ADAFISHER: ADAPTIVE SECOND ORDER OPTIMIZATION VIA FISHER INFORMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

First-order optimization methods are currently the mainstream in training deep neural networks (DNNs). Optimizers like Adam incorporate limited curvature information by employing the diagonal matrix preconditioning of the stochastic gradient during the training. Despite their widespread, second-order optimization algorithms exhibit superior convergence properties compared to their first-order counterparts e.g. Adam and SGD. However, their practicality in training DNNs are still limited due to increased per-iteration computations and suboptimal accuracy compared to the first order methods. We present AdaFisher–an adaptive second-order optimizer that leverages a block-diagonal approximation to the Fisher information matrix for adaptive gradient preconditioning. AdaFisher aims to bridge the gap between enhanced convergence capabilities and computational efficiency in second-order optimization framework for training DNNs. Despite the slow pace of second-order optimizers, we showcase that AdaFisher can be reliably adopted for image classification, language modelling and stand out for its stability and robustness in hyperparameter tuning. We demonstrate that AdaFisher outperforms the SOTA optimizers in terms of both accuracy and convergence speed.

1 INTRODUCTION

Deep Neural network (DNN) optimization often struggles with the challenge of generalizing across varied architectures and complex data distributions. Current methods such as Adam optimizer (Kingma & Ba, 2017) and its variants (AdamP (Heo et al., 2020), AdaInject (Dubey et al., 2022), AdaBelief (Zhuang et al., 2020) and YOGI Zaheer et al. (2018)) require extensive hyperparameter tuning and often fail to generalize efficiently. In response, DNN training typically minimizes an empirical loss function $\mathcal{L}(\theta)$, updating parameters θ using the expression $\theta_{t+1} = \theta_t - \alpha G_t^{-1} \nabla \mathcal{L}(\theta_t)$ at time step t, where G_t represents curvature information, i.e. Hessian or Fisher Information Matrix (FIM) (Amari & Nagaoka, 2000). The Hessian matrix interfaces with the deterministic Newton method (Martens, 2020), whereas the FIM harmonizes with the Natural Gradient Descent (NGD) approach (Amari & Nagaoka, 2000) as a statistical method. These curvature information crucially optimize the gradient's preconditioning by accurately rescaling and orienting it. This adjustment significantly accelerates convergence by ensuring more direct progress towards minima, thus enhancing training efficiency and reducing the number of required iterations.

While first-order methods such as SGD (Kiefer & Wolfowitz, 1952) simplify G_t by treating it as the identity matrix, second-order methods employ curvature matrices to enhance the optimization process. Although these matrices accelerate convergence by effectively navigating saddle points and swiftly moving towards minima (Foret et al., 2021), they require higher computational resources. In fact, when the number of learnable parameters increases, the curse of dimensionality associated with curvature matrix G_t makes the entire process completely intractable for routine deep learning training tasks. Noteworthy approaches, such as Adagrad (Duchi et al., 2011), Adadelta (Zeiler, 2012), RMSProp (Hinton et al.), and Adam family utilize a simple diagonal approximation of the empirical FIM, which often results in convergence to suboptimal local minima and poor generalization (Wilson et al., 2018; Luo et al., 2019). Advanced methods like AdaHessian (Yao et al., 2021) and Shampoo (Gupta et al., 2018) improve on this by integrating structured matrices such as the diagonal Hessian or tensor-based preconditioners to enhance optimization. However, these second-order approaches including K-FAC (Martens & Grosse, 2020; Eschenhagen et al., 2024) still face challenges of high computational demands, lack generalization and the need for extensive hyperparameter tuning when applied to large models (Ma et al., 2019).

To address these challenges, we present AdaFisher, an adaptive second-order optimizer, as an innovative solution to address the generalization challenges raised in training DNNs. Substituting the second moment of Adam by a novel **diagonal block-Kronecker** approximation of the FIM, it strikes a balance between simplicity and generalization by introducing one additional hyperparameter compared to Adam but fewer than K-FAC, AdaHessian or Shampoo. This feature, combined with comparable memory and time requirements than the first-order methods, enhances AdaFisher's practicality for achieving effective generalization in DNN optimization. As illustrated in Figure 1, AdaFisher not only converges more rapidly but also reaches a superior local minimum by effectively navigating through saddle points compared to its counterparts. Further details regarding the visualization can be found in Appendix C. Our main contributions are as follows: **[C1]** We empirically showcase the diagonal dominance of Kronecker factors' energy and provide fresh insights of FIM in optimization; **[C2]** We introduce a diagonal block-Kronecker approximation of the FIM applicable to various layers, including normalization layers, enhancing model adaptability; **[C3]** We demonstrate AdaFisher's robustness and stability across diverse settings, proving its effectiveness; **[C4]** We showcase AdaFisher's superior performance against traditional optimizers empirically in image classification and language modeling; **[C5]** We develop a new technique that visualizes trajectories across different optimizers for better understanding of model behavior in the loss landscape. Additionally, we introduce an explainable FIM measure from AdaFisher, enabling comparative analysis of optimizer behavior.

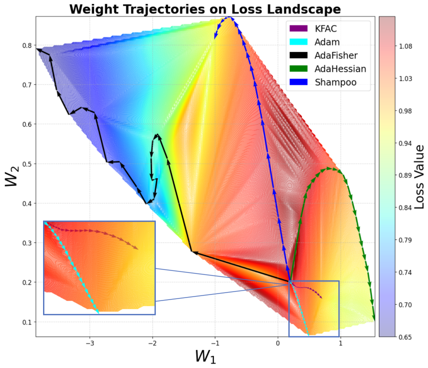

Figure 1: Visualization of optimization paths for various optimizers on a loss surface, comparing their convergence efficiency.

2 BACKGROUND

We consider a supervised learning framework with a dataset $\mathbf{D}$ containing N i.i.d samples, $\mathbf{D} := \{x_n, y_n\}_{n=1}^{N}$ where $x_n \in \mathbb{R}^d$ and $y_n \in \mathbb{R}^C$. Let $f_\theta : \mathbb{R}^d \to \mathbb{R}^C$ be a L-layer neural network parametrized by θ where $\theta_i = concat(W_i, b_i) \in \mathbb{R}^{P_i}$, and $P_i = P_i^{out} \times (P_i^{in} + 1)$. Let $\mathcal{L} : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ be the loss function defined by negative log likelihood, i.e. $\mathcal{L}(y, f_\theta(x)) := -\log p_\theta(y|x)$ where $p_\theta(y|x)$ is the likelihood of the neural network f_θ. The network computes its output $h_L = f_\theta(x)$ according to: $a_i = \theta_i \bar{h}_{i-1}$, $h_i = \phi_i(a_i)$, $\forall\, i \in \{1, \dots, L\} \mid h_0 = x_n$ where $\bar{h}_i = [1, h_i^T]^T \in \mathbb{R}^{P_i^{in}+1}$ terminated by $z := h_L \in \mathbb{R}^{P_L^{out}}$. For a given input target pair (x, y), the gradient of the loss $\mathcal{L}(y, f_\theta(x))$ concerning the weights are computed by the backpropagation algorithm (Lecun, 2001). For convenience, we adopt the special symbol $s_i = \nabla_{a_i}\mathcal{L}$ for the pre-activation derivative. Starting from $\nabla_{h_L}\mathcal{L} = \partial_z \mathcal{L}(y, z = h_L)$, we perform: $s_i := \nabla_{a_i}\mathcal{L} = \nabla_{h_i}\mathcal{L} \odot \phi_i'(a_i)$, $\nabla_{\theta_i}\mathcal{L} = s_i \bar{h}_{i-1}^T$, $\nabla_{\bar{h}_{i-1}}\mathcal{L} = \theta_i^T s_i \quad | \forall i \in \{L, \dots, 1\}$, where $\odot$ denotes the element-wise product. Finally, the gradient $\nabla_\theta \mathcal{L}$ is retrieved by: $\nabla_\theta \mathcal{L} = [\text{vec}(\nabla_{\theta_1}\mathcal{L})^T, \text{vec}(\nabla_{\theta_2}\mathcal{L})^T, \dots, \text{vec}(\nabla_{\theta_L}\mathcal{L})^T]^T$. Optimization of a DNN can be recast as a problem of finding the parameter set θ that maximizes the likelihood, or equivalently, minimizes the negative log-likelihood of the observed data. This Maximum Likelihood Estimation approach can be expressed as an unconstrained optimization problem: $\min_\theta \; J(\theta) = \sum_{n=1}^{N} \mathcal{L}(y_n, f_\theta(x_n))$, where $J(\theta)$ denotes the objective function, corresponding to the negative log-likelihood of the data. The FIM, utilized in lieu of the Hessian for Newton's method (Holmgren, 1996), approximates the curvature of the log-likelihood function (Amari, 1998). It is defined as

$$F = \sum_{n=1}^{N} \mathbb{E}_{y \sim p(y|f_\theta(x_n))} \left[\nabla_\theta \log p_\theta(y|x_n) \nabla_\theta \log p_\theta(y|x_n)^T \right] = \mathbb{E}\left[\nabla_\theta \mathcal{L} (\nabla_\theta \mathcal{L})^T \right], \tag{1}$$

where F measures the expected information that an observable y conveys about the parameter θ. For brevity, we write $\mathbb{E}$ instead of $\mathbb{E}_{y \sim p(y|f_\theta(x_n))}$. The K-FAC approach further simplifies FIM calculation using a block-diagonal approximation in DNNs, known as Empirical FIM (EFIM), denoted $\hat{F}$. In Eq. (1), F is construed as a block matrix with dimensions $L \times L$, where each (i, j)th block

$F_{i,j}$ is articulated by $F_{i,j} = \mathbb{E}[\text{vec}(\nabla_{\theta_i}\mathcal{L})\text{vec}(\nabla_{\theta_j}\mathcal{L})^T]$. By harnessing the vectorization identity $\text{vec}(uv^T) = v \otimes u$, we express $\text{vec}(\nabla_{\theta_i}\mathcal{L})$ as $\bar{h}_{i-1} \otimes s_i$ (Petersen & Pedersen, 2008), where $\nabla_{\theta_i}\mathcal{L}$ is defined as $s_i\bar{h}_{i-1}^T$. By segmenting the FIM into discrete layer-specific blocks, we can effectuate a systematic factorization of each block

$$\hat{F}_{i,j} = \mathbb{E}[\text{vec}(\nabla_{\theta_i}\mathcal{L})\text{vec}(\nabla_{\theta_j}\mathcal{L})^T] = \mathbb{E}[\bar{h}_{i-1}\bar{h}_{j-1}^T \otimes s_i s_j^T] \approx \mathbb{E}[\bar{h}_{i-1}^T\bar{h}_{j-1}] \otimes \mathbb{E}[s_i^T s_j],$$

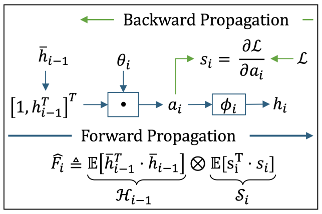
Backward Propagation
$\bar{h}_{i-1}$
θ_i
$s_i = \frac{\partial \mathcal{L}}{\partial a_i}$
$\mathcal{L}$
$[1, h_{i-1}^T]^T$
$\cdot$
a_i
ϕ_i
h_i
Forward Propagation
$\hat{F}_i \triangleq \underbrace{\mathbb{E}[\bar{h}_{i-1}^T \cdot \bar{h}_{i-1}]}_{\mathcal{H}_{i-1}} \otimes \underbrace{\mathbb{E}[s_i^T \cdot s_i]}_{\mathcal{S}_i}$

Figure 2: Illustration of EFIM computation using K-FAC for a given layer i.

where i, j span the layer indices from 1 to L. Here, $\mathbb{E}[\bar{h}_{i-1}^T\bar{h}_{j-1}]$ and $\mathbb{E}[s_i^T s_j]$ are empirically approximated using batch statistics, simplifying the computation for large-scale DNNs (Tang et al., 2021). Notably, $\bar{h}_{i-1} \in \mathbb{R}^{M \times (P_i^{in}+1)}$ and $s_i \in \mathbb{R}^{M \times P_i^{out}}$, where M is the size of a batch, rendering $\hat{F}_{i,j} \in \mathbb{R}^{(P_i^{in}+1)P_i^{out} \times (P_i^{in}+1)P_i^{out}}$. Initially, K-FAC estimates the expectation of the Kronecker product under the presumption that activations and pre-activation derivatives are mutually independent, succinctly represented as the Kronecker product of the individual expectations: $\hat{F}_{i,j} = \mathcal{H}_{i-1,j-1} \otimes \mathcal{S}_{i,j}$, where $\mathcal{H}_{i-1,j-1} = \mathbb{E}[\bar{h}_{i-1}^T\bar{h}_{j-1}]$ and $\mathcal{S}_{i,j} = \mathbb{E}[s_i^T s_j]$, denoting the Kronecker factors. The assumption for the block-diagonal structure posits that weight derivatives across distinct layers are uncorrelated, expressed as: $F \approx \hat{F} = \text{diag}(\hat{F}_{1,1}, \ldots, \hat{F}_{L,L}) = \text{diag}(\hat{F}_1, \ldots, \hat{F}_L)$. Figure 2 shows EFIM computation via K-FAC.

3 METHODOLOGY

Our methodology consists of four primary components: **(i)** Analyzing the Kronecker factors' structure in Section 3.1 and showcase their diagonal dominance; **(ii)** Introducing a novel approximation of the FIM that retains only the diagonals of the Kronecker factors, detailed in Section 3.2; **(iii)** Enhancing the Adam optimizer by incorporating our diagonal FIM approximation as an alternative to the conventional second moment, described in Section 3.3; **(iv)** Providing a theoretical proof of AdaFisher's convergence under both convex and non-convex conditions in Section 3.4.

3.1 DIAGONAL CONCENTRATION OF KRONECKER FACTORS

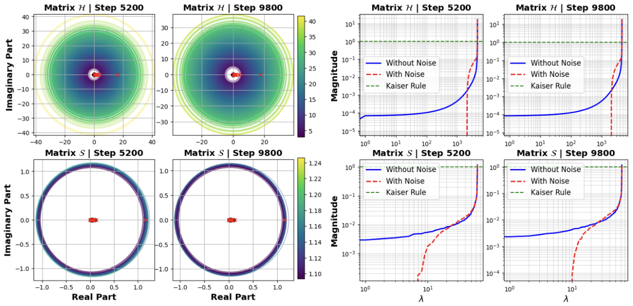

Figure 3: Gersgorin disks and eigenvalue perturbations in the 37th Convolutional Layer of a ResNet-18 at steps 5200 (middle of training) and 9800 (end of training). Left: Gersgorin circles; Right: Eigenvalue spectrum with/without noise.

Inspired by Gersgorin's Circle Theorem (Horn & Johnson, 2012), we empirically conclude the diagonal concentration property of the Kronecker factors from the eigenvalue distribution and its perturbation under Gaussian noise. For demonstration, we focus on the eigenvalue spectrum of weight matrices from the 37th layer of ResNet-18 (He et al., 2015), after training for 50 epochs on CIFAR-10 (Krizhevsky et al., 2009). As Figure 3 illustrates, the eigenvalues (denoted as red crosses) predominantly cluster within the Gersgorin discs, which are centered along the matrix's diagonal elements (denoted as black circles), signifying substantial diagonal dominance. This phenomenon is quantitatively supported by the Gersgorin's Circle Theorem, which posits that every eigenvalue λ of a complex square matrix $\mathcal{A}$ lies within at least one of the Gersgorin discs $D(a_{ii}, R_i)$, where $R_i = \sum_{j \neq i} |a_{ij}|$ represents the radius computed as the sum of the absolute values of the off-diagonal entries of the ith row. Next, we introduce Gaussian noise $\mathcal{N}(0, \sigma^2)$, $\sigma = 10^{-3}$ to the off-diagonal elements. The perturbed matrix $\hat{\mathcal{M}}$ is then expressed as: $\hat{\mathcal{M}} = \mathcal{A} + \mathcal{E}, \quad$ where $\mathcal{E} = [e_{ij}]$ and $e_{ij} \sim \mathcal{N}(0, \sigma^2)$ for $i \neq j$. This allows us to scrutinize the noise-induced perturbation on eigenvalues, which are pivotal for comprehending the dynamics and stability of the system. This demonstrates that the introduction of noise to the off-diagonal elements induces only minimal perturbations in the eigenvalues, particularly those surpassing the

Kaiser criterion, which remain virtually unchanged (Braeken & Van Assen, 2017). This negligible shift corroborates the robustness of the matrix's diagonal dominance. An extensive discussion of this analysis is available in Appendix A.1.

3.2 EFFICIENT COMPUTATION OF THE FIM

In the realm of optimization, NGD offers a geometrically nuanced adaptation of the classical steepest descent approach, transitioning the focus from parameter space to the model's distribution space underpinned by the adoption of a *Riemannian metric*, Amari & Nagaoka (2000). The formulation of NGD is articulated as

$$\triangle\theta_t = F_t^{-1}\nabla J(\theta_t), \tag{2}$$

where F_t denotes the FIM at time t distinguished from F_i, the FIM at layer i. One of the distinguishing features of NGD within this framework is its reparameterization invariance, a direct consequence of leveraging the model's distribution properties rather than its parameters. Nevertheless, the direct computation of the FIM poses significant challenges due to its computational demands.

For the efficient approximation of the FIM, we present the following: (1) a methodology for calculating the Kronecker factors for normalization layers, and (2) diagonal approximation of the Kronecker factors. Beyond conventional layers such as convolutional and linear layers that are well established (Grosse & Martens, 2016; Martens & Grosse, 2020), the Kronecker factors for BatchNorm (Ioffe & Szegedy, 2015) and LayerNorm (Ba et al., 2016) normalization can be derived with Proposition 3.1.

Proposition 3.1. *Consider a neural network layer indexed by i, and a mini-batch $\mathbb{B} \subset \mathbf{D}$ of size M ($|\mathbb{B}| = M$). The empirical statistics of the Kronecker factors for the normalization layers can be characterized by*

$$\mathcal{H}_{i-1} = \frac{\left(\sum_{\mathbb{B}}\sum_{\mathcal{T}}\bar{h}_{i-1}\right)^T\left(\sum_{\mathbb{B}}\sum_{\mathcal{T}}\bar{h}_{i-1}\right)}{(M|\mathcal{T}|)^2},\ \mathcal{S}_i = \frac{\left(\sum_{\mathbb{B}}\sum_{\mathcal{T}}s_i\right)\left(\sum_{\mathbb{B}}\sum_{\mathcal{T}}s_i\right)^T}{M}$$

Here, $\mathcal{T}$ represents the spatial size dimension for Batch Norm layer or for LayerNorm layer, it signifies the product of the number of heads and the per-head dimension.

For the proof of this proposition and the extended computation of other type of layers are given in Appendix A.2 and Section A.3. Note that in the context of online and stochastic optimization, the Kronecker factors for a given layer i can be estimated using an Exponentially Moving Average (EMA) scheme across batches, defined by

$$\mathcal{H}_{i-1} = \gamma\mathcal{H}_{i-2} + (1-\gamma)\hat{\mathcal{H}}_{i-1},\ \mathcal{S}_i = \gamma\mathcal{S}_{i-1} + (1-\gamma)\hat{\mathcal{S}}_i, \tag{3}$$

where $0 < \gamma < 1$ is the exponential decay factor, and $\hat{\mathcal{H}}_{i-1}$, $\hat{\mathcal{S}}_i$ represent the current estimates of the Kronecker factors derived from the latest mini-batch of data. This EMA scheme is commonly used in methods involving diagonal or block-diagonal approximations to the curvature matrix (e.g. LeCun et al. (2012); Park et al. (2000); Schaul et al. (2013). Such schemes have the desirable property that they allow the curvature estimation to depend on much more data than what can be reasonably processed in a single mini-batch.

Investigation from Section 3.1 suggest that the FIM's critical information predominantly resides along its diagonal. Building upon these insights, we propose a novel approximation for the FIM that conceptualizes the Kronecker factors as diagonal matrices, denoted as $\tilde{F}_{D_i}$ for layer i.

Proposition 3.2. *Assume that $\mathcal{H}_{i-1}$ and $\mathcal{S}_i$ can be closely approximated by diagonal matrices, denoted by $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ respectively at layer i, such that $\mathcal{H}_{D_{i-1}} = Diag(\mathcal{H}_{i-1})$, $\mathcal{S}_{D_i} = Diag(\mathcal{S}_i)$ where Diag denote the diagonal of a matrix. Then the Empirical FIM can be defined by*

$$\tilde{F}_{D_i} \triangleq \mathcal{H}'_{D_{i-1}} \otimes \mathcal{S}'_{D_i} + \lambda, \tag{4}$$

where $\mathcal{H}'_{D_{i-1}}$ and $\mathcal{S}'_{D_i}$ denote the Min-Max normalization of $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ (Patro & Sahu, 2015) and λ is a regularization parameter.

This approximation strikes a balance between computational time and space complexity, and the accuracy of performance, as discussed in Section 4. We set the regularization parameter $\lambda = 0.001$, which acts as a damping factor following Tikhonov regularization principles, enhancing computational stability and conditioning of the FIM. For foundational details on λ, refer to Martens & Grosse (2015) and for methodology, see Appendix A.2. The closed-form solution for the augmented gradient $\triangle\theta_t$ is derived from the diagonal approximation of the FIM, given by $\triangle\theta_t = \tilde{F}_{D_t}^{-1}\nabla J(\theta_t)$. Detailed derivation in Appendix A.2, this represents the AdaFisher optimizer's update rule, focusing on the diagonal elements to reduce computational overhead while maintaining a reasonable FIM approximation. This strategic simplification enhances the efficiency of the optimization process, crucial for training deep neural networks where computational resources are limited.

