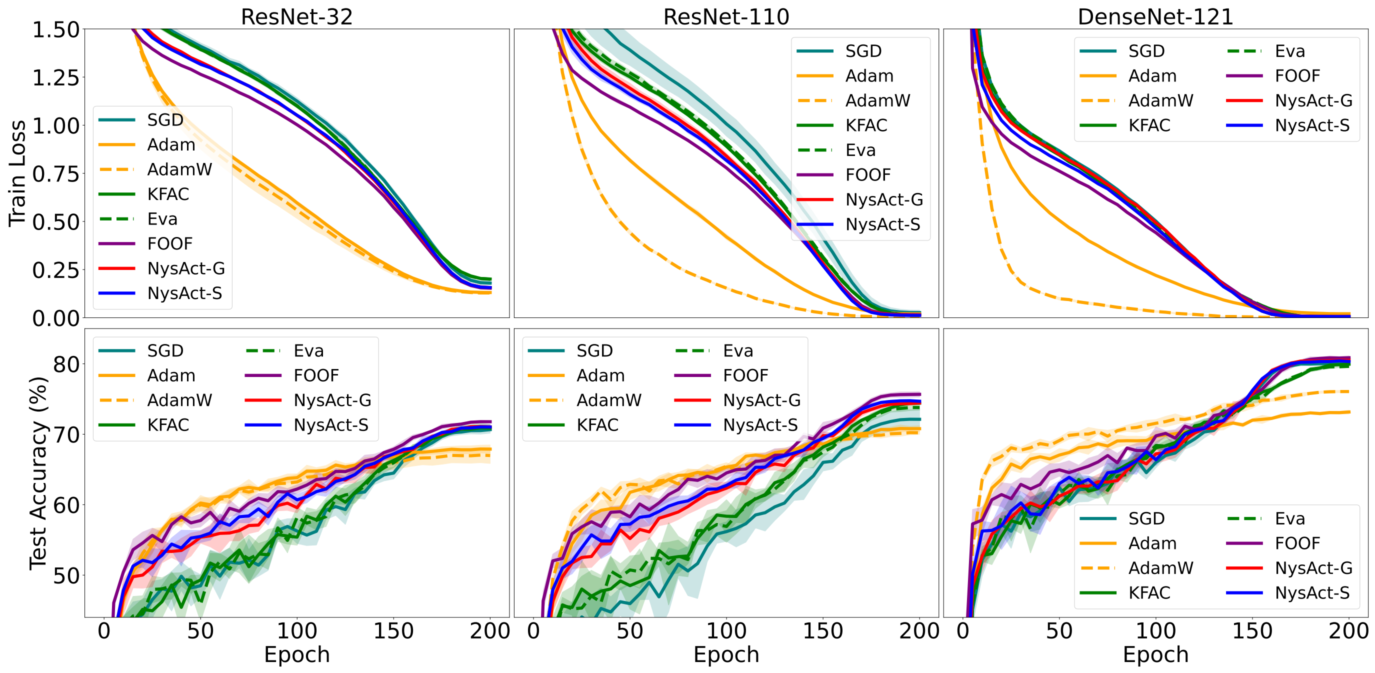


Fig. 2: Comparison of training loss and test accuracy of optimizers on CIFAR-100 dataset

on CIFAR-10 and ResNet-110 on CIFAR-100, each trained for 100 epochs across three different runs.

Essential Hyperparameters. Figure 3 provides a comprehensive comparison of NysAct with KFAC, Eva, and FOOF, focusing on essential hyperparameter tuning. The first subplot highlights that all methods, including NysAct, perform best at a learning rate of 0.1. At the lower end, with a learning rate of 0.001, Eva and FOOF struggle, showing a drop in performance. On the higher end, at a learning rate of 1.0, KFAC and Eva experience significant performance degradation. In contrast, NysAct maintains stable and consistent performance across the entire range of learning rates tested. In the second subplot, NysAct, along with the other methods, demonstrates consistent performance across different EMA coefficients, with the exception of KFAC. This suggests that NysAct's performance remains largely unaffected by changes in the EMA coefficient, whereas KFAC exhibits noticeable fluctuations, particularly at a coefficient value of 0.9. In the third subplot, NysAct's test accuracy increases as damping values rise within the range of 0.1 to 10.0. At a damping value of 0.01, both FOOF and NysAct experience a slight dip in performance relative to KFAC and Eva. However, when the damping value reaches 10.0, KFAC displays significant variability and a marked decline in performance. The observation in the last subplot aligns with the broader trend in optimization, where large-batch training often leads to degraded network performance, as highlighted in previous research [9, 13]. Among the methods compared, NysAct experiences a moderate decrease in test accuracy as the batch size grows, showing a more stable performance relative to other baselines.

