Revisiting Scalable Hessian Diagonal Approximations for Applications in Reinforcement Learning

Mohamed Elsayed [1 2] **Homayoon Farrahi** [1 2] **Felix Dangel** [3] **A. Rupam Mahmood** [1 2 4]

Abstract

Second-order information is valuable for many applications but challenging to compute. Several works focus on computing or approximating Hessian diagonals, but even this simplification introduces significant additional costs compared to computing a gradient. In the absence of efficient exact computation schemes for Hessian diagonals, we revisit an early approximation scheme proposed by Becker and LeCun (1989, *BL89*), which has a cost similar to gradients and appears to have been overlooked by the community. We introduce *HesScale*, an improvement over BL89, which adds negligible extra computation. On small networks, we find that this improvement is of higher quality than all alternatives, even those with theoretical guarantees, such as unbiasedness, while being much cheaper to compute. We use this insight in reinforcement learning problems where small networks are used and demonstrate HesScale in second-order optimization and scaling the step-size parameter. In our experiments, HesScale optimizes faster than existing methods and improves stability through step-size scaling. These findings are promising for scaling second-order methods in larger models in the future.[1]

1. Introduction

Second-order information—the entries of the Hessian matrix—is paramount in a wide spectrum of applications, including preconditioning in optimization (Martens and Grosse 2015, Yao et al. 2021, Shen et al. 2024), and estimating the importance of weights or neurons (Elsayed and Mahmood 2024) in pruning (LeCun et al. 1989, Hassibi and Stork 1992, Singh and Alistarh 2020). However, computing the Hessian entries is generally expensive, preventing its usage in systems that require small memory and computation. This is reflected in the minimal adoption of second-order methods compared to their first-order counterparts.

Many second-order methods rely on some approximation of the Hessian entries to make computation less prohibitive. For example, a type of truncated-Newton method called Hessian-free methods (Martens et al. 2010) exploits the Hessian-vector product (Pearlmutter 1994, Bekas et al. 2007). However, such methods might require many iterations per update or additional techniques to achieve stability when used in optimization, adding computational overhead (Martens and Sutskever 2011). Some variations only approximate the diagonals of the Hessian matrix using stochastic estimation with matrix-free computations (e.g., Chapelle et al. 2011, Martens et al. 2012, Yao et al. 2021), but they may produce low-quality approximations of the Hessian entries (Jahani et al. 2021). Other methods impose probabilistic modeling assumptions and estimate a block diagonal Fisher information matrix (Martens and Grosse 2015, Botev et al. 2017), but they are more expensive to compute.

A long-overlooked approach is deterministic diagonal approximations to the Hessian. Specifically, the method proposed by Becker and LeCun (1989), which we call *BL89*, can be implemented as efficiently as first-order methods. We call this method, along with other cheap methods (e.g., Yao et al. 2021) with the same computational and memory complexities as the gradient, *scalable second-order methods*, distinguishing them from methods with superlinear computational or memory complexity (e.g., Mizutani and Dreyfus 2008, Botev et al. 2017, Dangel et al. 2020a). Despite the promise of scalable second-order optimization, the approximation quality of BL89 is shown to be poor (Hassibi and Stork 1992, Martens et al. 2012). A scalable second-order method with a high-quality approximation is still needed.

Scalable second-order methods can benefit reinforcement learning (RL) in at least three ways: sample efficiency, stability, and robustness. Second-order optimization can help with sample efficiency due to faster convergence (Wu et al. 2017). Moreover, step-size scaling schemes (e.g., Martens

[1]Department of Computing Science, University of Alberta, Edmonton, Canada [2]Alberta Machine Intelligence Institute [3]Vector Institute, Toronto, Canada [4]CIFAR AI Chair. Correspondence to: Mohamed Elsayed <mohamedelsayed@ualberta.ca>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]Code is available at: `https://github.com/mohmdelsayed/HesScale`

and Grosse 2015) use Hessian approximations to improve the stability of reinforcement learning methods that can suffer from instability due to large updates (Dohare et al. 2023). Another benefit of step-size scaling is achieving robustness against step-size sensitivity, allowing for tuning-free optimization (Mahmood et al. 2012), which is crucial for learning with real-life robots (Mahmood et al. 2018).

In this paper, we introduce a refinement of BL89, which we call *HesScale*. Similarly to BL89, our method is scalable, with small computational and memory overhead, but maintains higher approximation accuracy than BL89 and many other methods for Hessian diagonal approximation. We provide two applications of HesScale: second-order optimization and step-size scaling. In supervised learning and RL tasks, we demonstrate that second-order optimization with HesScale achieves better sample efficiency compared to existing scalable second-order methods. In RL tasks, we show that step-size scaling with HesScale improves both robustness and stability of the base learning method.

2. Background

Here, we describe the Hessian matrix for neural networks and some existing methods for estimating it. Generally, Hessian matrices can be computed for any scalar-valued function that is twice differentiable. If $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is such a function, then for its argument $\boldsymbol{\psi} \in \mathbb{R}^n$, the Hessian matrix $\boldsymbol{H} \in \mathbb{R}^{n \times n}$ of f with respect to $\boldsymbol{\psi}$ is given by $H_{i,j} = \partial^2 f(\boldsymbol{\psi}) / \partial \psi_i \partial \psi_j$. Here, element i of vector $\boldsymbol{v}$ is denoted by v_i, and element (i, j) of matrix $\boldsymbol{M}$ is denoted by $M_{i,j}$. For optimization in deep learning, the function f is typically the objective function, and the vector $\boldsymbol{\psi}$ is commonly the weight vector of a neural network. Computing and storing an $n \times n$ matrix, where n is the number of weights in a neural network, is prohibitively expensive. Therefore, many methods exist for approximating the Hessian matrix or parts of it with less memory footprint, computational requirement, or both. A common technique is to utilize the structure of the function to reduce the computations needed. For example, some approximate a layer-wise block diagonal Hessian. The computation further simplifies when activation functions are assumed to be piece-wise linear. This assumption results in the *Generalized Gauss-Newton* (GGN, Schraudolph 2002) approximation. However, computing and storing the GGN matrix or its block diagonals is still too demanding.

Many approximation methods were developed to reduce the storage and computation requirements of the block-diagonal GGN matrix. For example, under probabilistic modeling assumptions, the *Kronecker-factored Approximate Curvature* (KFAC, Martens and Grosse 2015) method writes each GGN's diagonal block $\boldsymbol{G}$ as a Kronecker product of two matrices of smaller sizes as: $\boldsymbol{G} \approx \boldsymbol{A} \otimes \boldsymbol{B}$, where $\boldsymbol{A} = \mathbb{E}[\boldsymbol{h}\boldsymbol{h}^\top]$, $\boldsymbol{B} = \mathbb{E}[\boldsymbol{g}\boldsymbol{g}^\top]$, $\boldsymbol{h}$ is the activation vector, and $\boldsymbol{g}$ is the gradient of the loss with respect to the pre-activation vector. The $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices can be estimated by Monte Carlo estimation. KFAC is also more efficient when used in optimization than other methods approximating GGN block diagonals since it requires inverting only the small matrices using the Kronecker product property $(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$. Despite KFAC's gains in efficiency, it is still costly since storing the Kronecker matrices can become prohibitively expensive for large-scale problems. Additionally, its Kronecker structure introduces approximation errors. Alternative approaches that achieve high accuracy exploit the GGN's outer-product structure (Dangel et al. 2022, Yang et al. 2022, Dangel et al. 2020b), but they suffer from a superlinear scaling in the network's output dimension in both memory and compute, which limits their ability to scale.

Restricting calculations to Hessian diagonals provides some curvature information with relatively little computation. However, it has been shown that the exact computation for diagonals of the Hessian typically has quadratic complexity with the unlikely existence of algorithms that can compute the exact diagonals with less than quadratic complexity (Martens et al. 2012). Some stochastic methods provide a way to compute unbiased estimates of the exact Hessian diagonals. For example, the AdaHessian (Yao et al. 2021) algorithm uses Hutchinson's estimator $\text{diag}(\boldsymbol{H}) = \mathbb{E}[\boldsymbol{z} \circ (\boldsymbol{H}\boldsymbol{z})]$, where $\boldsymbol{z}$ is a multivariate random variable with a Rademacher distribution and the expectation can be estimated using Monte Carlo estimation. Similarly, the GGN-MC method (Dangel et al. 2020b) uses the relationship between the Fisher information matrix and the Hessian matrix under probabilistic modeling assumptions to have an MC approximation of the diagonal of the GGN matrix. Although these stochastic approximation methods are scalable, that is, with linear computational and memory complexity in the number of parameters and network outputs, they suffer from low approximation quality (see Fig. 2), improving which requires many sampling and factors of additional computations.

3. The Proposed HesScale Method

We present our method for approximating the diagonal of the Hessian at each layer in feed-forward networks, where a backpropagation rule is used to utilize the Hessian of previous layers. Here, we derive the backpropagation rule for fully connected networks. A similar derivation for fully connected networks with the mean squared error was presented for BL89 (Becker and LeCun 1989). However, ours is a refinement of BL89 in that we use the exact diagonals of the Hessian matrix at the last layer. We show that the computational complexity can still be linear in the network's output dimension for some common loss functions. We defer the derivation of Hessian diagonals for the convolutional neural networks to Appendix F.

We use the non-convex stochastic optimization setting where there is an objective we need to minimize. The stochastic objective given a sample S is denoted by $\mathcal{L}(S, \mathcal{W})$, where S is a random variable that can be the input-output pair in supervised learning or a transition tuple in reinforcement learning, and $\mathcal{W}$ is a set of learnable parameters. The *learner* maintains a single or multiple neural network and is required to optimize the objective by changing these network parameters. Specifically, the learner is required to minimize the objective $\mathbb{E}_S(\mathcal{L}(S, \mathcal{W}))$.

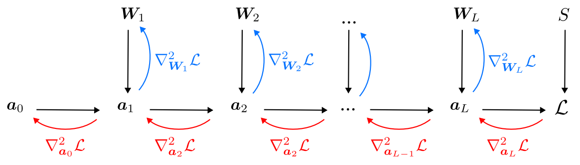

$\boldsymbol{W}_1$
$\boldsymbol{W}_2$
...
$\boldsymbol{W}_L$
S
$\nabla^2_{\boldsymbol{W}_1}\mathcal{L}$
$\nabla^2_{\boldsymbol{W}_2}\mathcal{L}$
$\nabla^2_{\boldsymbol{W}_L}\mathcal{L}$
$\boldsymbol{a}_0$
$\boldsymbol{a}_1$
$\boldsymbol{a}_2$
...
$\boldsymbol{a}_L$
$\mathcal{L}$
$\nabla^2_{\boldsymbol{a}_0}\mathcal{L}$
$\nabla^2_{\boldsymbol{a}_2}\mathcal{L}$
$\nabla^2_{\boldsymbol{a}_2}\mathcal{L}$
$\nabla^2_{\boldsymbol{a}_{L-1}}\mathcal{L}$
$\nabla^2_{\boldsymbol{a}_L}\mathcal{L}$

Figure 1. Backpropagating the exact Hessian information in a neural network. Red arrows represent the direction of influence while backpropagating the Hessian of the loss w.r.t. pre-activations which are then used to compute the Hessian of the loss w.r.t. the weights at each layer, denoted by the blue arrows. Black arrows denote the direction of influence during the forward pass.

