# A Unified View of Optimizers from an Approximated Curvature Perspective

Zangwei Zheng

zangwei@u.nus.edu

This paper discusses optimizers on the discriminative neural network $f(\theta)$ with cross-entropy loss $\mathcal{L}$ over estimated likelihood $p(x|\theta)$. For simplicity, we do not consider some optimization techniques (weight decay, momentum, etc.). Titles with a star$(*)$ mean there are no real-world corresponding optimizers.

The analysis of optimizers in deep learning has three forms: 1. One-step analysis by a local expansion of $f(\theta)$ 2. Convergence rate by bounding $|\mathcal{L}(\theta) - \mathcal{L}(\theta^*)|$. 3. Analyze SDE by dynamic systems. In this paper, we will focus on the first forms. By analyzing the local behavior, we can have a better knowledge of optimizers.

## 1 Optimization without Stochasticity

**(General Update)** Let $g = \nabla f(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[g_x], H = \nabla^2 f(\theta)$, we consider expansion $f(\theta + d) \approx f(\theta) + g^T d + \frac{1}{2} d^T H d$. Define $\Delta_d = f(\theta) - f(\theta + d) = -(g^T d + \frac{1}{2} d^T H d)$.

Eigenvalue decomposition for $H$ is $H = Q\text{diag}(\lambda_1, \cdots, \lambda_n)Q^T, Q = [e_1, \cdots, e_n]$. By a change of basis, we have $\hat{d} = Q^T d$ and $d = Q\hat{d} = \sum_{i=1}^n \hat{d}_i \vec{e}_i \doteq \sum_i \vec{d}_i$. Then $\Delta_d = -\sum(\hat{g}_i \hat{d}_i + \frac{1}{2}\hat{d}_i^2 \lambda_i) \doteq -\sum \Delta_d^i$, where $\hat{g}_i$ is the decomposition of gradient $g$, $\Delta_d^i$ describes the changes contributed on each eigenvector.

The norm defined by $H$ is $||d||_H^2 = d^T H d$, which is the norm in the space of $[\lambda_1 e_1, \cdots, \lambda_n e_n]$. The Taylor expansion requires a limited norm of $d$. And along each eigenvector, we have $||\vec{d_i}||_H^2 = \lambda_i \hat{d}_i^2$.

Taken together, we will focus on the following values.

| | | |
|---|---|---|
| $d$ | $\Delta_d = -(g^T d + \frac{1}{2} d^T H d)$ | $\|\|d\|\|_H^2 = d^T H d$ |
| $\hat{d}_i$ | $\Delta_d^i = \hat{g}_i \hat{d}_i + \frac{1}{2}\hat{d}_i^2 \lambda_i$ | $\|\|\vec{d_i}\|\|_H^2 = \lambda_i \hat{d}_i^2$ |

Generally speaking, when designing optimizers, we have the following goals:

- Maximize $\Delta_d$.

- Limit $||d||_H^2$ to avoid diverging (too large update), inaccurate Taylor approximation, and large value of directional sharpness (the second term $\frac{1}{2}||d||_H^2$ in $\Delta_d$).
- $||\vec{d_i}||_H^2$ have similar values for a high conditional number problem.
- $\Delta_d^i$ have similar values for a high conditional number problem.

**(Pure Newton's Method)** When $f$ is convex and $H \succeq 0$, maximizing $\Delta_d$ leads to

$$d = -\tilde{g} = -H^{-1}g \qquad \Delta_d = \frac{1}{2}||\tilde{g}||_H^2 \qquad ||d||_H^2 = g^T H^{-1} g$$
$$\hat{d}_i = -\frac{\hat{g}_i}{\lambda_i} \qquad \Delta_d^i = \frac{\hat{g}_i^2}{2\lambda_i} \qquad ||\vec{d_i}||_H^2 = \frac{\hat{g}_i^2}{\lambda_i}$$

During the training of neural networks, most of $H$'s eigenvalues are positive [9]. Thus, we still consider the positive semi-definite $H$ case.