Inverse Update Frequencies. In Figure 4, we assessed the effects of varying inverse update frequencies for the preconditioning matrix, testing intervals of 5, 10, 50, and 100 steps. The results suggest that increasing the update frequency does not significantly compromise the test accuracy of NysAct, while it contributes to reducing computational overhead. For update frequencies of 10 steps or more, NysAct achieved the fastest training time while maintaining the second-best test accuracy, just behind FOOF. At a frequency of 5 steps, Eva exhibited the fastest overall training time, with FOOF being the slowest. NysAct demonstrated a slightly faster training time than FOOF. KFAC is absent from this subplot due to its frequent failures in inverting the preconditioners. Notably, at 50 and 100 steps, Eva, despite being the lightest and most scalable variant of KFAC, became slower than FOOF in this settings.

Impact of Rank on NysAct. Figure 5 presents the impact of the rank hyperparameter in NysAct. The subplots on the left display the results for NysAct-G, while those on the right show the results for NysAct-S. In both cases, NysAct outperforms SGD in test accuracy and closely follows FOOF. When comparing the sketching methods, Gaussian sampling exhibits larger variability in test accuracy compared to subcolumns sampling, though both methods achieve similar performance, around 74% test accuracy. As the rank size increases, there is a subtle trend of improved test accuracy for both sampling methods, aligning with the expectation that higher-rank approximations better capture the original matrix's properties. The findings suggest that NysAct effectively approximates the exact activation covariance matrix with low ranks, as evidenced by the minimal difference in test accuracy between rank-5 and rank-20 approximations, with overlapping

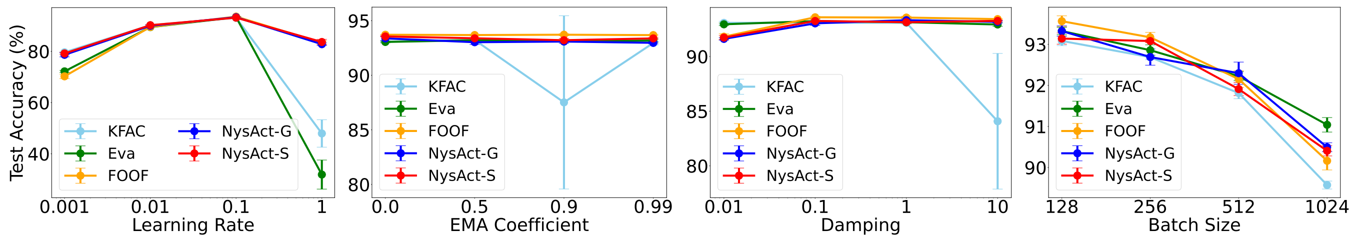


Fig. 3: Comparison of the effects of learning rate, EMA coefficient, damping, and mini-batch size on gradient preconditioning methods, training ResNet-32 on CIFAR-10 for 100 epochs.

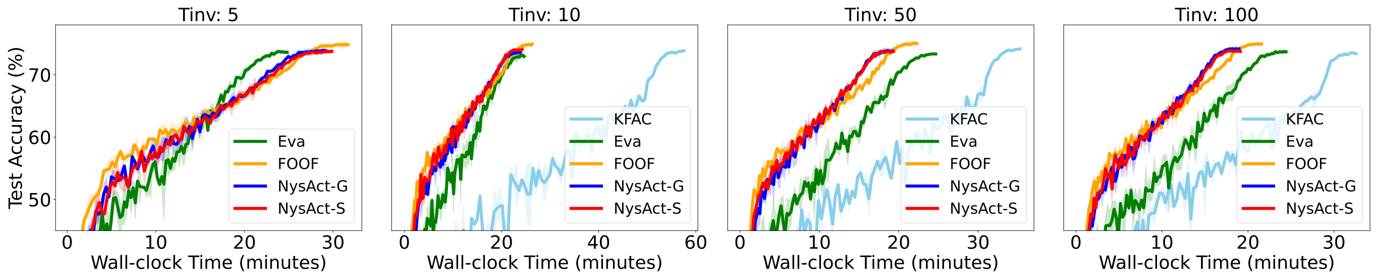


Fig. 4: Comparison of wall-clock time for different inverse update frequencies during ResNet-110 training on the CIFAR-100 dataset

error bars indicating negligible variance.