Table 1: Summary of the first, second moments, regret bound and Applicability used in Adam Kingma & Ba (2017), AdaHessian Yao et al. (2021), K-FAC Martens & Grosse (2020), Shampoo Gupta et al. (2018), and AdaFisher for updating model parameters $\theta_{t+1} = \theta_t - \eta m_t / \sqrt{v_t}$. Here β_1 and β_2 are first and second moment hyperparameters. L_t and R_t refer to the preconditioning method used by Shampoo Gupta et al. (2018), $g_t = \text{vec}(G_t)$, and T denotes the total number of steps. Note that Trans. denotes Transformers

Optimizer	m_t	v_t	Regret Bound	Applicability	
				CNNs	Trans.
Adam	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\left(\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}g_i g_i}{1-\beta_2^t}\right)^{1/2}$	$O(\log T\sqrt{T})$	✓	✓
AdaHessian	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\left(\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}D_i^{(s)}D_i^{(s)}}{1-\beta_2^t}\right)^{1/2}$	$O(\log T\sqrt{T})$	✓	✓
K-FAC	$\hat{F}^{-1}g_t$	1	$O(\sqrt{T})$	✓	×
Shampoo	$L_t^{-1/4}G_t R_t^{-1/4}$	1	$O(\sqrt{T})$	✓	×
AdaFisher	$\frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}$	$\tilde{F}_{D_t}$	$O(\log T\sqrt{T})$	✓	✓

3.3 AUGMENTING FIM INTO ADAM

Adam, which combines the methodologies of RMSProp and momentum (Sutskever et al., 2013), updates parameters following $\theta_{t+1} = \theta_t - \alpha_t \frac{m_t}{v_t}$. Here, α_t represents the learning rate, while m_t and v_t denote the first and second moment estimates, respectively. Although Adam is widely used, its approximation of the second moment using simple diagonal elements of second order statistics through squared gradients (Kunstner et al., 2019) can mirror stability challenges observed in simpler methods like SGD (Ruder, 2016). To overcome these limitations, we introduce AdaFisher, which utilizes a more refined diagonal block-Kronecker approximation of the FIM, derived using the K-FAC framework. This enhancement significantly improves curvature understanding and optimization dynamics. AdaFisher distinguishes itself from Adam by incorporating a higher fidelity approximation of the FIM, enhancing both optimization efficiency and model robustness in complex scenarios. As demonstrated in Figure 4, AdaFisher's FIM values exhibit narrow variations and lower mean values during training, suggesting a convergence towards flatter local minima. In contrast, Adam shows broader variations, indicating less efficient convergence. For more details regarding the convergence behavior refer to Appendix B.1. Moreover, AdaFisher omits the square root and the traditional EMA applied over the second moment, since the FIM naturally incorporates an EMA of its Kronecker factors (detailed in Eq. (3)). The exclusion of the square root aligns with the theoretical definition of second order methods, as using a square root deviates from the second-order Taylor expansion approximation that these methods aim to follow. A comparative summary of different moment estimates, m_t and v_t, along with their regret bounds and applicability across various optimizers, is presented in Table 1. Building on the principles of AdamW (Loshchilov & Hutter, 2019), which modifies Adam by integrating weight decay directly into the weight update step to counteract suboptimal decay behaviors and boost optimization performance, we introduce AdaFisherW. This variant adapts the AdamW framework to further enhance the opti-

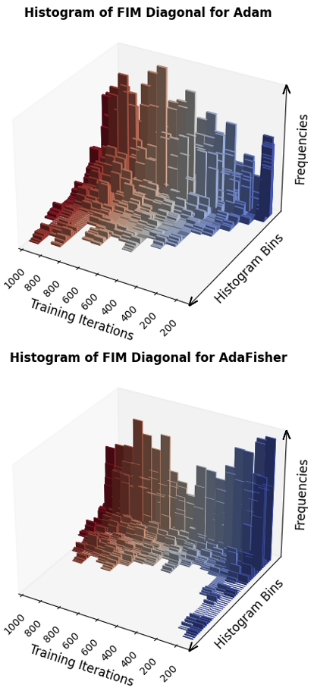

Figure 4: Comparison of FIM Diagonal Histograms during ResNet18 Training on CIFAR10: The figure displays the FIM diagonal elements for the first convolutional layer with Adam and AdaFisher over 1,000 training iterations.

mizer by leveraging curvature information from AdaFisher. Finally, AdaFisher is compatible with multi-GPU environments, with a distributed version detailed in Appendix A.4. The implementation for both AdaFisher[1] variants are delineated in the pseudo-code presented in Algorithm 1.

3.4 CONVERGENCE ANALYSIS

In this section, we provide a theoretical analysis of AdaFisher's convergence in both convex optimization and non-convex stochastic optimization. We first present a standard convergence behaviour of Eq. (2) for a simple strongly convex and strictly smooth function $f(J)$.

Proposition 3.3 (Convergence in convex optimization)**.** *For FIM defined in Eq. (4), the updating scheme* $\triangle\theta_t = \tilde{F}_t^{-1}\nabla J(\theta_t)$ *converges. Further, if* ∇J *is Lipschitz, the convergence rate is bounded.*

For non-convex case, we adopt the similar derivations of Chen et al. (2019), since AdaFisher belongs to the family of generalized Adam-type methods.

Proposition 3.4 (Convergence in non-convex stochastic optimization)**.** *Under the assumptions: (i)* J *is lower bounded and differentiable;* $||\nabla J(\theta) - \nabla J(\theta')||_2 \leq L||\theta - \theta'||_2$, $||\tilde{F}_{D_t}||_\infty < L$, $\forall t, \theta, \theta'$, *(ii) Both the true and stochastic gradient are bounded, i.e.* $||\nabla J(\theta_t)||_2 \leq \lambda$ *and* $||g_t||_2 \leq \lambda$, $\forall t$ *for some* $\lambda > 0$, *(iii) Unbiased and independent noise in* g_t, *i.e.* $g_t = \nabla J(\theta_t) + \zeta_t$, $\mathbb{E}[\zeta_t] = 0$, *and* $\zeta_i \perp \zeta_j$, $\forall i \neq j$. *Assume* $\eta_t = \frac{\eta}{\sqrt{t}}$, $\beta_t \leq \beta \leq 1$ *is non-increasing,* $\frac{\tilde{F}_{D_{t-1}}[j]}{\eta_{t-1}} \leq \frac{\tilde{F}_{D_t}[j]}{\eta_t}$, $\forall t \in [T], j \in [d]$, *we then have*

$$\min_{t\in[T]} \mathbb{E}[||\nabla J(\theta_t)||_2^2] \leq \frac{L}{\sqrt{T}}(C_1\eta^2\lambda^2(1+\log T) + C_2 d\eta + C_3 d\eta^2 + C_4) \quad (5)$$

where C_1, C_2, C_3 *are constants independent of* d *and* T, C_4 *is a constant independent of* T, *the expectation is taken w.r.t all the randomness corresponding to* $\{g_t\}$.

Proposition 3.4 implies the convergence rate for AdaFisher in the non-convex case is at $O(\log T/\sqrt{T})$, which is similar to Adam-type optimizer. While DNNs often include nonsmooth components like ReLU and max pooling, which create nondifferentiable points in the loss landscape, optimizers like AdaFisher handle these cases effectively as shown by our results in Section 4.

Algorithm 1 AdaFisher optimization algorithm. Good default settings for the tested machine learning problems are $\alpha = 0.001$ (learning rate), $\lambda = 0.001$ (Tikhonov damping parameter), $\gamma = 0.92$ (Exponentially decaying factor). [Default parameters are: $\beta = 0.9$ (Exponentially decaying factor of Adam), κ (weight decay) (Kingma & Ba (2017), Loshchilov & Hutter (2019))].

```
Require: Step size $\alpha$; Exponential decay rate for Kronecker factors $\gamma \in [0,1)$; Tikhonov damping parameter $\lambda$; Exponential decay rate for first moments $\beta$ in $[0,1)$; Initial parameters $\theta$
  Initialize 1st moment variable $m = 0$; FIM $\tilde{F}_{D_i} = \mathbf{I}$; time step $t = 0$
while stopping criterion not met do
    Sample a minibatch of $M$ examples from the training set $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$
    Compute $\mathcal{H}_{D_{i-1}}, \mathcal{S}_{D_i}$ for $i \in \{1,\dots,L\}$ using Section A.3 (notice that: $\mathcal{H}_{D_0} = x$)
    Compute EMAs of $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ using Eq.(3)
    Compute $\tilde{F}_{D_i}$ for $i \in \{1,\dots,L\}$ using Eq. (4)
    $h_t \leftarrow \frac{1}{M}\sum_i \nabla_{\theta_t}\mathcal{L}(f(x^{(i)};\theta_t), y^{(i)})$ (Compute gradient)
    $m_{t+1} \leftarrow \frac{\beta m_t + (1-\beta)h_t}{1-\beta^t}$ (Update and correct biased first moment)
    Case AdaFisher: $\Delta\theta_t = -\alpha\tilde{F}_{D_t}^{-1}m_t$; Case AdaFisherW: $\Delta\theta_t = -\alpha\left(\tilde{F}_{D_t}^{-1}m_t + \kappa\theta_t\right)$
    $\theta_{t+1} \leftarrow \theta_t + \Delta\theta_t$ (Apply update)
    $t \leftarrow t + 1$
end while
```

4 RESULTS

To evaluate AdaFisher, we conduct experiments on six benchmark datasets across Image Classification for CV and Language Modelling for NLP that are commonly used to evaluate optimization

---

[1] PyTorch implementation is available in the supplementary materials.

algorithms: CIFAR-10, CIFAR100 (Krizhevsky et al., 2009), TinyImageNet (Le & Yang, 2015), and ImageNet (Deng et al., 2009) for image classification; Wikitext-2 (Merity et al., 2016) and Penn Treebank (PTB) (Marcus et al., 1993) for language modelling. The six baseline methods we compare with are SGD, Adam/AdamW, K-FAC, AdaHessian, and Shampoo. For CIFAR experiments, we report the average over 5 runs. We also perform a transfer learning task using the ImageNetV1 weights Paszke et al. (2019). Detailed descriptions of the experimental setups (including hyperparameters, datasets, and data augmentation), results, and analyses are provided in Appendix D.

Table 2: Performance metrics (mean, std) of different networks and optimizers on CIFAR10 and CIFAR100 using batch size 256 with a 200-epoch AdaFisher training cutoff.

Network	CIFAR10						CIFAR100					
	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet18	$95.64_{0.1}$	$94.85_{0.1}$	$95.44_{0.1}$	$95.17_{0.2}$	$94.08_{0.2}$	**$96.25_{0.2}$**	$76.56_{0.2}$	$75.74_{0.1}$	$71.79_{0.2}$	$76.03_{0.3}$	$76.78_{0.2}$	**$77.28_{0.2}$**
ResNet50	$95.71_{0.1}$	$94.45_{0.2}$	$95.54_{0.1}$	$95.66_{0.1}$	$94.59_{0.1}$	**$96.34_{0.2}$**	$78.01_{0.1}$	$74.65_{0.5}$	$75.81_{0.3}$	$77.40_{0.4}$	$78.07_{0.4}$	**$79.77_{0.4}$**
ResNet101	$95.98_{0.2}$	$94.57_{0.1}$	$95.29_{0.6}$	$96.01_{0.1}$	$94.63_{0.1}$	**$96.39_{0.1}$**	$78.89_{0.2}$	$75.56_{0.3}$	$73.38_{0.2}$	$77.01_{0.4}$	$78.83_{0.2}$	**$80.65_{0.4}$**
DenseNet121	$96.09_{0.1}$	$94.86_{0.1}$	$96.11_{0.1}$	$96.12_{0.1}$	$95.66_{0.1}$	**$96.72_{0.1}$**	$80.13_{0.4}$	$75.87_{0.4}$	$74.80_{0.9}$	$79.79_{0.2}$	$80.24_{0.3}$	**$81.36_{0.3}$**
MobileNetV3	$94.43_{0.2}$	$93.32_{0.1}$	$92.86_{3.1}$	$94.34_{0.1}$	$93.81_{0.2}$	**$95.28_{0.1}$**	$73.89_{0.3}$	$70.62_{0.3}$	$56.58_{4.5}$	$73.75_{0.3}$	$70.85_{0.3}$	**$77.56_{0.1}$**
Tiny Swin	$82.34_{0.2}$	$87.37_{0.6}$	$84.15_{0.2}$	$64.79_{0.5}$	$63.91_{0.4}$	**$88.74_{0.4}$**	$54.89_{0.4}$	$60.21_{0.4}$	$56.86_{0.5}$	$34.45_{0.4}$	$30.39_{1.2}$	**$66.05_{0.5}$**
FocalNet	$82.03_{0.2}$	$86.23_{0.1}$	$64.18_{0.2}$	$38.94_{0.8}$	$37.96_{0.7}$	**$87.90_{0.1}$**	47.76_{03}	$52.71_{0.5}$	$32.33_{0.3}$	$9.98_{0.6}$	$9.18_{0.1}$	**$53.69_{0.3}$**
CCT-2/3×2	$78.76_{0.3}$	$83.89_{0.4}$	–	$33.08_{2.3}$	$35.16_{0.4}$	**$84.94_{0.3}$**	$54.05_{0.4}$	$59.78_{0.5}$	–	$7.17_{0.2}$	$8.60_{0.1}$	**$62.91_{0.5}$**

*Note that Adam and AdaFisher were used for all CNN architectures, while AdamW and AdaFisherW were applied for all ViT experiments.

4.1 IMAGE CLASSIFICATION

We commence our analysis by assessing the convergence and generalization capabilities of various models on image classification tasks. Specifically, we deploy ResNet architectures (ResNetX where $X \in \{18, 50, 101\}$), MobileNetV3 (Howard et al., 2019), Tiny Swin (Liu et al., 2021), FocalNet (Yang et al., 2022) and CCT-2/3×2 (Hassani et al., 2021) on CIFAR10 and CIFAR100, while utilizing standard ResNet50 for TinyImageNet and ImageNet. The performance outcomes for CIFAR datasets are detailed in Table 2. Our empirical evaluation of AdaFisher optimizer across these models and datasets illustrates its efficiency in optimizing in image classification, surpassing contemporary SOTA optimizers, including SGD, Adam, AdamW, AdaHessian, Shampoo, and K-FAC. We employ the Wall-Clock-Time (WCT) method with a cutoff of 200 epochs for AdaFisher's training, except for ImageNet, where we use a 90-epoch WCT for Adam, which surprisingly matched AdaFisher's training duration. Results confirm AdaFisher's superior classification accuracy on both CNNs and VITs. Please note that the results for Tiny ImageNet are described in Appendix D.2.4.

ImageNet Training. Training on ImageNet typically requires multiple GPUs and large batch sizes. Our study showcases that AdaFisher achieves superior validation accuracy on a single GPU than its counterparts in scenarios marked by the light blue region. This performance outstrips traditional approaches like SGD, LAMB (You et al., 2019), and LARS (You et al., 2017), which typically utilize batch sizes of 16K. While AdaFisher attains SOTA results on a single GPU, it further excels when scaled up in a distributed setting with larger batch sizes. The results, benchmarked using 256 batch size and a WCT of 90 Adam training epochs, are detailed in Table 3 and illustrated in Figure 5. Distributed AdaFisher curves are illustrated in Figure 15. The light blue highlights in the table represent our experiments with a batch size of 256 on a single GPU. The light green indicates results from a distributed version of AdaFisher employing larger batch sizes, whereas the orange reflects results from SOTA methods using a higher batch size of 16K, SGD with a batch size of 256 and AdamW with batch size of 1024. Overall, AdaFisher demonstrates robust generalization capabilities, in contrast to K-FAC, which tends to overfit.

Table 3: Validation of ImageNet-1K / ResNet50 by different optimizers reported on Top-1 and Top-5 accuracy.

Optimizers	Batch size	Top-1	Top-5
Adam	256	67.78	88.37
K-FAC	256	70.96	89.44
Shampoo	256	72.82	91.42
AdaFisher	256	**76.95**	**93.39**
AdaFisher	512	**77.01**	**93.45**
AdaFisher	1024	**77.09**	**93.56**
SGD Goyal et al. (2017)	256	76.40	-
AdamW Chen et al. (2024)	1024	76.34	-
LAMB You et al. (2019)	$16K$	76.66	93.22
SGD You et al. (2019)	$16K$	75.20	-
LARS Huo et al. (2021)	$16K$	75.1	-

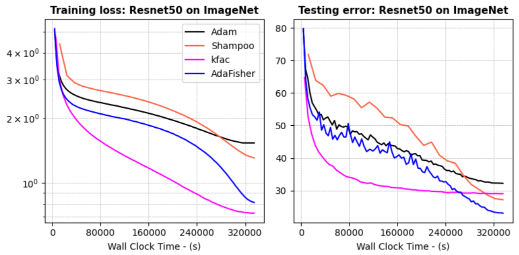

Figure 5: Training loss and validation error of ResNet-50 on ImageNet. AdaFisher consistently achieves lower test error as compared to its counterparts.

4.2 TRANSFER LEARNING

Owing to global warming concerns, more sustainable practices are essential in training DNNs. Employing pretrained models from ImageNet-1k by PyTorch for transfer learning on datasets like CIFAR10 and CIFAR100 reduces the carbon footprint and leverages prior computations, promoting eco-friendly AI methodologies. We applied these pretrained weights across various CNN architectures, to train on these datasets. The results, presented in Table 4, highlight the significant advantages of using AdaFisher, consistently achieving top accuracy across both datasets. More details can be found in Appendix D.2.3.

Table 4: Performance comparison of different networks and optimizers on CIFAR10 and CIFAR100 using ImageNet-1K pretrained weights. Evaluation is based on wall clock time of 50 training epochs with AdaFisher.

	CIFAR10						CIFAR100					
Network	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet50	$96.50_{0.2}$	$96.45_{0.2}$	$96.35_{0.3}$	$96.45_{0.1}$	$96.03_{0.4}$	$\mathbf{97.13}_{\mathbf{0.2}}$	$82.12_{0.1}$	$82.01_{0.4}$	$80.64_{0.9}$	$80.55_{0.4}$	$81.70_{0.2}$	$\mathbf{82.23}_{\mathbf{0.2}}$
ResNet101	$97.07_{0.2}$	$96.70_{0.1}$	$96.65_{0.2}$	$96.84_{0.1}$	$96.63_{0.1}$	$\mathbf{97.22}_{\mathbf{0.1}}$	$84.01_{0.1}$	$82.43_{0.2}$	$81.36_{0.8}$	$82.26_{0.3}$	$82.65_{0.2}$	$\mathbf{84.47}_{\mathbf{0.2}}$
DenseNet121	$94.80_{0.1}$	$94.77_{0.1}$	$93.08_{0.1}$	$94.41_{0.2}$	$94.76_{0.1}$	$\mathbf{95.03}_{\mathbf{0.1}}$	$75.98_{0.2}$	$75.65_{0.3}$	$71.06_{0.9}$	$76.10_{0.3}$	$76.08_{0.2}$	$\mathbf{76.92}_{\mathbf{0.3}}$
MobileNetV3	$91.76_{0.3}$	$90.92_{0.3}$	$86.45_{2.5}$	$91.72_{0.2}$	$91.39_{0.3}$	$\mathbf{92.78}_{\mathbf{0.2}}$	$71.86_{0.4}$	$66.11_{0.8}$	$59.69_{2.3}$	$69.85_{0.4}$	$68.87_{0.3}$	$\mathbf{72.38}_{\mathbf{0.4}}$

4.3 LANGUAGE MODEL

We employ the WikiText-2 dataset, which encompasses approximately 100 million tokens derived from over 2 million words extracted from a curated set of 'Good' and 'Featured' articles on Wikipedia. Additionally, we utilize the PTB dataset, renowned for its extensive collection of English words with part-of-speech tags, which has been widely used in natural language processing tasks for training and benchmarking language models. Our experiments utilize a scaled-down version of GPT-1 (Radford et al., 2019), featuring four self-attention layers with masking capabilities with a total of 28,351,488 learnable parameters. More details about hyperparameters and model can be found in Appendix D.3. The perplexity (PPL) on the test set, corresponding to the best-performing model during validation, is documented in Table 5. Similar to approaches in image classification, we apply the WCT method with 50 epochs training time of AdaFisher as the cutoff period. Notice that Shampoo did not achieve convergence despite using optimal hyperparameters, and the K-FAC was unable to train with ASDL library (Osawa et al., 2023).

Table 5: Language Modelling performance (PPL) on Wikitext-2 and PTB test dataset (lower is better).