**(Newton's Method)** In convex optimization, we know if $||g||$ is too large, the pure Newton's method may not converge [1]. A learning rate $\alpha \in (0, 1]$ can be introduced to dampen the update.

$$d = -\alpha\tilde{g} \qquad \Delta_d = \alpha(1 - \frac{\alpha}{2})||\tilde{g}||_H^2 \qquad ||d||_H^2 = \alpha^2||\tilde{g}||_H^2$$
$$\hat{d}_i = -\alpha\frac{\hat{g}_i}{\lambda_i} \qquad \Delta_d^i = \alpha(1 - \frac{\alpha}{2})\frac{\hat{g}_i^2}{\lambda_i} \qquad ||\vec{d_i}||_H^2 = \alpha^2\frac{\hat{g}_i^2}{\lambda_i}$$

**(Natural Gradient Descent)** In our setting, the Fisher Information Matrix (FIM) $F$ equals Newton-Gaussian Matrix $G$ [7], and $G$ approximates Hessian $H$ [10], namely $H \approx F = G \succeq 0$. Thus, Newton's Method and the Natural Gradient Descent (NGD) are almost identical here.

In addition, $||d||_{\text{KL}} = \text{KL}[p(\theta)||p(\theta + d)] \approx \frac{1}{2}||d||_H$, which means Newton's method and Natural Gradient Descent have the same distance measure.

Calculating $F = \mathbb{E}_{x\sim p(x|\theta)}[gg^T]$ requires sampling from $p(x|\theta)$ instead of the data distribution $\mathcal{D}$. For computational efficiency, many works use Empirical Fisher in ML $\tilde{F} = \mathbb{E}_{x\sim\mathcal{D}}[gg^T]$. However, $\tilde{F}$ is only a good approximation of $F$ when $f$ is well-learned. From now on, we will use the terms 'Newton's method' and 'Natural Gradient Descent' interchangeably.

**(Normalized Natural Gradient Descent)** If we want the learning rate to control the update length in Newton's method (or KL divergence changes in NGD) by $||\alpha H^{-1}g||_H = \epsilon$, we have $\alpha = \frac{\epsilon}{||\tilde{g}||_H}$. Thus, we have

$$d = -\epsilon\frac{H^{-1}g}{\sqrt{g^T H^{-1} g}} \qquad \Delta_d = \epsilon||\tilde{g}||_H - \frac{\epsilon^2}{2} \qquad ||d||_H^2 = \epsilon^2$$

**(Gradient-Norm Normalized NGD)** In training neural networks, only a few eigenvalues are large while most of them are near 0. Thus we have a high conditional number. If we want the same update length in each eigenbasis $||\vec{d_i}||_H$, by introducing learning rate $s_i$ on each eigenbasis to make $||\vec{d_i}||_H = ||s_i\tilde{g}_i|| = \epsilon\hat{g}_i$. This leads to $s_i = \epsilon\sqrt{\lambda_i}$. To make $\lambda_i \to \sqrt{\lambda_i}$, we can use $H^{-1} \to H^{-1/2}$.

$$d = -\epsilon H^{-1/2} g \qquad \Delta_d = \epsilon(H^{-1/2} - \frac{\epsilon}{2} I)||g||_2 \qquad ||d||_H^2 = \epsilon^2 ||g||_2^2$$

$$\hat{d}_i = -\epsilon \frac{\hat{g}_i}{\sqrt{\lambda_i}} \qquad \Delta_d^i = \epsilon(\frac{1}{\sqrt{\lambda_i}} - \frac{\epsilon}{2})\hat{g}_i^2 \qquad ||\vec{d_i}||_H^2 = \epsilon^2 \hat{g}_i^2$$

. As we can see, the introduced learning rate $s_i$ makes $||d||_H^2$ controlled by the gradient norm. Namely, this method takes advantage of gradient norm length. This serves as an explanation for adaptive optimizers (e.g., Adam [5]). To make $\Delta_d^i > 0$, we have $\epsilon < \frac{2}{\sqrt{\max_i \lambda_i}}$. This conclusion is in accordance with the preconditioned sharpness in the "adaptive edge of stability" [2].

