

Towards Enhancing RMSProp with Forward-Looking Gradient Updates for Complex Loss Landscapes

ANONYMIZED for ICLR 2026

Keywords: deep learning, optimization, gradient descent, nonlinear activations, integrated gradient

Abstract

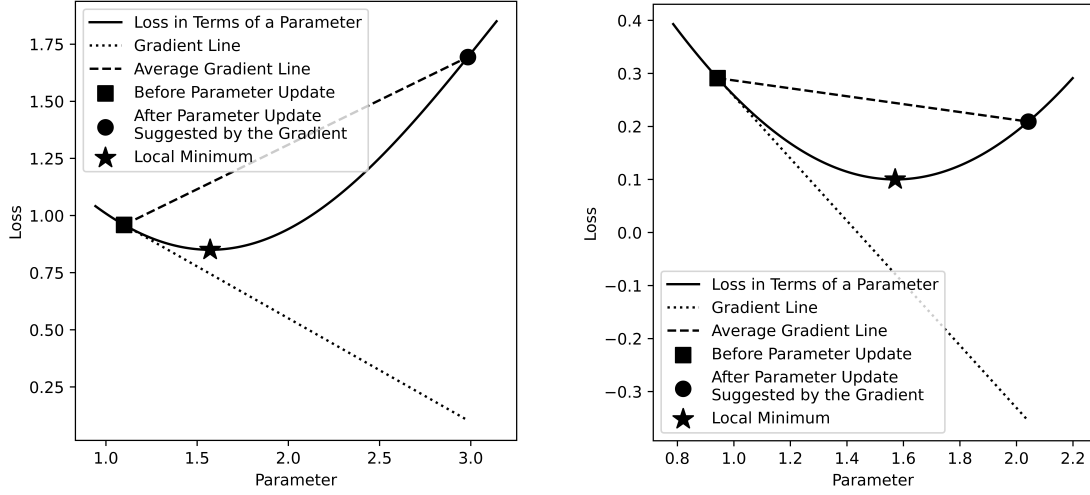
This paper introduces a novel algorithm for training deep neural networks with many non-linear layers. Our method utilizes an approximated integrated gradient that is averaged over the range of the weight update to more accurately capture the loss change resulting from parameter updates. Unlike standard gradients, this average gradient improves learning efficiency in certain scenarios. We incorporate the approximated average gradient into RMSProp and compare the resulting algorithm to conventional RMSProp and Adam. We evaluate the approach on deep models lacking skip connections, such as those with many nonlinear activations and no residual structure, where traditional methods typically encounter difficulties. These models that focus on extracting high-order features create a

loss landscape more akin to that of a biological brain. Our method significantly improves sample efficiency on MNIST, Fashion MNIST, and IMDB benchmarks for both convolutional and fully connected architectures, across various initialization schemes. While our approach incurs moderately higher computational and memory costs compared to standard RMSProp, its performance on shallow models remains comparable. Nevertheless, our main contributions are: (a) The introduction of the average gradient concept as an efficient alternative to computing high-order derivatives. (b) A fast formula for its estimation, accompanied by a formal proof. (c) An example algorithm that leverages this formula to enhance the efficiency of RMSProp for some models, as validated by our evaluation.

1 Introduction

In this research, we focus on solving deep learning problems by calculating the precise influence of potential updates on the loss for each model parameter separately. Our goal is to obtain more precise information about the influence of each parameter on loss than what the gradient provides. Each potential update of a model parameter influences the locally-optimal direction of other model parameters during the same weight update, highlighting the complexity of the problem. The average gradient (defined in Appendix A), unlike the gradient, stores the accurate contribution of each model parameter to the loss delta related to a given weight update (see Fig. 1). Therefore, the average gradient can be utilized to efficiently minimize the loss. In this research, we propose a very fast algorithm to approximate the average gradient. We prove its

approximation accuracy, validate the proof using our handcrafted metric to compare batch–loss minimization efficiency between methods, and test our method on various domains, models, and weight initialization techniques. Our algorithm in its current form primarily targets very deep models with numerous nonlinear layers. A brief summary of our approach is presented in Fig. 2.



(a) *Example 1.* The average gradient suggests a different direction for updating a particular parameter.

(b) *Example 2.* If the average gradient decreases in the same direction as the gradient, it additionally provides more information about the loss landscape.

Figure 1: *Comparison of Gradient and Average Gradient.* The average gradient accurately reflects the influence of a parameter update on loss because it precisely predicts the loss trend between the two points. The plots illustrate a simple case involving only one model parameter; however, they can also be interpreted as representing the loss contribution of a parameter during a weight update in a model with multiple parameters.

Due to the tendency to increase model depth along with its width (M. Tan & Le, 2019)

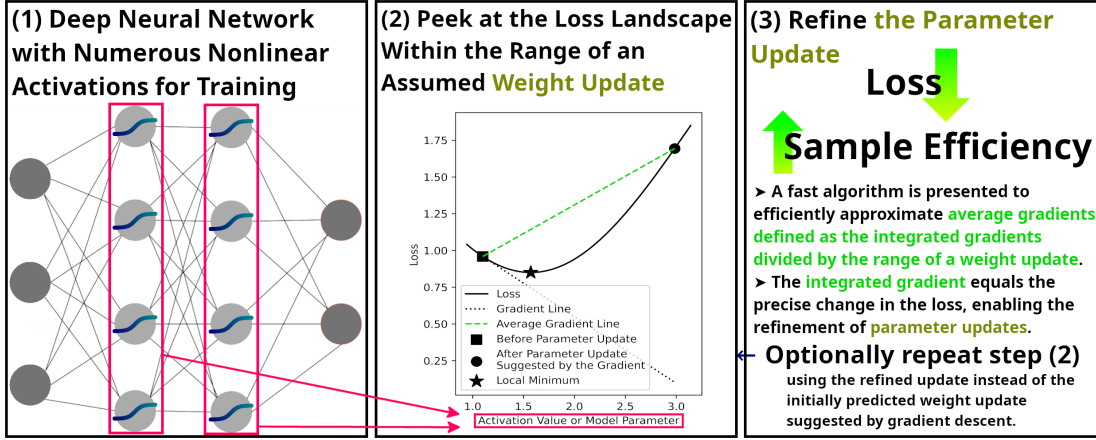


Figure 2: *Simplified Overview of the Proposed Approach.*

and the popularity of certain nonlinear activation functions, our approach may offer insights for future improvements in practical deep learning. Our primary objective is to significantly improve sample efficiency, even at the expense of increased computational time, which is essential for practical applications in fields such as deep reinforcement learning and reinforcement learning from human feedback (Kirk et al., 2023). These methods have been used in popular chatbots, such as OpenAI’s ChatGPT (OpenAI et al., 2023) and Anthropic’s Claude (Kirk et al., 2023).

However, we were not able to test our algorithm on large models because doing so would require a complete reimplementation of the backend logic of current deep learning frameworks, which is not feasible within a reasonable timeframe. Instead, we focused on evaluating our algorithm using various weight initialization techniques, datasets, and a selection of model architectures, particularly those known to be challenging.

The remainder of the paper is organized as follows. We begin with a review of related work (Sec. 2), followed by a description of our method (Sec. 3.1) and an overview of the experimental setup (Sec. 3.2). Our results are presented in Sec. 4 and analyzed in

a dedicated discussion (Sec. 5). We summarize our findings in Sec. 6, where we also discuss neurobiological implications, address the limitations of our study, and outline directions for future work. Extended definitions, proofs, and additional results are provided in the appendices. Our code is publicly available at [ANONYMIZED for ICLR 2026].

2 Related Work

2.1 Optimization of Deep Neural Networks

Gradient optimization dominates deep learning with optimizers like Stochastic Gradient Descent (Liu, Gao, & Yin, 2020), RMSProp (Tieleman & Hinton, 2012), or Adam (Kingma & Ba, 2014). The leading algorithms for training do not change frequently over the years. However, our algorithm and its variants can be used in conjunction with first-order optimizers, since the gradient can be replaced by a locally averaged gradient computed over a range of parameter values.

Gradient averaging is commonly used in machine learning, but in a distinct scenario than in our approach. Momentum is the running average of gradients over subsequent batches (Dozat, 2016; Kingma & Ba, 2014; Liu et al., 2020). It prevents falling into local minimums and may accelerate learning. Similarly, averaging model parameters over many updates may stabilize learning (Li, Yang, Liang, Zhang, & Jordan, 2023; Merity, Keskar, & Socher, 2017; B. T. Polyak & Juditsky, 1992; Ruppert, 1988; Sun, Kashima, Matsuzaki, & Ueda, 2010); however, it requires an additional learning technique to compute the parameter updates. Crucially, the aforementioned techniques are independent of

our approach because they aggregate gradients or parameter values from multiple past learning iterations rather than focusing solely on the current iteration.

Accumulating gradients over a batch is inherent in machine learning. In practice, it is equivalent to averaging gradients computed for multiple inputs. However, this approach alone does not take into account the information about a parameter update (Fig. 1), which remains unknown during its computation. Consequently, it does not ensure accurate computation of the influence of an unknown parameter update on the loss, particularly for large update steps. Nevertheless, batching remains fully compatible with our method and is employed in our implementation.

Our approach is more closely related to some second-order optimization methods (H. H. Tan & Lim, 2019) rather than momentum-based or parameter-averaging techniques. This is due to the utilization of information about the curvature of a loss function during each parameter update (Fig. 1). Current methods in deep neural networks focus on approximating second-order derivatives while keeping computational demands acceptable, as exemplified by algorithms such as KFAC (Grosse & Martens, 2016; Martens & Grosse, 2015) and Shampoo (Gupta, Koren, & Singer, 2018). Building on this direction, the development of the Soap optimizer (Vyas et al., 2024) has even surpassed Adam (Kingma & Ba, 2014) in both computational efficiency and sample efficiency for certain large language models. In contrast, our goal is to introduce a fundamentally different fast curvature estimator.