Optimizer	Test PPL	
	WikiText-2	PTB
AdamW	175.06	44.70
AdaHessian	407.69	59.43
Shampoo	1727.75	–
AdaFisherW	**152.72**	**41.15**

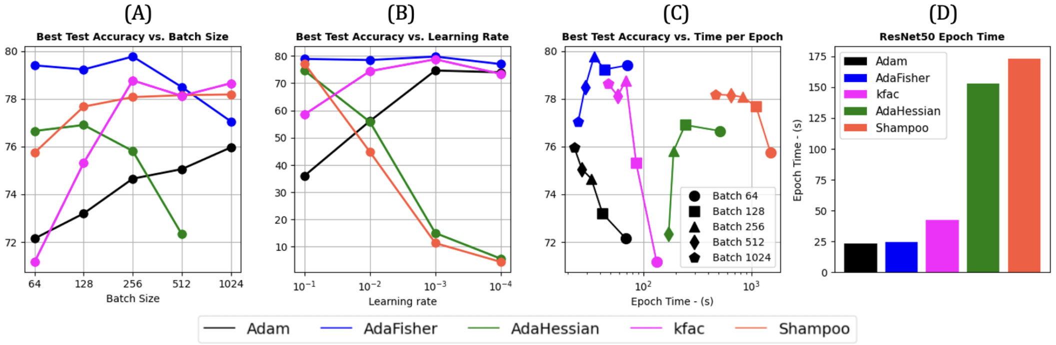

Figure 6: Performance comparison of AdaFisher and other optimizers using the ResNet50 network on the CIFAR100 dataset. (A) Test accuracy by batch size. (B) Accuracy vs. learning rates. (C) Accuracy related to epoch time across batch sizes. (D) Epoch time for different optimizers with batch size of 256.

4.4 STABILITY ANALYSIS

In this section, we assess AdaFisher's stability under varying learning rates and batch sizes using ResNet50 on CIFAR100 and compare its performance to other optimizers. Improved stability indicates a reduced need for hyperparameter tuning while maintaining high performance. To ensure a fair comparison, all methods were evaluated using a consistent experimental setup, with parameters tailored to each optimizer's strengths. However, we exclude AdaHessian results for a batch size of 1024 due to its significant computational cost.

Batch Size Analysis. We examine the impact of batch size on AdaFisher's performance, as shown in Panels (A) and (C) of Figure 6. AdaFisher maintains high test accuracy across various batch sizes, excelling particularly at smaller sizes despite some sensitivity to larger ones. Panel (C) highlights AdaFisher's efficiency, achieving high accuracy with shorter epoch times compared to Adam, detailed further in Panel (D) where AdaFisher shows competitive epoch durations against other optimizers. These results, discussed in Appendix D.2.7, underscore AdaFisher's effective performance across batch size variations without adjusting other hyperparameters.

Learning Rate Stability. This analysis evaluates the impact of learning rate variations on AdaFisher's performance, as depicted in Panel (B) of Figure 6. AdaFisher demonstrates superior stability, particularly at lower learning rates, maintaining consistent performance across a broad spectrum. This stability alleviates the need for meticulous learning rate adjustments, thereby streamlining model training in various computational environments. Additionally, AdaFisher's stability across various learning rates can be attributed to its effective approximation of the curvature matrix.

Ablation Studies. We further conduct extensive ablation studies on additional components of AdaFisher, including the convergence efficiency, our novel approximation of the FIM, the significance of EMA for Kronecker factors, the impact of the square root, the stability across learning rate schedulers and the updated computation of the FIM for normalization layers. These analyses are thoroughly detailed in Appendix B.

5 RELATED WORK

Tractable Approximation of the FIM. Efficient approximations of the FIM for neural network optimization have evolved significantly, beginning with block-diagonal strategies exemplified by TONGA (Roux et al., 2007) and extending to the Kronecker-factored approaches like K-FAC. Further innovations have emerged, such as SK-FAC (Tang et al., 2021), EVA (Zhang et al., 2023), which accelerates the computation of the FIM, and Eschenhagen et al. (2023), who propose a generalized framework for FIM computation that enhances preconditioning methods like Shampoo. More recently, Liu et al. (2023), Huang et al. (2024) and Duvvuri et al. (2024) have introduced tractable solutions for computing the FIM. AdaFisher distinguishes itself by integrating enhanced FIM computations with novel diagonal Kronecker factors, enriching the Adam optimization framework. This integration, outlined in Proposition 3.2 and detailed in Appendix A.3, advances the fusion of second-order optimization principles with first-order methods. This builds upon innovations like AdaHessian, which incorporates Hessian diagonals into the Adam framework.

Adaptive First-Order Methods. Building upon the diagonal approximation heritage of the FIM, AdaFisher extends traditional diagonally-scaled first-order methods such as AdaGrad (Duchi et al., 2011), AdamP, AdaInject, AdaBelief, and Adam. These methods have inspired advancements like AMSGrad (Reddi et al., 2019), AdaBound (Luo et al., 2019), RAdam (Liu et al., 2019), and enhanced AdaBelief, FOOF (Benzing, 2021) and INNAprop (Bolte et al., 2024), improving both theoretical rigor and practical effectiveness. A recent study by Jiang et al. (2024a) illustrates that first-order adaptive methods can bias training trajectories and effectively navigate through saddle points. In response, Mishchenko & Stich (2023) propose an empirical solution involving the addition of noise to mitigate these biases. Leplat et al. (2022) introduces a novel method accelerating convergence via Gauss-Seidel type discretization. AdaFisher differentiates itself by eliminating the conventional square root in the second moment calculation, with benefits underscored by Lin et al. (2024a) and Malladi et al. (2022) in CNN architectures, and Zhang et al. (2024) demonstrates the critical role of Adam family optimizers in Transformer models. Its unique preconditioning, based on the Fisher Information, is elaborated in Algorithm 1.

6 CONCLUSION, LIMITATIONS AND FUTURE RESEARCH

In this work, we introduced AdaFisher, an adaptive optimizer that leverages a novel diagonal block-Kronecker approximation of the FIM to improve gradient rescaling and descent directions. Incorporated within the Adam framework, AdaFisher speeds up training, reduces hyperparameter sensitivity, and delivers higher accuracy and stability across image classification and language modeling tasks. Empirical and theoretical analyses demonstrate its superiority over current optimizers, with efficient space and time usage facilitating its application across diverse tasks. Notably, AdaFisher excels in SOTA comparisons on ImageNet under both single and multi-GPU setups. Although optimized for statistical tasks, AdaFisher is less suited for tasks involving non-exponential loss families due to its reliance on statistical data for FIM computation. Future work will expand testing to other models and areas, such as generative modeling (diffusion models) and graph neural networks, and developing CUDA kernels for Kronecker factors could greatly improve AdaFisher's scalability and performance.

REFERENCES

iris, 2018. URL `https://dx.doi.org/10.21227/rz7n-kj20`.

Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, Feb 1998. ISSN 0899-7667. doi: 10.1162/089976698300017746.

Shun-ichi Amari and Hiroshi Nagaoka. Methods of information geometry. 2000. URL `https://api.semanticscholar.org/CorpusID:116976027`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Frederik Benzing. Kronecker-factored second-order optimizers perform first-order descent on neurons. 2021.

Jérôme Bolte, Ryan Boustany, Edouard Pauwels, and Andrei Purica. A second-order-like optimizer with adaptive gradient scaling for deep learning. *arXiv preprint arXiv:2410.05871*, 2024.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Johan Braeken and Marcel ALM Van Assen. An empirical kaiser criterion. *Psychological methods*, 22(3):450, 2017.

Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima. *Advances in Neural Information Processing Systems*, 34:22405–22418, 2021.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36, 2024.

Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1x-x309tm`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Shiv Ram Dubey, SH Shabbeer Basha, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Adainject: Injection based adaptive gradient descent optimizers for convolutional neural networks. *IEEE Transactions on Artificial Intelligence*, 2022.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL `http://jmlr.org/papers/v12/duchi11a.html`.

Sai Surya Duvvuri, Fnu Devvrit, Rohan Anil, Cho-Jui Hsieh, and Inderjit S Dhillon. Combining axes preconditioners through kronecker approximation for deep learning. In *The Twelfth International Conference on Learning Representations*, 2024.

Muhammad ElNokrashy, Badr AlKhamissi, and Mona Diab. Depth-wise attention (dwatt): A layer fusion method for data-efficient classification. *arXiv preprint arXiv:2209.15168*, 2022.

Runa Eschenhagen, Alexander Immer, Richard E Turner, Frank Schneider, and Philipp Hennig. Kronecker-factored approximate curvature for modern neural network architectures. *arXiv preprint arXiv:2311.00636*, 2023.

Runa Eschenhagen, Alexander Immer, Richard Turner, Frank Schneider, and Philipp Hennig. Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, 36, 2024.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization, 2021.

Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.

Thomas George. NNGeometry: Easy and Fast Fisher Information Matrices and Neural Tangent Kernels in PyTorch, February 2021. URL `https://doi.org/10.5281/zenodo.4532597`.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers, 2016.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization, 2018.

Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. 2021. URL `https://arxiv.org/abs/2104.05704`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

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*, pp. 770–778, 2016.

Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. *arXiv preprint arXiv:2006.08217*, 2020.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

Richard A. Holmgren. *Newton's Method*, pp. 127–151. Springer New York, New York, NY, 1996. ISBN 978-1-4419-8732-7. doi: 10.1007/978-1-4419-8732-7_12. URL `https://doi.org/10.1007/978-1-4419-8732-7_12`.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2 edition, 2012.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

Keke Huang, Ruize Gao, Bogdan Cautis, and Xiaokui Xiao. Scalable continuous-time diffusion framework for network inference and influence estimation, 2024.

Zhouyuan Huo, Bin Gu, and Heng Huang. Large batch optimization for deep learning using new complete layer-wise adaptive rate scaling. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7883–7890, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Kaiqi Jiang, Dhruv Malik, and Yuanzhi Li. How does adaptive optimization impact local neural network geometry? *Advances in Neural Information Processing Systems*, 36, 2024a.

Zixuan Jiang, Jiaqi Gu, Hanqing Zhu, and David Pan. Pre-rmsnorm and pre-crmsnorm transformers: equivalent and efficient pre-ln transformers. *Advances in Neural Information Processing Systems*, 36, 2024b.

Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/46a558d97954d0692411c861cf78ef79-Paper.pdf`.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Yann Lecun. A theoretical framework for back-propagation. 08 2001.

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient backprop*, pp. 9–48. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, 2012. ISBN 9783642352881. doi: 10.1007/978-3-642-35289-8_3. Copyright: Copyright 2021 Elsevier B.V., All rights reserved.

Valentin Leplat, Daniil Merkulov, Aleksandr Katrutsa, Daniel Bershatsky, Olga Tsymboi, and Ivan Oseledets. Nag-gs: Semi-implicit, accelerated and robust stochastic optimizer. *arXiv preprint arXiv:2209.14937*, 2022.

Wu Lin, Felix Dangel, Runa Eschenhagen, Juhan Bae, Richard E Turner, and Alireza Makhzani. Can we remove the square-root in adaptive gradient methods? a second-order perspective. *arXiv preprint arXiv:2402.03496*, 2024a.

Wu Lin, Felix Dangel, Runa Eschenhagen, Juhan Bae, Richard E. Turner, and Alireza Makhzani. Can we remove the square-root in adaptive gradient methods? a second-order perspective, 2024b.

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

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

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

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate, 2019.

Linjian Ma, Gabe Montague, Jiayu Ye, Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. Inefficiency of k-fac for large batch size training, 2019.

Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the sdes and scaling rules for adaptive gradient algorithms. *Advances in Neural Information Processing Systems*, 35: 7697–7711, 2022.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL `https://aclanthology.org/J93-2004`.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2408–2417, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/martens15.html`.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature, 2020.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Konstantin Mishchenko and Sebastian U Stich. Noise injection irons out local minima and saddle points. In *OPT 2023: Optimization for Machine Learning*, 2023.

Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing*. Prentice-hall Englewood Cliffs, second edition, 1999.

Kazuki Osawa, Satoki Ishikawa, Rio Yokota, Shigang Li, and Torsten Hoefler. Asdl: A unified interface for gradient preconditioning in pytorch, 2023.

H Park, S.-I Amari, and K Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13(7):755–764, 2000. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(00)00051-4. URL `https://www.sciencedirect.com/science/article/pii/S0893608000000514`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

SGOPAL Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

K. B. Petersen and M. S. Pedersen. The matrix cookbook, October 2008. URL `http://www2.imm.dtu.dk/pubdb/p.php?3274`. Version 20081110.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

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

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Ernest Ryu and Stephen Boyd. A primer on monotone operator methods survey. *Applied and computational mathematics*, 15:3–43, 01 2016.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates, 2013.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost, 2018.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `https://proceedings.mlr.press/v28/sutskever13.html`.

Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):2917–2931, September 2020. ISSN 1558-2205. doi: 10.1109/tcsvt.2019.2935128. URL `http://dx.doi.org/10.1109/TCSVT.2019.2935128`.

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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13479–13487, 2021.

Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning, 2018.

Jianwei Yang, Chunyuan Li, Xiyang Dai, Lu Yuan, and Jianfeng Gao. Focal modulation networks, 2022.

Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 10665–10673, 2021.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. *Advances in neural information processing systems*, 31, 2018.

Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

Lin Zhang, Shaohuai Shi, and Bo Li. Eva: A general vectorized approximation framework for second-order optimization. *arXiv preprint arXiv:2308.02123*, 2023.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why transformers need adam: A hessian perspective. *arXiv preprint arXiv:2402.16788*, 2024.

Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels, 2018.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 18795–18806. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/d9d4f495e875a2e075a1a4a6e1b9770f-Paper.pdf`.

APPENDIX

CONTENTS

A Theory **16**
- A.1 Kronecker Factors: A Structural Examination (Continue) 16
- A.2 Proofs 19
- A.3 Computation of Kronecker Factors 23
- A.4 Distributed AdaFisher 23

B Ablation Studies **24**
- B.1 Evaluating Stability Across Learning Rate Schedulers, and Assessing Convergence Efficiency 25
- B.2 Component Analysis: Evaluating the Significance of AdaFisher's Elements 26

C Visualization **27**

D Experiments **28**
- D.1 Hardware 28
- D.2 Image Classification 28
  - D.2.1 Hyperparameter Tuning 29
  - D.2.2 Dataset Details 30
  - D.2.3 Transfer Learning 30
  - D.2.4 Results 31
  - D.2.5 Comparison with Other Relevant Methods 37
  - D.2.6 Comparison with Consistent Epoch Counts 37
  - D.2.7 Comparison of Training Speed and Memory Utilization 38
- D.3 Language Modelling 40
  - D.3.1 Dataset Details 40
  - D.3.2 Network Details 40
  - D.3.3 Hyperparameters 40
  - D.3.4 Results 40

E Impact Statement **40**

A THEORY

A.1 KRONECKER FACTORS: A STRUCTURAL EXAMINATION (CONTINUE)

In the realm of matrix theory, Gersgorin's Circle Theorem offers a principle for localizing the eigenvalues of a complex square matrix, asserting that each eigenvalue is situated within at least one Gersgorin disk. These disks are defined by the matrix's diagonal elements and the sum of the absolute values of the respective off-diagonal row entries. Formally, the theorem is stated as follows:

Theorem A.1 (Gersgorin Circle Theorem)**.** *Let $\mathcal{A}$ be a complex square matrix with eigenvalues λ. For each λ, there exists an index i such that:*

$$|\lambda - \mathcal{A}_{ii}| \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} |\mathcal{A}_{ij}|,$$

where the summation excludes the diagonal entry $\mathcal{A}_{ii}$.

For a detailed proof of Theorem A.1, the reader is referred to the seminal work by Horn and Johnson Horn & Johnson (2012). Extending the application of Gersgorin's Circle Theorem to the study of Kronecker factors within deep neural networks, we analyze these factors from both convolutional (37th) and linear (41st) layers of a ResNet-18 network, post-training on CIFAR10 dataset. As elucidated in Section 3.1, leveraging Theorem A.1 demonstrates that the eigenvalues of the Kronecker factors from the convolutional layer are predominantly concentrated along the diagonal. This observation is analogously applicable to the linear layer. Figure 3 showcases the Gersgorin disks for the 41st (linear) layer, with the eigenvalues (red crosses) significantly clustered within these disks (centered at the black circles), underscoring a pronounced diagonal dominance. Moreover, upon introducing Gaussian noise to the off-diagonal elements following this shceme: $\hat{\mathcal{M}} = \mathcal{A} + \mathcal{E}, \quad$ where $\mathcal{E} = [e_{ij}]$ and $e_{ij} \sim \mathcal{N}(0, \sigma^2)$ for $i \neq j$, the perturbation analysis elucidates that such stochastic variations engender only marginal displacements in the eigenvalues. Notably, those eigenvalues fulfilling the Kaiser criterion are minimally affected, substantiating the resilience of the diagonal dominance against noise-induced perturbations. Our next analysis focus centers on elucidating the behaviors of matrices through consecutive steps in the frequency domain, thereby highlighting the intricate patterns and transformations emergent from the training process. By deploying a Fast Fourier Transform (FFT) on $\mathcal{H}$ and $\mathcal{S}$, along with their noise-infused variants $\hat{\mathcal{H}}$ and $\hat{\mathcal{S}}$, we aim to dissect the spectral nuances of these factors. The deliberate addition of noise

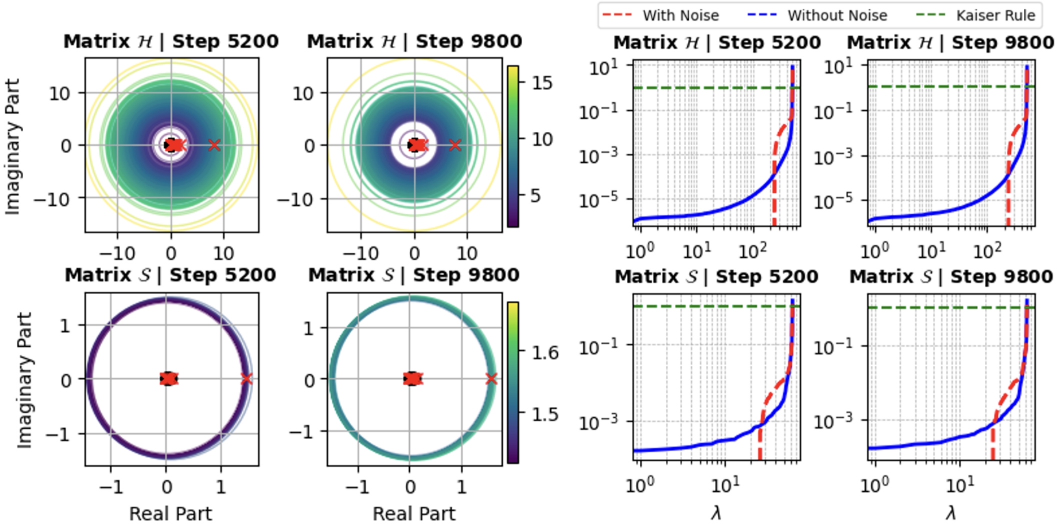

Figure 7: Gersgorin disks and eigenvalue perturbation analysis for matrices $\mathcal{H}$ and $\mathcal{S}$ at training steps 5200 (middle of training) and 9800 (end of training) in a ResNet-18 Network's Linear Layer (41st Layer). The left panel depicts the Gersgorin's circles in the complex plane, while the right panel illustrates the magnitude spectrum of eigenvalues with and without the influence of Gaussian noise.

to the off-diagonal serves as a probe to validate our hypothesis that the pivotal information of the Kronecker factors is predominantly concentrated along their diagonals. The minimal impact of such noise perturbations observed empirically underscores this diagonal dominance. Our analysis aims to juxtapose the frequency domain representations of both the uncontaminated and the noise-affected matrices at assorted iterative phases, thereby illuminating the inherent stability and tenacity of the Kronecker structures amidst stochastic disturbances.
Let A be a two-dimensional $m \times n$ matrix. The FFT of A, denoted as $\mathcal{F}(A)$, is computed as:

$$\mathcal{F}(A)_{kl} = \sum_{p=0}^{m-1} \sum_{q=0}^{n-1} A_{pq} \cdot e^{-2\pi i \left(\frac{pk}{m} + \frac{ql}{n} \right)}, \tag{6}$$

where $\mathcal{F}(A)_{kl}$ is the value of the FFT at the k-th row and l-th column of the transformed matrix, A_{pq} is the value of the original matrix at the p-th row and q-th column, and i is the imaginary unit (Oppenheim et al., 1999). Figure 8 demonstrates the Fourier spectral analysis of the Kronecker factors

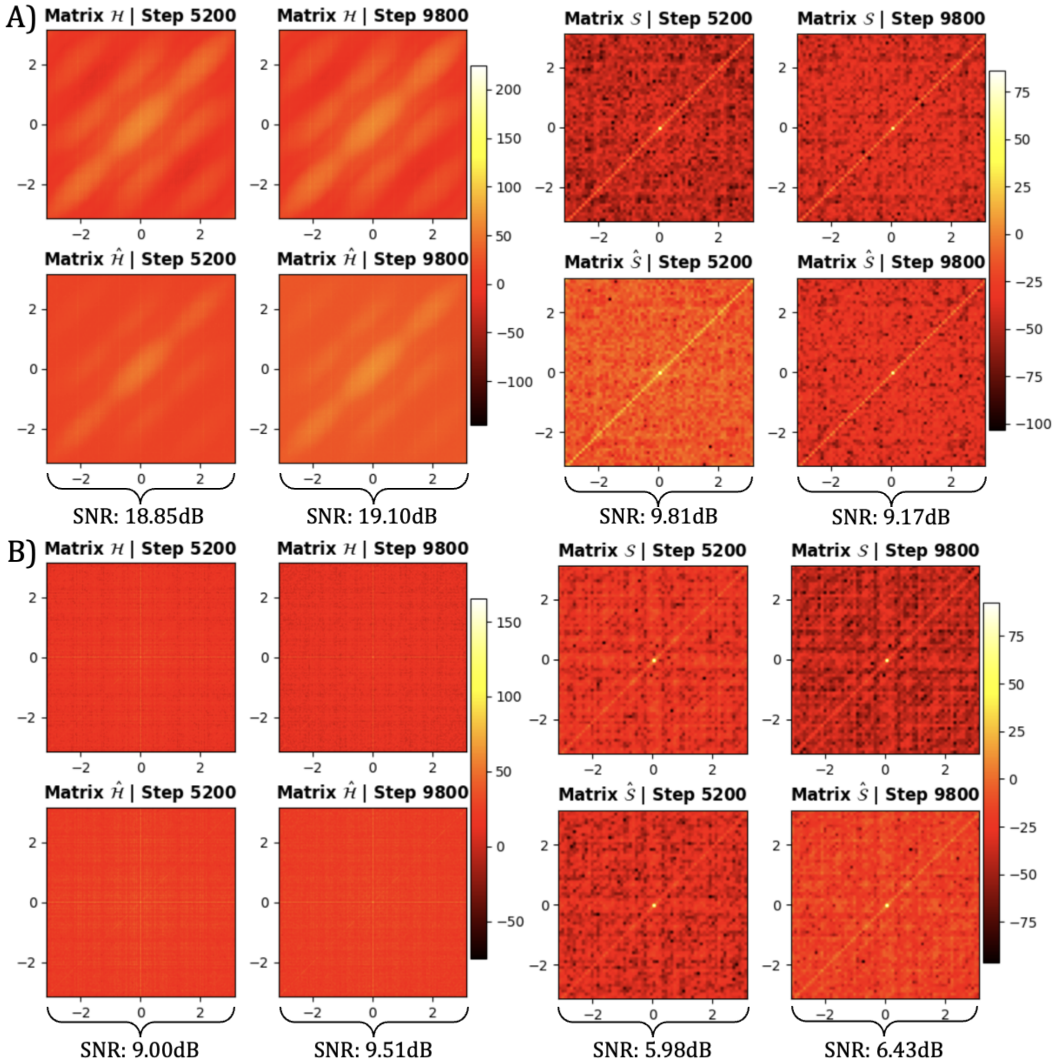

Figure 8: Comparative Visualization of FFT Outputs for Kronecker Factors in a ResNet-18 Network's Convolutional and Linear Layers. (A) FFT results for Kronecker factors $\mathcal{H}$ and $\hat{\mathcal{H}}$ from the 37th convolutional layer under noise-free conditions (top) and with Gaussian noise (bottom) at iterations 5200 (middle of training) and 9800 (end of training). (B) Analogous FFT results for Kronecker factors $\mathcal{S}$ and $\hat{\mathcal{S}}$ from the 41st linear layer, also contrasted between noise-free (top) and noisy conditions (bottom) at the same iterations.