Consider a neural network with L layers parametrized by the set of weights $\mathcal{W} = \{\boldsymbol{W}_1, ..., \boldsymbol{W}_L\}$, where $\boldsymbol{W}_l$ is the weight matrix at the l-th layer, and its element at the ith row and the jth column is denoted by $W_{l,i,j}$. During learning, the parameters of the neural network are changed to reduce the loss. During a forward pass, we get the activation $\boldsymbol{h}_l$ at layer l by applying the activation function $\boldsymbol{\sigma}$ to the pre-activation $\boldsymbol{a}_l$: $\boldsymbol{h}_l = \boldsymbol{\sigma}(\boldsymbol{a}_l)$. We simplify notations by defining $\boldsymbol{h}_0 \doteq \boldsymbol{x}$. The activation $\boldsymbol{h}_l$ is then multiplied by the weight matrix $\boldsymbol{W}_{l+1}$ of layer $l+1$ to produce the next pre-activation: $a_{l+1,i} = \sum_{j=1}^{|\boldsymbol{h}_l|} W_{l+1,i,j} h_{l,j}$.[2] We assume here that the activation function is element-wise activation for all layers except for the final layer L, where it becomes the softmax function. The backpropagation equations for the described network are given as (Rumelhart et al. 1986):

$$\frac{\partial \mathcal{L}}{\partial W_{l,i,j}} = \frac{\partial \mathcal{L}}{\partial a_{l,i}} \frac{\partial a_{l,i}}{\partial W_{l,i,j}} = \frac{\partial \mathcal{L}}{\partial a_{l,i}} h_{l-1,j}, \tag{1}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial a_{l,i}} &= \sum_{k=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial \mathcal{L}}{\partial a_{l+1,k}} \frac{\partial a_{l+1,k}}{\partial h_{l,i}} \frac{\partial h_{l,i}}{\partial a_{l,i}} \\ &= \sigma'(a_{l,i}) \sum_{k=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial \mathcal{L}}{\partial a_{l+1,k}} W_{l+1,k,i}. \end{aligned} \tag{2}$$

In the following, we write the equations for the exact Hessian diagonals with respect to weights $\partial^2\mathcal{L}/\partial W^2_{l,i,j}$, which requires the calculation of $\partial^2\mathcal{L}/\partial a^2_{l,i}$. Fig. 1 shows the general computational graph used to backpropagate the second-order information (Mizutani and Dreyfus 2008, Dangel et al. 2020a), but we only focus next on the diagonal part:

$$\begin{aligned} \frac{\partial^2 \mathcal{L}}{\partial W^2_{l,i,j}} &= \frac{\partial}{\partial W_{l,i,j}} \left(\frac{\partial \mathcal{L}}{\partial a_{l,i}} h_{l-1,j} \right) \\ &= \frac{\partial}{\partial a_{l,i}} \left(\frac{\partial \mathcal{L}}{\partial a_{l,i}} \right) \frac{\partial a_{l,i}}{\partial W_{l,i,j}} h_{l-1,j} \\ &= \frac{\partial^2 \mathcal{L}}{\partial a^2_{l,i}} h^2_{l-1,j}, \end{aligned}$$

$$\begin{aligned} \frac{\partial^2 \mathcal{L}}{\partial a^2_{l,i}} &= \frac{\partial}{\partial a_{l,i}} \left(\sigma'(a_{l,i}) \sum_{k=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial \mathcal{L}}{\partial a_{l+1,k}} W_{l+1,k,i} \right) \\ &= \sigma'(a_{l,i}) \sum_{k,p=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial^2 \mathcal{L}}{\partial a_{l+1,k} \partial a_{l+1,p}} \frac{\partial a_{l+1,p}}{\partial a_{l,i}} W_{l+1,k,i} \\ &\quad + \sigma''(a_{l,i}) \sum_{k=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial \mathcal{L}}{\partial a_{l+1,k}} W_{l+1,k,i} \\ &= \sigma'(a_{l,i})^2 \sum_{k,p=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial^2 \mathcal{L}}{\partial a_{l+1,k} \partial a_{l+1,p}} W_{l+1,p,i} W_{l+1,k,i} \\ &\quad + \sigma''(a_{l,i}) \sum_{k=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial \mathcal{L}}{\partial a_{l+1,k}} W_{l+1,k,i}. \end{aligned}$$

Since the calculation of $\partial^2\mathcal{L}/\partial a^2_{l,i}$ depends on the off-diagonal terms, the computation complexity becomes quadratic in the layer's width. Following BL89, we approximate the Hessian diagonals by ignoring the off-diagonal terms, leading to a backpropagation rule with linear computational complexity for our estimates $\widehat{\frac{\partial^2 \mathcal{L}}{\partial W^2_{l,i,j}}}$ and $\widehat{\frac{\partial^2 \mathcal{L}}{\partial a^2_{l,i}}}$:

$$\widehat{\frac{\partial^2 \mathcal{L}}{\partial W^2_{l,i,j}}} \doteq \widehat{\frac{\partial^2 \mathcal{L}}{\partial a^2_{l,i}}} h^2_{l-1,j}, \tag{3}$$

$$\begin{aligned} \widehat{\frac{\partial^2 \mathcal{L}}{\partial a^2_{l,i}}} &\doteq \sigma'(a_{l,i})^2 \sum_{k=1}^{|\boldsymbol{a}_{l+1}|} \widehat{\frac{\partial^2 \mathcal{L}}{\partial a^2_{l+1,k}}} W^2_{l+1,k,i} \\ &\quad + \sigma''(a_{l,i}) \sum_{k=1}^{|\boldsymbol{a}_{l+1}|} \frac{\partial \mathcal{L}}{\partial a_{l+1,k}} W_{l+1,k,i}. \end{aligned} \tag{4}$$

For the last layer, we use the exact Hessian diagonals $\widehat{\frac{\partial^2 \mathcal{L}}{\partial a^2_{L,i}}} \doteq \frac{\partial^2 \mathcal{L}}{\partial a^2_{L,i}}$ since it can be computed cheaply for some common loss functions. For example, the exact Hessian diagonals for cross-entropy loss with softmax is simply $\boldsymbol{q} - \boldsymbol{q} \circ \boldsymbol{q}$, where $\boldsymbol{q}$ is the predicted probability vector and $\circ$ denotes element-wise multiplication. We show this property with derivations for negative log-likelihood function with

[2]The bias term can be added by appending an additional column to a weight matrix and a 1 to each layer's input vector.

Gaussian and softmax distributions in Appendix B and for other RL-related loss functions in Appendix C.

We found empirically that this small change makes a large difference in the approximation quality, as shown in Fig. 2(a). Hence, unlike BL89, which uses a Hessian diagonal approximation of the last layer by Eq. 4, we use the exact values directly to achieve more approximation accuracy. We call this method for Hessian diagonal approximation *HesScale* and provide its pseudocode in Algorithm 1. Finally, we provide in Appendix E additional analysis on the special cases that make HesScale exact.

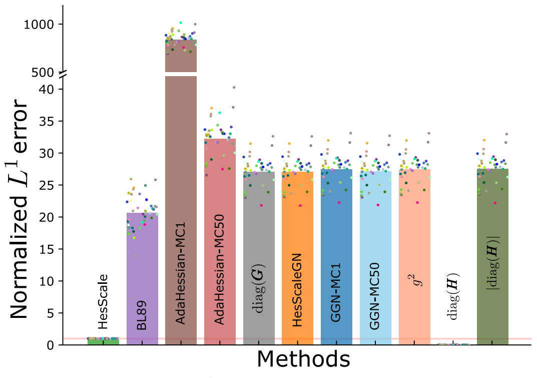
Algorithm 1 HesScale

```
Require: Neural network $f$ and a layer number $l$
Require: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}_{l+1}}$ and $\widehat{\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{a}_{l+1}^2}}$, unless $l = L$
Require: Loss function $\mathcal{L}$
if $l = L$ then
    Compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}_L}$ and $\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{a}_L^2}$
    Compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_L}$ using Eq. 1
    $\widehat{\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{a}_L^2}} \leftarrow \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{a}_L^2}$
    Compute $\widehat{\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{W}_L^2}}$ using Eq. 3
else if $l \neq L$ then
    Compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}_l}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_l}$ using Eq. 2 and Eq. 1
    Compute $\widehat{\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{a}_l^2}}$ and $\widehat{\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{W}_l^2}}$ using Eq. 4 and Eq. 3
end if
Return $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_l}$, $\widehat{\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{W}_l^2}}$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}_l}$, and $\widehat{\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{a}_l^2}}$
```

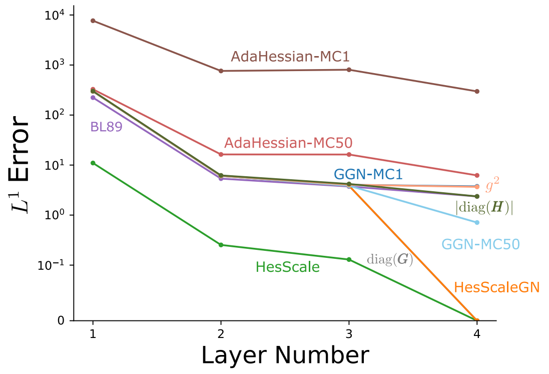
The computation can be reduced further by dropping the last term in Eq. 4, which corresponds to the Gauss-Newton approximation and is justified for piece-wise linear activation functions. We call this variation *HesScaleGN*.

4. Approximation Quality

We evaluate HesScale's approximation quality and compare it with other methods. We start by studying the approximation quality of Hessian diagonals compared to the true values. In our experiments, we implemented HesScale using the *BackPACK* framework (Dangel et al. 2020b), which allows easy implementation of backpropagation of statistics other than the gradient. To measure the approximation quality of the Hessian diagonals for different methods, we use the L^1 distance between the exact Hessian diagonals and their approximations, where $L^1(\mathbf{a}, \mathbf{b}) = \sum_i |a_i - b_i|$. Our task here is supervised classification, and data examples are sampled randomly from MNIST. We used a network of three hidden layers with *tanh* activations, each containing 32 units. The network weights and biases are initialized using Kaiming initialization (He et al. 2015). We trained the network with SGD using a batch size of 1. For each example pair, we compute the exact Hessian diagonals for each layer and their approximations from each method. All layers' errors are summed and then averaged over 1000 data examples for each method. In this experiment, we used 40 different initializations for the network weights, shown as colored dots in Fig. 2(a). In this figure, we show the average error incurred by each method normalized by the average error incurred by HesScale. Any approximation that incurs an averaged error above 1 has a worse approximation than HesScale, and any approximation with an error less than 1 has a better approximation than HesScale. Moreover, we show the layer-wise error for each method in Fig. 2(b).



(a) Normalized L^1 error with respect to HesScale



(b) Layer-wise L^1 error

Figure 2. (a) The average error for each method is normalized by the average error incurred by HesScale. Each colored point represents a different initialization. The norm of the vector of Hessian diagonals $|\operatorname{diag}(\boldsymbol{H})|$ is shown as a reference. (b) The average layer-wise error for each method is shown. HesScale(GN) has zero error at the last layer since it uses the exact entries there.

Different Hessian diagonal approximations are considered for comparison with HesScale. We included several deterministic and stochastic approximations for the Hessian diagonals. We also included the sample estimate of the Fisher Information Matrix done by squaring the gradients and denoted by g^2, which is adopted by many first-order methods (e.g., Kingma and Ba 2014). We compare HesScale with two stochastic approximation methods: AdaHessian

(Yao et al. 2021), and the Monte-Carlo (MC) estimate of the GGN matrix (GGN-MC, Dangel et al. 2020b). We also compare HesScale with two deterministic approximation methods: the diagonals of the exact GGN matrix (Schraudolph 2002) ($\mathrm{diag}(\boldsymbol{G})$) and the diagonal approximation by Becker and LeCun (1989) (BL89). Since AdaHessian and GGN-MC are already diagonal approximations, we use them directly and show the error with 1 MC sample (GGN-MC1 & AdaHessian-MC1) and with 50 MC samples (GGN-MC50 & AdaHessian-MC50).

HesScale provides a better approximation than the other deterministic and stochastic methods. For stochastic methods, we use many MC samples to improve their approximation. However, their approximation quality is still poor. Methods approximating the GGN diagonals do not capture the complete Hessian information since the GGN and Hessian matrices are different when the activation functions are not piece-wise linear, as is the case for our tanh-activated network. Although these methods approximate the GGN diagonals, their approximation is significantly better than the AdaHessian approximation. Among the methods for approximating the GGN diagonals, HesScaleGN approximates the exact GGN diagonals closely. This experiment clearly shows that HesScale achieves the best approximation quality (overall and across layers) compared to other stochastic and deterministic approximation methods.

4.1. Diagonality of Hessians w.r.t. Pre-activations

To understand why HesScale gives such a high approximation quality, we investigate the structure of the matrices propagated in the full Hessian backpropagation. The high approximation quality suggests that those matrices are diagonally dominant; thus, dropping the off-diagonal elements does not significantly harm the approximation. Similar to the metric in Balles et al. (2020), we introduce the metric $\rho \in [0, 1]$ that measures the diagonal dominance of a matrix $\boldsymbol{A}$ given by

$$\rho(\boldsymbol{A}) = \frac{\|\mathrm{diag}(\boldsymbol{A})\|_{\mathrm{F}}}{\|\boldsymbol{A}\|_{\mathrm{F}}},$$

where $\|.\|_{\mathrm{F}}$ is the Frobenius norm and $\mathrm{diag}(.)$ extracts the diagonal of a matrix. The metric ρ gives a value of 1 when $\boldsymbol{A}$ is a diagonal matrix and a value of 0 when $\boldsymbol{A}$ is a hollow matrix. We use ρ to to measure the diagonal dominance of $\{\nabla^2_{\boldsymbol{a}_i}\mathcal{L}\}_{i=1}^{L}$ (shown in Fig. 3) on an MLP of four hidden layers each containing 128 units. In Table 1, we show the diagonal dominance of those matrices before and after training. As a reference, we compute the diagonal dominance for a random matrix with standard Gaussian entries of the same size. We found that ρ of a random matrix is 0.09. The diagonal dominance metric for each matrix is averaged over 300 independent runs. We trained for 10000 iterations on EMNIST (Cohen et al. 2017) with a batch size of 32. Table 1 shows that $\{\nabla^2_{\boldsymbol{a}_i}\mathcal{L}\}_{i=1}^{L}$ are diagonally dominant before and after training, which explains the high approximation quality of HesScale compared to other approximations.