Alternatively, the curvature of the loss function can be *roughly* estimated using gradients from the two most recent iterations, which may slightly improve learning (Dubey et al., 2019; Roy et al., 2021; Zheng et al., 2024). Moreover, generalization can

be improved by leveraging predicted future gradients. In this approach, weight values are first predicted following multiple successive parameter updates, and ultimately, the final update is performed using only the gradient obtained from a single backward pass on these predicted weights (Guan, Li, Shi, & Meng, 2024).

One method is mathematically analogous to performing gradient averaging over a range that approximately corresponds to potential weight updates. In this approach, the arithmetic mean of two gradients is computed: one evaluated at the current weights and another at weights perturbed in a direction defined by the Hadamard product of the model parameters’ absolute values and the gradient (Tseng, Liu, Lee, & Zeng, 2022). This strategy resulted in improved generalization. However, in designing our algorithm, we opted for additional enhancements, including significant gains in sample efficiency, a fast and accurate approximation of the average gradient, and resolutions to critical issues limiting very deep architectures—namely, singularity problems (O. Oyedotun, Al Ismaeil, & Aouada, 2021; Yasrab, 2019) and the shattered gradient problem (Balduzzi et al., 2017) (as discussed in Section 2.3). Moreover, we chose to approximate the gradient integral over the range spanning the pre-update and post-update states, in order to accurately capture the influence of a parameter update.

2.2 Explainability Techniques

The integrated gradient, mathematically closely related to the average gradient, is employed in several neural network explainability techniques (Khorram, Lawson, & Fuxin, 2021; Sattarzadeh et al., 2021; Sundararajan, Taly, & Yan, 2017). However, the approximation algorithms for the integral of the gradient used in the literature are very inefficient

to compute for every parameter update of a model due to the calculation of the Riemann sum (Hughes-Hallett, Gleason, Lock, & Flath, 2021), in contrast to our approach. Moreover, to the best of our knowledge, these techniques have not been used to train neural networks. Importantly, the question of "how much each variable like feature or model parameter influences output?", is common for training and explainability techniques.

2.3 Optimization Issues in Deep Learning

The problem of gradient inefficiency in very deep models lacking residual connections is well documented in the literature (Balduzzi et al., 2017; O. Oyedotun et al., 2021; O. K. Oyedotun, Al Ismaeil, & Aouada, 2022; Yasrab, 2019). Exploding or vanishing gradients (Bengio, Simard, & Frasconi, 1994) are not a limitation in this case (Balduzzi et al., 2017), as they can be effectively addressed using batch normalization (Ioffe & Szegedy, 2015), proper weight initialization (Glorot & Bengio, 2010; He, Zhang, Ren, & Sun, 2015), or dynamic control of gradient magnitudes (Basodi, Ji, Zhang, & Pan, 2020; Zaeemzadeh, Rahnavard, & Shah, 2020a). Simplifying a model by suppressing interactions of spatially independent features (Balduzzi et al., 2017) reduces the variance of the loss landscape, allowing for efficient training even with numerous layers. However, in very deep neural networks without skip connections, the problem of information loss during backpropagation persists (O. Oyedotun et al., 2021; O. K. Oyedotun et al., 2022; Yasrab, 2019), a phenomenon commonly referred to as singularity problems. The development of a training algorithm that efficiently navigates complex loss landscapes offers a universal and elegant solution to the challenges of training deep neural architectures—challenges that largely stem from the limitations of commonly used gradient descent

optimizers. In this study, we identify one particularly challenging architecture (denoted as Model B in Tab. 1) and use it to evaluate our algorithm. We aim to address the aforementioned optimization issues to unlock enhanced performance across a broader range of models, particularly very deep ones, and to pave the way for optimization tools tailored to deep models with human-like capabilities.

3 Methods

3.1 Algorithm

All of the best and most popular optimizers for training large neural networks rely on the gradient. Consequently, they explicitly ignore how loss function in terms of model parameters behaves in the range between before and after a potential weight update (Fig. 1). The definition of the gradient implies, that it reflects the accurate influence on loss only for learning rates approaching to zero, which does not hold in practice. Consequently, gradient-based optimizers do not calculate the accurate influence on loss of a potential weight update, which may significantly slow down the learning of very deep models with many nonlinear operators, as our experiments show. The average gradient solves the described problem. Mathematically, a key property of the average gradient is its direct proportionality to the loss delta (Fig. 1).

In our algorithm, given a plain feedforward neural network, the average gradient is approximated and propagated according to the equation proven in Appendix B:

$$\begin{aligned} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\vartheta_k} \ell(\vartheta) &\approx \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k(\vartheta)}{\partial \vartheta_k} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_{k+1}(\vartheta)}{\partial \mathbf{x}_k(\vartheta)}. \\ \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_{k+2}(\vartheta)}{\partial \mathbf{x}_{k+1}(\vartheta)} &\cdot \dots \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_n(\vartheta)}{\partial \mathbf{x}_{n-1}(\vartheta)} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\mathbf{x}_n(\vartheta)} \ell(\vartheta) \end{aligned} \quad (1)$$

where ℓ is a loss function, ϑ_k are parameters of a layer no. k and $(\mathbf{x}_k, \mathbf{x}_{k+1}, \dots, \mathbf{x}_n)$ are inputs and outputs of subsequent layers of a neural network. The notation $\nabla_{\mathbf{x}} f$ refers to the gradient of some function f for an argument \mathbf{x} . $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ denotes the Jacobian matrix of the vector-valued function \mathbf{f} , where the i^{th} column corresponds to the gradient vector $\nabla_{\mathbf{x}} f_i$. The symbol \cdot denotes standard matrix multiplication. The average operator \mathcal{AVG} of gradients or Jacobians is defined in Appendix A and aligns with intuition. The averages are aggregated with respect to the model parameters in the range from θ to θ' . The average gradients are propagated in the same manner as the gradients in the standard backpropagation algorithm. The computation based on Equation 1 is fast and memory efficient because the procedure is similar to the standard backpropagation of gradients, which is done according to:

$$\nabla_{\theta_k} \ell = \frac{\partial \mathbf{x}_k}{\partial \theta_k} \cdot \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \cdot \dots \cdot \frac{\partial \mathbf{x}_n}{\partial \mathbf{x}_{n-1}} \cdot \nabla_{\mathbf{x}_n} \ell \quad (2)$$

In our algorithm’s two-iteration variant (Algorithm 1), the first iteration performs standard backpropagation (as defined in Equation 2) over layer outputs $\mathbf{x}_{(\cdot)}(\theta)$ and model parameters θ . During this iteration, an optimizer (RMSProp in our experiments) updates the parameters, yielding new weight values θ' . Then it is assumed that the absolute value of the parameter delta $|\theta - \theta'|$ of the RMSProp optimizer is good enough to retain it. The second backpropagation is performed for eventual negations of update directions only, where, conversely, the *average* gradient is propagated. Importantly, the range on which the gradient is averaged equals $[\theta, \theta']$ (between parameters before and after the estimated potential update). The average derivatives of each nonlinear activation are

calculated as follows:

$$\begin{aligned} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,i}(\vartheta)}{\partial x_{k-1,i}(\vartheta)} &= \int_0^1 \frac{\partial x_{k,i}(\theta + t \cdot (\theta' - \theta))}{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))} dt \\ &= \frac{\int_0^1 \frac{\partial x_{k,i}(\theta + t \cdot (\theta' - \theta))}{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))} dt \cdot \int_0^1 \frac{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))}{\partial t} dt}{\int_0^1 \frac{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))}{\partial t} dt} \end{aligned}$$

where $x_{k-1,i}$ and $x_{k,i}$ are the input and output of a nonlinear activation, respectively.

Both variables depend on the complete set of model parameter values, denoted by

ϑ . We approximate $x_{k-1,i}(\theta + t \cdot (\theta' - \theta))$ as an affine (first-order) function of t , i.e.

$x_{k-1,i}(\theta + t \cdot (\theta' - \theta)) \approx \alpha t + \beta$, which implies $\frac{\partial x_{k-1,i}}{\partial t} = \alpha = \text{const}$ and therefore

$\int_0^1 \frac{\partial x_{k-1,i}}{\partial t} dt = \int_0^1 \alpha dt = \alpha = \frac{\partial x_{k-1,i}}{\partial t}$. Finally, this constant can be carried into the

remaining integral $\int_0^1 \frac{\partial x_{k,i}}{\partial x_{k-1,i}} dt$, yielding:

$$\begin{aligned} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,i}(\vartheta)}{\partial x_{k-1,i}(\vartheta)} &\approx \frac{\int_0^1 \frac{\partial x_{k,i}(\theta + t \cdot (\theta' - \theta))}{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))} \cdot \frac{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))}{\partial t} dt}{\int_0^1 \frac{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))}{\partial t} dt} \\ &= \frac{\int_0^1 \frac{\partial x_{k,i}(\theta + t \cdot (\theta' - \theta))}{\partial t} dt}{\int_0^1 \frac{\partial x_{k-1,i}(\theta + t \cdot (\theta' - \theta))}{\partial t} dt} = \frac{x_{k,i}(\theta') - x_{k,i}(\theta)}{x_{k-1,i}(\theta') - x_{k-1,i}(\theta)} \end{aligned} \quad (3)$$

The main advantage of our approximation versus alternative schemes is that we never

linearize the partial derivative $\frac{\partial x_{k,i}}{\partial x_{k-1,i}}$ in θ , θ' , or t ; consequently, it captures the true

dynamics of the gradients more accurately. Moreover, the calculation is computationally

efficient. Division-by-zero cases are handled as described in Algorithm 1.