$\mathcal{H}$ and $\mathcal{S}$ over two distinct iterative stages of training—5200 and 9800 for a convolutional and linear layers (37th and 41st of a ResNet-18 network respectively). Each Kronecker factor is analyzed via FFT in both a pristine, noise-free condition and a Gaussian noise-affected state, with the associated Signal-to-Noise Ratios (SNRs) detailed in Eq. (7). In the noise-free FFT spectra, a pronounced diagonal energy concentration is manifest in the $\mathcal{H}$ and $\mathcal{S}$ factors of the convolutional layer, indicative of significant informational preservation along the diagonal. In contrast, the linear layer exhibits a less pronounced but still discernible diagonal energy distribution, suggesting a more diffuse yet still noteworthy diagonal information structure. With the addition of noise, the matrices $\hat{\mathcal{H}}$ and $\hat{\mathcal{S}}$ still display a notable diagonal pattern, indicating minimal SNR deterioration. This observation supports the proposition that the kronecker factors primarily encode their information along the diagonal, and the introduction of noise into the off-diagonal elements has a limited impact. The SNR between a matrix $\mathcal{M}$ and $\hat{\mathcal{M}}$ is computed using the formula:

$$\text{SNR} = 10 \cdot \log_{10} \left(\frac{\sum_{i=1}^{N} |\mathcal{M}_{ii}|^2}{\sum_{j>i}^{N} |\hat{\mathcal{M}}_{ij}|^2} \right), \tag{7}$$

where $\mathcal{M}_{ii}$ denotes the diagonal elements of $\mathcal{M}$, and $\hat{\mathcal{M}}_{ij}$ represents the upper triangular elements of $\hat{\mathcal{M}}$ excluding the diagonal (Oppenheim et al., 1999). The observed reduction in SNR from step 5200 to step 9800 for the Kronecker factor $\mathcal{S}$ in the convolutional layer, under noisy conditions, could suggest an incremental integration of noise effects across iterations. Conversely, for the remaining factors, an increase in SNR throughout the training process is detected, which may indicate an enhancement in signal clarity. Nevertheless, the integrity of the diagonal concentration of energy remains predominantly intact, demonstrating the underlying robustness of the network's feature extraction capability against noise perturbations. Ultimately, the spectral analyses validate the hypothesis that the Kronecker factors' informational content is predominantly diagonal and resistant to the effects of off-diagonal Gaussian noise. This durability is sustained through successive iterations, maintaining the primary spectral characteristics of the Kronecker factors. Figure 9 offers a visual exposition of the Kronecker Product Factors $\mathcal{H}$ and $\mathcal{S}$ at progressive iteration junctures—specifically steps 5200 and 9800 for a convolutional and linear layers (37th and 41st of a ResNet-18 network respectively). The initial duo of plots in each (A) and (B), delineate the Kronecker factor $\mathcal{H}$ at the aforementioned steps, elucidating the matrix's structure at two distinct evolutionary stages. The next duo plots in (A) and (B) represents the Kronecker factor $\mathcal{S}$ at different steps of training. This visual examination, in conjunction with the preceding spectral analyses, articulates an integrated story of

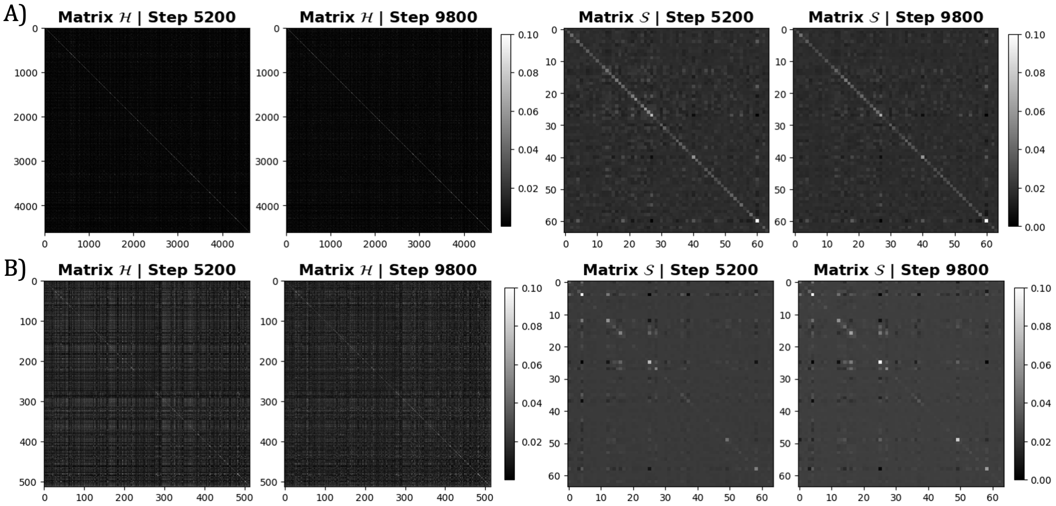

Figure 9: Visualization of Kronecker Factors $\mathcal{H}$ and $\mathcal{S}$ for convolutional (A) and linear (B) layers at different iteration steps within a ResNet-18 network. For the convolutional layer (37th layer), the first two plots in (A) represent factor $\mathcal{H}$ at steps 5200 (middle of training) and 9800 (end of training), elucidating the matrix's structure at these stages. The subsequent two plots display factor $\mathcal{S}$, highlighting changes in granularity and contrast with iteration progression. Similarly, in (B) for the linear layer (41st position), we observe the structural evolution of factor $\mathcal{H}$ and $\mathcal{S}$ over the same iterations, with variations in pattern density and clarity. These visualizations collectively underscore the dynamic nature of the Kronecker factors' architecture as training advances.

the developmental trajectory of the Kronecker factors. The enduring diagonal salience observed in both $\mathcal{H}$ and $\mathcal{S}$ underscores the notion that the informational energy of the Kronecker factors is predominantly concentrated along the diagonal. This persistent feature accentuates the structural stability and the focused nature of information encoding within the network's layers.

A.2 PROOFS

Proposition A.1. *Consider a neural network layer indexed by i, and a mini-batch $\mathbb{B} \subset \mathbf{D}$ of size M ($|\mathbb{B}| = M$). The empirical statistics of the Kronecker factors for the normalization layers can be characterized as follows:*

$$\mathcal{H}_{i-1} = \frac{\left(\sum_{\mathbb{B}} \sum_{\mathcal{T}} \bar{h}_{i-1}\right)^T \left(\sum_{\mathbb{B}} \sum_{\mathcal{T}} \bar{h}_{i-1}\right)}{(M|\mathcal{T}|)^2}, \; \mathcal{S}_i = \frac{\left(\sum_{\mathbb{B}} \sum_{\mathcal{T}} s_i\right)\left(\sum_{\mathbb{B}} \sum_{\mathcal{T}} s_i\right)^T}{M} \tag{8}$$

Here, $\mathcal{T}$ represents the spatial size dimension for Batch Norm layer and for LayerNorm layer, it signifies the product of the number of heads and the per-head dimension.

Proof. The justification of our approach will be split into two parts: the first for the Batch Normalization layer and the second for the Layer Normalization.

Part 1: Batch Normalization

Batch Normalization is a solution to the problem of internal covariance shifting for normalizing the layer inputs. For each activation $\bar{h}_{i-1}$ they introduce a pair of parameters ν_i and β_i which scale and shift the normalized value:

$$y_i = \nu_i \bar{h}_{i-1} + \beta_i \tag{9}$$

The FIM for Batch Normalization captures the sensitivity of the output with respect to the parameters νi and β_i. We introduce rescaling and shifting operations into the FIM formulation to adapt for BatchNorm parameters, enabling efficient FIM approximation. For the multiplication operation in BatchNorm (scaling factor), we adjust the FIM calculation using Eq. (8) incorporating the batch size and spatial dimension. This normalization ensures FIM accounts for the BatchNorm scaling factors. Similarly, for the addition operation involving bias terms, we seamlessly integrate biases into the FIM formulation, capturing their impact on gradient computation. The shape of the Kronecker factors are defined as:

$$\mathcal{H}_{i-1} \in \mathbb{R}^{2\times 2}, \; \mathcal{S}_i \in \mathbb{R}^{c_i \times c_i}$$

where c_i refers to the channel dimension of layer i.

Part 2: Layer Normalization

Layer Normalization normalizes the inputs across the features instead of the batch dimension and the same equation, Eq. (9), is also used for Layer Normalization. Similar to Batch Normalization we introduce rescaling and shifting operations into the FIM formulation but adapted to the normalization across features rather than the batch. In fact, $\mathcal{T}$ refers here to the product of the number of heads and the per-head dimension rather than the spatial size dimension.The shape of the Kronecker factors for LayerNorm are:

$$\mathcal{H}_{i-1} \in \mathbb{R}^{2\times 2}, \; \mathcal{S}_i \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{T}|}$$

□

Proposition A.2. *Let $\mathcal{H}_{i-1}$ and $\mathcal{S}_i$ represent the Kronecker factors for a given layer index i within a neural network, where these factors exhibit semi-diagonal characteristics indicating energy concentration predominantly along the diagonal, as elaborated in Section 3.1. Define g_i as the gradient obtained through backpropagation at layer i. Assume that $\mathcal{H}_{i-1}$ and $\mathcal{S}_i$ can be closely approximated by diagonal matrices, denoted by $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$ respectively at layer i, such that $\mathcal{H}_{D_{i-1}} = Diag(\mathcal{H}_{i-1})$, $\mathcal{S}_{D_i} = Diag(\mathcal{S}_i)$ where $Diag(\mathcal{M})$ denote the diagonal approximation of a matrix $\mathcal{M}$, which retains only the main diagonal. Therefore we define the Empirical FIM as:*

$$\tilde{F}_{D_i} \triangleq \mathcal{H}'_{D_{i-1}} \otimes \mathcal{S}'_{D_i} + \lambda, \tag{10}$$

where $\mathcal{M}'$ denotes the Min-Max normalization technique Patro & Sahu (2015) for $\mathcal{M} = \mathcal{H}_{D_{i-1}}$ or $\mathcal{S}_{D_i}$. The regularization parameter λ set to 0.001, serves as damping factors, in alignment with the principles of Tikhonov regularization, to enhance computational stability and improve the conditioning of the matrix. The foundational aspects of the K-FAC optimization approach are detailed in Martens & Grosse (2015). For a comprehensive account of the methodology and construction details, please consult Appendix A.2. Then, the closed-form solution for the augmented gradient $\hat{g}_i$, derived from the diagonal approximation of the FIM, is given by: $\hat{g}_i = \tilde{F}_{D_i}^{-1} g_i$.

Proof. The justification of our approach comprises two principal components: the rationale for adopting a diagonal approximation of the Kronecker factors, and the methodology for normalization and regularization of these factors.

Part 1: Diagonalization of Kronecker Factors

The assumption of independent neuronal activity within layers is foundational to our approach. This assumption posits that the covariance matrices $\mathcal{H}$ and $\mathcal{S}$, encapsulating the second-order statistics of activations and sensitivities, respectively, are diagonal. This diagonal nature arises because independence among random variables implies a covariance of zero for any pair of distinct variables, thereby nullifying all off-diagonal elements of these covariance matrices.

Consider matrices A and B, each being diagonal with elements a_{ii} and b_{jj}, respectively. The Kronecker product $A \otimes B$, by definition, generates elements $a_{ii} b_{jj}$ at the corresponding (i, j) positions. For diagonal A and B, this product maintains non-zero values exclusively at diagonal positions where $i = j$, resulting in:

$$A \otimes B = \text{diag}(a_{11} b_{11}, \ldots, a_{nn} b_{mm}), \tag{11}$$

yielding a purely diagonal matrix. Moreover, we have empirically demonstrated that the energy of the Kronecker factors is concentrated along the diagonal, as detailed in Sections 3.1 and A.1. These arguments supports our initial premise.

Part 2: Normalization and Regularization

Normalization plays a pivotal role in machine learning algorithms, particularly in ensuring numerical stability and improving convergence properties of optimization algorithms. When dealing with matrices such as $\mathcal{H}_{D_i}$ and $\mathcal{S}_{D_i}$, which exhibit a diagonal structure, normalization not only aids in adjusting the scale of matrix values but also addresses the issue of varying scales among different features. The adoption of Min-Max normalization for the diagonal elements $\mathcal{A}_i$ is especially advantageous as it standardizes the data to a fixed interval, commonly $[0, 1]$, which is crucial for many gradient-based optimization methods. The transformed matrix $\tilde{\mathcal{A}}_i$ is mathematically defined as:

$$\tilde{\mathcal{A}}_i = \frac{\mathcal{A}_i - \min(\mathcal{A}_i)}{\max(\mathcal{A}_i) - \min(\mathcal{A}_i)}, \tag{12}$$

where $\mathcal{A}_i$ represents the diagonal elements from either $\mathcal{H}_{D_i}$ or $\mathcal{S}_{D_i}$. This approach ensures that all elements are scaled uniformly, preserving their relative magnitudes and distances. The numerator, $\mathcal{A}_i - \min(\mathcal{A}_i)$, shifts the values so that the minimum element is zero. The denominator, $\max(\mathcal{A}_i) - \min(\mathcal{A}_i)$, scales the range of values to fit between zero and one. Compared to other normalization methods, such as z-score normalization (Patro & Sahu, 2015), Min-Max normalization offers the distinct benefit of bounding the values, which prevents problems associated with unbounded ranges that can adversely affect learning processes, particularly in networks sensitive to input magnitude. Moreover, Min-Max normalization is advantageous in scenarios where the parameters are influenced by activation functions like sigmoid or tanh, which are sensitive to input scale and function optimally within a defined range of $[0, 1]$ or $[-1, 1]$. Thus, normalization, specifically using the Min-Max method, is crucial for maintaining computational stability in algorithms by ensuring that all input features contribute equally to the analysis without any undue influence from outliers or disproportionately large feature values. This uniformity facilitates faster convergence during training and mitigates the risk of encountering vanishing or exploding gradient issues in neural networks.

Together, these components substantiate the proposition, demonstrating that our methodological innovations not only adhere to theoretical expectations but also offer practical advantages in computational stability and efficiency. $\square$

Proposition A.3. *For the FIM defined in Eq. (10), the updating scheme $\triangle\theta_t = \tilde{F}_t^{-1}\nabla J(\theta_t)$ converges. Moreover, if ∇J is Lipschitz, i.e., $||\nabla J(\theta) - \nabla J(\theta')||_2 \leq L||\theta - \theta'||$ for any θ and θ', then for the k-step iteration with a fixed step size $\delta \leq 1/L$, then*

$$J(\theta^{(k)}) - J(\theta^*) \leq \frac{||\theta^{(0)} - \theta^*||_2^2}{2\delta k},$$

where $J(\theta^*)$ is the optimal value.

Proof. For convenience, we denote $g_t := \nabla J(\theta_t)$. We follow the same proof as in Yao et al. (2021). Assume that $J(\theta)$ is a strongly convex and strictly smooth function in $\mathbb{R}^d$, such that there exist positive constants α and β so that $\alpha I \leq \nabla^2 J(\theta) \leq \beta I$ for all w. We can show that the update formulation $\triangle\theta_t = \tilde{F}_t^{-1} g_t$ converges by showing that with the proper learning rate:

$$\triangle\theta_t := J(\theta_{t+1}) - J(\theta_t) \leq -\frac{\alpha}{2\beta^2}||g_t||^2$$

Note that when $k = 0$ or 1, the convergence rate is the same as gradient descent or Newton method, respectively. Our proof is similar to Boyd & Vandenberghe (2004) for Newton method. We denote $\lambda(\theta_t) = (g_t^T \tilde{F}_t^{-1} g_t)^{1/2}$. Since $J(\theta)$ is strongly convex, we have

$$\begin{aligned} J(\theta_t - \eta\triangle\theta_t) &\leq J(\theta_t) - \eta g_t^T \triangle\theta_t + \frac{\eta^2\beta||\triangle\theta_t||^2}{2} \\ &\leq J(\theta_t) - \eta\lambda(\theta_t)^2 + \frac{\beta}{2\alpha}\eta^2\lambda(\theta_t)^2. \end{aligned}$$

The second inequality come from the fact that

$$\lambda(\theta_t)^2 = \triangle\theta_t^T \tilde{F}_t \triangle\theta_t \geq \alpha||\triangle\theta_t||^2.$$

Therefore, the step size $\hat{\eta} = \alpha/\beta$ will make f decrease as follows,

$$J(\theta_t - \hat{\eta}\triangle\theta_t) - J(\theta_t) \leq -\frac{1}{2}\hat{\eta}\lambda(\theta_t)^2.$$

Since $\alpha I \preceq \tilde{F}_t \preceq \beta I$, we have

$$\lambda(\theta_t)^2 = g_t^T \tilde{F}_t^{-1} g_t \geq \frac{1}{\beta}||g_t||^2.$$

Therefore,

$$J(\theta_t - \hat{\eta}\triangle\theta_t) - J(\theta_t) \leq -\frac{1}{2\beta}\hat{\eta}||g_t||^2 = -\frac{\alpha}{2\beta^2}||g_t||^2 \tag{13}$$

Since F_{D_t} is positive definite, hence Eq. (13) holds true. For the bound on convergence rate, we refer to Ryu & Boyd (2016) for the details of the complete proof.

□

Proposition A.4 (Convergence in nonconvex stochastic optimization)**.** *Under the assumptions:*
(i) f is lower bounded and differentiable; $||\nabla J(\theta) - \nabla J(\theta')||_2 \leq L||\theta - \theta'||_2$, $||\tilde{F}_{D_t}||_\infty < L, \forall t, \theta, \theta'$.
(ii) Both the true and stochastic gradient are bounded, i.e. $||\nabla J(\theta_t)||_2 \leq \lambda$ and $||g_t||_2 \leq \lambda$, $\forall t$ for some $\lambda > 0$.
(iii) Unbiased and independent noise in g_t, i.e. $g_t = \nabla J(\theta_t) + \zeta_t$, $\mathbb{E}[\zeta_t] = 0$, and $\zeta_i \perp \zeta_j$, $\forall i \neq j$.

Assume $\eta_t = \frac{\eta}{\sqrt{t}}$, $\beta_t \leq \beta \leq 1$ is non-increasing, $\frac{\tilde{F}_{D_{t-1}}[j]}{\eta_{t-1}} \leq \frac{\tilde{F}_{D_t}[j]}{\eta_t}$, $\forall t \in [T], j \in [d]$, we then have

$$\min_{t\in[T]} \mathbb{E}[||\nabla J(\theta_t)||_2^2] \leq \frac{L}{\sqrt{T}}(C_1\eta^2\lambda^2(1 + \log T) + C_2 d\eta + C_3 d\eta^2 + C_4) \tag{14}$$

where C_1, C_2, C_3 are constants independent of d and T, C_4 is a constant independent of T, the expectation is taken w.r.t all the randomness corresponding to $\{g_t\}$.

