Extending Hessian-Aware Scaling of Gradient Descent

Arthur Pollet 325074 — Mikuláš Vanoušek 394827 — Matya Aydin 388895

***Abstract*—Gradient descent's performance depends heavily on the learning rate. Recent work has demonstrated that convergence with a unit step size can be achieved by incorporating second-order information, specifically through a Hessian-aware scaling technique called MRCG. In this report, we build upon this approach by introducing a stochastic variant. We also incorporate momentum and a preconditioned update direction.**

I. Introduction

Gradient descent is a widely used optimization algorithm. However, choosing an appropriate step size remains a key challenge. Incorporating second-order information can reduce sensitivity to the step size, improving convergence. In this report, we review relevant prior work and describe how we extend it. Finally, we present empirical results demonstrating the effectiveness of our approach.

II. Previous Work

A. Optimization and Gradient Descent

We begin by recalling the framework established in [1], where we wish to minimize a function f:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) \tag{1}$$

Gradient descent performs iterative updates:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \nabla f(\mathbf{x}_k)$$

Here, γ_k is the step size (also called the learning rate). In most practical applications, particularly in non-convex settings, if the step size is chosen correctly, gradient descent converges to a stationary point. However, choosing the right step size is challenging: If γ_k is too small, progress will be slow, requiring many iterations. If γ_k is too large, the algorithm may overshoot the minimum or even diverge entirely.

B. Hessian Aware Scaling

To address this issue, one can use second-order information — that is, how the gradient itself changes — to adaptively choose better step sizes. This involves analyzing the curvature of the function, which gives us insight into how steep or flat the landscape is around a point.

Curvature affects how we adjust step sizes during optimization. **Strong positive curvature (SPC)** indicates a steep, bowl-shaped region where gradients change rapidly—small steps should be performed. **Limited positive curvature (LPC)** corresponds to flatter regions where gradients vary slowly, allowing for larger steps. **Negative curvature (NC)** occurs near saddle points, where the surface curves downward in some directions—naive gradient descent may become unstable here, and special care is needed to escape efficiently. The **MRCG** [1] algorithm (see 2) uses efficiently computed estimates of curvature to determine the step size (see 1).

In the SPC case, we use one of three possible scalings:

$$s^{CG} = \frac{\|\mathbf{g}\|^2}{\langle \mathbf{g}, \mathbf{Hg}\rangle},\ s^{MR} = \frac{\langle \mathbf{g}, \mathbf{Hg}\rangle}{\|\mathbf{Hg}\|^2} \tag{2}$$

or

$$s^{GM} = \sqrt{s^{MR} s^{CG}} = \frac{\|\mathbf{g}\|}{\|\mathbf{Hg}\|} \tag{3}$$

The scaling s^{CG} draws inspiration from Conjugate Gradient (CG). It results from minimizing the local quadratic approximation under the constraint that the direction $\mathbf{p}$ is in the direction opposite to $\mathbf{g}$:

$$\min_{\mathbf{p}=-s\mathbf{g}} \langle \mathbf{p}, \mathbf{g}\rangle + \frac{\langle \mathbf{p}, \mathbf{Hp}\rangle}{2} \tag{4}$$

The scaling s^{MR} uses the same constraint but minimizes $\|\mathbf{Hp} + \mathbf{g}\|^2$. It draws inspiration from Minimum Residual (MR). Equation 3 is the geometric mean between these two quantities. We choose between these three scalings randomly, as seen on line 5 of Algorithm 1.

The condition $\langle \mathbf{g}, \mathbf{Hg}\rangle > \sigma\|\mathbf{g}\|^2$ gives a lower bound on the curvature of f along the direction of $\mathbf{g}$. Hence σ defines the threshold between strong and weak curvature. It acts as a Lipschitz gradient constant and $\frac{1}{\sigma}$ upper bounds the step size in the SPC and LPC cases. Finally, we know $\langle \mathbf{g}, \mathbf{Hg}\rangle < 0$ if $\mathbf{H}$ means curvature of f is negative around $\mathbf{x}$.

The MRCG algorithm only uses second-order information for the scaling in the SPC case. Crucially, we do not need to compute the full Hessian H, as the scaling only depends on Hg, which can be computed efficiently (see `jax.jvp`).