VII. Conclusion

We introduced NysAct, a scalable gradient preconditioning method that effectively reduces the computational complexity associated with activation covariance-based preconditioning while maintaining a fast convergence rate and strong generalization performance. Our extensive empirical evaluations on image classification tasks demonstrate that NysAct significantly improves end-to-end training time compared to other advanced preconditioning methods such as KFAC, Eva, and FOOF. Furthermore, NysAct delivers better test accuracy compared to first-order methods such as SGD and Adam(W). By addressing the limitations of both first- and second-order methods, NysAct offers an optimal blend between them, making it a scalable yet powerful optimization choice for deep learning tasks.

References

[1] Frederik Benzing. Gradient descent on neurons and its link to approximate second-order optimization. In *Proceedings of the International Conference on Machine Learning*, 2022.

[2] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *ArXiv*, abs/1902.05509, 2019.

[3] Ekin Dogus Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE conference on computer vision and pattern recognition*, 2009.

[5] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *ArXiv*, abs/1708.04552, 2017.

[6] Zachary Frangella, Pratik Rathore, Shipu Zhao, and Madeleine Udell. Sketchysgd: Reliable stochastic optimization via randomized curvature estimates, 2024.

[7] Elias Frantar, Eldar Kurtic, and Dan Alistarh. M-FAC: Efficient matrix-free approximations of second-order information. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

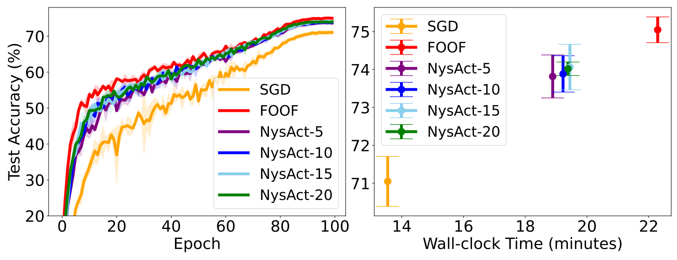
[8] Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks. In *Proceedings of the Conference on Neural Information Processing Systems*, volume abs/2006.08877, 2020.

[9] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *ArXiv*, 2017.

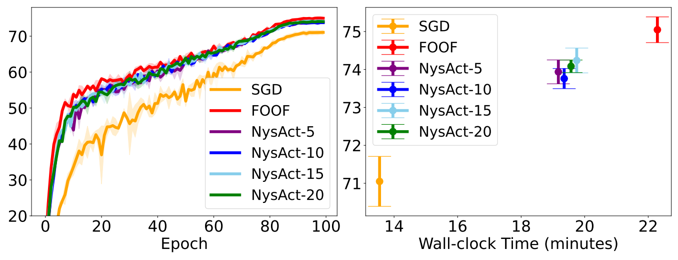
[10] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[12] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch:



(a) NysAct-Gaussian Sampling



(b) NysAct-Subcolumns Sampling

Fig. 5: Comparison of test accuracy and wall-clock time for different ranks of NysAct during ResNet-110 training on the CIFAR-100 dataset. The errorbar plots illustrate the comparison of test accuracy and wall-clock time for each method at their optimal epoch during the 100-epoch training period, along with the corresponding computational time in minutes.

better training with larger batches. *ArXiv*, abs/1901.09335, 2019.

[13] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Neural Information Processing Systems*, 2017.

[14] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[18] Hong Liu, Zhiyuan Li, David Leo Wright Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. In *The Twelfth International Conference on Learning Representations*, 2024.

[19] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations*, 2016.

[20] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations*, 2019.

[21] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the International Conference on Machine Learning*, 2015.

[22] Samuel G. Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 754–762, 2021.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 2019.

[24] Archan Ray, Nicholas Monath, Andrew McCallum, and Cameron Musco. Sublinear time approximation of text similarity matrices. In *AAAI Conference on Artificial Intelligence*, 2021.