**(Eigenbasis Normalized NGD)*** Since $\hat{g}_i = \frac{g^T \vec{e}_i}{\lambda_i}$, $\vec{e}_i$ with higher $\lambda_i$ tends to have smaller $\hat{g}_i$ and less sensitive to noise. However, we do not know how $\hat{g}_i$ is distributed. Thus, let's further consider a case in which we want to control $||d_i||_H = \epsilon$. Therefore, $s_i = \epsilon \frac{\sqrt{\lambda_i}}{|g_i|}$. Define $\text{sign}(x) = \frac{x}{|x|}$, One possible way to achieve this update is

$$d = -\epsilon H^{-1/2} Q \text{sign}(Q^T g)$$

$$\hat{d}_i = -\epsilon \cdot \text{sign}(\hat{g}_i) \qquad \Delta_d^i = -\epsilon|\hat{g}_i| + \frac{1}{2}\epsilon^2 \lambda_i \qquad ||\vec{d_i}||_H^2 = \epsilon^2$$

A potential benefit here is the directional sharpness along each eigenbasis is decoupled from $g_i$. We are not sure about the performance of this method.

**(Quasi-Newton Method)** There are three main challenges for Newton's method in training:

- $H$ needs $O(n^2)$ storage.
- Calculating $H$ needs $\Theta(kn^2)$, where $k$ is a large constant for calculating one element of $H$.
- Calculating $H^{-1}$ needs $O(n^3)$.

Thus, practically, we use an approximation for calculating the Hessian matrix, which is called the Quasi-Newton method. We summarize approximations into three categories:

- **Calculating Approximation (CA)**: use $G, F, \tilde{F}$ to approximate $H$.
- **Structure Approximation (SA)**: assume $H$ to be diagonal, block-diagonal, block-tridiagonal, etc.

- **Iterative Approximation (IA)**: use an iterative method to update $H$, or use momentum to keep track of $H$. Besides, in the neural network, which contains many function compositions, assumptions can be made to calculate $H$ layer by layer efficiently.

Classic Quasi-Newton methods use the secant method (IA): $H\Delta\theta \approx \Delta\nabla\theta$. Besides, the SR1 method assumes each iterative update for $H$ is rank-one, and the BFGS method uses a rank-two update. However, this still requires $O(n^2)$ computation and storage. L-BFGS [6] uses nearly $k$ $\Delta\theta$ and $\Delta\nabla\theta$ to efficiently calculate $H^{-1}g$ on the fly. However, this requires $O(kn)$ storage, which is still large for neural networks.

**(Gradient Descent)** One way to obtain gradient descent is to consider the expansion $f(\theta + d) \approx f(\theta) + g^T d + \frac{L}{2}||\Delta\theta||_2^2$ for a $L$-smooth $f$. Since $L$ bounds the largest eigenvalue, $\lambda_i < L$, from another perspective, we can view GD as NGD with a SA that $H$ is a diagonal matrix with apriori max eigenvalues $L$, namely $H \approx H' = LI$. Therefore, $||d||_{H'}^2 = L||d||_2^2$ and $\tilde{g} = \frac{g}{L}$.

$$
\begin{aligned}
d = -\frac{\alpha}{L}g \qquad & \Delta'_d = \frac{\alpha}{L}(1 - \frac{\alpha}{2})||g||_2^2 \qquad & ||d||_{H'}^2 = \frac{\alpha^2}{L}||g||_2^2 \\
& \Delta_d = \frac{\alpha}{L}||g||_2^2 - \frac{\alpha^2}{L^2}||g||_H^2 \qquad & ||d||_H^2 = \frac{\alpha^2}{L^2}g^T H g \\
\hat{d}_i = -\frac{\alpha}{L}\hat{g}_i \qquad & \Delta_d^{i'} = \frac{\alpha}{L}(1 - \frac{\alpha}{2})\hat{g}_i \qquad & ||\vec{d_i}||_H^2 = \frac{\alpha^2}{L^2}\lambda_i \hat{g}^2
\end{aligned}
$$

With approximated Hessian $H'$, the first line analyzes the $\Delta'_d$ and $||d||_{H'}^2$ under $H'$, which can be directly obtained by using Newton's method's results. On the other hand, the second line analyzes the real values.

To make $\Delta'_d > 0$, we have $\alpha < 2$. Define learning rate $\eta = \frac{\alpha}{L}$ we have $\eta < \frac{2}{L}$. In addition, the optimal learning rate is $\eta = \frac{||g||^2}{||g||_H^2}$.

