addHessian: Combining quasi-Newton method with first-order method for neural network training

Sota Yasuda\textsuperscript{1a)}, S. Indrapriyadarsini\textsuperscript{2}, Hiroshi Ninomiya\textsuperscript{3}, Takeshi Kamio\textsuperscript{4}, and Hideki Asai\textsuperscript{5}

\begin{itemize}
  \item \textsuperscript{1}Graduate School of Integrated Science and Technology, Shizuoka University, 3-5-1 Johoku, Naka-ku, Hamamatsu, Shizuoka 432-8561, Japan
  \item \textsuperscript{2}Graduate School of Science and Technology, Shizuoka University, 3-5-1 Johoku, Naka-ku, Hamamatsu, Shizuoka 432-8561, Japan
  \item \textsuperscript{3}Graduate School of Electrical and Information Engineering, Shonan Institute of Technology, 1-1-25 Tsujido-nishikaigan, Fujisawa, Kanagawa 251-8511, Japan
  \item \textsuperscript{4}Graduate School of Information Sciences, Hiroshima City University, 3-4-1 Ozukahigashi, Asaminami-ku, Hiroshima 731-3194, Japan
  \item \textsuperscript{5}Research Institute of Electronics, Shizuoka University, 3-5-1 Johoku, Naka-ku, Hamamatsu, Shizuoka 432-8561, Japan
\end{itemize}

\textsuperscript{a)}yasuda.sota.16@shizuoka.ac.jp

Received October 18, 2021; Revised December 17, 2021; Published April 1, 2022

Abstract:
First-order methods such as SGD and Adam are popularly used in training Neural networks. On the other hand, second-order methods have shown to have better performance and faster convergence despite their high computational cost by incorporating the curvature information. While second-order methods determine the step size by line search approaches, first-order methods achieve efficient learning by devising a way to adjust the step size. In this paper, we propose a new learning algorithm for training neural networks by combining first-order and second-order methods. We investigate the effectiveness of our proposed method when combined with popular first-order methods - SGD, Adagrad, and Adam, through experiments using image classification problems.

Key Words: neural networks, training algorithms, quasi-Newton method;

1. Introduction
Neural networks (NN) have been widely used in several large-scale applications. Most of these applications use large amounts of data for training the NN model. In such large-scale applications, a full batch approach though effective can pose serious limitations on the memory resources and computation cost. Thus training NNs
using large datasets has been a challenging task and stochastic or mini-batch training strategies are preferred in large-scale optimization problems. Several studies have been focusing on effective stochastic optimization methods. Choice of the optimization algorithm used for training the NN plays an important role. Gradient-based algorithms are popularly used in training NNs and can be broadly categorized into first and second-order methods. First-order methods such as SGD [1], Adagrad [2] and Adam [3] are popularly used due to their simplicity and low computational complexity. On the other hand, second-order methods such the quasi-Newton methods [4] have shown faster convergence with the use of the Hessian information. Both first and second-order methods have been actively researched upon for stochastic settings. A common problem in stochastic training is variance due to the updates being made based on gradient estimates obtained from small sub-samples (mini-batch) of the training data. Another problem in stochastic optimization is the choice of the learning rate. Most of the popular first-order methods implement effective adaptive learning rate strategies while second-order methods use line-search techniques or simple decay schedule. In this paper, we propose a new learning algorithm for training NNs by combining first-order and second-order methods. Our proposed algorithm thus benefits from the adaptive learning rate of first-order methods and fast convergence of second-order methods while attempting variance reduction.

2. Background

NN training is an iterative process where the parameters of the network \( w_t \) are updated in order to minimize the error function

\[
E(w) = \frac{1}{b} \sum_{i \in X} E_i(w),
\]