[25] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[26] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2015.

[27] Zedong Tang, Fenlong Jiang, Maoguo Gong, Hao Li, Yue Wu, Fan Yu, Zidong Wang, and Min Wang. Skfac: Training neural networks with faster kronecker-factored approximate curvature. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13474–13482, 2021.

[28] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herv'e J'egou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2020.

[29] Hoang Tran and Ashok Cutkosky. Better SGD using second-order momentum. In *Advances in Neural Information Processing Systems*, 2022.

[30] Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Fixed-rank approximation of a positive-semidefinite matrix from streaming data. In *Neural Information Processing Systems*, 2017.

[31] Christopher K. I. Williams and Matthias W. Seeger. Using the nyström method to speed up kernel machines. In *Neural Information Processing Systems*, 2000.

[32] Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer,

and Michael W. Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[33] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Young Joon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[34] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[35] Lin Zhang, Shaohuai Shi, and Bo Li. Eva: Practical second-order optimization with kronecker-vectorized approximation. In *International Conference on Learning Representations*, 2023.

[36] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *AAAI Conference on Artificial Intelligence*, abs/1708.04896, 2017.

APPENDIX

A. Convegence Analysis

In this section, we analyze the convergence properties of NYSACT. To simplify the analysis, we focus on feed-forward networks composed of linear layers, though these results can be extended to other types of layers as well.

We make the following standard assumptions in stochastic optimization.

Assumption A.1. (Smoothness) The loss function $\mathcal{L}$ is continuously differentiable and L-smooth, i.e., for all $\mathbf{W}_1$ and $\mathbf{W}_2$,

$$\|\nabla\mathcal{L}(\mathbf{W}_1) - \nabla\mathcal{L}(\mathbf{W}_2)\|_F \leq L\|\mathbf{W}_1 - \mathbf{W}_2\|_F$$

Assumption A.2. (Gradient Properties) The stochastic gradient is an unbiased gradient estimate of the true gradient:

$$\mathbb{E}\left[\mathbf{G}(\mathbf{W})\right] = \nabla\mathcal{L}(\mathbf{W}),$$

where $\mathbf{G}$ is the gradient in matrix form. The variance of the stochastic gradient is bounded by a constant σ^2:

$$\mathbb{E}\left[\|\mathbf{G}(\mathbf{W}) - \nabla\mathcal{L}(\mathbf{W})\|_F^2\right] \leq \sigma^2.$$

Additionally, the norm of the true gradient is bounded, i.e., there exists a constant $C > 0$ such that

$$\|\nabla\mathcal{L}(\mathbf{W})\| \leq C.$$

Assumption A.3. (Nyström Approximation) The Nyström approximated activation covariance $\tilde{\mathbf{A}}$ of the exact activation covariance $\mathbf{A} = \mathbb{E}[\mathbf{a}\mathbf{a}^\intercal]$ satisfies the approximation bound

$$\|\mathbf{A} - \tilde{\mathbf{A}}\|_F \leq \epsilon\|\mathbf{A}\|_F$$

for some small approximation error $\epsilon > 0$, and the eigenvalues of $\tilde{\mathbf{A}}$ are bounded as $0 \leq \lambda_{\min}(\tilde{\mathbf{A}}) \leq \lambda_i \leq \lambda_{\max}(\tilde{\mathbf{A}})$.

The update rule for NYSACT is given by

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \eta\mathbf{G}_k(\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1}.$$

Using the Assumption A.1, the Taylor expansion of the loss around $\mathbf{W}_k$ gives

$$\mathcal{L}(\mathbf{W}_{k+1}) \leq \mathcal{L}(\mathbf{W}_k) + \langle\mathbf{G}_k, \mathbf{W}_{k+1} - \mathbf{W}_k\rangle + \frac{L}{2}\|\mathbf{W}_{k+1} - \mathbf{W}_k\|_F^2.$$

Substituting $\mathbf{W}_{k+1} - \mathbf{W}_k = -\eta\mathbf{G}_k(\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1}$, we have

$$\mathcal{L}(\mathbf{W}_{k+1}) \leq \mathcal{L}(\mathbf{W}_k) + \langle\mathbf{G}_k, \mathbf{W}_{k+1} - \mathbf{W}_k\rangle + \frac{L}{2}\|\mathbf{W}_{k+1} - \mathbf{W}_k\|_F^2$$