We assume that each activation is represented as a distinct k^{th} layer, i.e., $\mathbf{f}_k : \mathbb{R}^m \rightarrow$

\mathbb{R}^m , where m is the length of both representations \mathbf{x}_{k-1} and \mathbf{x}_k . Its input is the output

from the $(k-1)$ th layer, given by $\mathbf{x}_{k-1}(\vartheta) = [x_{k-1,1}(\vartheta), x_{k-1,2}(\vartheta), \dots, x_{k-1,n}(\vartheta)]$ which

depends on the model parameters ϑ . Then the average gradient of layer \mathbf{f}_k is given by:

$$\begin{aligned} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_{k-1}} &= \text{diag}([\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,1}(\vartheta)}{\partial x_{k-1,1}(\vartheta)}, \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,2}(\vartheta)}{\partial x_{k-1,2}(\vartheta)}, \dots, \\ &\quad \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,n}(\vartheta)}{\partial x_{k-1,n}(\vartheta)}]) \end{aligned} \quad (4)$$

where each term $\mathcal{AVG}_{(\cdot)}(\cdot)$ is approximated in Eq. 3, and $\text{diag}(\cdot)$ denotes a diagonal matrix constructed from the input vector.

Although the average gradient for parameterized layers can be efficiently approximated, we defer its implementation for future work. Instead of calculating the average gradient for all layers, we restrict our analysis to the activation layers, which is sufficient for a satisfactory improvement in the performance of deep models. Assuming that the k^{th} layer consists solely of an activation function and that the $(k - 1)$ th layer is either fully-connected or convolutional, we extend the approximations as follows:

$$\begin{aligned}\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_{k-2}} &\approx \frac{\partial \mathbf{x}_{k-1}}{\partial \mathbf{x}_{k-2}} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_{k-1}} \\ \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k}{\partial \vartheta_{k-1}} &\approx \frac{\partial \mathbf{x}_{k-1}}{\partial \vartheta_{k-1}} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k}{\partial \mathbf{x}_{k-1}}\end{aligned}\tag{5}$$

This equation intuitively moves the approximated values closer to the gradient. Our goal is to obtain an approximation of the average gradient that is more accurate than using the gradient directly as an estimate. Therefore, if the precision of the approximation remains between that of the gradient and our current estimate in Eq. 1, it satisfies our objective.

A variant of our algorithm employs n backpropagation iterations, during which the approximated average gradient is iteratively refined—with each iteration using the preceding estimate as input. The goal of these subsequent iterations is to synchronize the range of gradient averaging with the parameter update (i.e., ensuring $\theta'^{(t)} \approx \theta^{(t+1)}$, using the notation introduced in Algorithm 1). The intuition is that with each iteration, the averaged derivatives of the nonlinear activations become better aligned with the actual parameter update. Consequently, the average gradient is progressively refined to match the corresponding parameter update more precisely, forming an iterative loop of improvement. Note that each iteration is typically marginally slower to compute than a

Algorithm 1 *Single Step in the Two-Iteration Algorithm* assuming a plain feedforward neural network.

Input: $f = (f_1, f_2, \dots, f_n)$ Model with n layers

$\theta^{(t)}$ Parameters of the model

$s^{(t)}$ *optimizer's* state

Output: $\theta^{(t+1)}, s^{(t+1)}$

- 1: Sample a batch $B^{(t)}$.
 - 2: $(\mathbf{x}_1^{(t)}, \mathbf{x}_2^{(t)}, \dots, \mathbf{x}_n^{(t)}) = f(\theta^{(t)}, B^{(t)})$ {Compute outputs of each layer of the model.
Assume that \mathbf{x}_i is a vector.}
 - 3: $(G_\theta^{(t)}, G_{\mathbf{x}}^{(t)}) = \nabla_{(\theta^{(t)}, \mathbf{x}^{(t)})} \ell(\mathbf{x}_n^{(t)}, B^{(t)})$ {Calculate gradients.}
 - 4: $(\theta'^{(t)}, s^{(t+1)}) \leftarrow \text{optimizer}(s^{(t)}, \theta^{(t)}, G_\theta^{(t)})$ {Estimate the next update.}
 - 5: $(\mathbf{x}'_1, \mathbf{x}'_2, \dots, \mathbf{x}'_n) = f(\theta'^{(t)}, B^{(t)})$
 - 6: **for** $i \leftarrow n$ **to** 1 **step** -1 **do** {Iterate over layers to compute estimated average gradient. Initialize $G'_{\mathbf{x},n} \leftarrow \nabla_{\mathbf{x}'_n} \ell(\mathbf{x}'_n, B^{(t)})$ }
 - 7: **if** f_i is a nonlinear activation **then** $\{G'_{\theta,i} = \emptyset\}$
 - 8: $G'_{\mathbf{x},i-1} \leftarrow \frac{\mathbf{x}'_i - \mathbf{x}_i^{(t)}}{\mathbf{x}'_{i-1} - \mathbf{x}_{i-1}^{(t)}} \circ G'_{\mathbf{x},i}$ {Equation 3. If $(x'_{i-1,j} = x_{i-1,j}^{(t)})$, then

$$G'_{\mathbf{x},(i-1,j)} \leftarrow \frac{\partial x_{i,j}^{(t)}}{\partial x_{i-1,j}^{(t)}} G'_{\mathbf{x},(i,j)}$$
}
 - 9: **else**
 - 10: $G'_{\mathbf{x},i-1} \leftarrow \frac{\partial \mathbf{x}_i^{(t)}}{\partial \mathbf{x}_{i-1}^{(t)}} G'_{\mathbf{x},i}$ {Equation 5}
 - 11: $G'_{\theta,i} \leftarrow \frac{\partial \mathbf{x}_i^{(t)}}{\partial \theta_i^{(t)}} G'_{\mathbf{x},i}$ {Equation 5}
 - 12: **end if**
 - 13: **end for**
 - 14: $\theta^{(t+1)} \leftarrow \theta^{(t)} + (\theta'^{(t)} - \theta^{(t)}) \circ \text{sign}(G_\theta'^{(t)})$
-

single backpropagation procedure.

An interesting way of comparing the gradient-based RMSProp optimization with our algorithm is to examine the average loss deltas for all weight updates of both algorithms. The purpose of the evaluation approach is to validate the proof in Appendix B. However, such a comparison focuses on loss measurements for a *single batch* each time, making it an imperfect predictor of performance on the *whole dataset*. The first iteration of our method is the gradient-based RMSProp procedure, hence the change in loss for RMSProp $\Delta_{RMSProp}$ is known for both the same model parameters and data as in the case of the loss delta of our method. Therefore, the sum of loss differences after the updates of both approaches can be easily and measurably compared relatively to the sum of loss deltas of RMSProp:

$$\begin{aligned} \mathcal{RD}_{AG,RMSProp} &= \frac{\mathcal{AVG}_{b \in B} \Delta_{AG} - \mathcal{AVG}_{b \in B} \Delta_{RMSProp}}{|\mathcal{AVG}_{b \in B} \Delta_{RMSProp}|} \\ &= \frac{\sum_{b \in B} (\ell_b(\theta'_{AG,b}) - \ell_b(\theta_b))}{|\sum_{b \in B} (\ell_b(\theta'_{RMSProp,b}) - \ell_b(\theta_b))|} - \text{sign}(\sum_{b \in B} (\ell_b(\theta'_{RMSProp,b}) - \ell_b(\theta_b))) \end{aligned} \quad (6)$$

$\mathcal{RD}_{AG,RMSProp}$ is the relative difference in avg. loss deltas of *RMSProp* and the method based on the average gradient (*AG*). The \mathcal{AVG} operator denotes the arithmetic average. B is the set of all batches. $\Delta_{AG,b}$ is the loss delta assuming a batch b after our algorithm's update of model parameters θ_b to new values $\theta'_{AG,b}$. Notation for RMSProp is analogous. ℓ_b is the loss, assuming data of a batch b . $\text{sign}(\cdot)$ is the sign function. \mathcal{RD} would not be as useful when using momentum because the metric compares the aggregated loss of a single batch per parameter update, whereas momentum contributes to a decrease in loss over many batches per a single parameter update. Without the momentum, \mathcal{RD} significantly increases the statistical confidence in comparing training algorithms because, for *the same* model weights, the losses are compared for each weight

update. Keeping the same parameter values for each loss delta reduces the variance of \mathcal{RD} , resulting in a decrease in errors when comparing methods.

3.2 Models and Training

We tested our algorithm on three different models, hereafter referred to as A, B, and C (see Tab. 1). The first two models were trained using the MNIST and Fashion MNIST (Xiao, Rasul, & Vollgraf, 2017) image datasets. Although their image characteristics differ significantly, the input sizes are identical, enabling us to cross-evaluate models A and B. In contrast, Model C was trained on a natural language processing task, the IMDb dataset (Maas et al., 2011).

Epoch counts were tailored so that training achieved minimal or near-minimal test loss values before the final epoch of gradient-based RMSProp training. Throughout our experiments, we exclusively used cross-entropy loss with a uniform batch size of 128.

Our primary objective is to compare our method with both Adam (Kingma & Ba, 2014) and RMSProp (Tieleman & Hinton, 2012). Adam has become the de facto baseline in deep learning in recent years. Additionally, comparing with RMSProp enables us to more precisely delineate the performance difference between using the standard gradient and the average gradient.

Table 1: *Models*

Model	Architecture Summary	Datasets	Parameter Count
Model A	6×Convolution 2D + ELU 1×Linear	MNIST, Fashion MNIST	17506

The purpose is to assess learning performance on shallow convolutional architectures, whose loss landscapes exhibit reduced nonlinearities compared to those of deeper models.

Model B	2×Convolution 2D + ELU 2×Max Pooling 2D 28×Linear + Tanh	MNIST, Fashion MNIST	8228
---------	--	-------------------------	------

The aim is to evaluate performance on a computationally feasible optimization task that presents singularity-related challenges (O. Oyedotun et al., 2021; Yasrab, 2019), which arise from the repetition of 10 neurons with tanh activation across all linear layers. This setup effectively simulates the conditions of optimizing a deeper and wider Multilayer Perceptron architecture with numerous spatially independent connections, thus increasing the complexity of the optimization process (Balduzzi et al., 2017).