Once the scaling is set, we use line search to adapt the step size α for global convergence, this can be done by ensuring that the Armijo condition is satisfied:

$$f(\mathbf{x} + \alpha\mathbf{p}) \leq f(\mathbf{x}) + \rho\alpha\langle \mathbf{p}, \mathbf{g}\rangle \tag{5}$$

With $\rho \in (0, \frac{1}{2})$, a hyperparameter that controls by how much f will decrease after selecting α with line search. In fact, by construction, we have $\langle \mathbf{p}, \mathbf{g}\rangle \leq 0$. The complete algorithm is presented in Algorithm 2. θ is a hyperparameter controling the decay of the selected step size in Algorithm 3 and its growth in Algorithm 4.

III. Extensions

A. Stochastic Variant of MRCG

The original Hessian-aware method, as presented in Section II, relies on access to the full gradient $\nabla f(\mathbf{x})$ and the full Hessian $\nabla^2 f(\mathbf{x})$. While this is feasible for small-scale problems or synthetic experiments, it quickly becomes computationally expensive or intractable for large-scale machine learning tasks. To overcome this limitation, we propose a stochastic extension of MRCG.

In stochastic optimization, rather than computing the gradient and Hessian over the entire dataset, we approximate them using a randomly sampled mini-batch of data. Formally, given a dataset $\mathcal{D} = \{(\mathbf{a}_i, b_i)\}_{i=1}^n$ and a loss function $\ell(\mathbf{a}, b; \mathbf{x})$, we define:

$$\tilde{\nabla} f(\mathbf{x}) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla \ell\left(\mathbf{a}_i, b_i; \mathbf{x}\right)$$

$$\tilde{\nabla}^2 f(\mathbf{x}) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla^2 \ell\left(\mathbf{a}_i, b_i; \mathbf{x}\right)$$

where $\mathcal{B} \subset \mathcal{D}$ is a mini-batch sampled uniformly at random at each iteration. Under standard assumptions, these are unbiased estimators of the full gradient and Hessian, and they significantly reduce the computational cost.

B. First-Order Hessian Approximation

Drawing inspiration from [2], we can use the sum of the previous (stochastic) gradients to approximate the diagonal of the Hessian in Algorithm 1 as:

$$\hat{H}_k = \begin{bmatrix} \sqrt{[G_k]_1} & 0 & \cdots & 0 \\ 0 & \sqrt{[G_k]_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sqrt{[G_k]_d} \end{bmatrix}^{-1} \tag{6}$$

With $[G_k]_d = \sum_{i=0}^k [g_i]_d^2$. This approximation enables fast computation of Hessian-vector products in $\mathcal{O}(d)$ time via element-wise multiplication, and only the diagonal entries are stored, ensuring memory efficiency. Note that this diagonal approximation yields a strictly positive definite matrix, which avoids issues of singularity. For comparison, we also consider an alternative variant where we accumulate the raw gradients: $[G_k^{\text{plain}}]_j = \sum_{i=0}^k [g_i]_j$. Although neither approach yields a true second-order method, both serve as first-order approximations to the curvature. We compare these two strategies in Section IV-C.

C. Momentum

A straightforward extension is adding momentum by replacing $\mathbf{p}_k$ with an exponential moving average of the previous iterates:

$$\hat{\mathbf{p}}_k = \beta \hat{\mathbf{p}}_{k-1} + (1-\beta)\mathbf{p}_k$$

Results are presented in Section IV-D.

D. Preconditioned direction

Drawing inspiration from [3], we can replace the output of Algorithm 1 with a rescaled version, using a moving average of the second-order moment given by Equation 6:

$$\tilde{\mathbf{p}}_k = \frac{\mathbf{p}_k}{\beta \hat{H}_{k-1} + (1-\beta) g_k^2 + \epsilon} \tag{7}$$

where the division is taken element-wise, as $\hat{H}_{k-1}$ is a vector and ϵ is a small tolerance set to 10^{-9} to avoid division by zero. This rescaling adjusts the update direction by penalizing ill-conditioned coordinates, which can otherwise slow down convergence. As a result, it tends to favor larger step sizes in well-scaled directions during the line search. We evaluate the impact of this preconditioning in Section IV-E.

IV. Results

A. Reproducing MRCG

We reproduce the multi-class ℓ_2-regularized logistic-regression benchmark from [1]. Figure 1 shows that our implementation of MRCG decisively outperforms fixed-step GD, Armijo line-search GD, and Heavy-Ball. However, the Adam optimizer outperforms MRCG. Crucially, MRCG achieves this performance without any hand tuning or prior knowledge of problem constants.