Rearranging terms gives

$$\leq \mathcal{L}(\mathbf{W}_k) - \eta\langle\mathbf{G}_k, \mathbf{G}_k(\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1}\rangle + \frac{\eta L}{2}\|\mathbf{G}_k(\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1}\|_F^2. \tag{3}$$

Since $\tilde{\mathbf{A}}$ can be viewed as a perturbed matrix of $\mathbf{A}$ as $\tilde{\mathbf{A}} = \mathbf{A} + (\tilde{\mathbf{A}} - \mathbf{A})$, where we assume the perturbation norm $\|\tilde{\mathbf{A}} - \mathbf{A}\|_F$ is sufficiently small, the difference between the exact and approximated preconditioners can be obtained using matrix perturbation theory as

$$(\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1} = (\mathbf{A}_k + \rho\mathbf{I})^{-1} - (\mathbf{A}_k + \rho\mathbf{I})^{-1}(\tilde{\mathbf{A}}_k - \mathbf{A}_k)(\mathbf{A}_k + \rho\mathbf{I})^{-1}.$$

This yields the following error bound as

$$\begin{aligned}\langle\mathbf{G}_k, \mathbf{G}_k(\mathbf{A}_k + \rho\mathbf{I})^{-1}\rangle - \langle\mathbf{G}_k, \mathbf{G}_k(\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1}\rangle &= \mathrm{Tr}(\mathbf{G}_k((\mathbf{A}_k + \rho\mathbf{I})^{-1} - (\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1})\mathbf{G}_k^\intercal) \\ &= \mathrm{Tr}(\mathbf{G}_k(\mathbf{A}_k + \rho\mathbf{I})^{-1}(\tilde{\mathbf{A}}_k - \mathbf{A}_k)(\mathbf{A}_k + \rho\mathbf{I})^{-1}\mathbf{G}_k^\intercal)\end{aligned}$$

Since $\mathrm{Tr}(\mathbf{ABC}) \leq \|\mathbf{A}\|_F\|\mathbf{B}\|_2\|\mathbf{C}\|_F$, where $\|\cdot\|_2$ denotes the spectral norm, and $\|\mathbf{AB}\|_2 \leq \|\mathbf{A}\|_2\|\mathbf{B}\|_2$ holds,

$$\begin{aligned}&\leq |\mathbf{G}_k\|_F\|\mathbf{A}_k + \rho\mathbf{I})^{-1}\|_2\|\tilde{\mathbf{A}}_k - \mathbf{A}_k\|_2\|\mathbf{A}_k + \rho\mathbf{I})^{-1}\|_2\|\mathbf{G}_k\|_F \\ &= \|\mathbf{G}_k\|_F^2\|\mathbf{A}_k + \rho\mathbf{I})^{-1}\|_2^2\|\tilde{\mathbf{A}}_k - \mathbf{A}_k\|_2\end{aligned}$$

Assumption A.3 gives

$$\begin{aligned}&\leq \epsilon\|\mathbf{G}_k\|_F^2\|\mathbf{A}_k + \rho\mathbf{I}\|_2^{-2}\|\mathbf{A}_k\|_F \\ &= \frac{\epsilon}{(\lambda_{\max}(\mathbf{A}_k) + \rho)^2}\|\mathbf{G}_k\|_F^2\|\mathbf{A}_k\|_F.\end{aligned}$$

Using the above, we rewrite (3) as

$$\mathcal{L}(\mathbf{W}_{k+1}) \leq \mathcal{L}(\mathbf{W}_k) - \eta\langle\mathbf{G}_k, \mathbf{G}_k(\mathbf{A}_k + \rho\mathbf{I})^{-1}\rangle + \frac{\eta\epsilon}{(\lambda_{\max}(\mathbf{A}_k) + \rho)^2}\|\mathbf{G}_k\|_F^2\|\mathbf{A}_k\|_F + \frac{\eta L}{2}\|\mathbf{G}_k(\tilde{\mathbf{A}}_k + \rho\mathbf{I})^{-1}\|_F^2.$$