**(Normalized Gradient Descent)** With $||d||_{H'} = \epsilon$, we have $\alpha = \frac{\epsilon L^{1/2}}{||g||_2}$.

$$
d = -\epsilon L^{-1/2}\frac{g}{||g||_2} \qquad \Delta'_d = \epsilon\frac{||g||_2^2}{L} - \frac{\epsilon^2}{2} \qquad ||d||_{H'}^2 = \epsilon^2
$$

The new learning rate $\eta' = \epsilon L^{-1/2}$.

**(Sign Gradient Descent)** When approximating $H$ as $LI$, the counterpart for Gradient-Norm Normalized NGD is still Gradient Descent since $||\vec{d_i}||_H^2 = \epsilon\hat{g}_i$ leads to $\hat{d}_i = -\epsilon\frac{\hat{g}_i}{\sqrt{\lambda_i}}$. With $\lambda_i = L$, it is equivalent to use $\alpha = \epsilon\sqrt{L}$.

The counterpart for Eigenbasis Normalized NGD requires $||\vec{d_i}||_H^2 = \epsilon$, which can be obtained by setting $d = -\epsilon H^{-1/2}Q\text{sign}(Q^T d)$. Since here $Q = I$, we have $d = -\epsilon L^{-1/2}\text{sign}(g)$. Let $\eta = \epsilon L^{-1/2}$, we have the sign gradient descent $d = -\eta \cdot \text{sign}(g)$.

Table 1: Optimizers comparison in view of approximated curvature information.

| **Optimizer** | **Length Limitation** | **Calculating Approximation** | **Structure Approximation** | **Iterative Approximation** |
|---|---|---|---|---|
| Newton's Method | | | | |
| Natural Gradient Descent | | $F$ | | |
| Normalized NGD | $\|\|d\|\|_H = \epsilon$ | | | |
| GN-Normalized NGD | $\|\|\vec{d_i}\|\|_H = \epsilon g_i$ | | | |
| Eigenbasis Normalized NGD | $\|\|\vec{d_i}\|\|_H = \epsilon$ | | | |
| Gradient Descent | | | $LI$ | |
| Normalized GD | $\|\|d\|\|_{H'} = \epsilon$ | | $LI$ | |
| Sign GD | $\|\|\vec{d_i}\|\|_{H'} = \epsilon$ | | $LI$ | |
| L-BFGS | | | rank-2 update | iterative (secant assumption) compute on last $k$ items |
| K-FAC | | $\tilde{F}$ | block-diagonal | $a_i a_j \perp \frac{\partial \mathcal{L}}{g_i}\frac{\partial \mathcal{L}}{g_j}$, momentum |
| Shampoo | $\|\|\vec{d_i}\|\|_{H'} = \epsilon g_i$ | $\tilde{F}$ | $L^{-1/4} g R^{-1/4}$ | momentum |
| Adaptive (Adam, Adagrad) | $\|\|\vec{d_i}\|\|_{H'} = \epsilon g_i$ | $\tilde{F}$ | diagonal | momentum |
| AdaHessian | $\|\|\vec{d_i}\|\|_{H'} = \epsilon g_i$ | | diagonal | Monte Carlo, momentum |
| Sophia | | $F$ | diagonal | momentum |

**(Diagonal Natural Gradient Descent)** We approximate $H$ by a diagonal matrix, namely $H = \text{diag}(\lambda_1, \cdots, \lambda_n)$. This is better than the $LI$ matrix, and we still have a bunch of good properties: $Q = I, \|d\|_H^2 = \vec{\lambda}^T d^2$, where $\vec{\lambda} = [\lambda_1, \cdots, \lambda_n]^T$ and $(\cdot)^2$ means element-wise square. $\tilde{g} = g/\vec{\lambda}$, where $\cdot/\cdot$ means elemental-wise division. Thus,

- Natural Gradient Descent: $d = -\alpha g/\vec{\lambda}$, e.g. Sophia [7].
- Normalized NGD: $d = -\frac{\epsilon}{\vec{\lambda}^T d^2} g/\vec{\lambda}$.
- Gradient-Norm Normalized NGD: $d = -\epsilon g/\vec{\lambda}^{1/2}$, e.g., Adagrad [8], Adam [5], AdaHessian [13].
- Eigenbasis Normalized NGD: $d = -\epsilon \cdot \text{sign}(g)/\vec{\lambda}^{1/2}$.