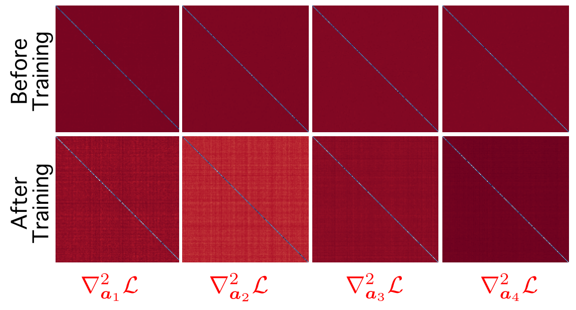


Figure 3. Heat maps of Hessian of the loss w.r.t. pre-activations. $\{\nabla^2_{\boldsymbol{a}_i}\mathcal{L}\}_{i=1}^{4}$ visually appear diagonally dominant. Red represents a small magnitude, and blue represents a large magnitude.

Table 1. Diagonal dominance before and after training

Layer Number	ρ (before training)	ρ (after training)
1	0.94	0.75
2	0.91	0.54
3	0.89	0.81
4	0.91	0.98

5. Two applications of HesScale

We utilize HesScale in two applications of second-order information: optimization and step-size scaling.

5.1. Second-order Optimization

We introduce an efficient optimizer based on HesScale, which we call *AdaHesScale* (Algorithm 2). We follow the same style introduced in Adam (Kingma and Ba 2014) in using the squared diagonal approximation instead of the squared gradients to update the moving average. Moreover, we introduce another optimizer based on HesScaleGN, which we call *AdaHesScaleGN*. For the convergence proof of methods with Hessian diagonals, we refer the reader to Appendix A. In addition, we provide a scalability comparison for AdaHesScale and AdaHesScaleGN against other optimization methods in Appendix D.

5.2. Step-size Scaling for Robustness and Stability

Scaling the step size has been a well-known approach for improving the robustness of supervised learning (e.g., Mahmood et al. 2012) and reinforcement learning (e.g., Dabney and Barto 2012). K-FAC (Martens and Grosse 2015) implemented a step-size scaling procedure based on trust region analysis, which was later studied with Adam (Clarke et al. 2023). Such a method has been combined with some RL

Algorithm 2 AdaHesScale

```
Require: Neural network $f$ with weights $\{W_1, ..., W_L\}$ and a dataset $\mathcal{D}$
Require: Small number $\epsilon \leftarrow 10^{-8}$
Require: Exponential decay rates $\beta_1, \beta_2 \in [0, 1)$
Require: step size $\alpha$
Require: Initialize $\{W_1, ..., W_L\}$
Initialize time step $t \leftarrow 0$.
for $l$ in $\{L, L-1, ..., 1\}$ do
    $M_l \leftarrow 0$;   $V_l \leftarrow 0$
end for
for $(x, y)$ in $\mathcal{D}$ do
    $t \leftarrow t + 1$
    $r_{L+1} \leftarrow s_{L+1} \leftarrow \emptyset$
    for $l$ in $\{L, L-1, ..., 1\}$ do
        Compute Loss $\mathcal{L}(x, y)$
        $F_l, S_l, r_l, s_l \leftarrow \texttt{HesScale}(\mathcal{L}, l, r_{l+1}, s_{l+1})$
        $M_l \leftarrow \beta_1 M_l + (1 - \beta_1) F_l$
        $V_l \leftarrow \beta_2 V_l + (1 - \beta_2) S_l^2$
        $\hat{M}_l \leftarrow M_l / (1 - \beta_1^t)$
        $\hat{V}_l \leftarrow V_l / (1 - \beta_2^t)$
        $W_l \leftarrow W_l - \alpha \hat{M}_l \oslash \left(\sqrt{\hat{V}_l} + \epsilon\right)$
    end for
end for
```

Algorithm 3 AdaHesScale with step-size scaling

```
Require: Neural network $f$ with weights $\{W_1, ..., W_L\}$ and a dataset $\mathcal{D}$
Require: Small number $\epsilon \leftarrow 10^{-8}$
Require: Exponential decay rates $\beta_1, \beta_2 \in [0, 1)$
Require: step size $\alpha$, trust-region radius $\Delta$
Require: Initialize $\{W_1, ..., W_L\}$
Initialize time step $t \leftarrow 0$.
for $l$ in $\{L, L-1, ..., 1\}$ do
    $M_l \leftarrow 0$;   $V_l \leftarrow 0$;   $U_l \leftarrow 0$
end for
for $(x, y)$ in $\mathcal{D}$ do
    $t \leftarrow t + 1$
    $h \leftarrow 0$
    $r_{L+1} \leftarrow s_{L+1} \leftarrow \emptyset$
    for $l$ in $\{L, L-1, ..., 1\}$ do
        Compute Loss $\mathcal{L}(x, y)$
        $F_l, S_l, r_l, s_l \leftarrow \texttt{HesScale}(\mathcal{L}, l, r_{l+1}, s_{l+1})$
        $M_l \leftarrow \beta_1 M_l + (1 - \beta_1) F_l$
        $V_l \leftarrow \beta_2 V_l + (1 - \beta_2) S_l^2$
        $\hat{M}_l \leftarrow M_l / (1 - \beta_1^t)$
        $\hat{V}_l \leftarrow V_l / (1 - \beta_2^t)$
        $U_l \leftarrow \alpha \hat{M}_l \oslash \left(\sqrt{\hat{V}_l} + \epsilon\right)$
        $S \leftarrow \sqrt{\hat{V}_l} \circ U_l^2$
        $h \leftarrow h + \mathbf{1}^\top S \mathbf{1}$
    end for
    $\eta \leftarrow \min\left(1, \sqrt{\frac{2\Delta}{h}}\right)$
    for $l$ in $\{L, L-1, ..., 1\}$ do
        $W_l \leftarrow W_l - \eta U_l$
    end for
end for
```

algorithms, giving algorithms such as ACKTR (Wu et al. 2017). The step-size mechanism scales the step size down when the update to be applied becomes outside the trust region radius, making the optimizer less sensitive to the choice of step size and, therefore, improving robustness. More recently, Dohare et al. (2023) demonstrated the problem of extreme instability in deep RL methods when trained for extended periods. Such a problem could be mitigated if the optimizer becomes aware of the size of the update it makes, which is the quantity measured by those step-size scaling methods. Thus, we developed a step-size scaling method following the mechanism outlined in K-FAC and based on our HesScale approximation. The K-FAC mechanism scales down the step size by $\min\left(\alpha_{\max}, \sqrt{\frac{2\Delta}{u^\top H u}}\right)$, where $\alpha_{\max}$ is the maximum step size, Δ is the trust-region radius, and u is the regular update if no scaling is performed. Algorithm 3 shows our step-size scaling applied to AdaHesScale using a HesScale approximation of H.

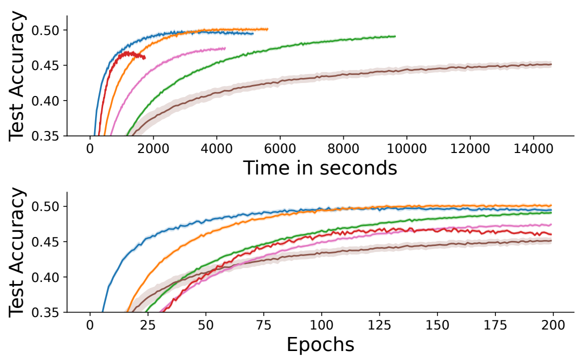
6. Supervised Learning Experiments

We compare the performance of our optimizers—AdaHesScale and AdaHesScaleGN—with three second-order optimizers: BL89, GGNMC, and AdaHessian. We also include comparisons with two first-order methods: Adam and SGD. We exclude K-FAC from our comparisons due to its relatively high cost.

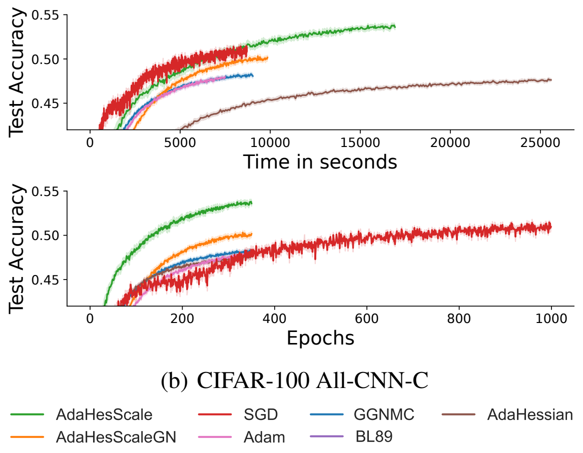
Our optimizers are evaluated in the supervised setting with two experiments using different architectures on the CIFAR-100 dataset. Instead of attempting to achieve state-of-the-art performance with specialized techniques and architectures, we followed the DeepOBS benchmarking work (Schneider et al. 2019) and compared the optimizers in their generic form using relatively simple networks to verify the validity of our method.

In the first experiment, we used the CIFAR-100 3C-3D task from DeepOBS. The network consists of three convolutional layers with *ReLU* activations, each followed by max pooling. After that, two fully connected layers (512 and 256 units) with *ReLU* activations are used. We use *ELU* instead of *ReLU*, which is used in DeepOBS, to differentiate between the performance of AdaHesScale and AdaHesScaleGN. We train each method for 200 epochs with a batch size of 128. In the second experiment, we use the CIFAR-100 ALL-CNN task from DeepOBS with the ALL-CNN-C network, which consists of 9 convolutional layers (Springenberg et al.

2014) with *ReLU* activations. Again, we use *ELU* instead of *ReLU*, which is used in DeepOBS, to differentiate between the performance of AdaHesScale and AdaHesScaleGN. We train each method for 350 epochs with a batch size of 256.



(a) CIFAR-100 3C3D



(b) CIFAR-100 All-CNN-C

Figure 4. CIFAR-100 3C3D and CIFAR-100 ALL-CNN classification tasks. (top) We show the time taken by each algorithm in seconds, and (bottom) we show the learning curves in the number of epochs. The shaded area represents the standard error. BL89 achieves lower than 0.35 and is not visible.

Fig. 4(a) and Fig. 4(b) show the results on CIFAR-100 ALL-CNN and CIFAR-100 3C3D tasks against the number of epochs and against time in seconds. In both experiments, we used $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for all adaptive methods. We run SGD for the time that matches the maximum time taken by any optimizer to have a fair comparison since it requires a small cost relative to other methods (Fig. 4(b)) unless the performance starts to go down within the predetermined number of epochs in the experiment (Fig. 4(a)).

The performance of each method is averaged over 30 independent runs. Each independent run has the same initialization across all algorithms. We performed a hyperparameter search for each method to find the best step size. Using each method's best step size on the validation set, we show the performance of the method against the time in seconds needed to complete the required number of epochs, which better depicts the computational efficiency of the methods.

In both CIFAR-100 3C3D and CIFAR-100 ALL-CNN, we notice that AdaHessian performed worse than all methods except BL89. This result is aligned with AdaHessian's inability to accurately approximate the Hessian diagonals, as shown in Fig. 2. Moreover, AdaHessian required more time than all other methods. While being time-efficient, AdaHesScaleGN consistently outperformed all methods in CIFAR-100 3C3D, and it outperformed most methods except AdaHesScale in CIFAR-100 ALL-CNN. Our experiments indicate that incorporating HesScale and HesScaleGN approximations in optimization methods can be of significant performance advantage in both computation and accuracy. AdaHesScale and AdaHesScaleGN outperformed other optimizers, likely due to their accurate approximation of the diagonals of the Hessian and GGN, respectively. We provide the sensitivity analysis in Appendix G.

7. Reinforcement Learning Experiments

We investigate the performance of AdaHesScale against other optimizers when used with two reinforcement learning algorithms, A2C (Mnih et al. 2016) and PPO (Schulman et al. 2017), on the MuJoCo environments (Todorov et al. 2012). We exclude optimizers based on GGN and GGNMC since their BackPACK implementation is limited to classification and regression. Then, we investigate the effect of step-size scaling on robustness and stability with AdaHesScale and Adam. In this section's experiments, we use MLPs of two hidden layers each containing 64 units with *tanh* activations, similar to the architectures used in CleanRL (Huang et al. 2022).

7.1. Performance

We start by focusing on a performance-based comparison. Fig. 5 shows the performance of A2C/PPO with AdaHesScale against A2C/PPO with Adam and AdaHessian on five environments. We add results for five other MuJoCo environments to Appendix H. In A2C, we observe that AdaHesScale significantly boosts performance on Ant, Walker2d, HalfCheetah, and Hopper compared to Adam and AdaHessian. In PPO, AdaHesScale performs similarly to Adam and outperforms AdaHessian in most environments.