where \( X \) is the training dataset with \( n = |X| \) samples, \( b \) is batch size, chosen such that \( b \ll n \) and \( E_i(w) \) is the error evaluated on the mini-batch sample \( i \). The parameters of the NN are updated iteratively as \( w_{t+1} = w_t + \nu_{t+1} \), where \( \nu_{t+1} \) is the update vector of the corresponding optimization algorithm. Optimization algorithms form the core of training NNs. The stochastic gradient descent (SGD) [1] is the simplest gradient based algorithm used in training NNs. SGD though simple, exhibits slow convergence and thus several algorithms have been proposed to speed up the training and improve the overall performance of the algorithm. Other popular first order methods such as the Adagrad [2] and Adam [3] methods devise strategies to adaptively tune the learning rate by using the history of past gradients. Second order methods have shown to further improve convergence despite their high computational cost. Furthermore, several methods have been proposed to effectively update the Hessian or second order curvature information while taking into consideration the storage and computational cost [4-6]. Also, methods such as the stochastic variance reduced gradient (SVRG), SVRG+II [7, 8] aim at reducing the variance introduced as a result of stochastic updates. There are other methods such as AdaHessian [9] that adaptively tune the Hessian matrix by using an estimate of the diagonal component of the Hessian matrix. It achieves low computational cost and reduces noise. Recently, SVRG OL [10] proposed a strategy that combines adaptivity and variance reduction by using nested loops - the outer loop computes the full gradient and contributes to variance reduction, and the inner loop performs SGD updates using the full and mini-batch gradient. Similarly, the Lookahead optimizer [11] also proposes a nested loop strategy by computing stochastic weight updates in the inner loop, and then computing the overall update in the outer loop using exponential moving averages to achieve stable learning. Inspired by these two methods, we propose a new stochastic algorithm - addHessian that uses the second order curvature information along with the first order update strategies, thus combining the adaptive learning rate of first-order methods and fast convergence of second-order methods while attempting variance reduction. We investigate the performance of our proposed algorithm with different first order methods and compare them with the conventional methods. In the following sub-sections, we briefly explain the update rules of the algorithms used in comparison.

Stochastic Gradient Descent

The stochastic gradient descent (SGD) method is one of the earliest and simplest stochastic gradient based algorithms. Its update vector \( \nu_t \) is shown as \( \nu_{t+1} = -\alpha_t \nabla E_i(w_t) \). The learning rate \( \alpha_t \) determines the step size along the direction of the gradient \( \nabla E_i(w_t) \). The step size \( \alpha_t \) is usually fixed with a small value such as 0.001 or updated with a simple decay strategy.
Adagrad

Adagrad method adjusts the learning rate adaptively based on the sum of past gradients and the update vector is as shown in Eq. (2) where \( v_{t+1,j} \) and \( \nabla E_t(w_{t,j}) \) are the  \( j \)-th elements of \( v_{t+1} \), \( w_t \) and \( \nabla E_t(w_t) \), respectively. \( \alpha \) is a global step size shared by all dimensions. The recommended value of \( \alpha \) is 0.001.

\[ v_{t+1,j} = -\frac{\alpha}{\sqrt{\sum_{i=1}^{t} (\nabla E_i(w_{i,j}))^2}} \nabla E_t(w_{t,j}). \]  

Adam

Adam uses the first and second moments of the gradient to adaptively tune the learning rates. That is, it uses bias corrected exponentially decaying average of past squared gradients \( \hat{\theta} \), and exponentially decaying average of past gradients \( \hat{m} \). The update equations of Adam are shown below. \( \epsilon = 10^{-8} \) and \( \beta_1 \) and \( \beta_2 \) are chosen to be 0.9 and 0.999, respectively.

\[ v_{t+1} = -\frac{\alpha \hat{m}_t}{\sqrt{\hat{\theta}_t} + \epsilon}, \]  

where \( \hat{m}_t = (\beta_1 m_{t-1} + (1-\beta_1) \nabla E_t(w_t)) / (1-\beta_1^t) \), and \( \hat{\theta}_t = (\beta_2 \theta_{t-1} + (1-\beta_2) (\nabla E_t(w_t))^2) / (1-\beta_2^t) \).  