For complete reproducibility, including dataset preprocessing, the oracle-call accounting scheme, and every hyperparameter—see Appendix A.

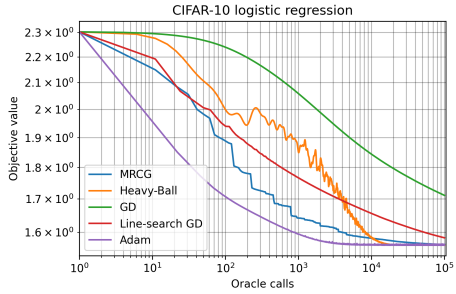

Fig. 1. Optimizers performance comparison

B. Stochastic Variant of MRCG

We compare stochastic gradient descent (SGD) to a stochastic version of MRCG on Scikit's California Housing dataset. The hyperparameters used are the same as in Section IV-A. We run the optimizers for 100 iterations using various batch sizes.

Figure 2 shows that low batch sizes lead to slower convergence in the objective value. Both algorithms converge monotonically with batches of size 5% of the dataset or higher. In this scenario, MRCG clearly outperforms SGD. Below 5%, SGD is less likely to produce updates that worsen the objective value. This confirms that stochastic approximations retain most of the benefits of Hessian-aware scaling while being scalable to larger datasets. Calculating the updates on the full dataset

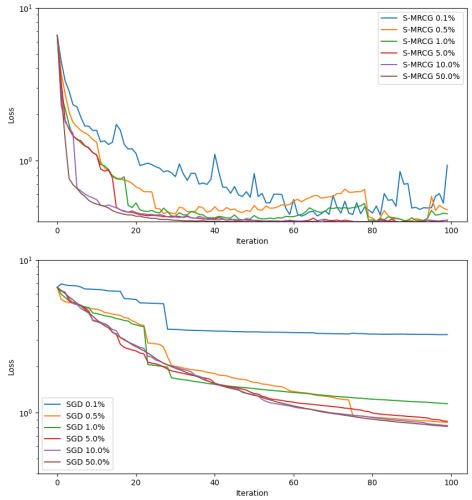

Fig. 2. Comparison of objective value convergence for different batch sizes for S-MRCG and SGD. The optimizer updates are stochastic, but the loss is calculated over the full dataset.

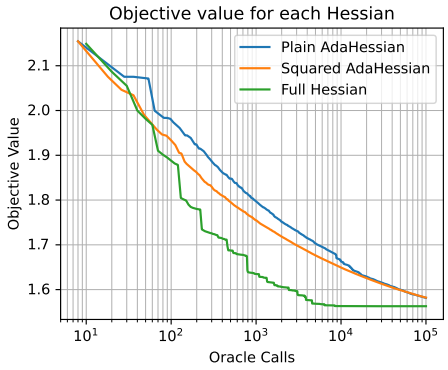

Fig. 3. Comparison of loss curves using different Hessian approximations

offers little benefit over the batch sizes of 10%, while being 10 times more computationally expensive.

C. First-Order Hessian Approximation

Figure 3 uses the same framework as in Section IV-A. The original MRCG algorithm requires fewer oracle calls overall. However, we observed shorter execution times for Hessian approximations. By only approximating the Hessian, we limit our ability to recognize SPC. Finally, as the problem is strongly convex, using the squared sum of the gradient yields better results than the plain sum, as it more accurately captures the Hessian's curvature.

D. Momentum

Figure 4 shows that adding momentum does not help convergence. We can draw similar conclusions to those in Section IV-C: Using an exponential moving average of the gradients instead of the raw gradient introduces a contradiction between the way the scaling is selected in Algorithm 1 and the step size that is selected by one of the line searches.