7.2. Robustness and Stability

Next, we study the effect of step-size scaling using the HesScale approximation on robustness and stability with AdaHesScale, which we call *Scaled AdaHesScale*, and Adam, which we call *Scaled Adam*. We used a trust-region radius $\Delta = 10^{-8}$ and applied the step-size scaling mechanism on both the actor and the critic networks. Fig. 6 shows the robustness of the methods for the step-size choice. For each

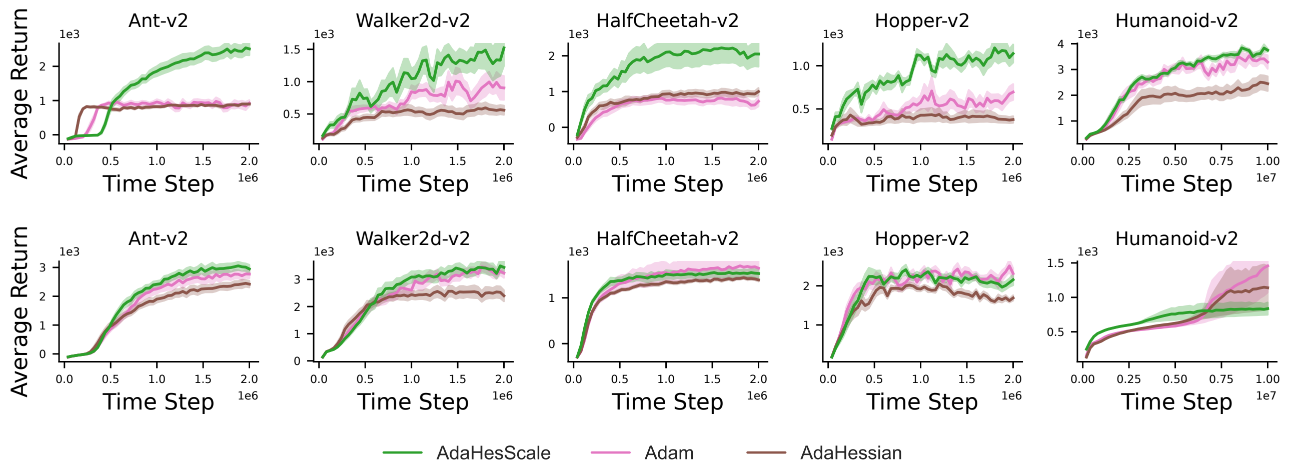


Figure 5. Performance of A2C (first row) and PPO (second row) with AdaHesScale, Adam, and AdaHessian on 5 MuJoCo environments. We show the undiscounted return averaged over 10 independent runs. The shaded area represents the standard error.

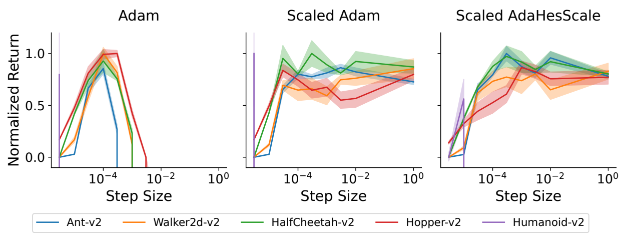


Figure 6. Robustness of HesScale-based step-size scaling with AdaHesScale and Adam on 5 MuJoCo environments. We show the undiscounted return averaged over 10 independent runs. The shaded area represents the standard error.

environment, we normalized the average return between the minimum and maximum values across both optimizers to have the same range $[0, 1]$. We observe that our step-size scaling makes both scaled optimizers insensitive to the step-size choice (compared to Adam) with PPO in all MuJoCo environments except for Humanoid, which does not benefit from the step-size scaling. We also redo the same experiment with the A2C algorithm in Appendix H.

Finally, we examine the effectiveness of step-size scaling in stability using the *UR-Reacher* real-robot task (Mahmood et al. 2018), where the goal is to move the end effector of the UR5 robotic arm in the 2d space and reach some specified point. In challenging environments, the default step size can lead to large updates, causing the policy to deteriorate with time (Dohare et al. 2023). However, the step-size scaling mechanism could mitigate this issue, which lowers the step size when the update is too big. Fig. 7 shows five runs for Adam and Scaled Adam each, where the learned policy by Adam deteriorates for many runs after reaching a maximum performance at around 80K steps, supporting the observation by Dohare et al. (2023). On the other hand, step-size scaling maintains consistent performance for Scaled Adam for all five runs, demonstrating its effectiveness in addressing the instability issue.

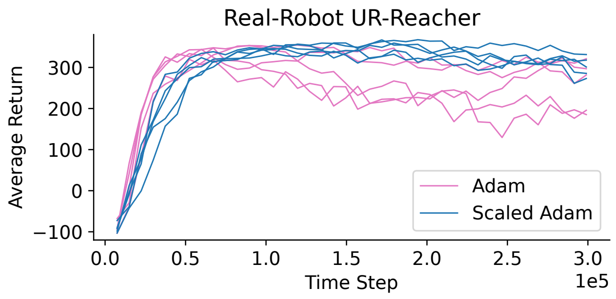


Figure 7. Performance of HesScale-based step-size scaling with Adam against standard Adam on UR-Reacher robotics task. We show the undiscounted return of five different independent runs.

8. Conclusion

HesScale is a scalable and efficient second-order method for approximating the diagonals of the Hessian at every network layer. Our results showed that our methods provide a more accurate approximation for the Hessian diagonals while requiring small additional computations. We demonstrated how HesScale can be used to build efficient second-order optimization methods for both supervised reinforcement learning. We also showed how HesScale can be used to build step-size scaling mechanisms to help improve the stability and robustness of reinforcement learning methods.

Acknowlegement

We gratefully acknowledge funding from the Canada CIFAR AI Chairs program, the Reinforcement Learning and Artificial Intelligence (RLAI) laboratory, the Alberta Machine Intelligence Institute (Amii), and the Natural Sciences and Engineering Research Council (NSERC) of Canada. We would also like to thank the Digital Research Alliance of Canada for providing the computational resources needed. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring Vector Institute.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Balles, L., Pedregosa, F., and Roux, N. L. (2020). The geometry of sign gradient descent. *arXiv preprint arXiv:2002.08056*.

Becker, S. and LeCun, Y. (1989). Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school, Published–1989*, pages 29–37.

Bekas, C., Kokiopoulou, E., and Saad, Y. (2007). An estimator for the diagonal of a matrix. *Applied numerical mathematics*, 57(11-12):1214–1229.

Botev, A., Ritter, H., and Barber, D. (2017). Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pages 557–565. PMLR.

Chan, A., Silva, H., Lim, S., Kozuno, T., Mahmood, A. R., and White, M. (2022). Greedification operators for policy optimization: Investigating forward and reverse kl divergences. *The Journal of Machine Learning Research*, 23(1):11474–11552.

Chapelle, O., Erhan, D., et al. (2011). Improved preconditioner for hessian free optimization. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, volume 201. Citeseer.

Clarke, R. M., Su, B., and Hernández-Lobato, J. M. (2023). Adam through a second-order lens. *arXiv preprint arXiv:2310.14963*.

Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. (2017). Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE.

Dabney, W. and Barto, A. (2012). Adaptive step-size for online temporal difference learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 872–878.

Dangel, F., Harmeling, S., and Hennig, P. (2020a). Modular block-diagonal curvature approximations for feedforward architectures. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Dangel, F., Kunstner, F., and Hennig, P. (2020b). BackPACK: Packing more into backprop. In *International Conference on Learning Representations (ICLR)*.

Dangel, F., Tatzel, L., and Hennig, P. (2022). ViViT: Curvature access through the generalized gauss-newton's low-rank structure. *Transactions on Machine Learning Research (TMLR)*.

Dohare, S., Lan, Q., and Mahmood, A. R. (2023). Overcoming policy collapse in deep reinforcement learning. In *Sixteenth European Workshop on Reinforcement Learning*.

Elsayed, M. and Mahmood, A. R. (2024). Addressing catastrophic forgetting and loss of plasticity in neural networks. In *The Twelfth International Conference on Learning Representations*.

Hassibi, B. and Stork, D. (1992). Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and AraÃšjo, J. G. (2022). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18.

Jahani, M., Rusakov, S., Shi, Z., Richtárik, P., Mahoney, M. W., and Takáč, M. (2021). Doubly adaptive scaled algorithm for machine learning using second-order information.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y., Denker, J., and Solla, S. (1989). Optimal brain damage. *Advances in neural information processing systems*, 2.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on robot learning*, pages 561–591. PMLR.

Mahmood, A. R., Sutton, R. S., Degris, T., and Pilarski, P. M. (2012). Tuning-free step-size adaptation. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 2121–2124. IEEE.

Martens, J. et al. (2010). Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742.

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

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1033–1040.

Martens, J., Sutskever, I., and Swersky, K. (2012). Estimating the hessian by back-propagating curvature. *arXiv preprint arXiv:1206.6464*.

Mizutani, E. and Dreyfus, S. E. (2008). Second-order stagewise backpropagation for hessian-matrix analyses and investigation of negative curvature. *Neural Networks*, 21(2-3):193–203.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Schneider, F., Balles, L., and Hennig, P. (2019). Deepobs: A deep learning optimizer benchmark suite. *arXiv preprint arXiv:1903.05499*.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shen, Y., Daheim, N., Cong, B., Nickl, P., Marconi, G. M., Bazan, C., Yokota, R., Gurevych, I., Cremers, D., Khan, M. E., et al. (2024). Variational learning is effective for large deep networks. *arXiv preprint arXiv:2402.17641*.

Singh, S. P. and Alistarh, D. (2020). Woodfisher: Efficient second-order approximation for neural network compression. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Advances in neural information processing systems*, 30.

Yang, M., Xu, D., Wen, Z., Chen, M., and Xu, P. (2022). Sketch-based empirical natural gradient methods for deep learning. *Journal of Scientific Computing*.

Yao, Z., Gholami, A., Shen, S., Mustafa, M., Keutzer, K., and Mahoney, M. (2021). Adahessian: An adaptive second order optimizer for machine learning. In *proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10665–10673.

Zaheer, M., Reddi, S., Sachan, D., Kale, S., and Kumar, S. (2018). Adaptive methods for nonconvex optimization. *Advances in neural information processing systems*, 31.

A. Convergence proof of AdaHesScale and AdaHesScaleGN

In this section, we provide a convergence proof for AdaHesScale and AdaHesScaleGN that resembles the proof given by Zaheer et al. (2018) for the convergence of adaptive optimization methods (e.g., Adam) in nonconvex problems. The following proof shows the convergence to a stationary point up to the statistical limit of the variance of the gradients, where $\|\nabla f(\boldsymbol{\theta})\|^2 \leq \delta$ represent a δ-accurate solution and is used to measure the stationarity of $\boldsymbol{\theta}$. Nonconvex optimization problems can be written as:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} f(\boldsymbol{\theta}) \doteq \mathbb{E}_{S\sim P}\left[\mathcal{L}(\boldsymbol{\theta}, S)\right]$$

where f is the expected loss, $\mathcal{L}$ is the sample loss, S is a random variable for samples and $\boldsymbol{\theta}$ is a vector of weights parametrizing $\mathcal{L}$. We assume that $\mathcal{L}$ is L-smooth, meaning that there exist a constant L that satisfy

$$\|\nabla\mathcal{L}(\boldsymbol{\theta}_1, s) - \nabla\mathcal{L}(\boldsymbol{\theta}_2, s)\| \leq L\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|,\ \forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^d, s \in \mathcal{S}. \tag{5}$$

Similar to the proof by Zaheer et al. (2018), we further assume that $\mathcal{L}$ has bounded gradients $|\nabla[\mathcal{L}(\boldsymbol{\theta}, s)]_i| \leq G, \forall i \in \mathbb{R}^d, s \in \mathcal{S}$ and bounded variance in the gradients $\mathbb{E}\|\nabla\mathcal{L}(\boldsymbol{\theta}, S) - \nabla f(\boldsymbol{\theta})\|^2 \leq \sigma^2, \forall \boldsymbol{\theta} \in \mathbb{R}^d$. Note that the assumption of L-smoothness on the sample loss result in L-smooth expected loss too, which is given by $\|\nabla f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_2)\| \leq L\|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|$.

Remember that the update rule of AdaHesScale and AdaHesScale-GN can be written as follows when the parameters are stacked in a single vector $\boldsymbol{\theta}$:

$$\theta_{t+1,i} = \theta_{t,i} - \alpha\frac{m_{t,i}}{\sqrt{v_{t,i}} + \epsilon},$$

where $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ are exponential moving averages of the first derivatives $\boldsymbol{g}_t$ and second derivatives $\boldsymbol{h}_t$, respectively. Similar to Zaheer et al. (2018), we write the proof for $\beta_1 = 0$ making $m_{t,i} = g_{t,i}$ for the sake of simplicity. However, this proof should extend to the general case.