B. Experimental Details

Hyperparameter Settings. Table VI outlines the hyperparameters utilized for training on CIFAR and ImageNet datasets. Here, η refers to the learning rate, β_1 and β_2 represent the EMA coefficients for the first and second moments, and ϵ is a small constant added for numerical stability in Adam-based optimizers. In gradient preconditioning methods, β_2 corresponds to the EMA coefficient for preconditioner updates. The damping factor, ρ, controls the regularization of the covariance matrix, while T_{cov} and T_{inv} define the update frequencies for the covariance and inverse matrices, respectively. Lastly, r denotes the approximation rank size utilized in NYSACT. For CIFAR datasets, For KFAC and Eva, hyperparameter values followed the recommendations in [35]. In contrast, for FOOF and NYSACT, we conducted a grid search over learning rates $[0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$ and damping factors $[0.01, 0.05, 0.1, 0.5, 1, 5, 10]$, while keeping the remaining settings consistent with other FIM-based preconditioners.

For ImageNet dataset, we scaled up the learning rate η by a factor of 5 for both SGD and preconditioning methods, using a mini-batch size of $1, 024$, compared to the CIFAR training, while we decrease weight decay to 0.00002. For both ResNets and DeiT, we tripled up the damping for KFAC and Eva to mitigate instability during preconditioner inversion. For all gradient preconditioning methods, we reduced the inversion frequency from 50 to 5, enabling the models to more frequently adjust to the changes in preconditioning matrices, particularly when training DeiT.

TABLE VI: Hyperparameter settings for CIFAR and ImageNet datasets training

Dataset	Optimizer	η	Momentum	β_1	β_2	Weight decay	ϵ	ρ	T_{cov}	T_{inv}	r
CIFAR	SGD	0.1	0.9	.	.	0.0005	.	.	.	.	.
	Adam	0.001	.	0.9	0.999	0.0005	1×10^{-8}	.	.	.	.
	AdamW	0.001	.	0.9	0.999	0.05	1×10^{-8}	.	.	.	.
	KFAC	0.1	0.9	.	0.95	0.0005	.	0.03	5	50	.
	Eva	0.1	0.9	.	0.95	0.0005	.	0.03	5	50	.
	FOOF	0.1	0.9	.	0.95	0.0005	.	1.0	5	50	.
	NYSACT	0.1	0.9	.	0.95	0.0005	.	1.0	5	50	10
Dataset	**Optimizer**	η	**Momentum**	β_1	β_2	**Weight decay**	ϵ	ρ	T_{cov}	T_{inv}	r
ImageNet	SGD	0.5	0.9	.	.	0.00002	.	.	.	.	.
	AdamW	0.001	.	0.9	0.999	0.05	1×10^{-8}	.	.	.	.
	KFAC	0.5	0.9	.	0.95	0.00002	.	0.1	5	50 / 5	.
	Eva	0.5	0.9	.	0.95	0.00002	.	0.1	5	50 / 5	.
	FOOF	0.5	0.9	.	0.95	0.00002	.	1.0	5	50 / 5	.
	NYSACT	0.5	0.9	.	0.95	0.00002 / 0.0001	.	1.0	5	50 / 5	20

Model Settings for ResNets and DeiT. Table VII summarizes the configurations used for training on the ImageNet dataset. Two architectures were explored: ResNet and DeiT-Small. For ResNet, we followed the PyTorch implementation [23], and for DeiT, we applied the settings from [28]. Both models incorporated advanced training techniques such as Random Erasing [36], Label Smoothing [26], Mixup/CutMix [33, 34], and Repeated Augmentation [12]. ResNet used TrivialAugment [22], while DeiT employed RandAugment [3] and Stochastic Depth [15]. Training was conducted at a resolution of 176 for ResNet and 224 for DeiT, with both models evaluated at a test resolution of 224. A mini-batch size of 1,024 was used with cosine learning rate decay and a 5-epoch warmup.

TABLE VII: Settings for ImageNet training

Architecture	ResNets	DeiT
Train Res	176	224
Test Res	224	224
Batch size	1,024	1,024
LR decay	cosine	cosine
Warmup epochs	5	5
Label Smoothing	0.1	0.1
Stochastic Depth	-	0.2
Repeated Augmentation	✓	✓
Horizontal flip	✓	✓
Random Resized Crop	✓	✓
Auto Augmentation	TrivialAugment	RandAugment(9/0.5)
Mixup	0.2	0.8
Cutmix	1.0	1.0
Random Erasing	0.1	0.25