Stochastic quasi-Newton Methods

Quasi-Newton methods achieve superlinear or quadratic convergence by incorporating the second-order curvature information. The Broyden-Fletcher-Goldfarb-Shannon (BFGS) algorithm is one of the most popular quasi-Newton methods. The update vector of the quasi-Newton method is given as \( u_t \) which contributes to adaptive learning rate. The hessian matrix \( H_t \) is symmetric positive definite and is iteratively approximated by the following BFGS formula [4].

\[ H_{t+1} = (I - s_T y / y_s) H_t (I - y_s T / y_T s) + s T y / y_T s, \]  

where \( I \) is the identity matrix, \( s_t, y_t \) are the curvature information vectors, and are given as \( s_t = w_t - w_{t-1} \), and \( y_t = \nabla E_t(w_t) - \nabla E_t(w_{t-1}) \). As the scale of the NN model increases, the cost of storing and updating the Hessian matrix \( H_t \) is expensive. Thus, a limited memory variant LBFGS designed for solving large-scale optimization problems is used. In the limited memory version, the Hessian matrix is defined by applying \( m \) BFGS updates using only the last \( m \) curvature pairs \( s_t, y_t \). As a result, the storage and computational cost is significantly reduced. The stochastic or online BFGS method [5] is a fast and scalable stochastic quasi-Newton method that was proposed to work well in a stochastic setting.

3. Proposed Method: addHessian

Both the first-order and second-order methods proposed thus far have their own advantages for use in training NNs. In this paper, we propose a new algorithm - addHessian, which is a combination of first-order and second-order methods. The proposed addHessian method is shown in Algorithm 1. As shown in the algorithm, the inner loop computes the stochastic gradients and the inner loop weights \( \hat{x}_{t+1} \) are updated using a first order algorithm \( A(f) \) which contributes to adaptive learning rate. The update vector \( u_t \) calculated as the difference of \( \hat{x}_{t+1} \) and \( \hat{x}_t \) is used in the final addHessian weight update as shown in step 11. In other words, the update equation for addHessian is the same as Eq. (9), with \( u_t \) in place of \( -\alpha r \nabla E(w_t) \). Since \( u_t \) is calculated from first order methods, the learning rate scheme is inherently incorporated. The limited memory scheme uses the two-loop recursion thereby reducing computation cost from \( d^2 \) to \( 6md \) and storage cost to \( 2md \). Further, since the curvature information is updated based on the full gradient calculated once in \( n/Lb \) iterations, the Hessian is less noisy. Here, \( n \) is the number of training samples, \( b \) is the batch size and \( L \) is the frequency of full
4. Simulation Results

We evaluate the performance of our proposed addHessian algorithm on CIFAR-10 [12] and MNIST [13] image classification problems using the LeNet-5 [14] architecture. We evaluate the performance of SGD, Adagrad and Adam methods when used as the inner loop optimizer in addHessian. All the methods were implemented on Tensorflow and evaluated against the conventional SGD, Adagrad, Adam, SVRG+II and oLBFGS methods over 10 independent trials with a batch size of $b = 128$ and limited memory $m = 4$. All hyperparameters were set to their default values.

4.1 Results on CIFAR-10 dataset

CIFAR-10 is a popular image classification dataset with 10 classes of $32 \times 32$ pixel images such as cat, dog, car, truck, etc. We first validate our proposed method by using SGD and Adagrad as the inner loop optimizer $A(f_t)$. We set the number of full gradients per epoch frequency $L = 1$. We compare the performance of addHessian+SGD and addHessian+Adagrad with the original SGD, Adagrad and second order SVRG+II and oLBFGS algorithms. The maximum number of epochs was set to 500. Table I shows the average, best, worst and median train and test accuracies at the end of 500 epochs over 10 independent trials. Fig. 1 shows the average train and test accuracy results over 500 epochs. From the results, we could confirm that our proposed addHessian could speed up and perform better than the original SGD, Adagrad, SVRG+II, and oLBFGS methods. Also, it was observed that the performance of SVRG+II dropped after around 50 epochs, which could be due to overtraining and overfitting. Upon further increasing the epoch to around 1000, we observed overfitting when this model was trained with Adam, SGD and addHessian. The overfitting problem can be remedied by architectural changes such as batch normalization, dropout and weight decay. It has been experimentally confirmed that the performance of the proposed method improves when some overfitting measures are taken, but