Proof. Follow Chen et al. (2019), as AdaFisher is an Adam-type method with the condition $||\eta_t m_t/\tilde{F}_{D_t}||_2 \leq G$ for some G (which can be obtained by $\eta_t < \eta$, $||g_t||_2 \leq \lambda$ and $||\tilde{F}_{D_t}||_2 \geq 1$), we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T}\eta_t\langle\nabla J(\theta_t), \nabla J(\theta_t)/\tilde{F}_{D_t}\rangle\right] \leq &\mathbb{E}\left[C_1\sum_{t=1}^{T}\left\|\frac{\eta_t g_t}{\tilde{F}_{D_t}}\right\|_2^2 + C_2\sum_{t=1}^{T}\left\|\frac{\eta_t}{\tilde{F}_{D_t}} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}}\right\|_1\right.\\ &\left. + C_3\sum_{t=1}^{T}\left\|\frac{\eta_t}{\tilde{F}_{D_t}} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}}\right\|_2^2\right] + C_4. \end{aligned} \tag{15}$$

We first bound non-constant terms in RHS of Eq. (15). For the term with C_1, since $||\tilde{F}_{D_t}||_2 \geq 1$, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta_t g_t}{\tilde{F}_{D_t}}\right\|_2^2\right] &\leq \mathbb{E}\left[\sum_{t=1}^{T}||\eta_t g_t||_2^2\right]\\ &= \mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta}{\sqrt{t}}g_t\right\|_2^2\right]\\ &\leq \eta^2\lambda^2\sum_{t=1}^{T}\frac{1}{t} \leq \eta^2\lambda^2(1+\log T).\end{aligned}$$

For the term with C_2, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta_t}{\tilde{F}_{D_t}} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}}\right\|_1\right] &= \mathbb{E}\left[\sum_{j=1}^{d}\sum_{t=2}^{T}\left(\frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}[j]} - \frac{\eta_t}{\tilde{F}_{D_t}[j]}\right)\right]\\ &= \mathbb{E}\left[\sum_{j=1}^{d}\frac{\eta_1}{\tilde{F}_{D_1}[j]} - \frac{\eta_T}{\tilde{F}_{D_T}[j]}\right]\\ &\leq \mathbb{E}\left[\sum_{j=1}^{d}\frac{\eta_1}{\tilde{F}_{D_1}[j]}\right] \leq d\eta\end{aligned}$$

where the first equality is due to $\frac{\tilde{F}_{D_{t-1}}[j]}{\eta_{t-1}} \leq \frac{\tilde{F}_{D_t}[j]}{\eta_t}, \forall t \in [T], j \in [d]$.

For the term with C_3, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta_t}{\tilde{F}_{D_t}} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}}\right\|_2^2\right] &= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{d}\left(\frac{\eta_t}{\tilde{F}_{D_t}[j]} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}[j]}\right)^2\right]\\ &= \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{d}\left|\frac{\eta_t}{\tilde{F}_{D_t}[j]} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}[j]}\right| \cdot \left|\frac{\eta_t}{\tilde{F}_{D_t}[j]} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}[j]}\right|\right]\\ &\leq \mathbb{E}\left[\sum_{t=1}^{T}\sum_{j=1}^{d}\left|\frac{\eta_t}{\tilde{F}_{D_t}[j]} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}[j]}\right| \cdot \left|\frac{\eta}{\sqrt{t}\tilde{F}_{D_t}[j]} - \frac{\eta}{\sqrt{t-1}\tilde{F}_{D_{t-1}}[j]}\right|\right]\\ &\leq \mathbb{E}\left[\eta\sum_{t=1}^{T}\sum_{j=1}^{d}\left|\frac{\eta_t}{\tilde{F}_{D_t}[j]} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}[j]}\right|\right]\\ &= \eta\mathbb{E}\left[\sum_{t=1}^{T}\left\|\frac{\eta_t}{\tilde{F}_{D_t}} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}}\right\|_1\right]\\ &\leq d\eta^2\end{aligned}$$

Hence

$$\begin{aligned}&\mathbb{E}\left[C_1\sum_{t=1}^{T}\left\|\frac{\eta_t g_t}{\tilde{F}_{D_t}}\right\|_2^2 + C_2\sum_{t=1}^{T}\left\|\frac{\eta_t}{\tilde{F}_{D_t}} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}}\right\|_1 + C_3\sum_{t=1}^{T}\left\|\frac{\eta_t}{\tilde{F}_{D_t}} - \frac{\eta_{t-1}}{\tilde{F}_{D_{t-1}}}\right\|_2^2\right] + C_4\\ &\leq C_1\eta^2\lambda^2(1+\log T) + C_2 d\eta + C_3 d\eta^2 + C_4\end{aligned} \tag{16}$$

Now we lower bound the LHS of Eq. (14). With the assumption $||\tilde{F}_{D_t}||_\infty \leq L$, we have

$$(\eta_t/\tilde{F}_{D_t})_j \geq \frac{\eta}{L\sqrt{t}}.$$

Thus

$$\mathbb{E}\left[\sum_{t=1}^{T} \eta_t \langle \nabla J(\theta_t), \nabla J(\theta_t)/\tilde{F}_{D_t} \rangle\right] \geq \mathbb{E}\left[\sum_{t=1}^{T} \frac{\eta}{L\sqrt{t}} ||\nabla J(\theta_t)||_2^2\right] \geq \frac{\sqrt{T}}{L} \min_{t\in[T]} \mathbb{E}[||\nabla J(\theta_t)||_2^2] \quad (17)$$

Combining Eq. (16) and (17) gives the desired result. □

A.3 COMPUTATION OF KRONECKER FACTORS

The Kronecker factors $\mathcal{H}$ and $\mathcal{S}$, which are integral to the AdaFisher optimizer, are computed following methodologies similar to those described in Grosse & Martens (2016). This section revisits the key equations used for this computation. For a given layer i in a neural network, consider a mini-batch $\mathbb{B} \subset \mathbf{D}$, where $|\mathbb{B}| = M$. The empirical Kronecker factors are computed as follows:

- For **fully connected layers**, the Kronecker factors are:

$$\mathcal{H}_{D_{i-1}} = \text{diag}\left(\frac{\overline{h}_{i-1}^T \overline{h}_{i-1}}{M}\right), \quad \mathcal{S}_{D_i} = \text{diag}\left(\frac{s_i^T s_i}{M}\right);$$

- For **convolutional layers**, the computation accounts for the spatial positions within the layer, denoted as $\mathcal{T}$:

$$\mathcal{H}_{D_{i-1}} = \text{diag}\left(\frac{[\![\overline{h}_{i-1}]\!]^T [\![\overline{h}_{i-1}]\!]}{M|\mathcal{T}|}\right), \quad \mathcal{S}_{D_i} = \text{diag}\left(\frac{s_i^T s_i}{M|\mathcal{T}|}\right);$$

  The algorithm employs the expansion operation denoted by $[\![\cdot]\!]$ (Grosse & Martens, 2016). *This operation essentially takes the patches surrounding spatial locations, stretches them into vectors, and compiles these vectors into a matrix*

- For **Normalization layers** (BatchNorm & LayerNorm) please refer to Proposition. 3.1
- For all **other type of layers** the Kronecker factors are:

$$\mathcal{H}_{D_{i-1}} = \mathbf{I}_{d_{i-1}}, \quad \mathcal{S}_{D_i} = \mathbf{I}_{d_i};$$

  where d_i denotes the dimension of the ith layer and $\mathbf{I}$ is the identity matrix.

Table 6: AdaFisher training time per epoch (s) across various numbers of GPUs on ResNet-50 ImageNet.

GPU amount	Batch Size	AdaFisher training time per epoch (s)
1	256	2882
2	512	1438
3	768	963
4	1024	720

A.4 DISTRIBUTED ADAFISHER

The efficacy of AdaFisher hinges on its innovative approximation of the FIM, denoted as $\tilde{F}$, which leverages Kronecker factors for computation. In a distributed setting, it is crucial to aggregate these Kronecker factors across multiple GPUs before updating the model parameters. Consider a training environment consisting of N GPUs. For any given layer i, the Kronecker factors are computed and aggregated across all GPUs as

$$(\mathcal{H}_{D_{i-1}})^{\text{SUM}} = \frac{1}{N}\sum_{n=1}^{N}(\mathcal{H}_{D_{i-1}})^n, \quad (\mathcal{S}_{D_i})^{\text{SUM}} = \frac{1}{N}\sum_{n=1}^{N}(\mathcal{S}_{D_i})^n \quad (18)$$

The theoretical justification for this aggregation lies in the linearity of expectation and the unbiasedness of the local Kronecker factor estimates. Specifically, if each $(\mathcal{H}_{D_{i-1}})^n$ and $(\mathcal{S}_{D_i})^n$ are unbiased estimators of their respective true factors $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$, then the averaged factors $(\mathcal{H}_{D_{i-1}})^{\text{SUM}}$ and $(\mathcal{S}_{D_i})^{\text{SUM}}$ remain unbiased estimators of $\mathcal{H}_{D_{i-1}}$ and $\mathcal{S}_{D_i}$. Consequently, the aggregated EFIM for layer i can be calculated as

$$\tilde{F}_{D_i}^{\text{SUM}} = (\mathcal{H}'_{D_{i-1}})^{\text{SUM}} \otimes (\mathcal{S}'_{D_i})^{\text{SUM}} + \lambda$$

where λ is a regularization parameter added to ensure numerical stability. This methodology ensures that each GPU contributes to a comprehensive update of the model, enhancing both convergence and performance in large-scale distributed training environments. We assessed the distributed version of AdaFisher on ImageNet, utilizing batch sizes of 512 and 1024 (refer to Table 3 and Figure 15 for details). Our findings indicate that AdaFisher scales nearly linearly with the number of GPUs, as evidenced in Table 6. There remains scope for additional low-level optimizations within the implementation to further enhance performance.

B ABLATION STUDIES

Building on the ablative studies detailed in Section 4.4, this section extends our stability analysis to explore the impact of various learning rate schedulers and convergence efficiency, as discussed in Section B.1. Additionally, we conduct an in-depth examination of the key components of AdaFisher. This includes analyzing the effects of the EMA, the use of square root transformations, our novel approximation of the FIM, and the critical role of computing the FIM for normalization layers, all of which are detailed in Section B.2. We have consolidated the key findings of each ablation study in Table 7.

Table 7: Summary of Ablation Studies for AdaFisher Optimizer.

Ablation Study	**Component Studied**	**Key Findings**
Learning rate schedulers	Impact of Cosine Annealing, StepLR, and no scheduler on AdaFisher	AdaFisher maintains stable and efficient performance across various schedulers, demonstrating its robustness and adaptability in diverse training environments. For further details please refer to Section B.1.
Convergence Efficiencys	Performance and alignment of FIM with Hessian	AdaFisher shows marked performance improvements towards the end of training, with FIM alignment to the Hessian enhancing rapid convergence and stable generalization across training and testing phases. For further details please refer to Section B.1.
Square Root Utilization	Effect of omitting square root in update rules	Eliminating the square root enhances AdaFisher's performance and stability, outperforming both its own version with the square root and Adam without the square root, while also improving computational efficiency. For further details please refer to Section B.2.
EMA of Kronecker Factors	Utilization of EMA for curvature estimation	Using EMA on Kronecker factors enhances AdaFisher's curvature estimation, leveraging data from multiple mini-batches for continuous updates, demonstrating significant benefits in methods with diagonal or block-diagonal curvature approximations. For further details please refer to Section B.2.
Importance of Fisher Computation for Normalization Layers	Impact of EFIM in normalization layers	Incorporating Fisher computation in normalization layers significantly improves AdaFisher's generalization and stability by enhancing parameter sensitivity and gradient variability insights, crucial for optimizing training dynamics and model convergence. For further details please refer to Section B.2.
New Approximation of the FIM	Diagonal approximation of the FIM	Our novel method focuses on the diagonal elements of the FIM, enhancing computation efficiency without losing critical information. Validation shows our approximation closely aligns with the true Fisher, confirming its efficacy. For further details please refer to Section B.2.

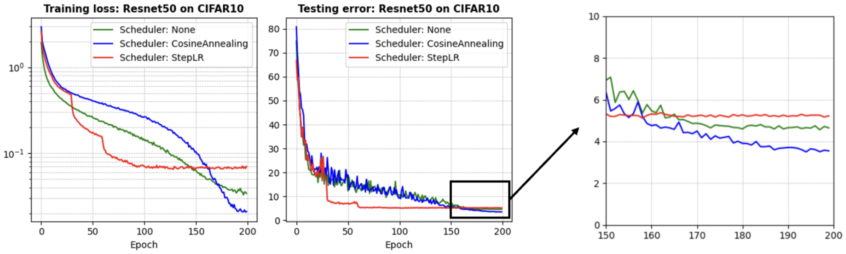

Figure 10: Performance comparison of AdaFisher using the ResNet50 on the CIFAR10 with batch size of 256 with different learning rate schedulers.

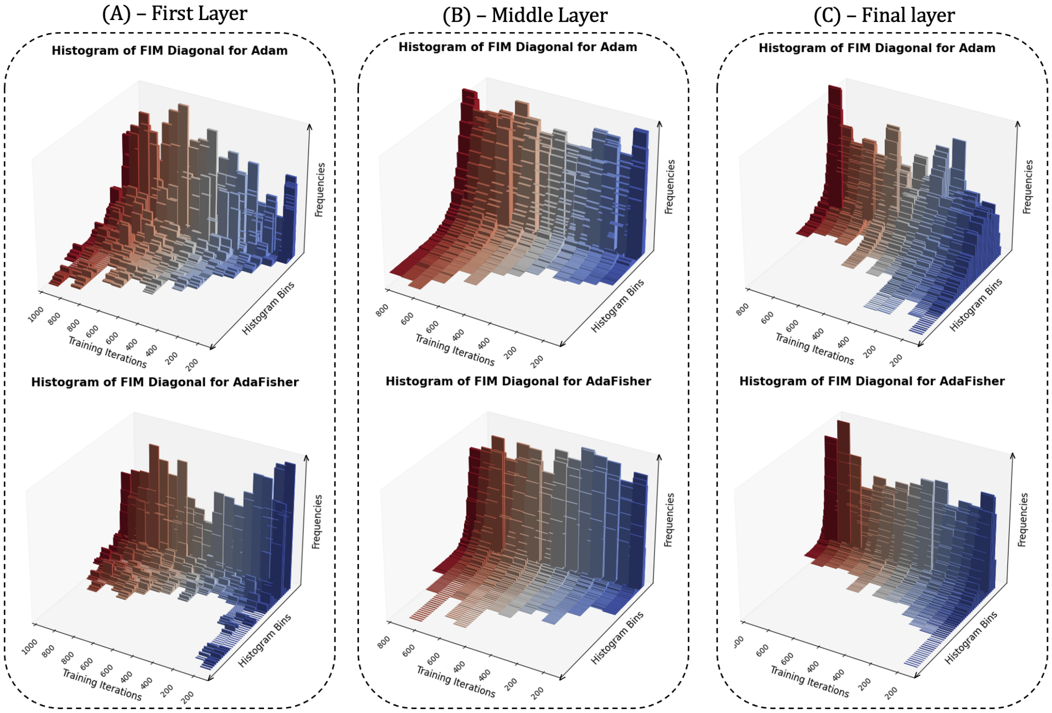

Figure 11: Comparison of FIM Diagonal Histograms during ResNet18 Training on CIFAR10 with Adam and AdaFisher over 1,000 training iterations. Panel (A) displays the FIM diagonal elements for the first convolutional layer; Panel (B) illustrates the FIM diagonal elements for the middle convolutional layer; Panel (C) shows the FIM diagonal elements for the last Linear layer.

B.1 EVALUATING STABILITY ACROSS LEARNING RATE SCHEDULERS, AND ASSESSING CONVERGENCE EFFICIENCY

Learning rate schedulers. This analysis evaluates the impact of different learning rate schedulers–Cosine Annealing, StepLR, and no scheduler—on the performance of AdaFisher, as depicted in Figure 10. AdaFisher exhibits remarkable robustness across these scheduling strategies. Notably, its performance remains stable and efficient whether it is paired with the gradual adjustments of Cosine Annealing, the abrupt changes of StepLR, or even in the absence of any scheduler. This underscores AdaFisher's adaptability and effectiveness in diverse training environments.

Convergence Efficiency. As training progresses, AdaFisher optimizer demonstrates a significant enhancement in performance compared to its counterparts, especially evident towards the end of the training period (see Appendix D.2.4). This rapid convergence is attributed to AdaFisher's approach by incorporating the FIM. Early and mid-training, the FIM serves as an approximation to the Hessian matrix, equivalent to the Generalized Gauss Newton Matrix (Eschenhagen et al., 2024). However, as the model approaches a local minimum, the FIM increasingly aligns precisely with the Hessian (Martens, 2020). This precise alignment accelerates convergence, markedly improving the optimizer's efficiency in the final phases of training. Additionally, AdaFisher's tendency to converge to flat local minima leads to more stable generalization when transitioning from training to testing distributions (Cha et al., 2021), contrasting sharply with other optimizers. To support these points, we analyze the training distribution of our diagonal block-Kronecker FIM during the training of ResNet18 on CIFAR10. Specifically, we examine the FIM distribution for the first (Panel A), middle (Panel B) convolutional layers and the last linear layer (Panel C), as shown in Figure 11. It is evident that for each layer, the FIM distribution with AdaFisher narrows to smaller values with fewer variations compared to that with Adam. This pattern demonstrates AdaFisher's convergence toward flatter local minima, as the Fisher Information, approximation of the Hessian, containing crucial curvature information.

B.2 COMPONENT ANALYSIS: EVALUATING THE SIGNIFICANCE OF ADAFISHER'S ELEMENTS

AdaFisher incorporates several key components, including a novel approximation of the FIM, the EMA of the Kronecker factors, the omission of the square root in the update rule, and a new EFIM formula for normalization layers. In this part, we elucidate each component and its significance within the AdaFisher optimizer.

Square Root Utilization. Recent studies, such as (Lin et al., 2024b), have reevaluated the necessity of the square root operation in the Adam family's update rules. These studies suggest that eliminating the square root does not affect convergence and may even narrow the generalization gap compared to SGD in CNN models. Our analysis, shown in panel (A) of Figure 12, investigates this aspect by comparing the performance of AdaFisher and Adam, both with and without the square root operation. The findings reveal that removing the square root not only boosts the performance and stability of both optimizers but also significantly enhances computational efficiency. Specifically, AdaFisher without the square root not only outperforms the version with the square root but also surpasses Adam without the square root. However, Adam without the square root typically requires an additional **scaling factor** proportional to the batch size, denoted as $f \propto$ batch size, to function correctly. Without this factor, Adam without the square root fails to learn effectively, making direct comparisons with AdaFisher invalid.

EMA of Kronecker Factors. As elucidated in Section 3.2, employing an EMA over the Kronecker factors facilitates a more sophisticated curvature estimation. This technique leverages data across multiple mini-batches, enabling continuous updates to the Fisher information rather than relying solely on the data from a single batch. Panel (B) of Figure 12 underscores, using ResNet-50 on CIFAR10 over 200 epochs, the benefits of using EMA on Kronecker factors, a strategy particularly advantageous in methods that utilize diagonal or block-diagonal approximations of the curvature matrix.

Importance of Fisher Computation for Normalization Layers. The integration of the EFIM in normalization layers, as detailed in Proposition 3.1, significantly enhances the generalization process. Panel (C) of Figure 12 illustrates the impact of incorporating Fisher computation in these layers during the training of AdaFisher with ResNet-50 on CIFAR10 over 200 epochs. In contrast, the identity matrix is employed when Fisher computation is omitted. The superior performance of AdaFisher when incorporating Fisher computation can be attributed to the critical role normalization layers play in adjusting the input distribution for each mini-batch. This adjustment substantially enhances the neural network's learning stability (Jiang et al., 2024b). By quantifying the information each output y carries about the parameters θ under the model distribution $p(y \mid x;\theta)$, the computation of the FIM in these layers provides valuable insights into parameter sensitivity and gradient variability. This insight is crucial for optimizing training dynamics and enhancing model convergence—areas that are often inadequately addressed by existing optimizers.

New Approximation of the FIM. In Proposition 3.2, we introduce a new methodology for approximating the FIM that diverges from the K-FAC optimizer. Unlike K-FAC, which utilizes the full Kronecker product, our approach focuses solely on the diagonal elements of the FIM, where, as demonstrated in Section 3.1, the energy of the Kronecker factors is predominantly concentrated. This method enables a more efficient computation of the FIM without sacrificing critical information. To validate our approach, we compare the true FIM diagonal with our approximation in convolutional and dense layers using a toy model composed of 2 convolutional layers and 2 linear layers on a subset of the MNIST dataset (Deng, 2012) over 50 epochs. Specifically, we calculate the true Fisher using the NNgeometry Python package (George, 2021), which facilitates the computation of the FIM, Gauss-Newton Matrix, or Neural Tangent Kernels applied to neural networks. We estimate $p(y \mid x)$ through Monte-Carlo sampling. During each epoch, we collected both the empirical and true Fisher information and calculated the Mean Absolute Error (MAE) between these two measures. Panel (D) of Figure 12 showcases the close approximation of AdaFisher's empirical diagonal to the true Fisher, thus validating the efficacy of our approximation method.