Since the expected loss f is L-smooth, we can write the following:

$$f(\boldsymbol{\theta}_{t+1}) \leq f(\boldsymbol{\theta}_t) + (\nabla f(\boldsymbol{\theta}_t))^\top(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) + \frac{L}{2}\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \tag{6}$$

$$= f(\boldsymbol{\theta}_t) - \alpha\sum_{i=1}^d\left(\nabla[f(\boldsymbol{\theta}_t)]_i\frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon}\right) + \frac{L\alpha^2}{2}\sum_{i=1}^d\frac{g_{t,i}^2}{(\sqrt{v_{t,i}} + \epsilon)^2}. \tag{7}$$

Next, we take the conditional expectation of $f(\boldsymbol{\theta}_{t+1})$ as follows:

$$\begin{aligned}
\mathbb{E}_t[f(\boldsymbol{\theta}_{t+1})|\boldsymbol{\theta}_t] &\leq f(\boldsymbol{\theta}_t) - \alpha\sum_{i=1}^d\left([\nabla f(\boldsymbol{\theta}_t)]_i\mathbb{E}_t\left[\frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon}\right]\right) + \frac{L\alpha^2}{2}\sum_{i=1}^d\mathbb{E}_t\left[\frac{g_{t,i}^2}{(\sqrt{v_{t,i}} + \epsilon)^2}\right]\\
&= f(\boldsymbol{\theta}_t) - \alpha\sum_{i=1}^d\left([\nabla f(\boldsymbol{\theta}_t)]_i\mathbb{E}_t\left[\frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon} - \frac{g_{t,i}}{\epsilon} + \frac{g_{t,i}}{\epsilon}\right]\right) + \frac{L\alpha^2}{2}\sum_{i=1}^d\mathbb{E}_t\left[\frac{g_{t,i}^2}{(\sqrt{v_{t,i}} + \epsilon)^2}\right]\\
&= f(\boldsymbol{\theta}_t) - \alpha\sum_{i=1}^d\left([\nabla f(\boldsymbol{\theta}_t)]_i\mathbb{E}_t\left[\frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon} - \frac{g_{t,i}}{\epsilon}\right]\right) - \alpha\sum_{i=1}^d\frac{[\nabla f(\boldsymbol{\theta}_t)]_i^2}{\epsilon} + \frac{L\alpha^2}{2}\sum_{i=1}^d\mathbb{E}_t\left[\frac{g_{t,i}^2}{(\sqrt{v_{t,i}} + \epsilon)^2}\right]\\
&\leq f(\boldsymbol{\theta}_t) + \alpha\left|\sum_{i=1}^d[\nabla f(\boldsymbol{\theta}_t)]_i\mathbb{E}_t\left[\frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon} - \frac{g_{t,i}}{\epsilon}\right]\right| - \alpha\sum_{i=1}^d\frac{[\nabla f(\boldsymbol{\theta}_t)]_i^2}{\epsilon} + \frac{L\alpha^2}{2\epsilon^2}\sum_{i=1}^d\mathbb{E}_t\left[g_{t,i}^2\right]\\
&\leq f(\boldsymbol{\theta}_t) + \alpha\sum_{i=1}^d|\nabla[f(\boldsymbol{\theta}_t)]_i|\,\mathbb{E}_t\left[\underbrace{\left|\frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon} - \frac{g_{t,i}}{\epsilon}\right|}_{T}\right] - \alpha\sum_{i=1}^d\frac{[\nabla f(\boldsymbol{\theta}_t)]_i^2}{\epsilon} + \frac{L\alpha^2}{2\epsilon^2}\sum_{i=1}^d\mathbb{E}_t\left[g_{t,i}^2\right]
\end{aligned}$$

Note that we used $\mathbb{E}_t[g_{t,i}] = [\nabla f(\boldsymbol{\theta}_t)]_i$ in the third line, and $\mathbb{E}[X] \leq \mathbb{E}[|X|]$ and $|\sum_i^d x_i| \leq \sum_{i=1}^d |x_i|$ in the fifth line. We also used $\frac{1}{(a+b)^2} \leq \frac{1}{b^2}, \forall a, b > 0$ for the last term in the fourth line.

Now, we can further bound the term T as follows:

$$\begin{aligned}T &= \left|\frac{g_{t,i}}{\sqrt{v_{t,i}}+\epsilon} - \frac{g_{t,i}}{\epsilon}\right| \\ &\leq \left|\frac{g_{t,i}}{\sqrt{v_{t,i}}+\epsilon}\right| + \left|\frac{g_{t,i}}{\epsilon}\right| \\ &\leq \frac{2|g_{t,i}|}{\epsilon}.\end{aligned}$$

Note that we used $\frac{1}{a+b} \leq \frac{1}{b}, \forall a, b > 0$. Next, let us go back to the main inequality and further bound it as follows:

$$\begin{aligned}\mathbb{E}_t[f(\boldsymbol{\theta}_{t+1})|\boldsymbol{\theta}_t] &\leq f(\boldsymbol{\theta}_t) + \frac{2\alpha}{\epsilon}\sum_{i=1}^{d}|\nabla[f(\boldsymbol{\theta}_t)]_i|\,\mathbb{E}_t\left[|g_{t,i}|\right] - \frac{\alpha}{\epsilon}\sum_{i=1}^{d}[\nabla f(\boldsymbol{\theta}_t)]_i^2 + \frac{L\alpha^2}{2\epsilon^2}\sum_{i=1}^{d}\mathbb{E}_t\left[g_{t,i}^2\right] \\ &\leq f(\boldsymbol{\theta}_t) + \frac{2\alpha G^2}{\epsilon} - \frac{\alpha}{\epsilon}\sum_{i=1}^{d}[\nabla f(\boldsymbol{\theta}_t)]_i^2 + \frac{L\alpha^2}{2\epsilon^2}\sum_{i=1}^{d}\mathbb{E}_t\left[g_{t,i}^2\right].\end{aligned}$$

We used in the second inequality the bounded sample-gradient assumption, $|g_{t_i}| \leq G$. Moreover, we can have the same bound for the expected gradient as follows:

$$\|\nabla f(\boldsymbol{\theta})\| = \|\mathbb{E}_S[\nabla\mathcal{L}(\boldsymbol{\theta}, S)]\| \leq \mathbb{E}_S[\|\nabla\mathcal{L}(\boldsymbol{\theta}, S)\|] \leq G,$$

which leads to bounded expected gradients in each dimension, $|\nabla[f(\boldsymbol{\theta})]_i| \leq G$.

From the bounded variance assumption, we know that the $\mathbb{E}[\|\boldsymbol{g}_t\|^2]$ is bounded as follows:

$$\mathbb{E}[\|\boldsymbol{g}_t\|^2] \leq \frac{\sigma^2}{b_t} + \|\nabla f(\boldsymbol{\theta}_t)\|^2.$$

We can further bound $\mathbb{E}_t[f(\boldsymbol{\theta}_{t+1})|\boldsymbol{\theta}_t]$ as follows:

$$\begin{aligned}\mathbb{E}_t[f(\boldsymbol{\theta}_{t+1})|\boldsymbol{\theta}_t] &\leq f(\boldsymbol{\theta}_t) + \frac{2\alpha G^2}{\epsilon} - \frac{\alpha}{\epsilon}\sum_{i=1}^{d}[\nabla f(\boldsymbol{\theta}_t)]_i^2 + \frac{L\alpha^2}{2\epsilon^2}\left(\frac{\sigma^2}{b_t} + \|\nabla f(\boldsymbol{\theta}_t)\|^2\right) \\ &= f(\boldsymbol{\theta}_t) + \frac{2\alpha G^2}{\epsilon} - \frac{\alpha}{\epsilon}\|\nabla f(\boldsymbol{\theta}_t)\|^2 + \frac{L\alpha^2}{2\epsilon^2}\left(\frac{\sigma^2}{b_t} + \|\nabla f(\boldsymbol{\theta}_t)\|^2\right) \\ &= f(\boldsymbol{\theta}_t) - \|\nabla f(\boldsymbol{\theta}_t)\|^2\left(\frac{\alpha}{\epsilon} - \frac{L\alpha^2}{2\epsilon^2}\right) + \frac{L\alpha^2\sigma^2 + 4\alpha\epsilon b_t G^2}{2b\epsilon^2}.\end{aligned}$$

Rearranging the inequality, taking expectations on both sides, and using the telescopic sum, we can write the following:

$$\left(\frac{2\epsilon\alpha - L\alpha^2}{2\epsilon^2}\right)\sum_{t=1}^{T}\mathbb{E}\|\nabla f(\boldsymbol{\theta}_t)\|^2 \leq f(\boldsymbol{\theta}_1) - \mathbb{E}[f(\boldsymbol{\theta}_{T+1})] + \frac{T(L\alpha^2\sigma^2 + 4\alpha\epsilon b_t G^2)}{2b\epsilon^2}.$$

Multiplying both sides by $\frac{2\epsilon^2}{T(2\epsilon\alpha - L\alpha^2)}$ and using the fact that f is the lowest at the global minimum $\boldsymbol{\theta}^*$: $f(\boldsymbol{\theta}_{T+1}) \geq f(\boldsymbol{\theta}^*)$ as follows:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\|\nabla f(\boldsymbol{\theta}_t)\|^2 \leq 2\epsilon^2\frac{f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}^*)}{T(2\epsilon\alpha - L\alpha^2)} + \frac{2\epsilon^2(L\alpha\sigma^2 + 4b_t G^2)}{b_t(2\epsilon - L\alpha)},$$

which shows the algorithm converges to a stationary point. However, in the limit $T \rightarrow \infty$, the algorithm has to have an increasing batch size similarly to Zaheer et al. (2018).

B. Hessian diagonals of the log-likelihood function for two common distributions

Here, we provide the diagonals of the Hessian matrix of functions involving the log-likelihood of two common distributions: a normal distribution and a categorical distribution with probabilities represented by a softmax function, which we refer to as a *softmax distribution*. We show that the exact computations of the diagonals can be computed with linear complexity since computing the diagonal elements does not depend on off-diagonals in these cases. In the following, we consider the softmax and normal distributions, and we write the exact Hessian diagonals in both cases.

B.1. Softmax distribution

Consider a cross-entropy function for a discrete probability distribution as $f \doteq -\sum_{i=1}^{|\boldsymbol{q}|} p_i \log q_i(\boldsymbol{\theta})$, where $\boldsymbol{q}$ is a probability vector that depends on a parameter vector $\boldsymbol{\theta}$, and $\boldsymbol{p}$ is a one-hot vector for the target class. For softmax distributions, $\boldsymbol{q}$ is parametrized by a softmax function $\boldsymbol{q} \doteq e^{\boldsymbol{\theta}} / \sum_{i=1}^{|\boldsymbol{q}|} e^{\theta_i}$. In this case, we can write the gradient of the cross-entropy function with respect to $\boldsymbol{\theta}$ as

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \boldsymbol{q} - \boldsymbol{p}.$$

Next, we write the exact diagonal elements of the Hessian matrix as follows:

$$\operatorname{diag}(\boldsymbol{H}_{\boldsymbol{\theta}}) = \operatorname{diag}(\nabla_{\boldsymbol{\theta}}(\boldsymbol{q} - \boldsymbol{p})) = \boldsymbol{q} - \boldsymbol{q}^2,$$

where $\boldsymbol{q}^2$ denotes element-wise squaring of $\boldsymbol{q}$, and ∇ operator applied to a vector denotes Jacobian. Computing the exact diagonals of the Hessian matrix depends only on vector operations, which means that we can compute with linear complexity in the network's output dimension. The cross-entropy loss is used with softmax distribution in many important tasks, such as supervised classification and discrete reinforcement learning control with parameterized policies (Chan et al. 2022).

B.2. Multivariate normal distribution with diagonal covariance

For a multivariate normal distribution with diagonal covariance, the parameter vector $\boldsymbol{\theta}$ is determined by the mean-variance vector pair: $\boldsymbol{\theta} \doteq (\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$. The log-likelihood of a random vector $\boldsymbol{x}$ drawn from this distribution can be written as

$$\begin{aligned}
\log q(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) &= -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{D}(\boldsymbol{\sigma}^2)^{-1}(\boldsymbol{x} - \boldsymbol{\mu}) - \frac{1}{2}\log(|\boldsymbol{D}(\boldsymbol{\sigma}^2)|) + c \\
&= -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{D}(\boldsymbol{\sigma}^2)^{-1}(\boldsymbol{x} - \boldsymbol{\mu}) - \frac{1}{2}\log(\sum_{i=1}^{|\boldsymbol{\sigma}|} \sigma_i^2) + c,
\end{aligned}$$

where $\boldsymbol{D}(\boldsymbol{\sigma}^2)$ gives a diagonal matrix with $\boldsymbol{\sigma}^2$ in its diagonal, $|\boldsymbol{M}|$ is the determinant of a matrix $\boldsymbol{M}$ and c is some constant. We can write the gradients of the log-likelihood function with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ as follows:

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}} \log q(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) &= \boldsymbol{D}(\boldsymbol{\sigma}^2)^{-1}(\boldsymbol{x} - \boldsymbol{\mu}) = (\boldsymbol{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}^2, \\
\nabla_{\boldsymbol{\sigma}^2} \log q(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) &= \frac{1}{2}\left[(\boldsymbol{x} - \boldsymbol{\mu})^2 \oslash \boldsymbol{\sigma}^2 - \mathbf{1}\right] \oslash \boldsymbol{\sigma}^2,
\end{aligned}$$

where $\mathbf{1}$ is an all-ones vector, and $\oslash$ denotes element-wise division. Finally, we write the exact diagonals of the Hessian matrix as

$$\begin{aligned}
\operatorname{diag}(\boldsymbol{H}_{\boldsymbol{\mu}}) &= \operatorname{diag}(\nabla_{\boldsymbol{\mu}}(\boldsymbol{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}^2) = -\mathbf{1} \oslash \boldsymbol{\sigma}^2, \\
\operatorname{diag}(\boldsymbol{H}_{\boldsymbol{\sigma}^2}) &= \operatorname{diag}\left(\nabla_{\boldsymbol{\sigma}^2}\left[\frac{1}{2}\left[(\boldsymbol{x} - \boldsymbol{\mu})^2 \oslash \boldsymbol{\sigma}^2 - \mathbf{1}\right] \oslash \boldsymbol{\sigma}^2\right]\right) = \left[0.5\mathbf{1} - (\boldsymbol{x} - \boldsymbol{\mu})^2 \oslash \boldsymbol{\sigma}^2\right] \oslash \boldsymbol{\sigma}^4.
\end{aligned}$$

Clearly, the gradient and the exact Hessian diagonals can be computed with linear complexity in the network's output dimension. Log-likelihood functions for normal distributions are used in many important problems, such as variational inference and continuous reinforcement learning control.