**(Approximated Natural Gradient Descent)** There are also works using other approximations, such as K-FAC [3] and Shampoo [4]. We list optimizers in Table 1 for comparison.

**(Clipped Natural Gradient Descent)** For different versions of Normalized NGD, we can relax the normalization to clipping. Let $C = \frac{\epsilon}{\alpha}$ be the clipping threshold. When we need a learning rate $\alpha \leq \epsilon t$, we use learning rate $\alpha \min(1, Ct)$. Thus:

- Clipped NGD: $\alpha \leq \frac{\epsilon}{||\tilde{g}||_H^2}, d = -\alpha \min(1, \frac{C}{||\tilde{g}||_H^2})\tilde{g}$.
- Gradient-Norm Clipped NGD: $\vec{d_i} = -\alpha \min(1, C\sqrt{\lambda_i})\frac{\hat{g}_i}{\lambda_i}$.
- Eigenbasis Clipped NGD: $\vec{d_i} = -\alpha \min(1, C\frac{\sqrt{\lambda_i}}{|\hat{g}_i|})\frac{\hat{g}_i}{\lambda_i}$.

The first one is the gradient-norm clipping technique. The latter two can only be efficiently calculated with a diagonal Hessian.

For diagonal NGD, Sophia uses an element-wise clipping $\hat{d}_i = -\alpha \min(1, \frac{\lambda_i \epsilon}{|\hat{g}_i|}) \frac{\hat{g}}{\lambda_i}$, which can be seen as an approximated Eigenbasis Clipped NGD.

# 2 Optimization with Stochasticity

**(General Update)** Define noise $\xi = g_x - g$. Since $g = \sum_x g_x$, we have $\mathbb{E}[\xi] = 0$. The covariance matrix $\Sigma_\xi = \mathbb{E}[\xi\xi^T]$. For a batch $b$ with batch size $B$, the noise $\xi_b = \frac{1}{B}\sum_{x\in b}\xi_x$. It is easy to see $\mathbb{E}[\xi_b] = 0, \Sigma_{\xi_b} = \frac{1}{B}\Sigma_\xi$. Similarly, for batch gradient $g_b$ we have $\mathbb{E}[g_b] = g, \Sigma_{g_b} = \frac{1}{B}\Sigma_\xi$.

Thanks to the Central Limit Theorem, with a large batch size, the $\xi$ is subject to Gaussian Distribution [12]. Recent studies show the covariance matrix can be approximated by $\Sigma_\xi \approx kH$ [11], where $H$ is the Hessian matrix, and $k \sim \mathcal{L}(\theta_t)$. Thus, we assume that $\xi_b \sim \mathcal{N}(\vec{0}, \frac{k}{B}H)$.

Generally, the update direction can be expressed as $d_b = Ag_b = Ag + A\xi_b \doteq d + d_\xi$. The component $d_\xi$ has covariance matrix $\mathbb{E}[(A\xi_b)^T(A\xi_b)] = \mathbb{E}[\mathrm{tr}(\xi_b^T A^T A\xi_b)] = \mathrm{tr}(A^T A\Sigma_{\xi_b})$. In a stochastic setting, we care about the expected version $\mathbb{E}[f(\theta + d_b)] \approx f(\theta) + g^T d + \frac{1}{2}d^T Hd + \frac{1}{2}\mathbb{E}[\xi^T H\xi]$. Compared with the case without stochasticity, the noise introduces an additional term $N_b = \frac{1}{2}\mathbb{E}[d_\xi^T H d_\xi] = \frac{1}{2}\mathrm{tr}(HA^T A\Sigma_{\xi_b})$. With approximation, we have $N_b = \frac{1}{2}\frac{k}{B}\mathrm{tr}(HA^T AH)$. Although in one step, we want to reduce $N_b$, some papers point out that higher $N_b$ can better jump out local minima.