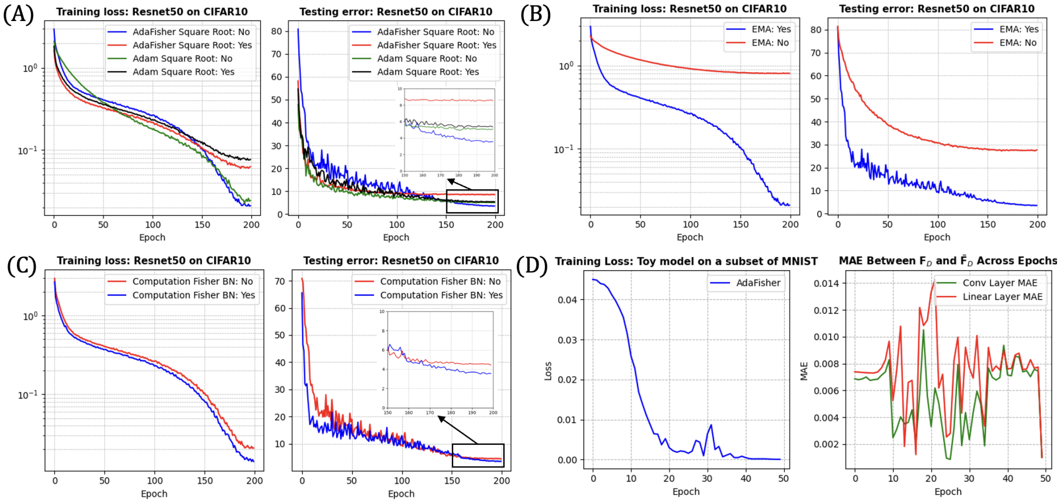

Figure 12: AdaFisher Component Analysis. (A) Comparison of MAE between the true FIM F_D and our approximation $\tilde{F}_D$ across convolutional and dense layers. (B) Performance comparison of AdaFisher with and without the EMA of Kronecker factors. (C) Assessment of AdaFisher's performance with and without the computation of EFIM for Batch Normalization (BN) layers.

C VISUALIZATION

The convergence rate of an optimizer is crucial, serving as an indicator of its robustness against saddle points and its ability to generalize effectively. In this section, we introduce a novel methodology for visualizing the convergence behavior of optimizers through a statistical model, as depicted in Figure 1. Initially, our process employs Principal Component Analysis (PCA) for dimensionality reduction, reducing the dataset dimensions from $\mathcal{D} \in \mathbb{R}^{m \times n}$ to $\hat{\mathcal{D}} \in \mathbb{R}^{m \times 2}$, following the protocol established in F.R.S. (1901). We then apply this reduced dataset to a toy model composed of an L-layer multi-layer perceptron (MLP). Notably, we focus on the first weight matrix W_1^e of this MLP, which resides in $\mathbb{R}^2$, where e denotes the epoch number. For consistency and to ensure comparability, all layers' weights are initialized identically across different optimizers. Following the training phase with various optimizers where we denote a set of optimizer results $\mathcal{O}$, we analyze both the

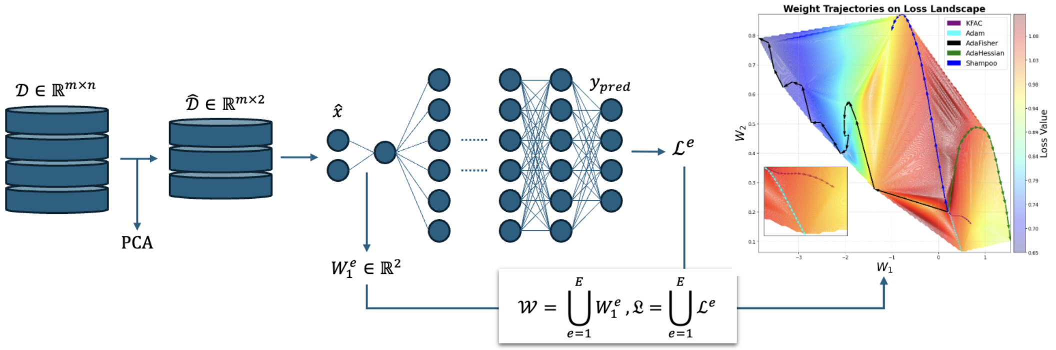
$\mathcal{D} \in \mathbb{R}^{m \times n}$
$\hat{\mathcal{D}} \in \mathbb{R}^{m \times 2}$
PCA
$\hat{x}$
y_{pred}
$\mathcal{L}^e$
$W_1^e \in \mathbb{R}^2$
$\mathcal{W} = \bigcup_{e=1}^{E} W_1^e, \mathfrak{L} = \bigcup_{e=1}^{E} \mathcal{L}^e$
Weight Trajectories on Loss Landscape
KFAC
Adam
AdaFisher
AdaHessian
Shampoo
W_1
W_2
Loss Value

Figure 13: Pipeline for visualization of optimization paths for various algorithms on a loss surface, comparing their convergence efficiency.

collection of first-layer weights, $\mathcal{W}$, and the evolution of the loss function, $\mathfrak{L}$ defined as:

$$\mathcal{W} = \begin{bmatrix} (W_1^1)^\top \\ (W_1^2)^\top \\ \vdots \\ (W_1^E)^\top \end{bmatrix}, \quad \mathfrak{L} = [\mathcal{L}_1^1, \mathcal{L}_1^2, \dots, \mathcal{L}_1^E]^\top$$

where $(W_1^e)^\top$ represents the weight vector at the e-th epoch, and $\mathcal{L}_1^e$ represents the loss at the e-th epoch, extracted from the optimization results $\mathcal{O}$. We construct a grid $(\mathbf{X}, \mathbf{Y})$ spanning the range of weight parameters, discretized into 200 linearly spaced points along each axis:

$$\mathbf{X}, \mathbf{Y} = \text{meshgrid}\left(\min(\mathcal{W}_{:,1}), \max(\mathcal{W}_{:,1}), \min(\mathcal{W}_{:,2}), \max(\mathcal{W}_{:,2}), 200\right)$$

Finally, we interpolate the loss values $\mathcal{L}$ over the grid using cubic interpolation to obtain a smooth loss surface $\mathbf{Z}$:

$$\mathbf{Z} = \text{griddata}(\mathcal{W}, \mathcal{L}, (\mathbf{X}, \mathbf{Y}), \text{method} =' cubic')$$

These elements are integral to the visualization process, which elucidates the optimizer's trajectory through the parameter space across training epochs. It is important to note that while we focus on the first layer's weight matrix for clarity, the methodology can be adapted to visualize the weights of any layer within the network. Figure 13 summarizes the pipeline.

In the experiment depicted in Figure 1, we selected the IRIS dataset (rz7, 2018), owing to its widespread recognition and compatibility with PCA application. Our model employs a 2-layer MLP architecture. We specifically attend to the weight matrix of the first layer, denoted by $W_1 \in \mathbb{R}^2$. This particular focus is informed by the empirical observation that the parameters of the first layer tend to exhibit a faster convergence rate compared to those of subsequent layers in the network. Such a phenomenon can be attributed to the more direct influence of the input features on the first layer's weights, which often results in a more pronounced and expedited learning dynamic. Given the classification nature of the task, we employed the Cross-Entropy loss function (Zhang & Sabuncu, 2018). The network was trained over 20 epochs using a suite of optimizers: Adam, AdaHessian, K-FAC, Shampoo, and AdaFisher. We standardized the learning rate across all optimizers at 1×10^{-3} to ensure comparability of results. Examination of Figure 1 reveals that AdaFisher's convergence is markedly superior to that of its counterparts, achieving rapid convergence to the local minimum of the loss landscape concerning the first weight parameter within a few iterations. Conversely, the alternative optimizers demonstrate convergence to less optimal local minima. Note that while the results may vary due to the stochastic nature of parameter initialization, AdaFisher typically converges to a better local minimum compared to its counterparts.

D EXPERIMENTS

D.1 HARDWARE

In total, we had a server with 6 NVIDIA RTX 6000 Ada Generation GPUS with 48 gigabytes of VRAM, and 128 gigabytes of RAM available for all experiments. All experiments described in this report were conducted on a system equipped with a single NVIDIA RTX 6000 Ada Generation GPU and 64 gigabytes of RAM, except for training AdaFisher on ImageNet with batch sizes of 512 and 1024, where four GPUs were utilized.

D.2 IMAGE CLASSIFICATION

We provide further results and detailed descriptions of our image classification experiments in this section. We conducted five trials with random initializations for the CIFAR experiments, and one trial each for Tiny ImageNet and ImageNet. We present the mean and standard deviation of the results for these trials.

Note on training time. Given that various optimizers demonstrate significantly different epoch durations, we have standardized our comparisons by restricting training to the total WCT consumed by 200 epochs using AdaFisher for both CIFAR and Tiny ImageNet experiments. Conversely, for ImageNet, we report the results based on 90 WCT training epochs using Adam, as, surprisingly, AdaFisher and Adam exhibited the same duration in this experiment. The final selected number of epochs for each optimizer is detailed in Table 8. Please note that we were unable to train AdaHessian on ImageNet due to the significant computational resources required by this optimizer.

Table 8: Comparison of the final epoch counts for various optimizers across different datasets.

	CIFAR10/100 & Tiny ImageNet						ImageNet			
Optimizers	SGD	Adam/AdamW	AdaHessian	K-FAC	Shampoo	AdaFisher/AdaFisherW	Adam	K-FAC	Shampoo	AdaFisher
Epochs	226	210	89	107	36	200	90	60	26	90

D.2.1 HYPERPARAMETER TUNING

Effective hyperparameter tuning is crucial for optimizing the performance of deep learning models. In this study, we systematically explored various hyperparameters for both CNNs and ViTs across multiple image classification tasks. The following subsections detail the tuning strategies employed for each model architecture and dataset.

CNNs. For all image classification tasks involving CNNs, we utilized ResNet18 as the backbone architecture and evaluated its performance on the CIFAR-10 dataset with a fixed batch size of 256 trained on 50 epochs. The hyperparameter tuning process encompassed the following components:

- **Optimizer Selection and Learning Rate Tuning:** Each optimizer was fine-tuned using ResNet18 on CIFAR-10. We performed a grid search to identify the optimal learning rate from the set $\{0.0001, 0.0003, 0.0005, 0.0009, \ldots, 0.1, 0.3, 0.5, 0.9\}$.
- **Learning Rate Scheduling:** A cosine annealing learning rate decay strategy was employed, aligning with the number of training epochs specified for each optimizer in Table 8. This approach follows the methodology proposed by Loshchilov & Hutter (2016) and was determined to be optimal for our experimental setup.
- **Weight Decay:** We applied a uniform weight decay of 5×10^{-4} across all optimizers for CIFAR-10 and Tiny ImageNet. An exception was made for MobileNetV3, where the weight decay was set to 1×10^{-5}. For experiments on ImageNet, the weight decay was established at 1×10^{-4}.
- **Damping Parameter Tuning:**
  - **AdaFisher, K-FAC, and Shampoo:**
    - **K-FAC and AdaFisher:** The damping parameter was searched within $\{0.0001, 0.0003, 0.0005, 0.0009, 0.001, 0.003, 0.005, 0.009, 0.01, 0.03, 0.05, 0.09\}$. This range was chosen based on prior research (Martens & Grosse, 2015) and our own experiments, which indicated optimal damping values around 1×10^{-3}.
    - **Shampoo:** The damping parameter was tuned within $\{1 \times 10^{-6}, 3 \times 10^{-6}, 5 \times 10^{-6}, 9 \times 10^{-6}, 1 \times 10^{-5}, 3 \times 10^{-5}, 5 \times 10^{-5}, 9 \times 10^{-5}, 1 \times 10^{-4}, 3 \times 10^{-4}, 5 \times 10^{-4}, 9 \times 10^{-4}\}$, as optimal values typically reside around 1×10^{-5}.
  - **AdaHessian:** The Hessian power was tuned within the range $\{0.1, 0.2, \ldots, 0.9, 1.0\}$.
  - **SGD:** The momentum of SGD was tuned within the range $\{0.1, 0.2, \ldots, 0.9, 1.0\}$.
  - **AdaFisher Decay Factors:** The decay factor γ for AdaFisher was tuned within $\{0.1, 0.2, \ldots, 0.9, 0.91, \ldots, 0.99\}$. The optimal value is: $\gamma = 0.92$.
- **Implementation Details:** For the Shampoo and K-FAC optimizers, we utilized the ASDL library as implemented in PyTorch provided by Osawa et al. (2023).

ViTs. For ViT-based image classification tasks, we employed the Tiny Swin Transformer on the CIFAR-10 dataset with a batch size of 256. The hyperparameter tuning strategy for ViTs included the following elements:

- **Weight Decay:** Weight decay values were set as indicated in the respective original publications for each model:
  - Tiny Swin: 1×10^{-2}
  - FocalNet: 5×10^{-2}
  - CCT-2/3×2: 6×10^{-2}
- **Learning Rate Tuning:** For SGD, AdaFisher, AdaHessian, K-FAC, and Shampoo optimizers, we conducted a grid search over the learning rates

$\{0.3, 0.15, 0.1, 0.05, 0.03, 0.015, 0.01, 0.005, 0.003, 0.0015, 0.001\}$, as these optimizers typically operate with higher learning rates compared to Adam-based optimizers. For AdamW, the learning rates were adopted from the original publications:

  - Tiny Swin and FocalNet: 1×10^{-4}
  - CCT-2/3×2: 5.5×10^{-5}

- **Damping Parameter Tunning:** We performed the same grid search over the damping parameter for K-FAC, Shampoo and AdaFisher, the Hessian power for AdaHessian, the momentum for SGD, and the decay factors for AdaFisher as explained in the CNNs part.

This meticulous hyperparameter tuning process ensures that each optimizer is optimally configured for the respective model architectures and datasets, thereby facilitating a fair and comprehensive comparison of their performance across different image classification tasks. The final learning rates for all optimizers and models are detailed in Table 9.

Table 9: Final selected learning rates for each optimizer, tuned using ResNet18 (for CNN) and Tiny Swin (for ViT) on CIFAR10 using a batch size of 256. We selected based on final validation top-1 accuracy.

Architecture	SGD	Adam	AdamW	AdaHessian	K-FAC	Shampoo	AdaFisher	AdaFisherW
CNNs	0.1	0.001	-	0.15	0.3	0.3	0.001	-
ViTs	0.01	-	0.0001/0.000055	0.01	0.003	0.003	-	0.001

D.2.2 DATASET DETAILS

CIFAR. The training/test sets for Cifar10/100 dataset contain 50k/10k images, respectively. We consider a batch size of 256. For CIFAR-related experiments, we perform 32×32 random-resize cropping, random horizontal flipping and cutout (DeVries & Taylor, 2017) as data augmentations. Please refer to Takahashi et al. (2020) for more details.

Tiny ImageNet. The training/test sets for TinyImageNet Le & Yang (2015) contains 100k/10k images. We perform 64×64 random-resize cropping and random horizontal flipping. The batch size is set to be 256.

ImageNet. The training/test sets for ImageNet Russakovsky et al. (2015) contains 1,281,167/150k images. We consider a batch size of 256, as we performed experiments on a single GPU instance without any GPU parallelism. We follow He et al. (2016) and perform random resized cropping to 224×244 and random horizontal flipping on the train set and 256×256 resizing with 224×224 center cropping on the test set.

Table 10: Final selected learning rates for each optimizer with ImageNet-1k pretrained weights, tuned using ResNet50 on CIFAR10 using a batch size of 256. We tuned by completing a full WCT epoch training cycle, and selected based on final validation top-1 accuracy.

SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
0.01	0.0001	0.15	0.3	0.03	0.001

Table 11: Final selected epoch counts for various optimizers across transfer learning task.

SGD	Adam/AdamW	AdaHessian	K-FAC	Shampoo	AdaFisher/AdaFisherW
58	55	22	27	18	50

D.2.3 TRANSFER LEARNING

Weights are initialized to the values provided by the publicly available checkpoints by PyTorch, except the first convolutional for the ResNet architecture and last dense layers for all networks, which change size to accomodate the new kernel size and number of classes respectively, that are randomly initialized. We train all models with weight decay $1e^{-4}$ as suggested in Wightman et al. (2021), expect for MobileNetV3 where weight decay is set to be $1e^{-5}$. Moreover, we did a grid search for each optimizer for selecting the best learning rate of the range $\{0.3, 0.15, 0.1, 0.03, 0.015, 0.01, \ldots, 1e-5\}$ where we tabulate the selected learning rate for each optimizer in Table 10. We use a batch size of 256 and cosine learning rate decay. We use the same augmentation policy (without Cutout) as in the previous experiments. The results were obtained using the WCT technique over 50 training epochs of AdaFisher, with the final epoch count detailed in Table 11. All other parameters remained unchanged.

Table 12: Performance of various networks and optimizers on TinyImegneNet using batch size 256. Reported using wall clock time of 200 AdaFisher training epochs as the cutoff.

Network	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet50	53.06	50.21	50.05	53.53	**57.41**
Big Swin	48.11	–	8.89	4.11	**48.86**

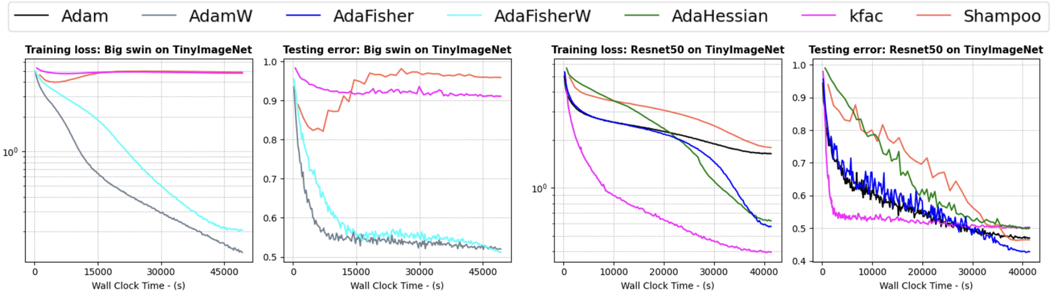

Figure 14: WCT training loss and testing error curves of several optimizers on Tiny ImageNet dataset, ResNet-50 and Big Swin with batch size of 256. AdaFisher consistently achieves lower test error as compared to Adam, AdaHessian, K-FAC and Shampoo. The final accuracy results are reported in Table 12.

Table 13: Performance metrics (mean, std) of different networks and optimizers on CIFAR10 and CIFAR100 using batch size 256 (a) without Cutout and (b) with Cutout. Reported using WCT of 200 AdaFisher training epochs as the cutoff.

(a) Without Cutout

Network	CIFAR10						CIFAR100					
	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet18	$94.89_{0.1}$	$93.64_{0.1}$	$94.05_{0.1}$	$94.04_{0.2}$	$94.52_{0.1}$	$\mathbf{95.02_{0.1}}$	$76.42_{0.1}$	$72.71_{0.2}$	$73.64_{0.2}$	$74.79_{0.2}$	$76.53_{0.1}$	$\mathbf{77.10_{0.2}}$
ResNet50	$95.07_{0.2}$	93.89_{02}	$94.26_{0.1}$	$94.25_{0.1}$	$94.92_{0.1}$	$\mathbf{95.42_{0.2}}$	$77.50_{0.2}$	$73.12_{0.7}$	$75.29_{0.3}$	$75.49_{0.2}$	$77.81_{0.2}$	$\mathbf{78.91_{0.9}}$
ResNet101	$94.77_{0.1}$	$93.14_{0.1}$	$94.73_{0.9}$	$94.23_{0.1}$	$94.22_{0.1}$	$\mathbf{95.51_{0.1}}$	$78.76_{0.2}$	$73.23_{0.4}$	$72.19_{0.2}$	$75.46_{0.3}$	$78.82_{0.1}$	$\mathbf{79.74_{0.3}}$
DenseNet121	$95.11_{0.1}$	$93.74_{0.2}$	$94.54_{0.1}$	$94.97_{0.1}$	$94.99_{0.1}$	$\mathbf{95.29_{0.1}}$	$78.61_{0.2}$	$75.38_{0.3}$	$72.54_{0.9}$	$77.09_{0.3}$	$78.70_{0.3}$	$\mathbf{79.03_{0.2}}$
MobileNetV3	$92.13_{0.2}$	$91.95_{0.1}$	$91.4_{3.1}$	$91.92_{0.1}$	$91.91_{0.2}$	$\mathbf{92.89_{0.1}}$	$73.81_{0.2}$	$65.64_{0.2}$	$60.78_{3.6}$	$69.87_{0.3}$	$68.01_{0.2}$	$\mathbf{73.15_{0.2}}$
Tiny Swin	$80.08_{0.2}$	$87.47_{0.2}$	$78.34_{0.2}$	$66.84_{0.3}$	$68.44_{0.2}$	$\mathbf{89.08}_{0.1}$	$57.43_{0.3}$	$62.20_{0.2}$	$54.12_{0.3}$	$36.12_{0.3}$	$33.75_{0.3}$	$\mathbf{66.47_{0.2}}$
FocalNet	$80.87_{0.2}$	$85.65_{0.1}$	$71.03_{0.3}$	$42.92_{0.2}$	$41.49_{0.2}$	$\mathbf{86.92}_{0.1}$	$45.66_{0.3}$	$52.88_{0.3}$	$38.05_{0.3}$	$11.23_{0.3}$	$11.06_{0.3}$	$\mathbf{52.9_{0.1}}$
CCT-2/3×2	$73.12_{0.2}$	$83.95_{0.1}$	–	$34.63_{1.1}$	$35.1_{0.8}$	$\mathbf{84.63_{0.3}}$	$52.12_{1.2}$	$60.14_{1.1}$	–	$8.06_{0.6}$	$9.76_{0.3}$	$\mathbf{60.63_{0.6}}$

*Note that Adam and AdaFisher were used for all CNN architectures, while AdamW and AdaFisherW were applied for all ViT experiments.