C. HesScale for reinforcement learning loss functions

C.1. Policy gradient loss

The policy gradient loss is computed as the multiplication between the negative log-likelihood and some scalar value that determines how good or bad the action or trajectory selected is. The policy gradient loss is given by

$$\mathcal{L}_{\text{PG}}(s, a) = -\log q(a|s; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)A$$

where A is the return, advantage, or TD error, depending on the algorithm used. The Hessian diagonals of such loss would differ by the multiplicative factor A from the Hessian diagonals w.r.t. the log-likelihood defined in Appendix B for both continuous or categorical distributions. The A2C (Mnih et al. 2016) algorithm uses the advantage function for A.

C.2. Value loss

The value loss used in policy gradient methods can be implemented as a regular regression error between the value and its bootstrapped target. Thus, the Hessian diagonals of such loss function would be ones since the Hessian matrix is the identity.

C.3. PPO policy loss

The proximal policy optimization (PPO, Schulman et al. 2017) methods depend on a surrogate loss different from the one presented in the previous section, unlike A2C (Mnih et al. 2016). The surrogate loss depends on the ratio between the current action probability and the old action probability. Thus, we focus in this section on deriving the gradient and Hessian diagonals of a multivariate normal distribution with diagonal covariance. The Hessian diagonals of such loss would differ by the multiplicative factor A from the Hessian diagonals w.r.t. the action probability. The action probability of a multivariate normal distribution with diagonal covariance is given by

$$p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2) = \frac{1}{\sqrt{2\pi|\boldsymbol{D}(\boldsymbol{\sigma}^2)|}} \exp^{\frac{1}{2}(\boldsymbol{a}-\boldsymbol{\mu})^\top \boldsymbol{D}(\boldsymbol{\sigma}^2)^{-1}(\boldsymbol{a}-\boldsymbol{\mu})}.$$

We can write the gradients of the Gaussian probability function with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ as follows:

$$\nabla_{\boldsymbol{\mu}} p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2) = p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2)(\boldsymbol{a}-\boldsymbol{\mu}) \oslash \boldsymbol{\sigma}^2,$$

$$\nabla_{\boldsymbol{\sigma}^2} p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2) = p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2)\left((\boldsymbol{a}-\boldsymbol{\mu})^2 \oslash \boldsymbol{\sigma}^2 - 1\right) \oslash (2\boldsymbol{\sigma}^2).$$

Finally, we write the exact diagonals of the Hessian matrix as

$$\operatorname{diag}(\boldsymbol{H}_{\boldsymbol{\mu}}) = \left((\boldsymbol{a}-\boldsymbol{\mu}) \circ \nabla_{\boldsymbol{\mu}} p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2) - p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2)\right) \oslash \boldsymbol{\sigma}^2,$$

$$\operatorname{diag}(\boldsymbol{H}_{\boldsymbol{\sigma}^2}) = (\boldsymbol{\sigma}^2 \circ \nabla_{\boldsymbol{\sigma}^2} p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2) - p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2))\left((\boldsymbol{a}-\boldsymbol{\mu})^2 \oslash \boldsymbol{\sigma}^2 - 1\right) \oslash (2\boldsymbol{\sigma}^4) - 0.5p(\boldsymbol{a};\boldsymbol{\mu},\boldsymbol{\sigma}^2)(\boldsymbol{a}-\boldsymbol{\mu})^2 \oslash \boldsymbol{\sigma}^6.$$

D. Scalability

We perform another experiment to evaluate the computational cost of our optimizers. Our Hessian approximation methods and corresponding optimizers have linear computational complexity, which can be seen from Eq. 4 and Eq. 3. However, computing second-order information in optimizers still incurs extra computations compared to first-order optimizers, which may impact how the total computations scale with the number of parameters. Hence, we compare the computational cost of our optimizers with others for various numbers of parameters. More specifically, we measure the update time of each optimizer, which is the time needed to backpropagate first-order and second-order information and update the parameters.

We designed two experiments to study the computational cost of first-order and second-order optimizers. In the first experiment, we used a neural network with a single hidden layer. The network has 64 inputs and 512 hidden units with *tanh* activations. We study the increase in computational time when increasing the number of outputs exponentially, which roughly doubles the number of parameters. The set of values we use for the number of outputs is $\{2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$. The results of this experiment are shown in Fig. 8(a). In the second experiment, we used a neural network with multi-layers, each containing 512 hidden units with *tanh* activations. The network has 64 inputs and 100 outputs. We study the increase in computational time when increasing the number of layers exponentially, which also roughly doubles the number of parameters. The set of values we use for the number of layers is $\{1, 2, 4, 8, 16, 32, 64, 128\}$. The results are shown in Fig. 8(b). The points in Fig. 8(a) and Fig. 8(b) are averaged over 30 updates. The standard errors of the means of these points are smaller than the width of each line. On average, we notice that the cost of AdaHessian, AdaHesScale, and AdaHesScaleGN are three, two, and 1.25 times the cost of Adam, respectively. It is clear that our methods are among the most computationally efficient approximation method for Hessian diagonals.

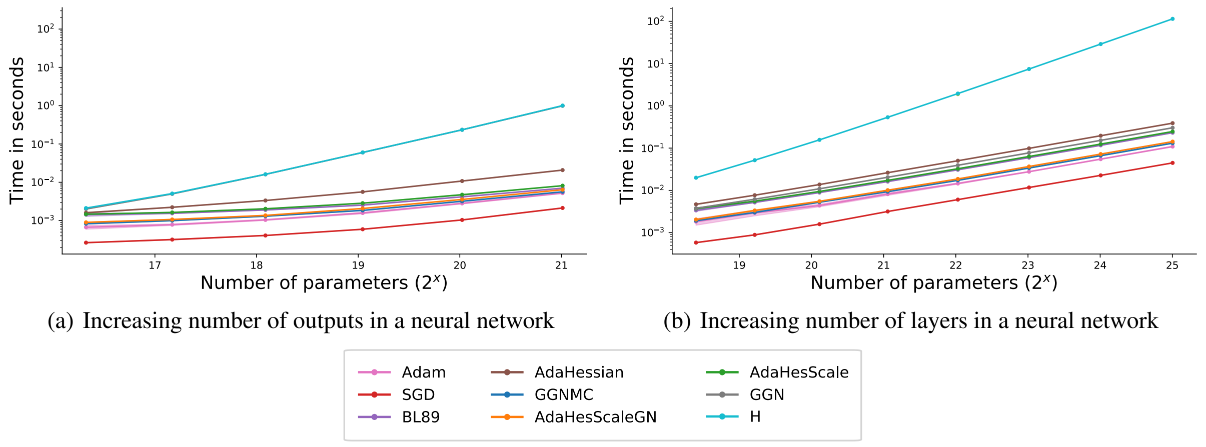


Figure 8. The average computation time for each step of an update is shown for different optimizers. The computed update time is the time needed by each optimizer to backpropagate gradients or second-order information and to update the parameters of the network. GGN overlaps with H in (a).

E. Conditions of Weight Structure for Hessian Diagonality

This section aims to provide a theoretical analysis of the weight structure that makes the Hessian blocks with respect to the pre-activations diagonal or dominantly diagonal. Assuming that the matrix $\boldsymbol{H}_l \in \mathbb{R}^{n\times n}$ containing second derivatives at the l-layer is diagonal, we look for the conditions on $\boldsymbol{W}_l \in \mathbb{R}^{n\times k}$ and $\boldsymbol{H}_l$ that guarantee $\boldsymbol{H}_{l-1} \in \mathbb{R}^{k\times k}$ to be diagonal. Formally, we want to analyze: $\min_{\boldsymbol{W}_l} \|(\boldsymbol{W}_l^\top \boldsymbol{H}_l \boldsymbol{W}_l) \circ \boldsymbol{I} - \boldsymbol{W}_l^\top \boldsymbol{H}_l \boldsymbol{W}_l\|_F^2$, where $\circ$ denotes the Hadamard product.

Let us start by writing the quantity J we want to minimize and simplify the expression. We let $\boldsymbol{A} = \boldsymbol{W}_l^\top \boldsymbol{H}_l \boldsymbol{W}_l$ and write the expression as follows:

$$\begin{aligned}
J &= \|(\boldsymbol{W}_l^\top \boldsymbol{H}_l \boldsymbol{W}_l) \circ \boldsymbol{I} - \boldsymbol{W}_l^\top \boldsymbol{H}_l \boldsymbol{W}_l\|_F^2 \\
&= \|\boldsymbol{A}_l \circ \boldsymbol{I} - \boldsymbol{A}_l\|_F^2 && \text{(substitute } \boldsymbol{W}_l^\top \boldsymbol{H}_l \boldsymbol{W}_l \text{ by } \boldsymbol{A}_l) \\
&= Tr\left((\boldsymbol{A}_l \circ \boldsymbol{I} - \boldsymbol{A}_l)^\top (\boldsymbol{A}_l \circ \boldsymbol{I} - \boldsymbol{A}_l)\right) && \text{(remember that } \|\boldsymbol{A}_l\|_F^2 = Tr(\boldsymbol{A}_l^\top \boldsymbol{A}_l)) \\
&= Tr\left((\boldsymbol{A}_l^\top \circ \boldsymbol{I} - \boldsymbol{A}_l^\top)(\boldsymbol{A}_l \circ \boldsymbol{I} - \boldsymbol{A}_l)\right) \\
&= Tr\left((\boldsymbol{A}_l^2 \circ \boldsymbol{I} - \boldsymbol{A}_l^2 \circ \boldsymbol{I} - \boldsymbol{A}_l^2 \circ \boldsymbol{I} + \boldsymbol{A}_l^\top \boldsymbol{A}_l)\right) && (\boldsymbol{A}_l^2 \text{ denotes element-wise squaring}) \\
&= Tr\left((\boldsymbol{A}_l^\top \boldsymbol{A}_l - \boldsymbol{A}_l^2 \circ \boldsymbol{I})\right). && \text{(expectation on the sum of squared off-diagonal elements)}
\end{aligned}$$

Now we can write $\nabla_{\boldsymbol{W}_l} J$ as follows:

$$\begin{aligned}
\nabla_{\boldsymbol{W}_l} J &= Tr\left(\boldsymbol{A}_l^\top \boldsymbol{A}_l - \boldsymbol{A}_l^2 \circ \boldsymbol{I}\right) \\
&= \nabla_{\boldsymbol{W}_l} Tr\left(\boldsymbol{A}_l^\top \boldsymbol{A}_l - \boldsymbol{A}_l^2 \circ \boldsymbol{I}\right) \\
&= \nabla_{\boldsymbol{W}_l} Tr\left(\boldsymbol{A}_l^\top \boldsymbol{A}_l\right) - \nabla_{\boldsymbol{W}_l} Tr\left(\boldsymbol{A}_l^2 \circ \boldsymbol{I}\right). && (Tr(\boldsymbol{A}+\boldsymbol{B}) = Tr(\boldsymbol{A}) + Tr(\boldsymbol{B}))
\end{aligned}$$

Taking the derivative with respect to $\boldsymbol{W}_l$ involves the 4-tensor of derivatives of elements of $\boldsymbol{A}_l$ with respect to $\boldsymbol{W}_l$, so using the index notation for the following calculations is more convenient. We drop the subscript l for clarity but emphasize that all matrices have a subscript l. We first write the trace of the first and second term in the index notation as follows:

$$A_{i,j} = \sum_{l=1}^{k} W_{l,i} W_{l,j} H_{l,l}, \qquad \Gamma = Tr(\boldsymbol{A}^\top \boldsymbol{A}) = \sum_{i=1}^{k}\sum_{j=1}^{k} A_{i,j}^2, \qquad \Lambda = Tr(\boldsymbol{A}^2 \circ \boldsymbol{I}) = \sum_{i=1}^{k} A_{i,i}^2.$$

We would like to compute $\nabla_{\boldsymbol{W}}\Gamma$ and $\nabla_{\boldsymbol{W}}\Lambda$. One can use the chain rule as follows:

$$\frac{\partial \square}{\partial W_{i,j}} = \sum_{m=1}^{k}\sum_{n=1}^{k} \frac{\partial \square}{\partial A_{m,n}} \frac{\partial A_{m,n}}{\partial W_{i,j}}$$

where $\square \in \{\Gamma, \Lambda\}$. Let us now calculate $\nabla_{\boldsymbol{A}}\Gamma$ and $\nabla_{\boldsymbol{A}}\Lambda$ as follows:

$$\begin{aligned}
\frac{\partial \Gamma}{\partial A_{m,n}} &= \frac{\partial}{\partial A_{m,n}} \sum_{i=1}^{k}\sum_{j=1}^{k} A_{i,j}^2 = 2A_{m,n}, \\
\frac{\partial \Lambda}{\partial A_{m,n}} &= \frac{\partial}{\partial A_{m,n}} \sum_{i=1}^{k} A_{i,i}^2 = 2A_{m,n}\delta_{m,n},
\end{aligned}$$

where $\delta_{m,n} = 1$ when $m = n$ and 0 otherwise. Let us now calculate the elements of the second term in the chain rule, which is a 4-tensor, as follows:

$$\begin{aligned}
\frac{\partial A_{m,n}}{\partial W_{i,j}} &= \frac{\partial}{\partial W_{i,j}} \left(\sum_{l=1}^{k} W_{l,m} W_{l,n} H_{l,l} \right) \\
&= \sum_{l=1}^{k} \frac{\partial W_{l,m}}{\partial W_{i,j}} W_{l,n} H_{l,l} + \sum_{l=1}^{k} \frac{\partial W_{l,n}}{\partial W_{i,j}} W_{l,m} H_{l,l} + \sum_{l=1}^{k} \frac{\partial H_{l,l}}{\partial W_{i,j}} W_{l,m} W_{l,n} \\
&= \sum_{l=1}^{k} W_{l,n} H_{l,l} \delta_{j,m} \delta_{i,l} + \sum_{l=1}^{k} W_{l,m} H_{l,l} \delta_{i,l} \delta_{j,n} + R_{i,j}^{m,n} \\
&= W_{i,n} H_{i,i} \delta_{j,m} + W_{i,m} H_{i,i} \delta_{j,n} + R_{i,j}^{m,n},
\end{aligned}$$

where $R_{i,j}^{m,n} = \sum_{l=1}^{k} \frac{\partial H_{l,l}}{\partial W_{i,j}} W_{m,l} W_{n,l}$ denotes the 4-tensor containing the derivative of the Hessian diagonals of the pre-activations with respect to the weights.