Model C	41×Convolution 2D + Tanh 5×Linear + Tanh	IMDb	14397
---------	---	------	-------

The objective is to evaluate performance on a computationally feasible NLP optimization task. In this task, we employ an architecture with tanh activations that is sufficiently deep to induce singularity-related issues (O. Oyedotun et al., 2021; Yasrab, 2019). However, these issues are less severe than in Model B, owing to the increased dimensionality of the internal representations in Model C. Moreover, the challenges related to spatial independence between neuronal connections (Balduzzi et al., 2017) are mitigated through the use of convolutional layers.

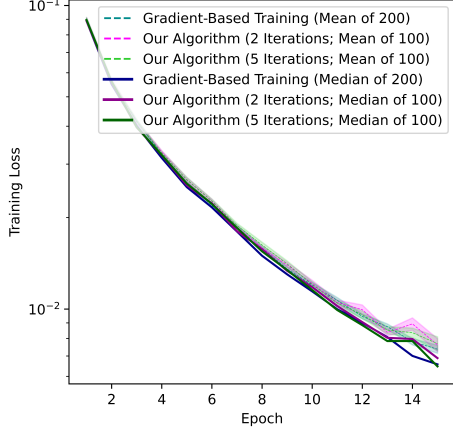
For every experimental configuration—each corresponding to a unique combination of model, optimizer (RMSProp (Tieleman & Hinton, 2012), Adam (Kingma & Ba, 2014), or variants of our method), dataset, and weight initialization technique (Glorot (Glorot & Bengio, 2010) or one with slightly vanishing gradients)—we conducted a separate learning-rate search. Additionally, for experiments on Model B, we used the same learning rate for both our method and gradient-based RMSProp, ensuring that any observed performance improvements could be attributed solely to our algorithm rather than to variations in the learning rate.

Nevertheless, since the method does not have any hyperparameters apart from the learning rate, it is less likely to overfit to a specific experimental setup (model, dataset, and learning rate) and show good results on it while experiencing deficient performance on other setups.

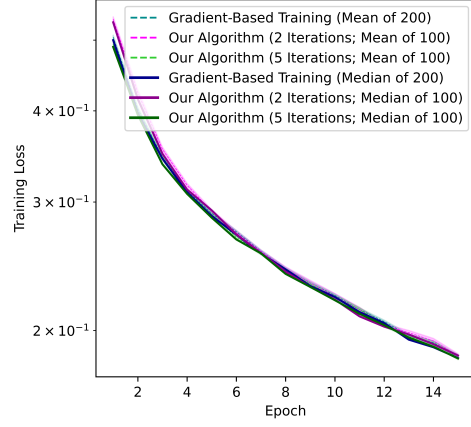
4 Results

For the shallow model A, all of the training algorithms are approximately equal (Fig. 3a; Fig. 3b). The relative difference in summed loss deltas (Eq. 6) revealed that the algorithm based on the average gradient is only marginally better than the standard RMSprop according to $\mathcal{RD} = 1.20\text{e-}3 \pm 2.7\text{e-}4$ (0.12% faster minimization of loss with 0.027% of SEM error) on MNIST and $\mathcal{RD} = 5.86\text{e-}3 \pm 2.79\text{e-}3$ on Fashion MNIST in the case of two iterations. For five iterations, $\mathcal{RD} = 6.47\text{e-}4 \pm 9.8\text{e-}5$ on MNIST and $\mathcal{RD} = 2.37\text{e-}3 \pm 4.5\text{e-}4$ on Fashion MNIST.

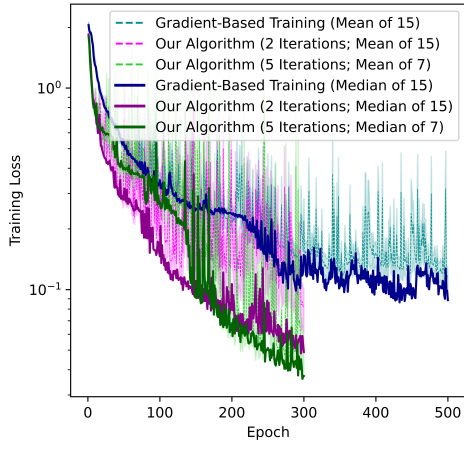
The results of Model B are much more interesting. The version of the algorithm with



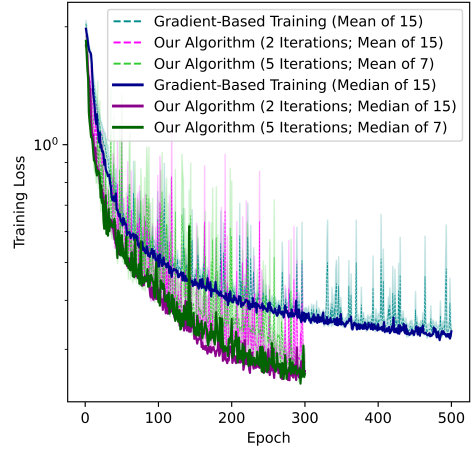
(a) *Model A on MNIST*



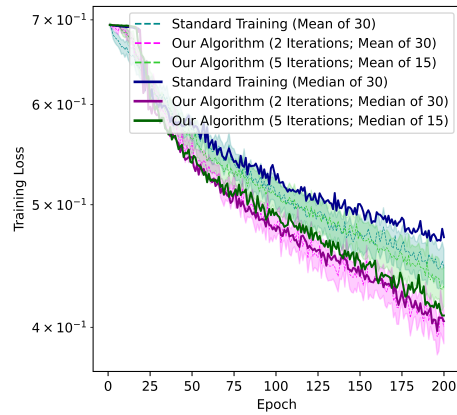
(b) *Model A on Fashion MNIST*



(c) *Model B on MNIST*



(d) *Model B on Fashion MNIST*



(e) *Model C on IMDB*

Figure 3: *Training losses*. Only mean curves contain confidence ranges (SEM).

two iterations is about three times faster at minimizing the median of training losses on both datasets (Fig. 3c; Fig. 3d; Tab. 3). Moreover, the mean losses are considerably lower than those observed with standard RMSProp training, even when the experiments are repeated using an alternative weight initialization method (more details are provided at the end of this section). Despite the minority of epochs with high oscillations, the method utilizing the average gradient is approximately two to three times faster in minimizing the mean loss, although this is not clearly visible in the plots. Furthermore, for both versions of our algorithm on both datasets, during from 49.3% to 70% of epochs, the average training loss was lower with statistical significance (SEM) than for the gradient-based RMSProp. Conversely, our algorithm was worse in that respect during from 0.667% to 2.33% of epochs with statistical significance. The average of minimal training losses on MNIST for the five iterations is 0.0393 ± 0.0058 , which is significantly lower than 0.0883 ± 0.0117 for the standard RMSProp. Meanwhile, the two iterations are also perform better than the gradient-based RMSProp, but without statistical significance, achieving 0.0747 ± 0.0188 . Even better averages of minimal training losses were obtained on Fashion MNIST, with the five-iteration and two-iteration versions achieving 0.254 ± 0.017 and 0.257 ± 0.014 respectively, compared to 0.314 ± 0.008 by the gradient-based training.

Plots of the test losses of Model B look very similar to the training losses (Appendix C), showing significant improvements in generalization, which correspond to the lower training losses. On MNIST, the average of best accuracies over training for five iterations is equal to $(97.87 \pm 0.09)\%$, which is significantly higher than $(96.80 \pm 0.78)\%$ and $(96.75 \pm 0.55)\%$ for the two-iteration version and gradient-based training, respec-

tively. On Fashion MNIST, the analogous results are $(88.09 \pm 0.35)\%$, $(87.54 \pm 0.55)\%$ and $(86.57 \pm 0.29)\%$, respectively.

For Model B, the \mathcal{RD} metric (Equation 6) provides a very high confidence of superiority of the average gradient for the high learning rates used for the trainings based on our algorithm. On MNIST for two and five iterations, it equals 10.41 ± 1.94 and 1.43 ± 0.29 , respectively. On Fashion MNIST, it is 0.58 ± 0.14 and 0.24 ± 0.04 for both variants, respectively.

Importantly, for Model B, the average-gradient algorithm dominated also for the learning rates that are optimal for the standard RMSProp training. Multiple metrics favored our algorithm with statistical significance, i.e., $\mathcal{RD} \in [0.0611 \pm 0.0004, 1.07 \pm 0.31]$, despite training counts equal to only two or three (for each of the four experiments).

The average gradient is superior for the deep convolutional model on the IMDB dataset. The performance of gradient-based RMSProp at epoch 200 is approximately equal to that of our two-iteration variant at epoch 127 using the median and average losses computed across runs. This equivalence implies that our two-iteration algorithm achieves about 58% higher sample efficiency than vanilla RMSProp at the final point of the training, and 35% higher sample efficiency at an average training point. Interestingly, performance of the four-iteration variant falls between the two-iteration variant and vanilla RMSProp. Moreover, the \mathcal{RD} metric—yielding $\mathcal{RD} = 0.0394 \pm 0.0053$ for the two-iteration variant versus $\mathcal{RD} = 0.0210 \pm 0.0018$ for the four-iteration variant—further supports the superiority of the two-iteration approach.

Repeated experiments with Model B employing Glorot weight initialization (Glorot & Bengio, 2010), coupled with separate learning-rate searches, confirmed that our

Table 2: *Minimal Training Losses of Model B Using Glorot Weight Initialization (Glorot & Bengio, 2010).* The confidence intervals represent the standard errors of the mean. Relative sample efficiency is computed separately for mean and median losses at each epoch, geometrically averaged, and finally arithmetically averaged over all epochs.