(b) With Cutout

Network	CIFAR10						CIFAR100					
	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher	SGD	Adam	AdaHessian	K-FAC	Shampoo	AdaFisher
ResNet18	$95.64_{0.1}$	$94.85_{0.1}$	$95.44_{0.1}$	$95.17_{0.2}$	$94.08_{0.2}$	$\mathbf{96.25_{0.2}}$	$76.56_{0.2}$	$75.74_{0.1}$	$71.79_{0.2}$	$76.03_{0.3}$	$76.78_{0.2}$	$\mathbf{77.28_{0.2}}$
ResNet50	$95.71_{0.1}$	$94.45_{0.2}$	$95.54_{0.1}$	$95.66_{0.1}$	$94.59_{0.1}$	$\mathbf{96.34_{0.2}}$	$78.01_{0.1}$	$74.65_{0.5}$	$75.81_{0.3}$	$77.40_{0.4}$	$78.07_{0.4}$	$\mathbf{79.77_{0.4}}$
ResNet101	$95.98_{0.2}$	$94.57_{0.1}$	$95.29_{0.6}$	$96.01_{0.1}$	$94.63_{0.1}$	$\mathbf{96.39_{0.1}}$	$78.89_{0.2}$	$75.56_{0.3}$	$73.38_{0.2}$	$77.01_{0.4}$	$78.83_{0.2}$	$\mathbf{80.65_{0.4}}$
DenseNet121	$96.09_{0.1}$	$94.86_{0.1}$	$96.11_{0.1}$	$96.12_{0.1}$	$95.66_{0.1}$	$\mathbf{96.72_{0.1}}$	$80.13_{0.4}$	$75.87_{0.4}$	$74.80_{0.9}$	$79.79_{0.2}$	$80.24_{0.3}$	$\mathbf{81.36_{0.3}}$
MobileNetV3	$94.43_{0.2}$	$93.32_{0.1}$	$92.86_{3.1}$	$94.34_{0.1}$	$93.81_{0.2}$	$\mathbf{95.28_{0.1}}$	$73.89_{0.3}$	$70.62_{0.3}$	$56.58_{4.5}$	$73.75_{0.3}$	$70.85_{0.3}$	$\mathbf{77.56_{0.1}}$
Tiny Swin	$82.34_{0.2}$	$87.37_{0.6}$	$84.15_{0.2}$	$64.79_{0.5}$	$63.91_{0.4}$	$\mathbf{88.74_{0.4}}$	$54.89_{0.4}$	$60.21_{0.4}$	$56.86_{0.5}$	$34.45_{0.4}$	$30.39_{1.2}$	$\mathbf{66.05_{0.5}}$
FocalNet	$82.03_{0.2}$	$86.23_{0.1}$	$64.18_{0.2}$	$38.94_{0.8}$	$37.96_{0.7}$	$\mathbf{87.90}_{0.1}$	47.76_{03}	$52.71_{0.5}$	$32.33_{0.3}$	$9.98_{0.6}$	$9.18_{0.1}$	$\mathbf{53.69_{0.3}}$
CCT-2/3×2	$78.76_{0.3}$	$83.89_{0.4}$	–	$33.08_{2.3}$	$35.16_{0.4}$	$\mathbf{84.94}_{0.3}$	$54.05_{0.4}$	$59.78_{0.5}$	–	$7.17_{0.2}$	$8.60_{0.1}$	$\mathbf{62.91_{0.5}}$

*Note that Adam and AdaFisher were used for all CNN architectures, while AdamW and AdaFisherW were applied for all ViT experiments.

D.2.4 RESULTS

Table 12 displays the results for the Tiny ImageNet dataset using ResNet50 and Big Swin networks, with visualizations provided in Figure 14. AdaFisher and AdaFisherW consistently outperform current SOTA optimizers. Notably, Figure 14 illustrates that although AdaFisher converges slower than K-FAC during ResNet50 training, it achieves superior generalization. This is evidenced by lower testing errors, suggesting that AdaFisher tends to converge to a flatter local minimum, enabling smoother transitions between training and testing datasets with minimal generalization loss. For further explanation, please see Cha et al. (2021). Please note that due to AdaHessian's high memory consumption, we were unable to train it on Big Swin. Table 13 presents the performance of various networks on CIFAR10/100 datasets using different optimizers, both with and without the cutout augmentation technique. AdaFisher and AdaFisherW consistently outperform their counterparts in both scenarios, demonstrating stable training and robustness to the augmentation techniques. The training losses and test errors for the CIFAR experiments, both with and without cutout, are visually represented in Figures 16, 17, 18, and 19. Figure 15 illustrates the training and validation error of the distributed version of AdaFisher on ImageNet across various batch sizes. AdaFisher not only outperforms its counterparts with smaller batch sizes (256), but it also continues to achieve superior generalization as batch sizes increase. Furthermore, these results reinforce the stability analysis concerning batch sizes presented in Section 4.4, extending it to a more challenging dataset.

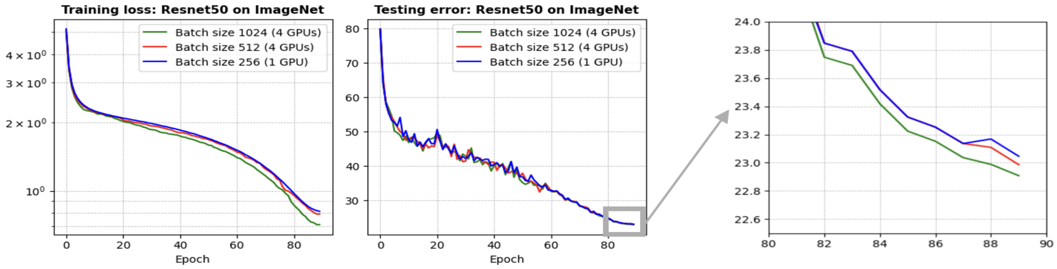

Figure 15: Performance of distributed AdaFisher using ResNet50 on ImageNet with different batch sizes for 90 epochs. The final accuracy results are reported in Table 3.

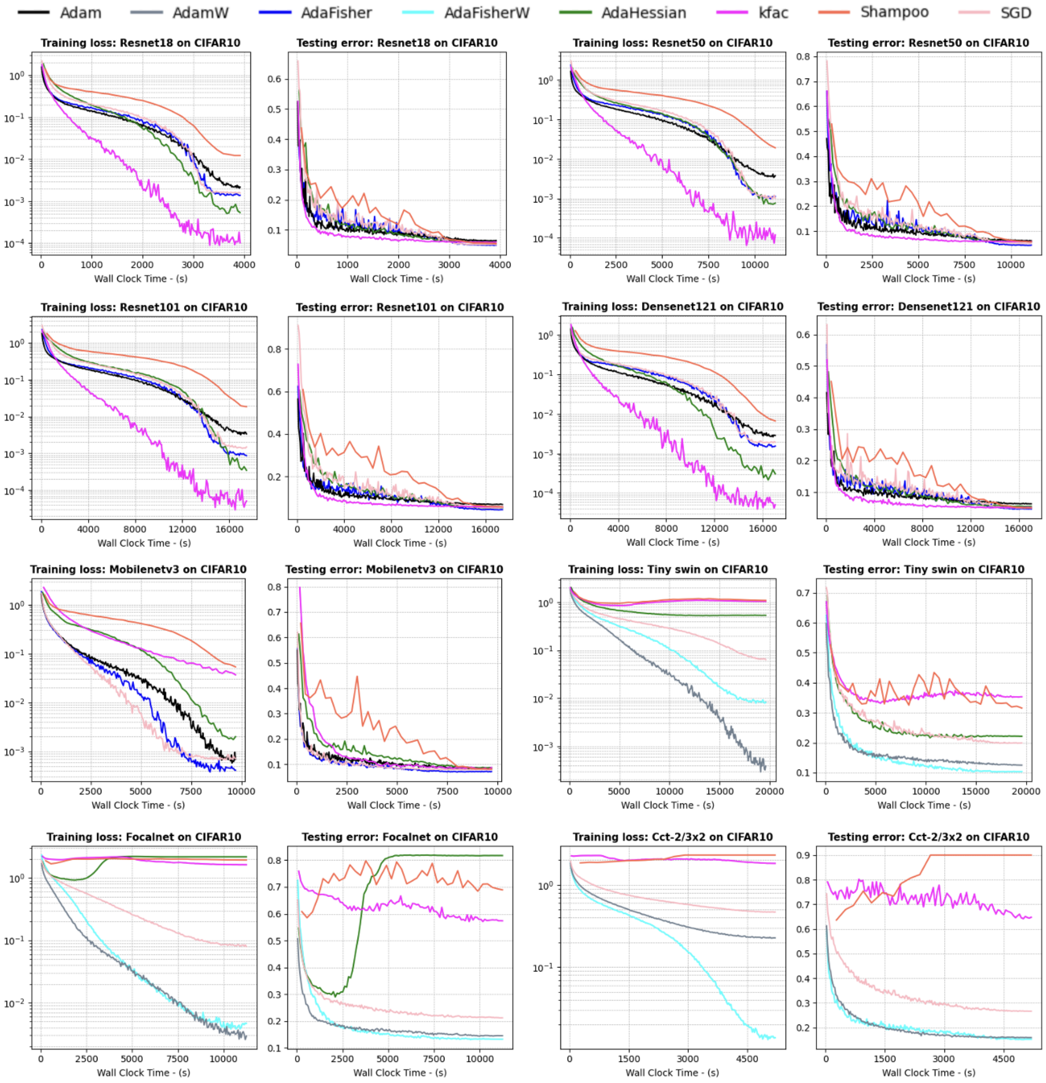

Figure 16: WCT training loss, test error, for CNNs and ViTs on CIFAR10 experiments, without Cutout. A batch size of 256 was used and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 13 (a).

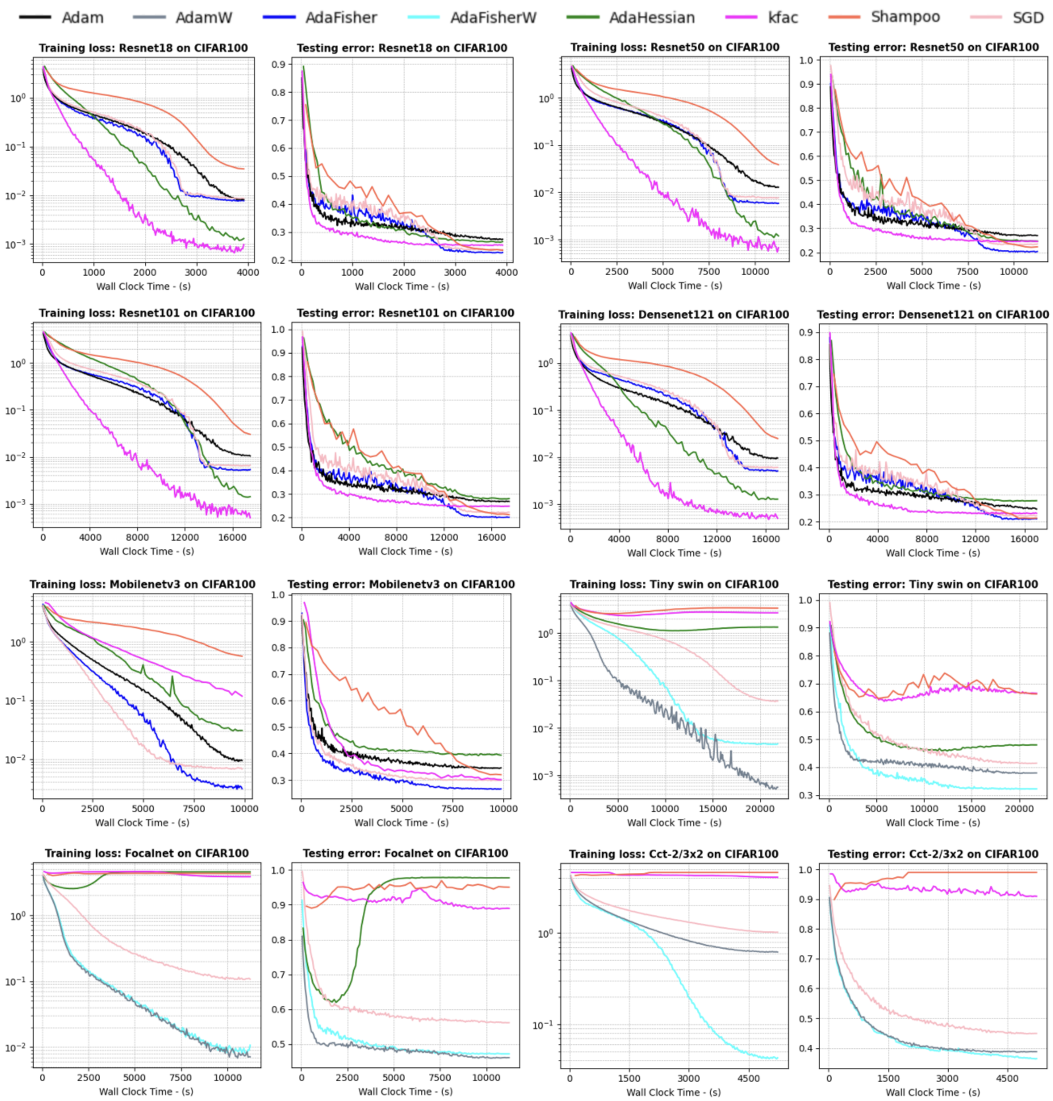

Figure 17: WCT training loss, test error, for CNNs and ViTs on CIFAR100 experiments, without Cutout. A batch size of 256 was used and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 13 (a).

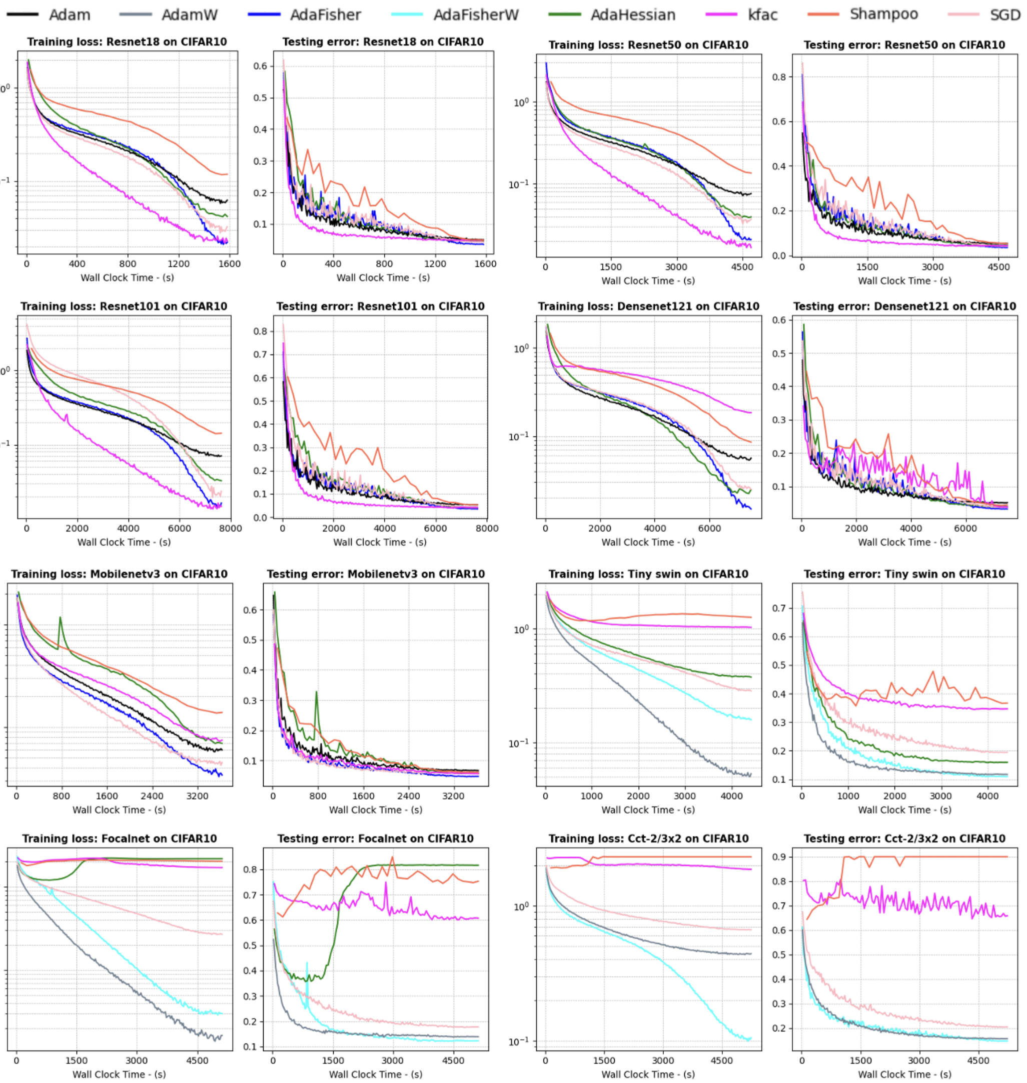

Figure 18: WCT training loss, test error, for CNNs and ViTs on CIFAR10 experiments, with Cutout. A batch size of 256 was used and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 13 (b).

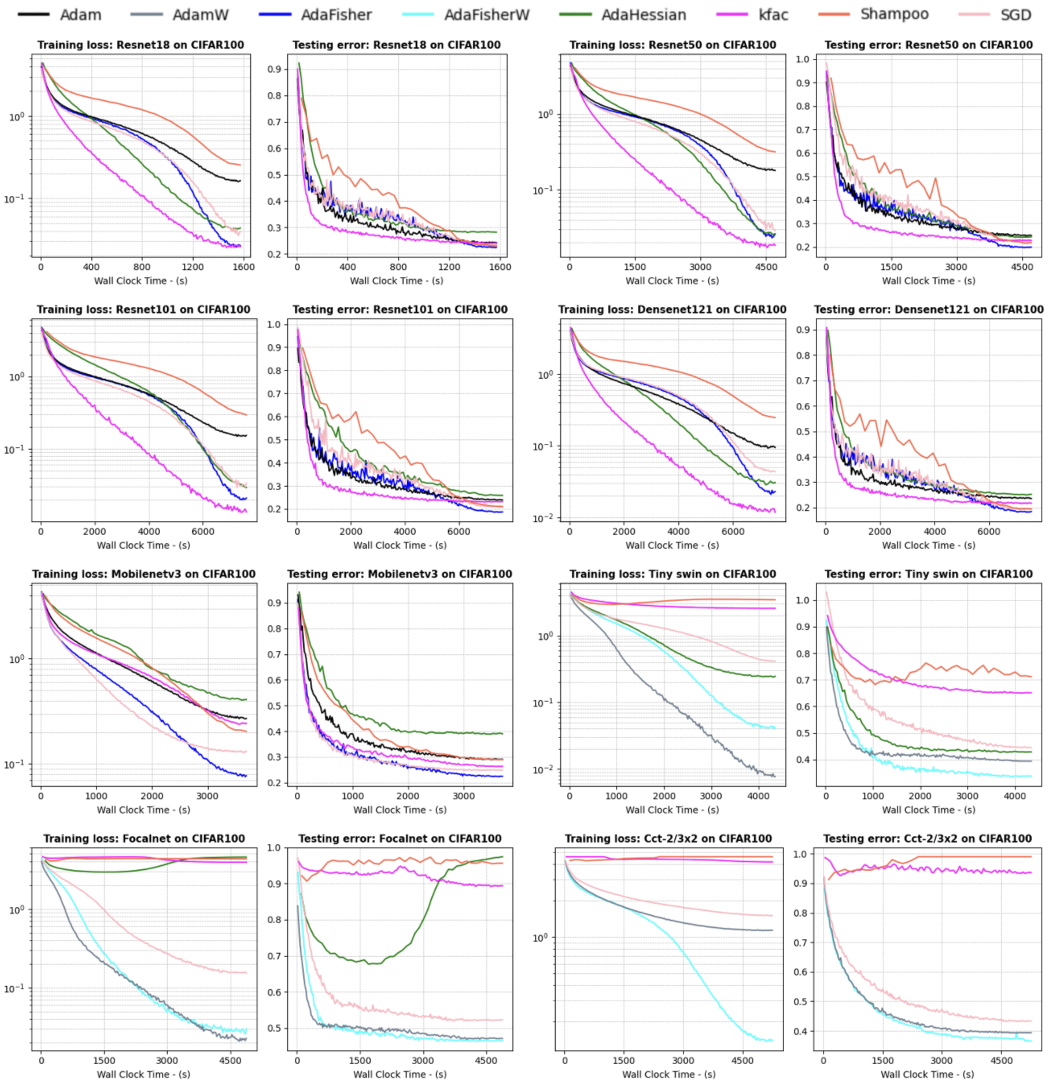

Figure 19: WCT training loss, test error, for CNNs and ViTs on CIFAR100 experiments, with Cutout. A batch size of 256 was used and all networks were tuned using ResNet18 applied on CIFAR10. The final accuracy results are reported in Table 13 (b).