If we use Calculating Approximation (CA) like $\tilde{F}$, our estimation of $H$ is inaccurate due to the noise in gradients. First, $F_b = \mathbb{E}_{x\sim p(x|\theta)}[g_b g_b^T] = F + \frac{1}{B}\Sigma_{\hat{\xi}}$, where $\Sigma_{\hat{\xi}} \neq \Sigma_x$ because the sampling distribution is different. Meanwhile, $\tilde{F}_b = \mathbb{E}_{x\sim D}[g_b g_b^T] = \tilde{F} + \frac{1}{B}\Sigma_x$. So, we must consider the error introduced by **Noisy Approximation (NA)**.

We still want to know what happens on each eigenbasis. Similar to the decomposition of $d$, we have $\hat{d}_\xi = Q^T d_\xi, d_\xi = Q\hat{d}_\xi = \sum_{i=1}^n \hat{d}_\xi \vec{e}_i \doteq \sum_i \vec{d}_{\xi i}$. Note that $||\vec{d}_{\xi i}||_2^2 = ||((A\xi)^T\vec{e}_i)\vec{e}_i||_2^2 = \frac{k}{B}\mathrm{tr}(AHA^T\vec{e}_i\vec{e}_i^T)$.

**(Newton's Method)** With $A = \alpha H^{-1}$, we have $N_b = \frac{1}{2}\frac{k}{B}\alpha$ and $||\vec{d}_{\xi i}||_2^2 = \frac{k\alpha}{B\lambda_i}$. While smaller components in the eigenbasis with large eigenvalues make update stable, this on the other hand weakens the ability to jump out of sharp minima.

**(Normalized Natural Gradient Descent)** With $A = \alpha H^{-1/2}$, we have $N_b = \frac{1}{2}\frac{k}{B}\alpha\mathrm{tr}(H)$ and $||\vec{d}_{\xi i}||_2^2 = \frac{k\alpha}{B}$. This makes noise length equal on every eigenbasis.

**(Gradient Descent)** With $A = \alpha I$, we have $N_b = \frac{1}{2}\frac{k}{B}\mathrm{tr}(HH)$, $||\vec{d}_{\xi i}||_2^2 = \frac{k\lambda_i}{B}\alpha$. Since $\mathrm{tr}(H)$ is usually smaller than 1, from Newton's method to gradient descent, we can see the noise term is decreasing ($N_b$) while the length on large eigenvalues grows. This can also explain why SGD flavors flat minima.

# References


[1] S. Boyd and L. Vandenberghe, "Convex optimization," in 1st ed. New York, NY, USA: Cambridge University Press, 2004, ch. 9-10.

[2] J. M. Cohen *et al.*, "Adaptive gradient methods at the edge of stability," *arXiv preprint arXiv:2207.14484*, 2022.

[3] R. Grosse and J. Martens, "A kronecker-factored approximate fisher matrix for convolution layers," in *International Conference on Machine Learning*, PMLR, 2016, pp. 573–582.

[4] V. Gupta, T. Koren, and Y. Singer, "Shampoo: Preconditioned stochastic tensor optimization," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1842–1850.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[6] D. Liu and J. Nocedal, "On the limited memory method for large scale optimization: Mathematical programming b," 1989.

[7] H. Liu, Z. Li, D. Hall, P. Liang, and T. Ma, "Sophia: A scalable stochastic second-order optimizer for language model pre-training," *arXiv preprint arXiv:2305.14342*, 2023.

[8] Y. Liu, W. J. Su, and T. Li, "On quantum speedups for nonconvex optimization via quantum tunneling walks," *Quantum*, vol. 7, p. 1030, 2023.

[9] L. Sagun, U. Evci, V. U. Guney, Y. Dauphin, and L. Bottou, "Empirical analysis of the hessian of over-parametrized neural networks," *arXiv preprint arXiv:1706.04454*, 2017.

[10] A. R. Sankar, Y. Khasbage, R. Vigneswaran, and V. N. Balasubramanian, "A deeper look at the hessian eigenspectrum of deep neural networks and its applications to regularization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 9481–9488.

[11] M. Wang and L. Wu, "The noise geometry of stochastic gradient descent: A quantitative and analytical characterization," *arXiv preprint arXiv:2310.00692*, 2023.

[12] Y. Wu, R. Luo, C. Zhang, J. Wang, and Y. Yang, "Revisiting the characteristics of stochastic gradient noise and dynamics," *arXiv preprint arXiv:2109.09833*, 2021.

[13] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," in *proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 10 665–10 673.