Dataset	Method	Min. Training Loss	Relative Sample Efficiency
MNIST	Adam (125 epochs)	0.0534 ± 0.0084	112%
	RMSProp (125 epochs)	0.0478 ± 0.0032	100%
	2 Iterations (50 epochs)	0.0245 ± 0.0006	444%
Fashion MNIST	Adam (125 epochs)	0.342 ± 0.018	101%
	RMSProp (125 epochs)	0.344 ± 0.010	100%
	2 Iterations (50 epochs)	0.253 ± 0.003	477%

method achieved a sample efficiency at least 4 times greater than that of gradient-based RMSProp across the datasets (see Tab. 2). Moreover, we tuned Adam (Kingma & Ba, 2014) using the same weight initialization method across all models and datasets. Adam’s performance was comparable to RMSProp. Tab. 3 summarizes the efficiency results for Adam and two iterations of our method. For Models B and C, training results are aggregated only for Glorot weight initialization, which is commonly used in practice.

Table 3: *Computational Efficiency Analysis of the Two-Iteration Variant Across Models.*

For Models B and C, only experiments employing Glorot weight initialization (Glorot & Bengio, 2010) are included. Computational complexity is asymptotically equivalent to memory complexity (up to a constant factor). Here, P denotes the number of parameters, d_l denotes the dimensionality of the l^{th} layer’s output, and B refers to the batch size, a definition that also applies when using gradient accumulation. We assume that a dedicated memory buffer of size P is allocated to store the gradients. Optimization speed is calculated by dividing sample efficiency by the epoch computation time.

	Relative Sample Efficiency Measured for Our Implementation	Estimated Relative Optimization Speed of Optimal Implementation	Relative Optimization Speed Measured for Our Implementation	Memory Complexity of a Step
RMSProp	100%	100%	100%	$O(3P + B \sum_{l=1}^n d_l)$
Adam	{107, 109, 109}%	{97, 99, 99}%	{98, 109, 109}%	$O(4P + B \sum_{l=1}^n d_l)$
2 Iterations	{99, 135 , 461 }%	{49, 68, 230 }%	{40, 41, 181 }%	$O(4P + 2B \sum_{l=1}^n d_l)$

5 Discussion

Our experiments directly reveal that adjusting the gradient on nonlinear activations can improve sample efficiency by multiple times in certain deep models. For the MNIST and Fashion MNIST benchmarks, the algorithm based on the average gradient demonstrates significant benefits compared to standard RMSProp and Adam training methods when applied to deep neural networks consisting of multiple fully connected layers with nonlinear activation functions: (a) An increase of up to 4.7-fold in training sample efficiency. (b) Significant reduction in wall-clock training time. (c) Stable

performance over a much wider range of learning rates results in significant benefits in terms of both reduced electricity consumption and time spent on hyperparameter searches. (d) Considerably better generalization, at least in a reasonable epoch count.

In contrast, for a deep sequential convolutional model trained on the IMDB dataset, sample efficiency improved by approximately 35% when using only two iterations of our algorithm. This improvement is especially significant in online learning, given that the associated increase in computational cost is moderate. The variant employing more iterations achieved efficiency values intermediate between those of vanilla RMSProp and the two-iteration variant.

However, we did not test our method on deep ResNet architectures, as these models behave similarly to ensembles of relatively shallow networks (Veit, Wilber, & Belongie, 2016). Nevertheless, our goal of identifying a use case for the average gradient has been achieved.

The \mathcal{RD} (Equation 6) confirms the outstanding results of the other measures. The score of $\mathcal{RD} = 10.41 \pm 1.94$, achieved by the two iterations on MNIST, corresponds to the average speed of batch-loss minimization that is $(1141 \pm 194)\%$ of the speed of the gradient-based RMSProp while using the same absolute values of weight updates. In the other cases of deep models, the average speed of batch-loss minimization ranges from $(2.10 \pm 0.18)\%$ to $(243 \pm 29)\%$. Therefore, even a relatively slight speedup in batch-loss minimization (such as 2.1% on the IMDB dataset) can contribute to a significantly higher gain in sample efficiency. Moreover, it is crucial to note that the highest of the mentioned gains occur at learning rates that are three times higher than the optimal rates for gradient-based training. Generally, high learning rate values may enable rapid learning because

model parameters are adjusted faster. Nevertheless, the average gradient is also superior in terms of the average speed of batch–loss minimization when using the optimal learning rates for gradient–based training across all tested models, with statistical significance.

Surprisingly, the algorithm version with five iterations is worse than the two iterations according to \mathcal{RD} with higher statistical confidence than for other measures. Across all experiments, the variant is computationally inefficient in terms of the resources required to reduce the loss to a certain level.

In the case of the shallow model incorporating nonlinear ELU activations, our method demonstrates only marginal improvements compared with gradient–based training using RMSProp and Adam. For shallow models, the average gradient converges toward the gradient, leading to comparable performance.

However, we did not evaluate our method on the most recent state–of–the–art models. This is because a considerable limitation of our work was the significant coding effort required to implement our method via low–level modifications of the backpropagation algorithm, while maintaining acceptable portability and computational efficiency. Consequently, we believe our results are particularly valuable compared to related work that relied on adjusted tools and simpler workflows. Furthermore, our method’s support for multiple simple architectures enabled several important experiments. We view these experiments as strong indicators of good performance in some feature–extraction scenarios, specifically those requiring numerous linear or convolutional layers combined with nonlinear activations. In fact, nearly all learning tasks depend on features that can be extracted by relatively shallow neural networks (Veit et al., 2016); notably, these simple features are implicitly computed by deep models with residual connections (Veit

et al., 2016). However, achieving general intelligence necessitates the development of significantly more complex representations. Therefore, we argue that our approach represents an important step toward more robust optimization techniques necessary for generally intelligent models.

Importantly, the number of backpropagations required to minimize the training loss to a predetermined level was significantly lower when using the average gradient compared to the gradient employed for Model B, across both datasets and weight initializations. This indicates that when singularity problems (O. Oyedotun et al., 2021; O. K. Oyedotun et al., 2022; Yasrab, 2019) and the shattered gradient issue (Balduzzi et al., 2017) are particularly pronounced, standard backpropagation becomes less optimal than the two backpropagations of our method. In contrast, this effect was not observed in related methods involving multiple backpropagations per step (Guan et al., 2024; Tseng et al., 2022), which may be due to their reliance on standard backpropagation. Nevertheless, improving sample–efficiency alone is considerably easier, as demonstrated for Model C.

6 Conclusions

Significant improvements in sample efficiency are evident under the following conditions: (a) when employing various weight initialization techniques; (b) on both natural language processing and computer vision benchmarks; (c) when evaluating both a deep convolutional architecture and a fully–connected network with nonlinear activation functions; (d) when compared with gradient–based optimizers, notably RMSProp and Adam. Furthermore, the computational cost associated with these improvements is

modest (as reported in Tab. 3). When using gradient accumulation, the Adam optimizer’s memory requirements scale in a manner comparable to those of the two–iteration variant of our method (see Tab. 3). These findings are particularly significant for the domains of online learning (Hoi, Sahoo, Lu, & Zhao, 2021) and reinforcement learning (Wang et al., 2024), where sample efficiency is critical.

For very deep models without residual connections, gradient–based training tends to be inefficient (Balduzzi et al., 2017; Zaeemzadeh, Rahnavard, & Shah, 2020b). In this work, we demonstrate techniques to mitigate this inefficiency, at least in certain scenarios. In general, the very deep structure of human brains enables the learning of universal and complex patterns. Therefore, accurately mimicking human brain model could potentially lead to satisfactory results. Our algorithm aims to improve learning in scenarios involving nonlinear neural structures that are very deep, a feature of provably efficient biological brains that distinguishes them from current AI models. Therefore, the method may contribute to the training of large models in the future, where sample efficiency is needed to learn new tasks on the fly, akin to how people or some animals do.

Our primary contribution is incorporating the average gradient as an efficient alternative to higher–order derivative computations, since both approaches exploit gradient dynamics. Our second major contribution is the efficient approximation of averaged integrated gradients—a key advancement toward practical applications. Finally, our third contribution involves enhancing RMSProp (Tieleman & Hinton, 2012) for specific models, demonstrating a special case of an algorithm that utilizes the average gradient for curvature estimation. Consequently, our results indicate not only that RMSProp can be significantly improved to outperform Adam (Kingma & Ba, 2014) on certain models,

but also that our novel, efficient curvature estimation technique substantially surpasses the performance of other popular deep learning optimizers in selected scenarios.

However, a key feature of our algorithm parallels processes observed in biological brains. In biological systems, when specific conditions indicate that a synaptic connection should be formed, such as when the Hebbian rule is satisfied (Bi & Poo, 1998; Caporale & Dan, 2008), the connection might be temporarily established (Becker & Tetzlaff, 2021; Yang, Pan, & Gan, 2009). Subsequently a further evaluation is performed, corresponding to the aggregation of synaptic tags across new connections (Frey & Morris, 1997; Redondo & Morris, 2011), to determine which connections should be maintained. Accordingly, both our algorithm and biological brains initially perform an update candidate step; however, the effect of this update is later reevaluated in light of the evolving behavior of the neural structure. Moreover, the process of synaptic-tag aggregation can be mathematically characterized by computing the integral or average of some learning signal for each neuron over a dynamic trajectory of network changes, as implemented in our method. These similarities, along with our results, suggest that the efficiency of biological learning may partly result from navigating through multiple potential network configurations by continuously evaluating transient connections.

In this paper, the proposed method was not evaluated on the most recent state-of-the-art models. This limitation is primarily due to the substantial coding effort required to implement our approach, which involves low-level modifications to the backpropagation algorithm. Ensuring both portability and computational efficiency under these constraints proved challenging. The primary difficulty was integrating current implementations of state-of-the-art deep models into the computation graph required for backpropagation

of the average gradient. Consequently, although our method may be more challenging to integrate into modern frameworks, we contend that our results are particularly significant when compared to related work that relied on preexisting high-level tools.