Now, we are able to compute $\nabla_{\boldsymbol{W}}\Gamma$ and $\nabla_{\boldsymbol{W}}\Lambda$ as follows:

$$\begin{aligned}
\frac{\partial \Gamma}{\partial W_{i,j}} &= \sum_{m=1}^{k}\sum_{n=1}^{k} \frac{\partial \Gamma}{\partial A_{m,n}} \frac{\partial A_{m,n}}{\partial W_{i,j}} \\
&= \sum_{m=1}^{k}\sum_{n=1}^{k} 2A_{m,n} \left(W_{i,n} H_{i,i} \delta_{j,m} + W_{i,m} H_{i,i} \delta_{j,n} + R_{i,j}^{m,n} \right) \\
&= 2\sum_{n=1}^{k} A_{j,n} W_{i,n} H_{i,i} + 2\sum_{m=1}^{k} A_{m,j} W_{i,m} H_{i,i} + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n} R_{i,j}^{m,n} \\
&= 4\sum_{n=1}^{k} A_{j,n} W_{i,n} H_{i,i} + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n} R_{i,j}^{m,n} \qquad \text{(note that } A_{m,i} = A_{i,m} \text{ since } \boldsymbol{A} \text{ is symmetric)}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \Lambda}{\partial W_{i,j}} &= \sum_{m=1}^{k}\sum_{n=1}^{k} \frac{\partial \Lambda}{\partial A_{m,n}} \frac{\partial A_{m,n}}{\partial W_{i,j}} \\
&= \sum_{m=1}^{k}\sum_{n=1}^{k} 2A_{m,n}\delta_{m,n} \left(W_{i,n} H_{i,i} \delta_{j,m} + W_{i,m} H_{i,i} \delta_{j,n} + R_{i,j}^{m,n} \right) \\
&= 2\sum_{m=1}^{k} A_{m,m} \left(W_{i,m} H_{i,i} \delta_{j,m} + W_{i,m} H_{i,i} \delta_{j,m} + R_{i,j}^{m,m} \right) \\
&= 2\sum_{m=1}^{k} A_{m,m} \left(2W_{i,m} H_{i,i} \delta_{j,m} + R_{i,j}^{m,m} \right) \\
&= 4A_{j,j} W_{i,j} H_{i,i} + 2\sum_{m=1}^{k} A_{m,m} R_{i,j}^{m,m},
\end{aligned}$$

$$\begin{aligned}
\frac{\partial(\Gamma-\Lambda)}{\partial W_{i,j}} &= 4\sum_{n=1}^{k} A_{j,n}W_{i,n}H_{i,i} + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n} - 4A_{j,j}W_{i,j}H_{i,i} - 2\sum_{m=1}^{k} A_{m,m}R_{i,j}^{m,m} \\
&= 4\sum_{n=1}^{k} A_{j,n}W_{i,n}H_{i,i} - 4A_{j,j}W_{i,j}H_{i,i} + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n} - 2\sum_{m=1}^{k} A_{m,m}R_{i,j}^{m,m} \\
&= 4\sum_{n=1}^{k} A_{j,n}W_{i,n}H_{i,i} - 4\sum_{n=1}^{k} A_{j,j}W_{i,j}H_{i,i}\delta_{j,n} + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n} - 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n}\delta_{m,n} \\
&= 4\sum_{n=1}^{k} A_{j,n}W_{i,n}H_{i,i}(1-\delta_{j,n}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n}(1-\delta_{m,n}).
\end{aligned}$$

We can finally write the element of the gradient $\nabla_{\boldsymbol{W}} J$ as follows:

$$\begin{aligned}
[\nabla_{\boldsymbol{W}} J]_{i,j} &= \nabla_{\boldsymbol{W}}(\Gamma-\Lambda) \\
&= 4\sum_{n=1}^{k} A_{j,n}W_{i,n}H_{i,i}(1-\delta_{j,n}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n}(1-\delta_{m,n}).
\end{aligned}$$

We find the conditions on $\boldsymbol{W}$ and $\boldsymbol{H}$ that minimizes J by setting $\nabla_{\boldsymbol{W}} J$ to zero. There are two notable cases where the gradient equals zero.

E.1. When $\boldsymbol{W}$ is diagonal

If $\boldsymbol{W}$ is diagonal, we can write it as $W_{i,j} = \delta_{i,j}$. Therefore, the gradient elements are reduced to:

$$\begin{aligned}
[\nabla_{\boldsymbol{W}} J]_{i,j} &= 4\sum_{n=1}^{k} \left(A_{j,n}W_{i,n}H_{i,i}\right)(1-\delta_{j,n}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n}\left(1-\delta_{m,n}\right) \\
&= 4\sum_{n=1}^{k} A_{j,n}H_{i,i}\delta_{i,n}(1-\delta_{n,j}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n}\left(1-\delta_{m,n}\right) \\
&= 4\sum_{n=1}^{k}\left(\sum_{l=1}^{k} W_{l,j}W_{l,n}H_{l,l}\right)H_{i,i}\delta_{i,n}(1-\delta_{n,j}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k}\left(\sum_{l=1}^{k} W_{l,m}W_{l,n}H_{l,l}\right)R_{i,j}^{m,n}\left(1-\delta_{m,n}\right) \\
&= 4\sum_{n=1}^{k}\left(\sum_{l=1}^{k} \delta_{l,j}\delta_{l,n}H_{l,l}\right)H_{i,i}\delta_{i,n}(1-\delta_{n,j}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k}\left(\sum_{l=1}^{k} \delta_{l,m}\delta_{l,n}H_{l,l}\right)R_{i,j}^{m,n}\left(1-\delta_{m,n}\right) \\
&= 4\sum_{n=1}^{k}\left(\delta_{j,n}H_{j,j}\right)H_{i,i}\delta_{i,n}(1-\delta_{n,j}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k}\left(\delta_{m,n}H_{m,m}\right)R_{i,j}^{m,n}\left(1-\delta_{m,n}\right) \\
&= 4H_{j,j}H_{i,i}\delta_{i,j}(1-\delta_{i,j}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k} H_{m,m}R_{i,j}^{m,n}\delta_{m,n}\left(1-\delta_{m,n}\right) \\
&= 0 \qquad \text{(note that } \delta_{i,j}(1-\delta_{i,j}) = 0, \forall i,j).
\end{aligned}$$

E.2. When $\boldsymbol{H} = \alpha\boldsymbol{I}$ and $W_{i,j} \sim \mathcal{N}(0,\sigma), \forall\sigma$, the gradient becomes zero on expectation

We can write $\boldsymbol{H} = \alpha\boldsymbol{I}$ as $H_{i,j} = \alpha\delta_{i,j}$. Therefore, the gradient elements are reduced to:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{W}_l}[\nabla_{\boldsymbol{W}} J]_{i,j} &= \mathbb{E}_{\boldsymbol{W}_l}\left[4\sum_{n=1}^{k} A_{j,n}W_{i,n}H_{i,i}(1-\delta_{j,n}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n}(1-\delta_{m,n})\right] \\
&= \mathbb{E}_{\boldsymbol{W}_l}\left[4\alpha\sum_{n=1}^{k} A_{j,n}W_{i,n}\delta_{i,i}(1-\delta_{j,n}) + 2\sum_{m=1}^{k}\sum_{n=1}^{k} A_{m,n}R_{i,j}^{m,n}(1-\delta_{m,n})\right]
\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{E}_{\boldsymbol{W}_l}\Bigg[4\alpha^2 \sum_{n=1}^{k}\left(\sum_{l=1}^{k} W_{l,j}W_{l,n}\delta_{l,l}\right) W_{i,n}\delta_{i,i}(1-\delta_{j,n}) \\
&\qquad + 2\alpha \sum_{m=1}^{k}\sum_{n=1}^{k}\left(\sum_{l=1}^{k} W_{l,m}W_{l,n}\delta_{l,l}\right) R_{i,j}^{m,n}\,(1-\delta_{m,n})\Bigg] \\
&= \mathbb{E}_{\boldsymbol{W}_l}\left[4\alpha^2 \sum_{n=1}^{k}\left(\sum_{l=1}^{k} W_{l,j}W_{l,n}\right) W_{i,n}(1-\delta_{j,n}) + 2\alpha \sum_{m=1}^{k}\sum_{n=1}^{k}\left(\sum_{l=1}^{k} W_{l,m}W_{l,n}\right) R_{i,j}^{m,n}\,(1-\delta_{m,n})\right] \\
&= \mathbb{E}_{\boldsymbol{W}_l}\left[4\alpha^2 W_{i,j}^3 - 4\alpha^2 W_{i,j}^3 + 2\alpha \sum_{m=1}^{k}\left(\sum_{l=1}^{k} W_{l,m}W_{l,m}\right) R_{i,j}^{m,m} - 2\alpha \sum_{n=1}^{k}\left(\sum_{l=1}^{k} W_{l,m}W_{l,m}\right) R_{i,j}^{m,m}\right] \\
&= 0.
\end{aligned}
$$

Note that $\mathbb{E}[W_{i,j}^2 W_{i,l}] = 0, \forall l \neq j$.

F. HesScale with Convolutional Neural Networks

Here, we derive the Hessian propagation for convolutional neural networks (CNNs). Consider a CNN with $L-1$ layers followed by a fully connected layer that outputs the predicted output $\boldsymbol{q}$. The CNN filters are parameterized by $\{\boldsymbol{W}_1, ..., \boldsymbol{W}_L\}$, where $\boldsymbol{W}_l$ is the filter matrix at the l-th layer with the dimensions $k_{l,1} \times k_{l,2}$, and its element at the ith row and the jth column is denoted by $W_{l,i,j}$. For the simplicity of this proof, we assume that the number of filters at each layer is one; the proof can be extended easily to the general case. At the layer l, we get the activation matrix $\boldsymbol{H}_l$ by applying the activation function $\boldsymbol{\sigma}$ to the pre-activation $\boldsymbol{A}_l$: $\boldsymbol{H}_l = \boldsymbol{\sigma}(\boldsymbol{A}_l)$. We assume here that the activation function is element-wise activation for all layers except for the final layer L, where it becomes the softmax function. We simplify notations by defining $\boldsymbol{H}_0 \doteq \boldsymbol{X}$, where $\boldsymbol{X}$ is the input sample. The activation $\boldsymbol{H}_l$ is then convoluted by the weight matrix $\boldsymbol{W}_{l+1}$ of layer $l+1$ to produce the next pre-activation: $A_{l+1,i,j} = \sum_{m=0}^{k_{l,1}-1}\sum_{n=0}^{k_{l,2}-1} W_{l+1,m,n} H_{l,(i+m),(j+n)}$. We denote the size of the activation at the l-th layer by $h_l \times w_l$. The recursive formulation used for Hessian diagonals backpropagation is given in Theorem F.1.