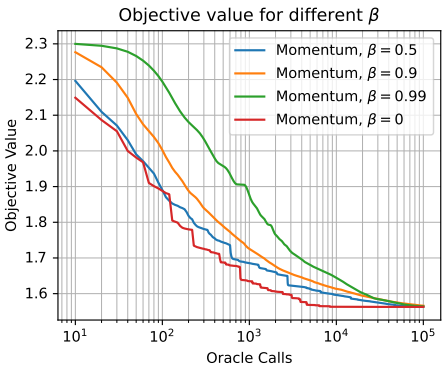

Fig. 4. Loss with different momentum values

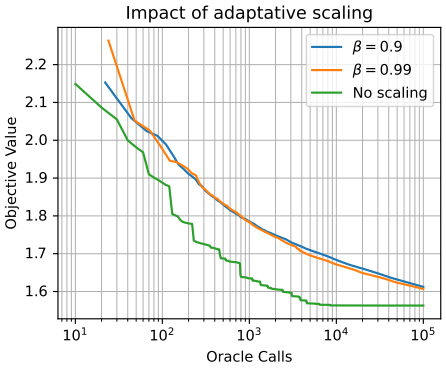

Fig. 5. Loss for different adaptative scalings

However, the loss remains monotonically decreasing, and all configurations converge to the same final value after 10^5 oracle calls.

E. Preconditioned direction

Figure 5 shows that varying β in the preconditioned direction defined in Equation 7 does not yield better performance. Despite producing larger step sizes in practice, modifying $\mathbf{p}$ coordinate-wise breaks its optimality given by Equation 4. These bigger step sizes might also explain the gap present after 10^5 oracle calls.

V. Conclusion

We successfully reproduced the MRCG algorithm and investigated several extensions to enhance its practicality. Our results indicate that a stochastic variant effectively scales the method to large datasets while largely preserving its performance benefits. Furthermore, employing a first-order Hessian approximation presents a promising trade-off, offering faster computation with a negligible loss in accuracy. Conversely, incorporating momentum or preconditioned directions proved counterproductive, as these modifications conflict with the algorithm's core mechanism of estimating curvature to set the step size. Future work could explore developing coordinate-specific scaling derived from the Hessian to further improve performance.

References

[1] O. Smee, F. Roosta, and S. J. Wright, "First-ish Order Methods: Hessian-aware Scalings of Gradient Descent," Feb. 2025, arXiv:2502.03701 [math]. [Online]. Available: http://arxiv.org/abs/2502.03701

[2] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, Jul. 2011, published July 2011; submitted March 2010, revised March 2011. [Online]. Available: https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014, dec. 22, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[4] B. A. Pearlmutter, "Fast exact multiplication by the Hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994. [Online]. Available: https://doi.org/10.1162/neco.1994.6.1.147

[5] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep. Technical Report TR-2009, 2009, dataset available at https://www.cs.toronto.edu/~kriz/cifar.html. [Online]. Available: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[6] Y. Nesterov and B. T. Polyak, "Cubic regularization of Newton method and its global performance," *Mathematical Programming*, vol. 108, no. 1, pp. 177–205, Aug. 2006.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, May 2015, arXiv:1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980

APPENDIX

Algorithm 1 Hessian-aware Scaling Selection

Inputs: Gradient $\mathbf{g}$, Hessian $\mathbf{H}$, and SPC scaling tolerance $\sigma > 0$.
Set the range for LPC scaling as $s_{\min}^{LPC} \in (0, 1/\sigma)$.
Set the range for NC scaling as $0 < s_{\min}^{NC} \leq s_{\max}^{NC} < \infty$.
if $\langle \mathbf{g}, \mathbf{Hg} \rangle > \sigma \|\mathbf{g}\|^2$ **then**
    choose $s^{SPC} \in \{s^{\text{MR}}, s^{\text{CG}}, s^{\text{GM}}\}$, set $\mathbf{p} = -s^{SPC}\mathbf{g}$, set `FLAG` = SPC.
else if $0 \leq \langle \mathbf{g}, \mathbf{Hg} \rangle \leq \sigma \|\mathbf{g}\|^2$ **then**
    choose $s^{LPC} \in [s_{\min}^{LPC}, 1/\sigma]$, set $\mathbf{p} = -s^{LPC}\mathbf{g}$, set `FLAG` = LPC.
else
    choose $s_{\text{NC}} \in [s_{\min}^{NC}, s_{\max}^{NC}]$, set $\mathbf{p} = -s^{NC}\mathbf{g}$, set `FLAG` = NC.
end if
Return $\mathbf{p}$, `FLAG`.