While we have not compared our algorithm to second-order optimization methods, the average gradient, introduced here as a novel curvature predictor, shows encouraging performance at this stage of our study. This stands in contrast to classical second-order techniques, which have been refined since at least the 17th century (B. Polyak, 2007). Nevertheless, it remains to be determined whether the average gradient, like the standard gradient, can be efficiently preconditioned using second-order statistics (Gupta et al., 2018; Vyas et al., 2024), an aspect that presents an avenue for future research. To the best of our knowledge, no previous work has incorporated the integrated gradient into an optimizer; however, our analysis reveals multiple promising directions. The average gradient can be estimated more precisely using Eq. 1 rather than sacrificing accuracy by relying on Eq. 5. Moreover, promising avenues for future research include exploring different integration ranges and leveraging various known optimizer enhancements.

References

Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K. W.-D., & McWilliams, B. (2017).

The shattered gradients problem: If resnets are the answer, then what is the question? In *International conference on machine learning* (pp. 342–350).
doi:10.48550/arXiv.1702.08591

Basodi, S., Ji, C., Zhang, H., & Pan, Y. (2020). Gradient amplification: An efficient

- way to train deep neural networks. *Big Data Mining and Analytics*, 3(3), 196–207.
doi:10.26599/BDMA.2020.9020004
- Becker, M. F., & Tetzlaff, C. (2021). The biophysical basis underlying the maintenance of early phase long-term potentiation. *PLoS computational biology*, 17(3), e1008813.
doi:10.1371/journal.pcbi.1008813
- Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2), 157–166.
doi:10.1109/72.279181
- Bi, G.-q., & Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24), 10464–10472. Retrieved from <https://www.jneurosci.org/content/18/24/10464>
doi:10.1523/JNEUROSCI.18-24-10464.1998
- Caporale, N., & Dan, Y. (2008). Spike timing-dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31(1), 25–46.
doi:10.1146/annurev.neuro.31.060407.125639
- Dozat, T. (2016). Incorporating nesterov momentum into adam. Retrieved from <https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ>
- Dubey, S. R., Chakraborty, S., Roy, S. K., Mukherjee, S., Singh, S. K., & Chaudhuri, B. B. (2019). diffgrad: an optimization method for convolutional neural networks. *IEEE transactions on neural networks and learning systems*, 31(11), 4500–4511.
doi:10.48550/arXiv.1909.11015
- Frey, U., & Morris, R. G. (1997). Synaptic tagging and long-term potentiation. *Nature*,

385(6616), 533–536. doi:10.1038/385533a0

- Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256). Retrieved from <https://proceedings.mlr.press/v9/glorot10a.html>
- Grosse, R., & Martens, J. (2016, 20–22 Jun). A kronecker-factored approximate fisher matrix for convolution layers. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (Vol. 48, pp. 573–582). New York, New York, USA: PMLR. Retrieved from <https://proceedings.mlr.press/v48/grosse16.html>
- Guan, L., Li, D., Shi, Y., & Meng, J. (2024). Xgrad: Boosting gradient-based optimizers with weight prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10), 6731–6747. doi:10.1109/TPAMI.2024.3387399
- Gupta, V., Koren, T., & Singer, Y. (2018, 10–15 Jul). Shampoo: Preconditioned stochastic tensor optimization. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (Vol. 80, pp. 1842–1850). PMLR. Retrieved from <https://proceedings.mlr.press/v80/gupta18a.html>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034). doi:10.48550/arXiv.1502.01852
- Hoi, S. C., Sahoo, D., Lu, J., & Zhao, P. (2021). Online learning: A comprehensive survey. *Neurocomputing*, 459, 249–289. doi:10.1016/j.neucom.2021.04.112

- Hughes-Hallett, D., Gleason, A. M., Lock, P. F., & Flath, D. E. (2021). *Applied calculus*. John Wiley & Sons. Retrieved from <https://books.google.com/books/about/Applied-Calculus.html?id=TtxCEAAAQBAJ>
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. , 448–456. doi:10.48550/arXiv.1502.03167
- Khorram, S., Lawson, T., & Fuxin, L. (2021). igos++ integrated gradient optimized saliency by bilateral perturbations. In *Proceedings of the conference on health, inference, and learning* (pp. 174–182). doi:10.48550/arXiv.2012.15783
- Kingma, D., & Ba, J. (2014, 12). Adam: A method for stochastic optimization. *International Conference on Learning Representations*. doi:10.48550/arxiv.1412.6980
- Kirk, R., Mediratta, I., Nalmpantis, C., Luketina, J., Hambro, E., Grefenstette, E., & Raileanu, R. (2023). Understanding the effects of rlhf on llm generalisation and diversity. *arXiv preprint arXiv:2310.06452*. doi:10.48550/arXiv.2310.06452
- Li, X., Yang, W., Liang, J., Zhang, Z., & Jordan, M. I. (2023, 25–27 Apr). A statistical analysis of polyak-ruppert averaged q-learning. In F. Ruiz, J. Dy, & J.-W. van de Meent (Eds.), *Proceedings of the 26th international conference on artificial intelligence and statistics* (Vol. 206, pp. 2207–2261). PMLR. Retrieved from <https://proceedings.mlr.press/v206/li23b.html>
- Liu, Y., Gao, Y., & Yin, W. (2020). An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, 33, 18261–18271. doi:10.48550/arXiv.2007.07989
- Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of*

- the association for computational linguistics: Human language technologies* (pp. 142–150). Association for Computational Linguistics. Retrieved from <https://aclanthology.org/P11-1015/>
- Martens, J., & Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning* (pp. 2408–2417). doi:10.48550/arXiv.1503.05671
- Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*. doi:10.48550/arXiv.1708.02182
- OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., ... Zoph, B. (2023, 03). Gpt-4 technical report. doi:10.48550/arXiv.2303.08774
- Oyedotun, O., Al Ismaeil, K., & Aouada, D. (2021, 02). Training very deep neural networks: Rethinking the role of skip connections. *Neurocomputing*, 441. doi:10.1016/j.neucom.2021.02.004
- Oyedotun, O. K., Al Ismaeil, K., & Aouada, D. (2022). Why is everyone training very deep neural network with skip connections? *IEEE Transactions on Neural Networks and Learning Systems*, 34(9), 5961–5975. doi:10.1109/TNNLS.2021.3131813
- Polyak, B. (2007). Newton’s method and its use in optimization. *European Journal of Operational Research*, 181(3), 1086-1096. doi:<https://doi.org/10.1016/j.ejor.2005.06.076>
- Polyak, B. T., & Juditsky, A. B. (1992). Acceleration of stochastic approxima-

- tion by averaging. *SIAM journal on control and optimization*, 30(4), 838–855.
doi:10.1137/0330046
- Redondo, R. L., & Morris, R. G. (2011). Making memories last: the synaptic tagging and capture hypothesis. *Nature reviews neuroscience*, 12(1), 17–30.
doi:10.1038/nrn2963
- Roy, S. K., Paoletti, M. E., Haut, J. M., Dubey, S. R., Kar, P., Plaza, A., & Chaudhuri, B. B. (2021). Angulargrad: A new optimization technique for angular convergence of convolutional neural networks. *arXiv preprint arXiv:2105.10190*.
doi:10.48550/arXiv.2105.10190
- Ruppert, D. (1988). *Efficient estimations from a slowly convergent robbins-monro process* (Tech. Rep.). Cornell University Operations Research and Industrial Engineering. Retrieved from <https://ecommons.cornell.edu/items/9a14790e-66a6-4460-9280-e9fb146fd02d>
- Sattarzadeh, S., Sudhakar, M., Plataniotis, K. N., Jang, J., Jeong, Y., & Kim, H. (2021). Integrated grad-cam: Sensitivity-aware visual explanation of deep convolutional networks via integrated gradient-based scoring. In *Icassp 2021-2021 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 1775–1779). doi:10.48550/arXiv.2104.02637
- Sun, X., Kashima, H., Matsuzaki, T., & Ueda, N. (2010). Averaged stochastic gradient descent with feedback: An accurate, robust, and fast training method. In *2010 ieee international conference on data mining* (pp. 1067–1072).
doi:10.1109/ICDM.2010.26
- Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep

- networks. In *International conference on machine learning* (pp. 3319–3328). doi:10.48550/arXiv.1703.01365
- Tan, H. H., & Lim, K. H. (2019). Review of second-order optimization techniques in artificial neural networks backpropagation. In *Iop conference series: materials science and engineering* (Vol. 495, p. 012003). doi:10.1088/1757-899X/495/1/012003
- Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105–6114). doi:10.48550/arXiv.1905.11946
- Tieleman, T., & Hinton, G. (2012). *Lecture 6.5 – rmsprop: Divide the gradient by a running average of its recent magnitude*. Retrieved from https://www.cs.toronto.edu/~hinton/coursera_lectures.html (Lecture slides for Neural Networks for Machine Learning (Coursera))
- Tseng, C.-H., Liu, H.-C., Lee, S.-J., & Zeng, X. (2022, 07). Perturbed gradients updating within unit space for deep learning. In *2022 international joint conference on neural networks (ijcnn)* (p. 01-08). doi:10.1109/IJCNN55064.2022.9892245
- Veit, A., Wilber, M. J., & Belongie, S. (2016). Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29. doi:10.48550/arXiv.1605.06431
- Vyas, N., Morwani, D., Zhao, R., Shapira, I., Brandfonbrener, D., Janson, L., & Kakade, S. M. (2024). Soap: Improving and stabilizing shampoo using adam. *ArXiv, abs/2409.11321*. Retrieved from <https://api.semanticscholar.org/CorpusID:272694107>
- Wang, X., Wang, S., Liang, X., Zhao, D., Huang, J., Xu, X., ... Miao, Q. (2024). Deep

- reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4), 5064–5078. doi:10.1109/TNNLS.2022.3207346
- Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*. doi:10.48550/arXiv.1708.07747
- Yang, G., Pan, F., & Gan, W.-B. (2009). Stably maintained dendritic spines are associated with lifelong memories. *Nature*, 462(7275), 920–924. doi:10.1038/nature08577
- Yasrab, R. (2019). Srnet: a shallow skip connection based convolutional neural network design for resolving singularities. *Journal of Computer Science and Technology*, 34, 924–938. doi:https://doi.org/10.1007/s11390-019-1950-8
- Zaeemzadeh, A., Rahnavard, N., & Shah, M. (2020a). Norm-preservation: Why residual networks can become extremely deep? *IEEE transactions on pattern analysis and machine intelligence*, 43(11), 3980–3990. doi:10.48550/arXiv.1805.07477
- Zaeemzadeh, A., Rahnavard, N., & Shah, M. (2020b). Norm-preservation: Why residual networks can become extremely deep? *IEEE transactions on pattern analysis and machine intelligence*, 43(11), 3980–3990. doi:10.48550/arXiv.1805.07477
- Zheng, H., Wang, B., Xiao, M., Qin, H., Wu, Z., & Tan, L. (2024). Adaptive friction in deep learning: Enhancing optimizers with sigmoid and tanh function. In *2024 IEEE 6th international conference on power, intelligent computing and systems (icpics)* (pp. 809–813). doi:10.48550/arXiv.2408.11839

A Definition of Average Gradient/Jacobian

Let us define the average gradient of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ over a parameter range

$[\mathbf{a}, \mathbf{b}]$, with $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$, via the mapping $\mathbf{g} : \mathbb{R}^m \rightarrow \mathbb{R}^n$, by

$$\mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f = \int_0^1 \nabla_{\mathbf{g}(\mathbf{a} + t \cdot (\mathbf{b} - \mathbf{a}))} f \, dt \quad (7)$$

which can alternatively be written by switching the integration variable to any component x_i of \mathbf{x} , via $x_i = a_i + t \cdot (b_i - a_i)$, yielding

$$\mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f = \frac{1}{b_i - a_i} \cdot \int_{a_i}^{b_i} \nabla_{\mathbf{g}(\mathbf{x})} f \, dx_i \quad (8)$$

However, any case in which a vector component would cause division by zero should be handled via Eq. 7.

If $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^l$, then by applying Eq. 7 component-wise we get

$$\begin{aligned} \mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \frac{\partial \mathbf{f}}{\partial \mathbf{g}(\mathbf{x})} &= [\mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f_1, \mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f_2, \dots, \mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f_l] \\ &= [\int_0^1 \nabla_{\mathbf{g}(\mathbf{a} + t \cdot (\mathbf{b} - \mathbf{a}))} f_1 \, dt, \int_0^1 \nabla_{\mathbf{g}(\mathbf{a} + t \cdot (\mathbf{b} - \mathbf{a}))} f_2 \, dt, \dots, \int_0^1 \nabla_{\mathbf{g}(\mathbf{a} + t \cdot (\mathbf{b} - \mathbf{a}))} f_l \, dt] \\ &= \int_0^1 \frac{\partial \mathbf{f}}{\partial \mathbf{g}(\mathbf{a} + t \cdot (\mathbf{b} - \mathbf{a}))} \, dt \end{aligned} \quad (9)$$

or, alternatively, using Eq. 8:

$$\begin{aligned} \mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \frac{\partial \mathbf{f}}{\partial \mathbf{g}(\mathbf{x})} &= [\mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f_1, \mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f_2, \dots, \mathcal{AVG}_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} \nabla_{\mathbf{g}(\mathbf{x})} f_l] \\ &= \frac{1}{b_i - a_i} \cdot [\int_{a_i}^{b_i} \nabla_{\mathbf{g}(\mathbf{x})} f_1 \, dx_i, \int_{a_i}^{b_i} \nabla_{\mathbf{g}(\mathbf{x})} f_2 \, dx_i, \dots, \int_{a_i}^{b_i} \nabla_{\mathbf{g}(\mathbf{x})} f_l \, dx_i] \\ &= \frac{1}{b_i - a_i} \int_{a_i}^{b_i} \frac{\partial \mathbf{f}}{\partial \mathbf{g}(\mathbf{x})} \, dx_i \end{aligned} \quad (10)$$

which holds for every i . However, if $b_i = a_i$, Eq. 9 must be used instead.

B Proof of Equation 1

Assumptions:

- The neural network is differentiable within the range relevant for computing the average gradient. However, with minor corrections, the reasoning remains valid under piecewise differentiability as well.
- Since we prove Equation 1, we assume that the neural network follows a sequential architecture without skip connections. However, the reasoning can be analogously extended to models with residual connections.
- We assume that correlations (and covariations) between gradient values of different layers and groups of layers in a neural network can be neglected. Note that this assumption is made by some state-of-the-art second-order optimization algorithms for deep neural networks, such as Shampoo (Gupta et al., 2018) and Soap (Vyas et al., 2024), which do not compute second-order statistics across different layers but instead track gradient correlations within each layer (Martens & Grosse, 2015). Therefore, the assumption does not impede achieving curvature estimates with acceptable precision.

Over the course of the proof, we assume a constant batch (or minibatch); therefore, we do not explicitly include it among the arguments of subsequent functions. The batch remains constant, as our goal is to compute the average gradient for this specific batch.

Since we do not compute the covariances of the Jacobians, we assume that each omitted covariance is zero. This assumption mirrors those made implicitly for gradient values across different layers in other state-of-the-art second-order methods (Gupta et

al., 2018; Vyas et al., 2024). By covariance, we mean that if we uniformly sample any random weights ϑ from the range of averaging, defined as $T \sim U(0, 1)$, $\vartheta = \theta + (\theta' - \theta)T$, then:

$$\begin{aligned}
\forall(k, i, j), \text{cov}\left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}}, \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta)\right) &= 0 = \mathbb{E}\left[\left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} - \mathcal{AVG}_{\omega \in [\theta, \theta']} \frac{\partial x_{k,j}(\omega)}{\partial \omega_{k,i}}\right) \cdot \right. \\
&\quad \left. (\nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) - \mathcal{AVG}_{\omega \in [\theta, \theta']} (\nabla_{x_{k,j}(\omega)} \ell(\omega)))\right] \\
&= (\theta'_{k,i} - \theta_{k,i})^{-1} \int_{\theta_{k,i}}^{\theta'_{k,i}} \left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} - \mathcal{AVG}_{\omega \in [\theta, \theta']} \frac{\partial x_{k,j}(\omega)}{\partial \omega_{k,i}}\right) \cdot \\
&\quad (\nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) - \mathcal{AVG}_{\omega \in [\theta, \theta']} (\nabla_{x_{k,j}(\omega)} \ell(\omega))) \, d\vartheta_{k,i}
\end{aligned} \tag{11}$$

where ϑ, ω are two instantiations of the model’s trainable parameters, and θ, θ' specify the lower and upper bounds of the averaging operation (see Appendix A for the definition of the \mathcal{AVG} operator). $x_{k,j}$ refers to the j^{th} output scalar of a layer no. k . ℓ denotes the loss function.

Note that multiple terms in Eq. 11 depend on all of the model parameters, denoted by $\vartheta, \omega, \theta$, and θ' , rather than on a particular scalar parameter—namely, the i^{th} parameter in the k^{th} layer: $\vartheta_{k,i}, \omega_{k,i}, \theta_{k,i}, \theta'_{k,i}$. Moreover, we use gradient notation to denote a scalar derivative (i.e., $\nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) = \frac{\partial \ell(\vartheta)}{\partial x_{k,j}(\vartheta)}$) to remain consistent with later notation referring to higher dimensions.

Cases of division by zero in the term $(\theta'_{k,i} - \theta_{k,i})^{-1}$ should be addressed either analogously to the corresponding cases discussed in Appendix A or by computing the limit as $\theta_{k,i}$ approaches $\theta'_{k,i}$.

We can further transform the equation:

$$\begin{aligned}
\text{cov} = 0 &= (\theta'_{k,i} - \theta_{k,i})^{-1} \int_{\theta_{k,i}}^{\theta'_{k,i}} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \nabla_{x_{k,j}} \ell(\vartheta) \, d\vartheta_{k,i} - \\
&(\theta'_{k,i} - \theta_{k,i})^{-1} \cdot \int_{\theta_{k,i}}^{\theta'_{k,i}} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \mathcal{AVG}_{\omega \in [\theta, \theta']} \nabla_{x_{k,j}(\omega)} \ell(\omega) \, d\vartheta_{k,i} - \\
&(\theta'_{k,i} - \theta_{k,i})^{-1} \cdot \int_{\theta_{k,i}}^{\theta'_{k,i}} \mathcal{AVG}_{\omega \in [\theta, \theta']} \frac{\partial x_{k,j}(\omega)}{\partial \omega_{k,i}} \cdot \nabla_{x_{k,j}(\omega)} \ell(\omega) \, d\vartheta_{k,i} + \\
&(\theta'_{k,i} - \theta_{k,i})^{-1} \cdot \int_{\theta_{k,i}}^{\theta'_{k,i}} \mathcal{AVG}_{\omega \in [\theta, \theta']} \frac{\partial x_{k,j}(\omega)}{\partial \omega_{k,i}} \cdot \mathcal{AVG}_{\omega \in [\theta, \theta']} \nabla_{x_{k,j}(\omega)} \ell(\omega) \, d\vartheta_{k,i}
\end{aligned} \tag{12}$$

where all $\mathcal{AVG}(\cdot)$ terms are constant scalars and can therefore be factored out of the integrals:

$$\begin{aligned}
\text{cov} = 0 &= (\theta'_{k,i} - \theta_{k,i})^{-1} \int_{\theta_{k,i}}^{\theta'_{k,i}} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \nabla_{x_{k,j}} \ell(\vartheta) \, d\vartheta - \\
&(\theta'_{k,i} - \theta_{k,i})^{-1} \cdot \int_{\theta_{k,i}}^{\theta'_{k,i}} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \, d\vartheta \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) - \\
&(\theta'_{k,i} - \theta_{k,i})^{-1} \cdot \int_{\theta_{k,i}}^{\theta'_{k,i}} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) \, d\vartheta \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} + \\
&\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta)
\end{aligned} \tag{13}$$