Theorem F.1. ***HesScale Computation with CNNs**. Under the zero second-order off-diagonals assumption in all layers of a neural network except for the last one, the second derivatives can be propagated with linear complexity in the number of network parameters and in the network's output dimension using the following equations:*

$$
\begin{aligned}
\widehat{\frac{\partial^2 \mathcal{L}}{\partial A_{l,i,j}^2}} &\doteq \sigma'(A_{l,i,j})^2 \sum_{m=0}^{k_{l+1,2}-1}\sum_{n=0}^{k_{l+1,2}-1} \widehat{\frac{\partial^2 \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}^2}} W_{l+1,m,n}^2 \\
&\quad + \sigma''(A_{l,i,j}) \sum_{m=0}^{k_{l+1,2}-1}\sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} W_{l+1,m,n}, \\
\widehat{\frac{\partial^2 \mathcal{L}}{\partial W_{l,i,j}^2}} &\doteq \sum_{m=0}^{h_l-k_{l,1}}\sum_{n=0}^{w_l-k_{l,2}} \widehat{\frac{\partial^2 \mathcal{L}}{\partial A_{l,m,n}^2}} H_{l-1,(i+m),(j+n)}^2.
\end{aligned}
$$

Proof. The backpropagation equations for the described network are given following Rumelhart et al. (1986):

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial A_{l,i,j}} &= \sum_{m=0}^{k_{l+1,1}-1}\sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} \frac{\partial A_{l+1,(i-m),(j-n)}}{\partial A_{l,i,j}} \\
&= \sum_{m=0}^{k_{l+1,1}-1}\sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} \sum_{m'=0}^{k_{l+1,1}-1}\sum_{n'=0}^{k_{l+1,2}-1} W_{l+1,m',n'} \frac{\partial H_{l,(i-m+m'),(j-n+n')}}{\partial A_{l,i,j}} \\
&= \sum_{m=0}^{k_{l+1,1}-1}\sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} W_{l+1,m,n}\sigma'(A_{l,i,j})
\end{aligned}
$$

$$= \sigma'(A_{l,i,j}) \sum_{m=0}^{k_{l+1,1}-1} \sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} W_{l+1,m,n}, \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial W_{l,i,j}} = \sum_{m=0}^{h_l - k_{l,1}} \sum_{n=0}^{w_l - k_{l,2}} \frac{\partial \mathcal{L}}{\partial A_{l,m,n}} \frac{\partial A_{l,m,n}}{\partial W_{l,i,j}} = \sum_{m=0}^{h_l - k_{l,1}} \sum_{n=0}^{w_l - k_{l,2}} \frac{\partial \mathcal{L}}{\partial A_{l,m,n}} H_{l-1,(i+m),(j+n)}. \tag{9}$$

In the following, we write the equations for the exact Hessian diagonals with respect to weights $\partial^2\mathcal{L}/\partial W_{l,i,j}^2$, which requires the calculation of $\partial^2\mathcal{L}/\partial A_{l,i,j}^2$ first:

$$\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial A_{l,i,j}^2} &= \frac{\partial}{\partial A_{l,i,j}} \left[\sigma'(A_{l,i,j}) \sum_{m=0}^{k_{l+1,1}-1} \sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} W_{l+1,m,n} \right] \\
&= \sigma'(A_{l,i,j}) \sum_{m,p=0}^{k_{l+1,2}-1} \sum_{n,q=0}^{k_{l+1,2}-1} \frac{\partial^2 \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)} \partial A_{l+1,(i-p),(j-q)}} \frac{\partial A_{l+1,(i-p),(j-q)}}{\partial A_{l,i,j}} W_{l+1,m,n} \\
&\quad + \sigma''(A_{l,i,j}) \sum_{m=0}^{k_{l+1,2}-1} \sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} W_{l+1,m,n}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial W_{l,i,j}^2} &= \frac{\partial}{\partial W_{l,i,j}} \left[\sum_{m=0}^{h_l - k_{l,1}} \sum_{n=0}^{w_l - k_{l,2}} \frac{\partial \mathcal{L}}{\partial A_{l,m,n}} H_{l-1,(i+m),(j+n)} \right] \\
&= \sum_{m,p=0}^{h_l - k_{l,1}} \sum_{n,q=0}^{w_l - k_{l,2}} \frac{\partial^2 \mathcal{L}}{\partial A_{l,m,n} \partial A_{l,p,q}} \frac{\partial A_{l,p,q}}{\partial W_{l,i,j}} H_{l-1,(i+m),(j+n)}
\end{aligned}$$

Since the calculation of $\partial^2\mathcal{L}/\partial A_{l,i,j}^2$ and $\partial^2\mathcal{L}/\partial W_{l,i,j}^2$ depend on the off-diagonal terms, the computation complexity becomes quadratic. Following Becker and LeCun (1989), we approximate the Hessian diagonals by ignoring the off-diagonal terms, which leads to a backpropagation rule with linear computational complexity for our estimates $\widehat{\frac{\partial^2 \mathcal{L}}{\partial W_{l,i,j}^2}}$ and $\widehat{\frac{\partial^2 \mathcal{L}}{\partial A_{l,i,j}^2}}$:

$$\begin{aligned}
\widehat{\frac{\partial^2 \mathcal{L}}{\partial A_{l,i,j}^2}} &\doteq \sigma'(A_{l,i,j})^2 \sum_{m=0}^{k_{l+1,2}-1} \sum_{n=0}^{k_{l+1,2}-1} \widehat{\frac{\partial^2 \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}^2}} W_{l+1,m,n}^2 \\
&\quad + \sigma''(A_{l,i,j}) \sum_{m=0}^{k_{l+1,2}-1} \sum_{n=0}^{k_{l+1,2}-1} \frac{\partial \mathcal{L}}{\partial A_{l+1,(i-m),(j-n)}} W_{l+1,m,n}, \qquad (10) \\
\widehat{\frac{\partial^2 \mathcal{L}}{\partial W_{l,i,j}^2}} &\doteq \sum_{m=0}^{h_l - k_{l,1}} \sum_{n=0}^{w_l - k_{l,2}} \widehat{\frac{\partial^2 \mathcal{L}}{\partial A_{l,m,n}^2}} H_{l-1,(i+m),(j+n)}^2. \qquad (11)
\end{aligned}$$

□

G. Additional Supervised Classification Results

Here, we provide the sensitivity of the step size for each method in Fig. 9.

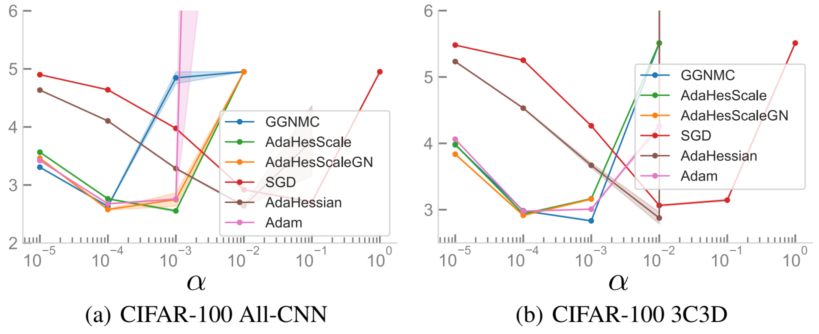


Figure 9. Parameter Sensitivity study for each algorithm on CIFAR-100 with All-CNN and 3C3D architectures. The range of step size is $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$. We choose β_1 to be equal to 0.9 and β_2 to be equal to 0.999. Each point for each algorithm represents the average test loss given a set of parameters.

H. Additional RL experiments

Here, we provide additional results on the performance of the A2C and PPO algorithms in Fig. 10. In addition, we provide a robustness analysis of the step size with the A2C and PPO algorithms using Scaled AdaHesScale against Scaled Adam in Fig. 11, 12, and 13.

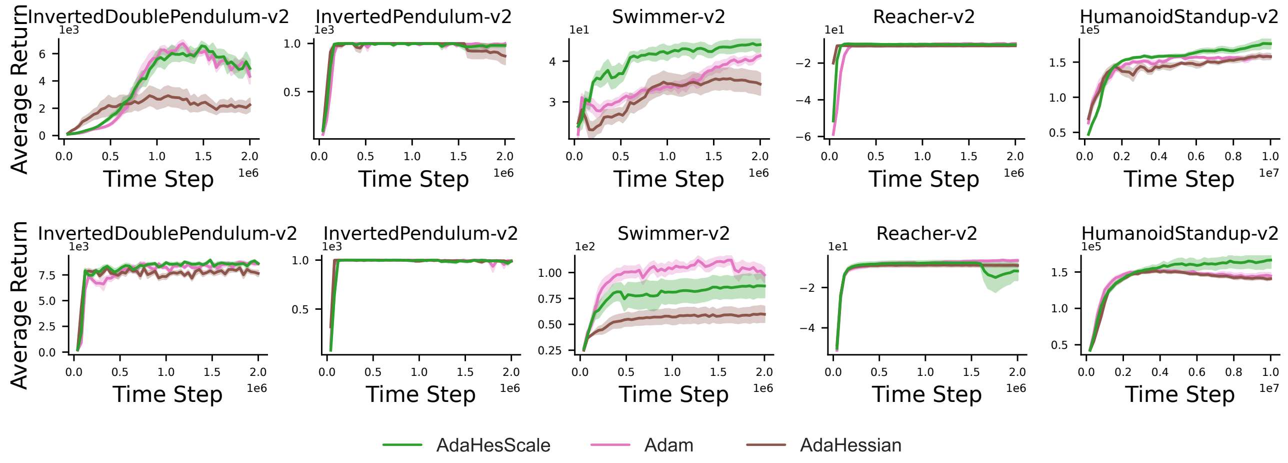


Figure 10. Performance of A2C (first row) and PPO (second row) with AdaHesScale, Adam, and with AdaHessian on 5 MuJoCo environments. We show the undiscounted return averaged over 10 independent runs. The shaded area represents the standard error.

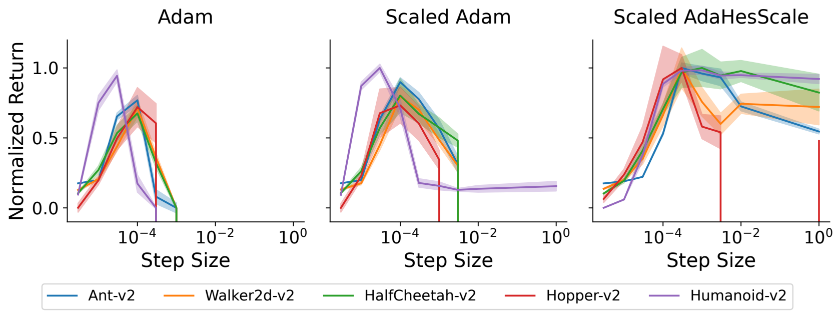


Figure 11. Robustness of HesScale-based step-size scaling with AdaHesScale and Adam on 5 MuJoCo environments using A2C. We show the undiscounted return averaged over 10 independent runs. The shaded area represents the standard error.

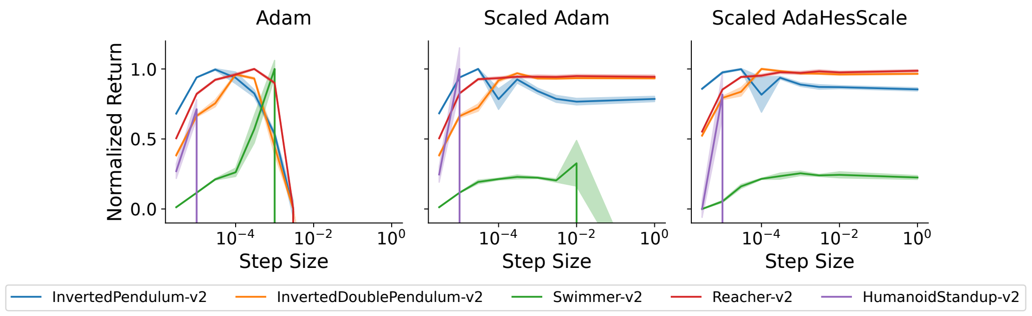


Figure 12. Robustness of HesScale-based step-size scaling with AdaHesScale and Adam on 5 additional MuJoCo environments using PPO. We show the undiscounted return averaged over 10 independent runs. The shaded area represents the standard error.

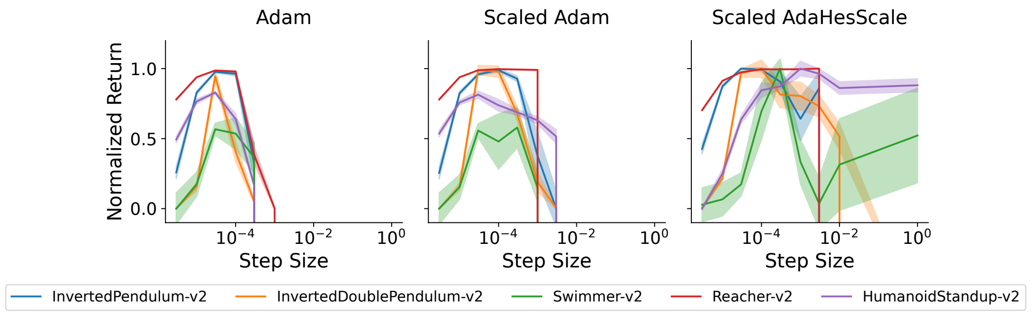


Figure 13. Robustness of HesScale-based step-size scaling with AdaHesScale and Adam on 5 additional MuJoCo environments using A2C. We show the undiscounted return averaged over 10 independent runs. The shaded area represents the standard error.