\begin{algorithm}
\caption{Proposed addHessian}
\begin{algorithmic}[1]
\State \textbf{Initialize}: $\Omega_0$ and $w_0$ with 10 runs of SVRG
\For{$k = 1, 2, \ldots, k_{max}$}
\State Compute full gradient $\Omega_k \leftarrow \nabla E(w_k)$
\State $s_k = w_k - w_{k-1}$, $y_k = \Omega_k - \Omega_{k-1} + s_k$
\State $x_0 \leftarrow w_k$, $\hat{x}_0 \leftarrow w_k$
\For{$i = 1, 2, \ldots, i_{max} = n/(Lb)$}
\State Sample $i$ from $X$ uniformly random
\State $f_i = \nabla E(x_i)$
\State Compute $\hat{x}_{i+1}$ using $A(f_i)$
\State $u_i = \hat{x}_{i+1} - \hat{x}_i$
\State Compute $H_k u_i$ using $s_k$, $y_k$ (Algorithm 2)
\State $x_{i+1} \leftarrow x_i + H_k u_i$
\EndFor
\State $w_{k+1} \leftarrow x_{i+1}$
\EndFor
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\caption{Two-loop Recursion}
\begin{algorithmic}[1]
\Require: gradient vector $u_i$, curvature pair $(S, Y)$ buffer
\State $\tau = \min(k, m)$
\For{$j = \tau, \ldots, 2, 1$}
\State $\sigma_j = (s_j^T u_i)/(y_j^T s_j)$
\State $g_j = u_i - \sigma_j y_j$
\EndFor
\State $g = \frac{1}{\tau} \sum_{j=1}^{\tau} s_j^T y_{k-j}$
\State $\beta = (y_j^T g_j)/(y_j^T s_j)$
\State $g_j = g_j - (\sigma_j - \beta) s_j$
\EndFor
\State $\eta_i = \gamma g$
\State $\text{return } \eta_i$
\end{algorithmic}
\end{algorithm}

\begin{table}[h]
\centering
\caption{Results on CIFAR-10 over 10 independent trials of 500 epochs}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{Type} & \textbf{Train Accuracy (%)} & \textbf{Test Accuracy (%)} \\
& Avg/Best/Worst/Median & Avg/Best/Worst/Median \\
\hline
SGD & Original & 60.74 / 63.33 / 58.18 / 60.85 & 56.67 / 58.89 / 54.39 / 56.94 \\
& addHessian & 74.11 / 77.95 / 70.05 / 73.60 & 59.96 / 61.53 / 55.90 / 60.68 \\
\hline
Adagrad & Original & 51.37 / 53.76 / 49.04 / 51.16 & 50.34 / 52.09 / 48.61 / 50.26 \\
& addHessian & 58.94 / 62.24 / 55.94 / 59.37 & 55.46 / 57.85 / 52.58 / 55.88 \\
\hline
SVRG+II & Original & 10.00 / 10.00 / 10.00 / 10.00 & 10.00 / 10.00 / 10.00 / 10.00 \\
& oLBFGS & 47.87 / 56.67 / 10.00 / 52.52 & 45.12 / 54.09 / 10.00 / 49.33 \\
\hline
\end{tabular}
\end{table}
further verification of the impact of the proposed method on overfitting is needed. However, since it is out of scope of this paper, we evaluated the performance of a simple LeNet-5 architecture when training is terminated early at 500 epochs. Thus, for further validation of our proposed method we use the example of MNIST.

4.2 Results on 28x28 MNIST dataset
The LeNet-5 architecture was originally proposed for the MNIST classification problem. Thus, we further evaluate our proposed addHessian on the popular 28 × 28 MNIST dataset with LeNet-5. The data set consists of 55,000 training samples and 10,000 test samples. In this example, we evaluate the performance of SGD, Adam and Adagrad when used in combination with addHessian in comparison with the conventional SGD, Adam, Adagrad, SVRG+II and oLBFGS methods. The number of full gradients per-epoch frequency is set to \( L = 5 \). Table II shows the average, best, worst and median train and test accuracy over 10 independent trials of 150 epochs. The results shows that the performance of the original method can be improved when used in combination with our proposed addHessian method. Fig. 2 shows the average train and test accuracy over 150 epochs. From the figure, it is clear that the proposed addHessian is able to achieve higher accuracies within fewer epochs compared to the plain original methods. Thus we can conclude that the proposed addHessian can speed up the NN training.