Several terms can be expressed in terms of $\mathcal{AVG}(\cdot)$ substitutions:

$$\begin{aligned}
\text{cov} = 0 &= \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \nabla_{x_{k,j}} \ell(\vartheta) \right) - \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \\
&\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) - \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} + \\
&\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) \, d\vartheta \\
&= \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \nabla_{x_{k,j}} \ell(\vartheta) \right) - \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta)
\end{aligned} \tag{14}$$

After rearranging one of the terms to the opposite side of the equation, we finally obtain:

$$\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \nabla_{x_{k,j}} \ell(\vartheta) \right) = \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{x_{k,j}(\vartheta)} \ell(\vartheta) \tag{15}$$

which describes scalar operations only. By introducing the horizontal vector $\frac{\partial \mathbf{x}_k(\vartheta)}{\partial \vartheta_{k,i}}$ in place of the scalar $\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}}$, and the vertical vector $\nabla_{x_k(\vartheta)} \ell(\vartheta)$ in place of the scalar

$\nabla_{\mathbf{x}_{k,j}(\vartheta)} \ell(\vartheta)$, we obtain:

$$\begin{aligned}
\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\vartheta_{k,i}} \ell(\vartheta) &= \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \sum_j \left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \nabla_{\mathbf{x}_{k,j}} \ell(\vartheta) \right) \\
&= \sum_j \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \left(\frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \nabla_{\mathbf{x}_{k,j}} \ell(\vartheta) \right) \\
&= \sum_j \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial x_{k,j}(\vartheta)}{\partial \vartheta_{k,i}} \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\mathbf{x}_{k,j}(\vartheta)} \ell(\vartheta) \\
&= \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k(\vartheta)}{\partial \vartheta_{k,i}} \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\mathbf{x}_k(\vartheta)} \ell(\vartheta)
\end{aligned} \tag{16}$$

where the summation runs over all differentiation paths. Since finite sums and integrals commute, the averaging operator \mathcal{AVG} may be interchanged with the sum, i.e. $\mathcal{AVG}(\sum_i f_i) = \sum_i \mathcal{AVG}(f_i)$.

Grouping the scalars into the vector $\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\vartheta_k} \ell(\vartheta) = [\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\vartheta_{k,1}} \ell(\vartheta), \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\vartheta_{k,2}} \ell(\vartheta), \dots]^T$ on the left-hand side, and the vectors $\frac{\partial \mathbf{x}_k(\vartheta)}{\partial \vartheta_{k,i}}$ into the matrix $\frac{\partial \mathbf{x}_k(\vartheta)}{\partial \vartheta_k}$ on the right-hand side yields:

$$\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\vartheta_k} \ell(\vartheta) = \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k(\vartheta)}{\partial \vartheta_k} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\mathbf{x}_k(\vartheta)} \ell(\vartheta) \tag{17}$$

Using the assumption that correlations between values of different Jacobians and gradients can be neglected, we write this in a form similar to Eq. 11:

$$\forall(k, i, j), \text{ cov}\left(\frac{\partial \mathbf{x}_{k+1,j}(\vartheta)}{\partial \mathbf{x}_{k,i}(\vartheta)}, \nabla_{\mathbf{x}_{k+1,j}(\vartheta)} \ell(\vartheta)\right) = 0 \tag{18}$$

We can apply analogous reasoning as presented for Eqs. 11 to 17, leading to:

$$\mathcal{AVG}_{\vartheta \in [\theta, \theta']} \left(\frac{\partial \mathbf{x}_{k+1}(\vartheta)}{\partial \mathbf{x}_k(\vartheta)} \nabla_{\mathbf{x}_{k+1}} \ell(\vartheta) \right) = \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_{k+1}(\vartheta)}{\partial \mathbf{x}_k(\vartheta)} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\mathbf{x}_{k+1}(\vartheta)} \ell(\vartheta) \tag{19}$$

Then, by applying Eq. 19 ($n - k$) times, this leads to:

$$\begin{aligned}
& \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \left(\frac{\partial \mathbf{x}_{k+1}(\vartheta)}{\partial \mathbf{x}_k(\vartheta)} \cdot \dots \cdot \frac{\partial \mathbf{x}_n(\vartheta)}{\partial \mathbf{x}_{n-1}(\vartheta)} \nabla_{\mathbf{x}_n} \ell(\vartheta) \right) \\
&= \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_{k+1}(\vartheta)}{\partial \mathbf{x}_k(\vartheta)} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \left(\frac{\partial \mathbf{x}_{k+2}(\vartheta)}{\partial \mathbf{x}_{k+1}(\vartheta)} \cdot \dots \cdot \frac{\partial \mathbf{x}_n(\vartheta)}{\partial \mathbf{x}_{n-1}(\vartheta)} \nabla_{\mathbf{x}_n} \ell(\vartheta) \right) \\
&= \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_{k+1}(\vartheta)}{\partial \mathbf{x}_k(\vartheta)} \cdot \dots \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_n(\vartheta)}{\partial \mathbf{x}_{n-1}(\vartheta)} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\mathbf{x}_n} \ell(\vartheta)
\end{aligned} \tag{20}$$

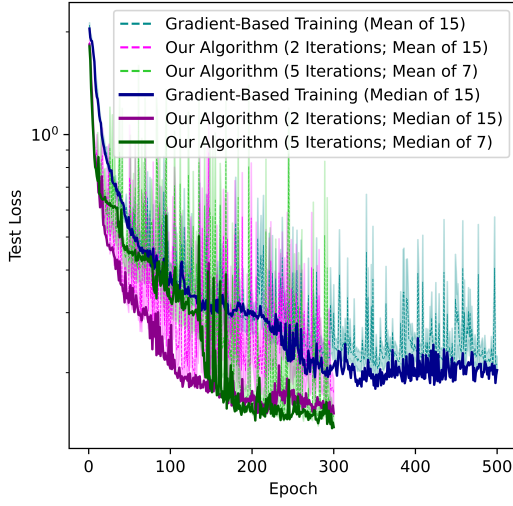
since Eq. 19 holds for all k , as indicated in Eq. 18. The symbol \cdot represents matrix multiplication.

Using Eq. 17 together with Eq. 20, we obtain:

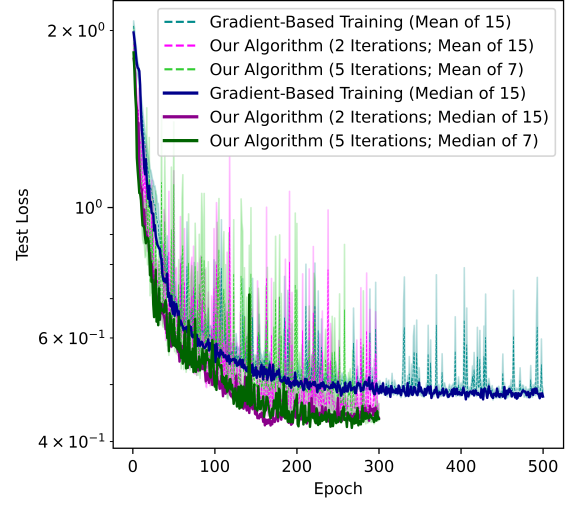
$$\begin{aligned}
& \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\vartheta_k} \ell(\vartheta) = \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_k(\vartheta)}{\partial \vartheta_k} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_{k+1}(\vartheta)}{\partial \mathbf{x}_k(\vartheta)} \cdot \\
& \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_{k+2}(\vartheta)}{\partial \mathbf{x}_{k+1}(\vartheta)} \cdot \dots \cdot \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \frac{\partial \mathbf{x}_n(\vartheta)}{\partial \mathbf{x}_{n-1}(\vartheta)} \mathcal{AVG}_{\vartheta \in [\theta, \theta']} \nabla_{\mathbf{x}_n(\vartheta)} \ell(\vartheta)
\end{aligned} \tag{21}$$

which matches Eq. 1. However, in the general case, our assumption about correlations leads to an approximation rather than equality in Eq. 1. At the start of the proof, we discuss why this assumption yields an acceptable approximation by referring to Vyas et al. (2024), Gupta et al. (2018) and Martens and Grosse (2015). \square

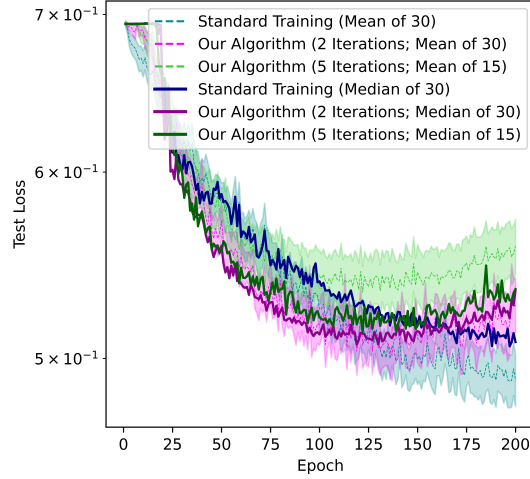
C Test Loss Curves



(a) *Model B on MNIST.*



(b) *Model B on Fashion MNIST.*



(c) *Model C on IMDB.*

Figure 4: *Test losses* of the training runs depicted in Fig. 3. Only mean curves contain confidence ranges (SEM).