Algorithm 2 MRCG: Scaled Gradient Descent With Line Search

Input: Line search parameter $\rho < 1/2$, termination tolerance $\varepsilon_g > 0$.
while $\|\mathbf{g}_k\| \geq \varepsilon_g$ **do**
    $[\mathbf{p}_k, \texttt{FLAG}] \leftarrow$ Call Algorithm 1 with $\mathbf{H}_k$, $\mathbf{g}_k$ and σ
    For `FLAG` = SPC/LPC, use Algorithm 3 to find $\alpha_k \in (0, 1]$ satisfying 5. For `FLAG` = NC, use Algorithm 4 to find $\alpha_k \in (0, \infty)$ satisfying 5.
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
end while

Algorithm 3 Backward Tracking Line Search

Input: Initial step size $\alpha_0 = 1$, Scaling parameter $0 < \theta < 1$.
$\alpha \leftarrow \alpha_0$
while 5 is not satisfied **do**
    $\alpha \leftarrow \theta\alpha$
end while
return α

‘

A. Oracle-call complexity measure

We plot every method's objective value against the total number of oracle calls, following standard practice in optimization. One forward evaluation of f costs 1 call; computing ∇f costs one additional call (total 2); a Hessian–vector product costs two additional calls (total 4) [4]. Counting oracle calls instead of wall-clock time removes the influence of implementation details and hardware.

B. Multi-class logistic regression on CIFAR 10

a) Dataset and preprocessing: We use the 50 000 training images of CIFAR 10 [5].

Algorithm 4 Forward/Backward Tracking Line Search

Input: Initial step size $\alpha_0 = 1$, Scaling parameter $0 < \theta < 1$.
$\alpha \leftarrow \alpha_0$
if 5 is not satisfied **then**
    Call Algorithm 3
else
    while 5 is satisfied **do**
        $\alpha \leftarrow \alpha/\theta$
    end while
    return $\theta\alpha$
end if

b) Objective: Let $\mathcal{D} = \{(a_i, b_i)\}_{i=1}^n \subset \mathbb{R}^d \times \{1, \ldots, C\}$ denote the training set. For each class $c \in \{1, \ldots, C\}$ we introduce a weight vector $x_c \in \mathbb{R}^{d^+}$ (the final component is the bias). We fix the last class as the reference by setting $\mathbf{x}_C = 0$ and optimise

$$\mathbf{X} = [\mathbf{x}_1^\top, \ldots, \mathbf{x}_{C-1}^\top]^\top \in \mathbb{R}^{d^+(C-1)}.$$

With $\lambda = 10^{-3}$ the regularised cross-entropy objective is

$$f(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{c=1}^{C-1} \mathbf{1}[b_i = c] \left(-\log \operatorname{softmax}\big(\mathbf{x}_c,\, a_i\big) \right) + \frac{\lambda}{2} \|\mathbf{X}\|^2,$$

where

$$\operatorname{softmax}\big(\mathbf{x}_c, a_i\big) = \frac{\exp\big(\langle \mathbf{x}_c,\, a_i \rangle\big)}{\sum_{j=1}^{C} \exp\big(\langle \mathbf{x}_j,\, a_i \rangle\big)} \quad \text{and} \quad \mathbf{x}_C \equiv 0.$$

We initialize at $x_0 = 0$ and include the bias term in every class's weight vector.

c) Convexity and smoothness surrogates: With the Frobenius bound $\|A\|_F^2 \geq \sigma_{\max}(A)^2$,

$$\mu_{\text{approx}} = \lambda, \qquad L_{\text{approx}} = \frac{C-1}{4n} \|A\|_F^2 + \lambda.$$

d) Common stopping rule: Optimization stops when $\|\nabla f(\mathbf{x}_k)\| \leq 10^{-4}$ or after 10^5 oracle calls, whichever comes first.

e) Hyperparameters:

- **Fixed-step GD**: $\alpha = 1/L_{\text{approx}}$.
- **Line-search GD**: Armijo backtracking with $\rho = 10^{-4}$, $\theta = 0.5$, and initial step $\alpha_0 = 1$.
- **Heavy-Ball momentum** [6]:

$$\alpha = \frac{4}{(\sqrt{L_{\text{approx}}} + \sqrt{\mu_{\text{approx}}})^2}, \quad \beta = \left(\frac{\sqrt{L_{\text{approx}}} - \sqrt{\mu_{\text{approx}}}}{\sqrt{L_{\text{approx}}} + \sqrt{\mu_{\text{approx}}}} \right)^2.$$

- **Adam** [7]: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$ and step size $\alpha = 10^{-3}$ (largest power of ten yielding stable convergence).
- **Scaled-gradient methods**: we set $\sigma = 0$ (strong convexity) and use the same backtracking parameters (ρ, θ) as line-search GD.