5. Conclusion
This paper proposed a new training algorithm addHessian that combines first-order and second-order methods. The effectiveness of the proposed method was confirmed on the CIFAR-10 and MNIST image classification problems using the LeNet-5 architecture. The experimental results show that the proposed method accelerates the learning process and improve the accuracy compared to the conventional methods while maintaining a moderate amount of computation. We compare the oBFGS method with the addHessian method. To account for the full gradient computation, we consider the computational cost per epoch. The error function and

| Type         | Train Accuracy (%) | Test Accuracy (%) |
|--------------|--------------------|-------------------|
|              | Avg/Best/Worst/Med | Avg/Best/Worst/Med |
| SGD          |                    |                   |
| Original     | 96.78/97.53/96.06 | 96.49/97.35/95.75 |
| addHessian   | 96.79/97.52/96.28 | 96.51/97.42/95.88 |
| Adam         |                    |                   |
| Original     | 99.98/100.00/99.90 | 98.76/98.97/98.56 |
| addHessian   | 99.96/100.00/99.87 | 98.77/99.01/98.51 |
| Adagrad      |                    |                   |
| Original     | 92.98/93.98/91.59 | 92.66/94.19/91.28 |
| addHessian   | 94.84/95.80/93.64 | 94.33/95.32/93.39 |
| SVRG+II      |                    |                   |
| Original     | 27.70/97.06/9.90  | 27.66/96.51/9.80  |
| oLBFGS       | 97.64/98.41/96.70 | 96.96/97.43/95.96 |

![Fig. 1: Results on CIFAR-10 with \( b = 128, m = 4 \) and \( L = 1 \)](image-url)
Gradient evaluations can be considered to be $nd$ and $d^2$ is necessary for Hessian calculation. Therefore, the computational cost of oBFGS is $2nd + nd^2/b$ and that of addHessian is $(L+1)nd + 6mbd/b$. The computational cost of addHessian depends on $L$, but it is comparable to other second-order methods. Further evaluation of the proposed method on larger models and datasets will be studied in future works.

![Graph](image)

Fig. 2: Results on MNIST with $b = 128$, $m = 4$ and $L = 5$

**References**

[1] H. Robbins, and S. Monro, “A stochastic approximation method.” *The annals of mathematical statistics*, 400-407, 1951.

[2] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of machine learning research*, vol. 12, pp. 21212159, July 2011.

[3] D.P. Kingma, and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint* arXiv:1412.6980, December 2014.

[4] J. Nocedal, and S.J. Wright, “Numerical optimization,” Springer, second edition, 2006.

[5] N.N. Schraudolph, J. Yu, and S. Gnter, “A stochastic quasi-Newton method for online convex optimization.” *Artificial intelligence and statistics*, PMLR, 2007.

[6] R.H. Byrd, S.L. Hansen, J. Nocedal, and Y. Singer, “A stochastic quasi-newton method for large-scale optimization,” *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 10081031, 2016

[7] R. Johnson and T. Zhang, “Accelerating stochastic gradient descent using predictive variance reduction.” *Advances in neural information processing systems* 26, 315-323, 2013.

[8] R. Kolte, M. Erdogdu, and A. Ozgur, “Accelerating SVRG via second-order information,” *NIPS workshop on optimization for machine learning*, 2015.

[9] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M.W. Mahoney, “ADAHESSIAN: An adaptive second order optimizer for machine learning,” *arXiv preprint* arXiv:2006.00719, 2020.

[10] A. Cutkosky, and R. Busa-Fekete, “Distributed stochastic optimization via adaptive SGD,” *arXiv preprint* arXiv:1802.05811, 2018.

[11] M.R. Zhang, J. Lucas, G. Hinton, and J. Ba, “Lookahead optimizer: k steps forward, 1 step back,” *arXiv preprint* arXiv:1907.08610, July 2019.

[12] A. Krizhevsky, and G. Hinton, “Learning multiple layers of features from tiny images,” 7, 2009.

[13] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database,” *AT&T Labs [Online], Available: http://yann.lecun.com/exdb/mnist*, 2010.

[14] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, “Object recognition with gradient-based learning,” In *Shape, contour and grouping in computer vision*, pp 319-345, Springer, Berlin, Heidelberg, 1999.