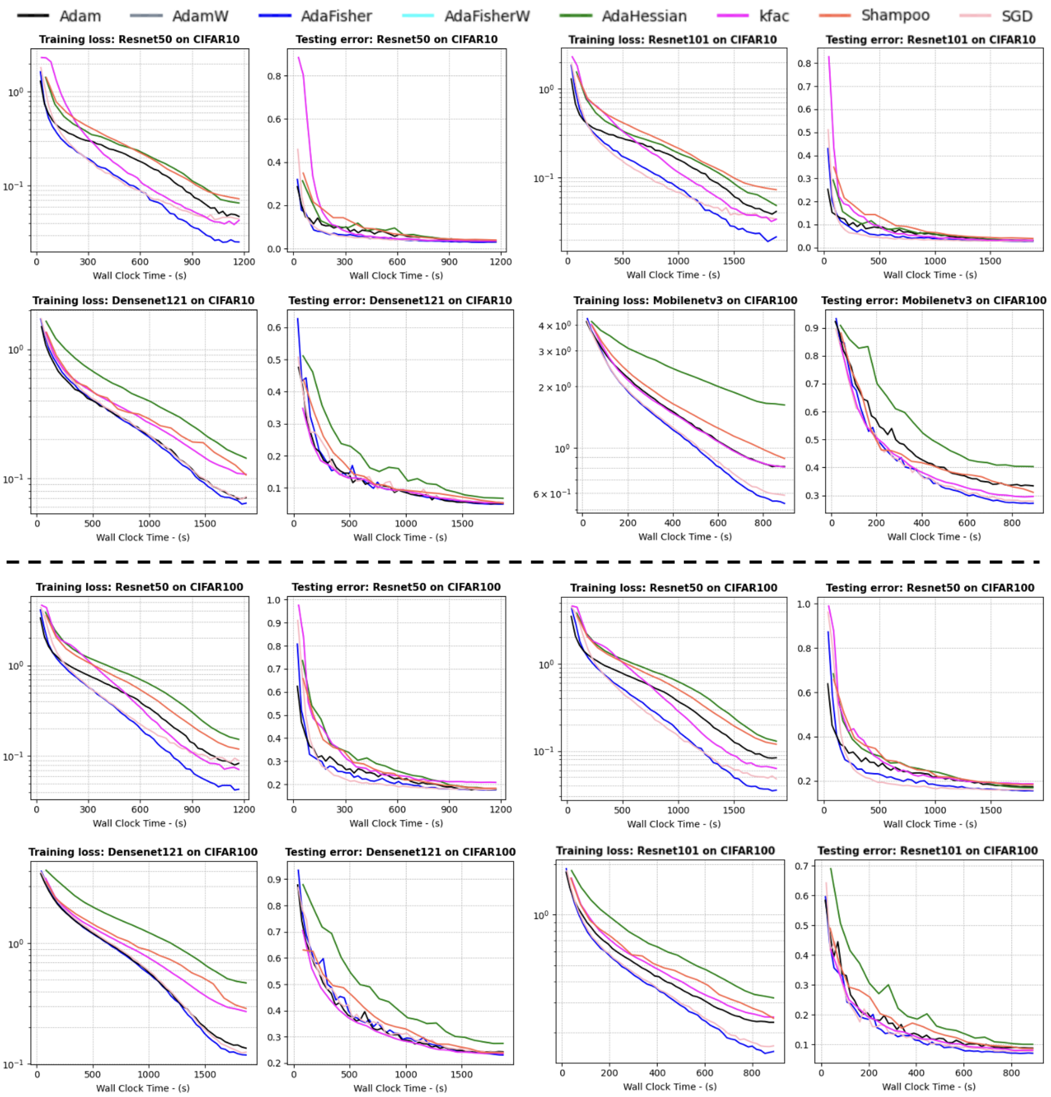

Figure 20: WCT training loss, test error, for CNNs on CIFAR10/100 experiments. A batch size of 256 was used and all networks were tuned using ResNet50 applied on CIFAR10. The final accuracy results are reported in Table 4.

D.2.5 COMPARISON WITH OTHER RELEVANT METHODS

In this work, we compare AdaFisher with six baseline optimizers for image classification: SGD, Adam/AdamW, AdaHessian, KFAC, and Shampoo. These baselines were selected because they either represent the current state of the art or utilize second-order gradients, making them suitable comparisons for evaluating second-order optimizers. However, other optimizers, such as AdaFactor Shazeer & Stern (2018) and EVA Zhang et al. (2023), are also relevant in this context. AdaFactor is an enhance Adam memory-efficient optimizer that approximates second-order moments using row and column factorizations, reducing memory consumption for large-scale models. EVA is a second-order optimizer designed to leverage the FIM with efficient matrix inversion techniques. Therefore, we compare experimentally AdaFisher against the optimizer baselines including Eva and AdaFactor. Regarding the hyperparameters for EVA, we used the optimal values reported in its original paper and trained the model for 119 epochs using the WCT technique. For AdaFactor, we fine-tuned the learning rate as described in Appendix D.2.1, identifying 0.001 as the optimal value, and trained the model for 216 epochs. Figure 21 illustrates the performance comparison on two distinct models: ResNet-18 with CIFAR-100 and MobileNetV3 with CIFAR10. The same data augmentation techniques were applied across all experiments, as detailed in Appendix D.2.2. The best test accuracies achieved are summarized in Table 14. AdaFisher demonstrates superior performance compared to the new optimizer baselines, outperforming both EVA and AdaFactor.

Table 14: Performance comparison of AdaFisher and other optimizers using ResNet-18 (CIFAR100) and MobileNet-V3 (CIFAR10). Reported using WCT of 200 AdaFisher training epochs as the cutoff.

Network—Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
MobileNet-V3	94.43	93.32	93.21	92.86	94.34	94.41	93.81	**95.28**
ResNet-18	76.56	75.74	69.45	71.79	76.03	76.69	76.78	**77.28**

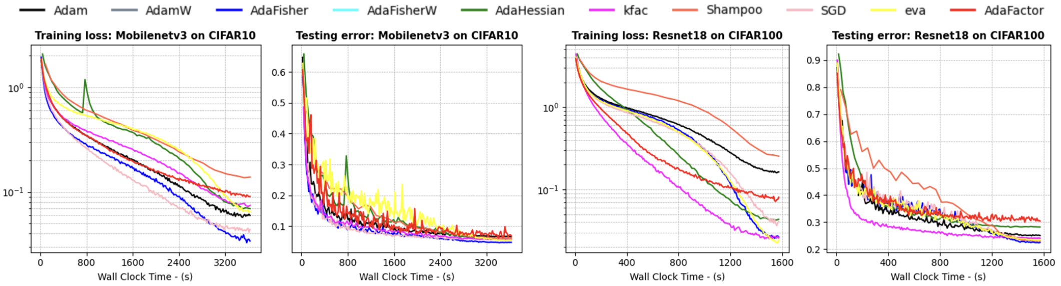

Figure 21: WCT training loss, test error, for ResNet-18 on CIFAR100 and MobileNet-V3 on CIFAR10. A batch size of 256 was used. The final accuracy and training time results are summarized in Table 14.

Table 15: Performance comparison of AdaFisher and other optimizers using (a) ResNet-18 and (b) MobileNet-V3 on CIFAR100 for 200 epochs.

(a) ResNet-18

Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
Test Acc	76.52	75.71	69.78	76.86	76.96	77.08	**77.35**	77.28
Training Time (min)	20.03	23.33	21.67	96.67	46.46	43.18	216.67	26.58

(b) MobileNet-V3

Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
Test Acc	73.42	70.53	71.08	62.36	75.16	75.48	70.65	**77.56**
Training Time (min)	50.03	56.63	54.22	206.28	116.86	96.78	487.21	60.12

(c) ResNet-50

Optimizer	SGD	Adam	AdaFactor	AdaHessian	K-FAC	Eva	Shampoo	AdaFisher
Test Acc	76.12	73.03	70.78	76.18	77.66	78.01	78.89	**78.91**
Training Time (min)	70.13	76.67	73.32	502.28	149.36	138.58	583.11	83.02

D.2.6 COMPARISON WITH CONSISTENT EPOCH COUNTS

We evaluated AdaFisher and its counterparts, including two prominent optimizers, Eva and Adafactor, over 200 epochs on ResNet-18, ResNet-50 and MobileNet-V3 using the CIFAR100 dataset.

Figure 22 illustrates the training loss and test error trends over epochs, along with the best test error achieved as a function of training time per epoch for all optimizers across both models. Table 15 summarizes the highest test accuracy and total training time for each method on both network architectures. Notably, while Shampoo achieved marginally better test accuracy than AdaFisher on ResNet-18, it required approximately eight times longer training time. Conversely, AdaFisher outperformed all baseline optimizers, including Shampoo, in the MobileNet-V3 and ResNet-50 experiments, achieving superior test accuracy while maintaining high efficiency comparable to first order optimizers.

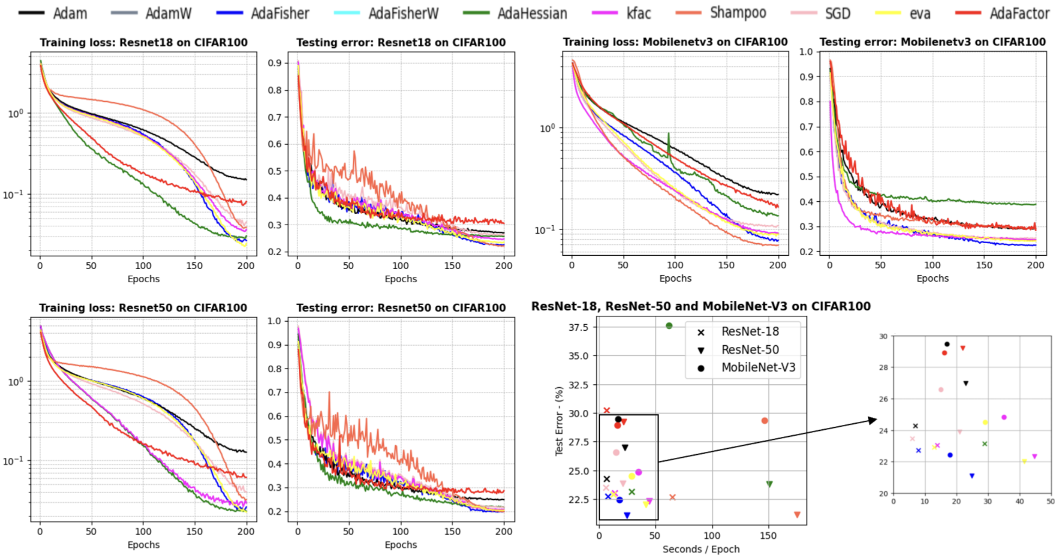

Figure 22: Performance comparison of AdaFisher and other well-finetuned optimizers at their best performances using ResNet-18 and MobileNet-V3 on CIFAR-100 for 200 epochs. A batch size of 256 was used. The final accuracy and training time results are summarized in Table 15.

D.2.7 COMPARISON OF TRAINING SPEED AND MEMORY UTILIZATION

As discussed in Section 4.4, AdaFisher emerges as a balanced trade-off between time complexity and performance. Similarly, its memory footprint is comparable to that of Adam, showcasing efficient VRAM utilization. We extend our stability analysis to the CIFAR10 dataset to provide a dataset-independent evaluation of performance metrics, as depicted in Figure 23. Additionally, we analyze the memory usage for different batch sizes using the ResNet-50 model on the CIFAR-10/100, presented in Figure 24. The analysis reveals that AdaFisher, while maintaining high accuracy levels, uses memory comparably to Adam, especially evident in higher batch sizes. This suggests that AdaFisher can achieve competitive performance without excessive VRAM consumption, making it an optimal choice for scenarios with memory constraints.

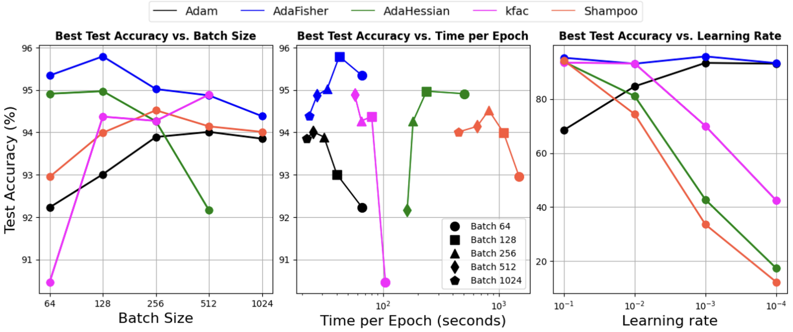

Figure 23: Performance comparison of AdaFisher and other optimizers across various batch sizes, epoch times and learning rate (with batch size of 256), evaluated using the ResNet50 on the CIFAR-10.

Epoch Times. Continuing our analysis of the time complexity for each optimizer, we present in Figure 25 the epoch times for various network architectures and datasets. Specifically, we compare

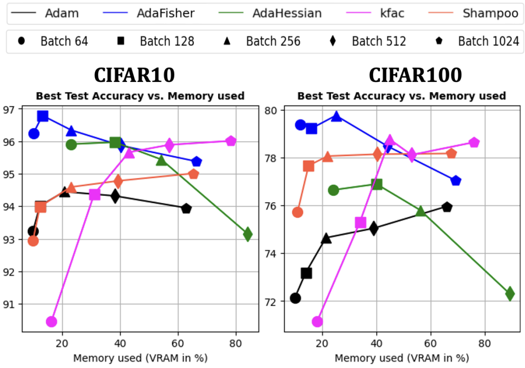

Figure 24: Performance comparison of AdaFisher and other optimizer regarding the memory used, assessed using ResNet50 and CIFAR10/100 across different batch sizes. This figure highlights how AdaFisher competes closely with Adam in terms of memory efficiency and performance.

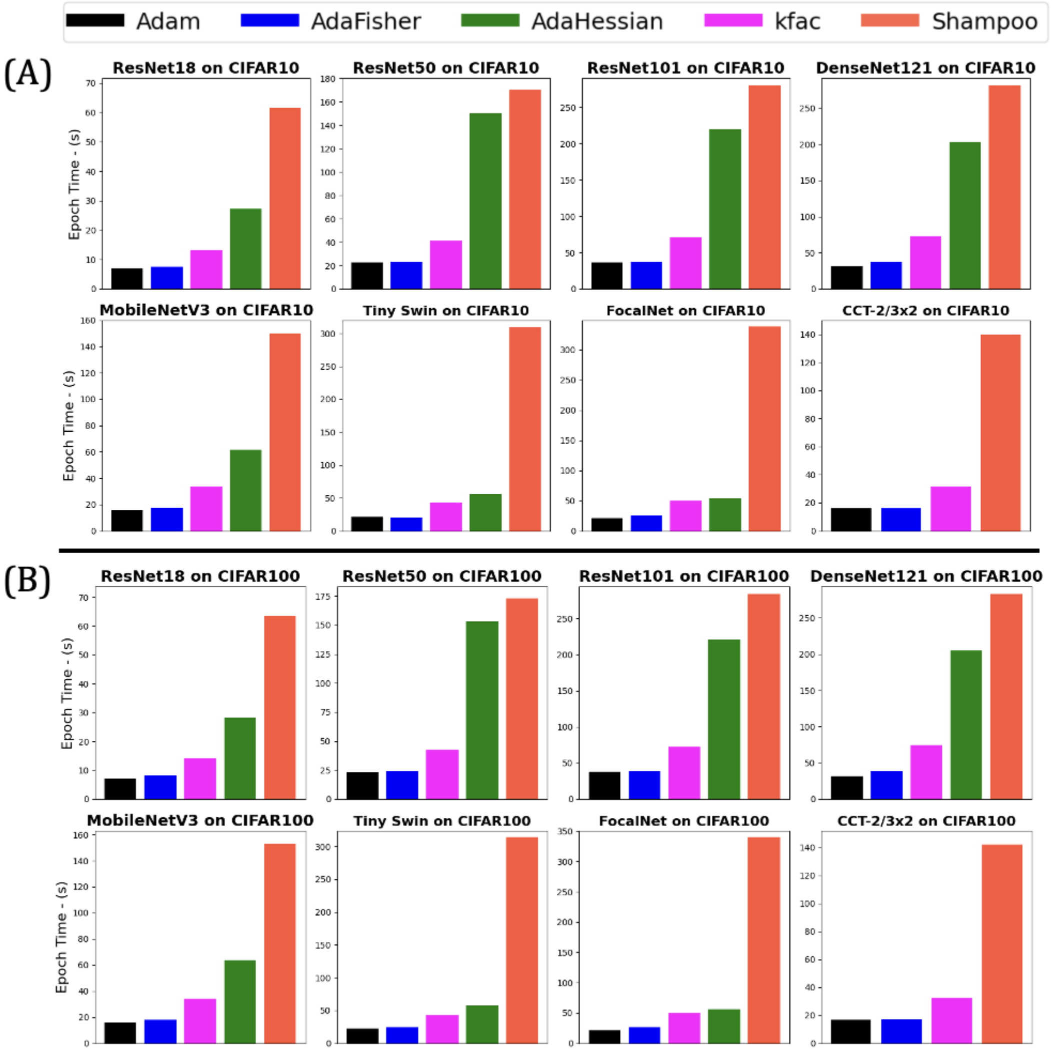

Figure 25: Epoch times for various networks on CIFAR10 (A) and CIFAR100 (B) using Adam, AdaFisher, K-FAC, AdaHessian and Shampoo.

the epoch times of Adam, AdaFisher, K-FAC, AdaHessian, and Shampoo optimizers on CIFAR10 and CIFAR100 datasets. As depicted in Figure 25 panel (A), AdaFisher demonstrates a comparable training time to Adam across multiple network architectures on the CIFAR10 dataset. This indicates that AdaFisher achieves efficient optimization without incurring significant additional computational cost. Similarly, in Figure 25 panel (B), we observe that the epoch times for AdaFisher remain close to those of Adam on the CIFAR100 dataset. While K-FAC and AdaHessian exhibit increased training times, Shampoo shows the highest epoch times across all tested networks. This further highlights the efficiency of AdaFisher as an optimizer, combining the advantages of advanced optimization techniques with practical training times.

D.3 LANGUAGE MODELLING

D.3.1 DATASET DETAILS

The Wikitext-2 dataset, derived from high-quality Wikipedia articles, contains over two million words and is structured into training, validation, and test sets. It is widely used for benchmarking language models in natural language processing, especially assessing perplexity to evaluate predictive performance. This dataset offers a balance between computational efficiency and linguistic complexity, making it ideal for practical language model training and evaluation.

D.3.2 NETWORK DETAILS

Network. We utilize a streamlined GPT-1 architecture which incorporates four self-attention layers, a reduction from the original twelve. This configuration retains core modeling capabilities while reducing complexity, encompassing a total of 28,351,488 learnable parameters.
Embeddings & Parameter Sharing. To expedite training, we employ pretrained embeddings from OpenAI's GPT, leveraging the benefits of parameter sharing for enhanced efficiency and faster convergence.

D.3.3 HYPERPARAMETERS

The model underwent training for 50 WCT epochs using AdaFisher on the WikiText-2 and PTB datasets, with the final epoch counts for each optimizer detailed in Table 16. For AdamW, we follow the learning rate setting in ElNokrashy et al. (2022). For the other optimizers we select the learning rate by doing a grid search of $\{0.3, 0.15, 0.1, 0.05, 0.03, 0.015, 0.01, \dots, 1e^{-5}\}$. We tabulate the learning rate the we use in Table 17. The batch size was configured to 32, and the weight decay was established at 0.1. Despite optimizing the configuration of hyperparameters, Shampoo failed to converge, and K-FAC could not be trained at all.

Table 16: Final selected epoch counts for various optimizers across language modelling task

AdamW	AdaHessian	Shampoo	AdaFisherW
55	18	12	50

Table 17: Final selected learning rates for each optimizer, tuned using GPT1 on WikiText-2 and PTB using a batch size of 32. We selected based on final validation PPL.

AdamW	AdaHessian	Shampoo	AdaFisherW
$5e^{-5}$	0.015	0.003	$1e{-}4$

D.3.4 RESULTS

Figure 26 displays the training loss and testing error curves, clearly showing that AdaFisher surpasses both Adam and AdaHessian in performance on the WikiText-2 and PTB datasets.

E IMPACT STATEMENT

AdaFisher represents a significant advancement in training efficiency, achieving superior accuracy on the ImageNet dataset using only a single GPU. This optimization is particularly beneficial for academia and students who may not have access to extensive computational resources. By enabling effective training with fewer GPUs, AdaFisher offers an accessible yet powerful solution, reducing hardware costs and making advanced machine learning more attainable for those with limited

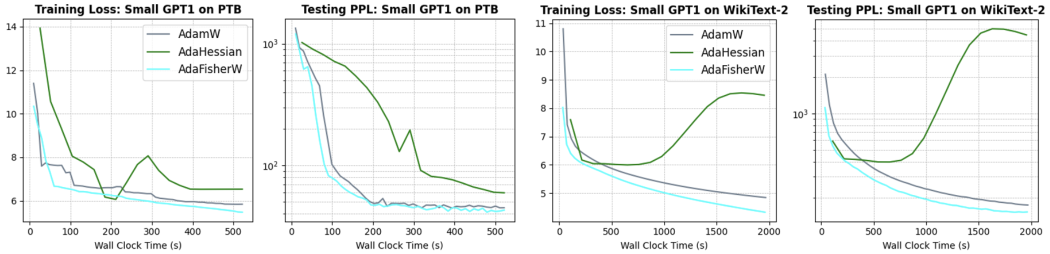

Figure 26: Training Loss and Test Perplexity of Small GPT-1 Model on WikiText-2 and PTB Datasets. Experiments were conducted using a batch size of 32 and optimal settings for all optimizers.

resources. This capability underscores AdaFisher's potential as a valuable tool in democratizing machine learning technology.