EA-CG: An Approximate Second-Order Method for Training Fully-Connected Neural Networks

Sheng-Wei Chen
HTC Research & Healthcare
sw_chen@htc.com

Chun-Nan Chou
HTC Research & Healthcare
jason.cn_chou@htc.com

Edward Y. Chang
HTC Research & Healthcare
edward_chang@htc.com

Abstract

For training fully-connected neural networks (FCNNs), we propose a practical approximate second-order method including: 1) an approximation of the Hessian matrix and 2) a conjugate gradient (CG) based method. Our proposed approximate Hessian matrix is memory-efficient and can be applied to any FCNNs where the activation and criterion functions are twice differentiable. We devise a CG-based method incorporating one-rank approximation to derive Newton directions for training FCNNs, which significantly reduces both space and time complexity. This CG-based method can be employed to solve any linear equation where the coefficient matrix is Kronecker-factored, symmetric and positive definite. Empirical studies show the efficacy and efficiency of our proposed method.

Introduction

Neural networks have been applied to solving problems in several application domains such as computer vision (He et al. 2016), natural language processing (Hochreiter and Schmidhuber 1997), and disease diagnosis (Chang et al. 2017). Training a neural network requires tuning its model parameters using Backpropagation. Stochastic gradient descent (SGD), Broyden-Fletcher-Goldfarb-Shanno and one-step secant are representative algorithms that have been employed for training in Backpropagation.

To date, SGD is widely used due to its low computational demand. SGD minimizes a function using the function's first derivative, and has been proven to be effective for training large models. However, stochasticity in the gradient will slow down convergence for any gradient method such that none of them can be asymptotically faster than simple SGD with Polyak averaging (Polyak and Juditsky 1992). Besides gradients, second-order methods utilize the curvature information of a loss function within the neighborhood of a given point to guide the update direction. Since each update becomes more precise, such methods can converge faster than first-order methods in terms of update iterations.

For solving a convex optimization problem, a second-order method can always converge to the global minimum in much fewer steps than SGD. However, the problem of neural-network training can be non-convex, thereby suffering from the issue of negative curvature. To avoid this issue, the common practice is to use the Gauss-Newton matrix with a convex criterion function (Schraudolph 2002) or the Fisher matrix to measure curvature since both are guaranteed to be positive semi-definite (PSD).

Although these two kinds of matrices can alleviate the issue of negative curvature, computing either the exact Gauss-Newton matrix or Fisher matrix even for a modestly-sized fully-connected neural network (FCNN) is intractable. Intuitively, the analytic expression for the second derivative requires $O(N^2)$ computations if $O(N)$ complexity is required to compute the first derivative. Thus, several pioneer works (LeCun et al. 1998; Amari, Park, and Fukumizu 2000; Schraudolph 2002) have used different methods to approximate either matrix. However, none of these methods have been shown to be computationally feasible and fundamentally more effective than first-order methods as reported in (Martens 2010). Thus, there has been a growing trend towards conceiving more computationally feasible second-order methods for training FCNNs.

We outline several notable works in chronological order herein. Martens proposed a truncated-Newton method for training deep auto-encoders. In this work, Martens used an $\mathcal{R}$-operator (Pearlmutter 1994) to compute full Gauss-Newton matrix-vector products and made good progress within each update. Martens and Grosse developed a block-diagonal approximation to the Fisher matrix for FCNNs, called Kronecker-factored Approximation Curvature (KFAC). They derived the update directions by exploiting the inverse property of Kronecker products. KFAC features that its cost in storing and inverting the devised approximation does not depend on the amount of data used to estimate the Fisher matrix. The idea is further extended to convolutional nets (Grosse and Martens 2016). Recently, Botev, Ritter, and Barber presented a block-diagonal approximation to the Gauss-Newton matrix for FCNNs, referred to as Kronecker-Factored Recursive Approximation (KFRA). They also utilized the inverse property of Kronecker products to derive the update directions. Similarly, Zhang et al. introduced a block-diagonal approximation of the Gauss-Newton matrix and used conjugate gradient (CG) method to derive the update directions.

However, these prior works either impose constraints on their applicability or require considerable computation in terms of memory or running time. On the one hand, sev-

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

eral of these notable methods relying on the Gauss-Newton matrix face the essential limit that they cannot handle non-convex criterion functions playing an important part in some problems. Use robust estimation in computer vision (Stewart 1999) as an example. Some non-convex criterion functions such as Tukey's biweight function are robustness to outliers and perform better than convex criterion functions (Belagiannis et al. 2015). On the other hand, in order to derive the update directions, some of the methods utilizing the conventional CG method may take too much time, and the others exploiting the inverse property of Kronecker products may require excessive memory space.

To remedy the aforementioned issues, we propose a block-diagonal approximation of the positive-curvature Hessian (PCH) matrix, which is memory-efficient. Our proposed PCH matrix can be applied to any FCNN where the activation and criterion functions are twice differentiable. Particularly, our proposed PCH matrix can handle non-convex criterion functions, which the Gauss-Newton methods cannot. Besides, we incorporate expectation approximation into the CG-based method, which is dubbed EA-CG, to derive update directions for training FCNNs in mini-batch setting. EA-CG significantly reduces the space and time complexity of the conventional CG method. Our experimental results show the efficacy and efficiency of our proposed method.

In this work, we focus on deriving a second-order method for training FCNNs since the shared weights of convolutional layers lead to the difficulties in factorizing its Hessian. We defer tackling convolutional layers to our future work. We also focus on the classification problem and hence do not consider auto-encoders. Our strategy is that once a simpler isolated problem can be effectively handled, we can then extend our method to address more challenging issues.

In summary, the contributions of this paper are as follows:

1. For curvature information, we propose the PCH matrix to improve the Gauss-Newton matrix for training FCNNs with convex criterion functions and overcome the non-convex scenario.
2. To derive the update directions, we devise effective EA-CG method, which does converge faster in terms of wall clock time and enjoys better testing accuracy than competing methods. Specially, the performance of EA-CG is competitive with SGD.

Truncated-Newton Method on Non-Convex Problems

Newton's method is one of the second-order minimization methods, and is generally composed of two steps: 1) computing the Hessian matrix and 2) solving the system of linear equations for update directions. The truncated-Newton method applies the CG method with restricted iterations to the second step of Newton's method. In this section, we first introduce the truncated-Newton method in the context of convex problems. Afterwards, we discuss the non-convex scenario of the truncated-Newton method and provide an important property that lays the foundation of our proposed PCH matrix.

Suppose we have a minimization problem

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}), \tag{1}$$

where f is a convex and twice-differentiable function. Since the global minimum is at the point that the first derivative is zero, the solution $\boldsymbol{\theta}^*$ can be derived from the equation

$$\nabla f(\boldsymbol{\theta}^*) = 0. \tag{2}$$

We can utilize a quadratic polynomial to approximate Problem 1 by conducting a Taylor expansion with a given point $\boldsymbol{\theta}^j$. Then, the problem turns out to be

$$\min_{\boldsymbol{d}} f(\boldsymbol{\theta}^j + \boldsymbol{d}) \approx f(\boldsymbol{\theta}^j) + \nabla f(\boldsymbol{\theta}^j)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \nabla^2 f(\boldsymbol{\theta}^j)\boldsymbol{d},$$

where $\nabla^2 f(\boldsymbol{\theta}^j)$ is the Hessian matrix of f at $\boldsymbol{\theta}^j$. After applying the aforementioned approximation, we rewrite Eq. (2) as the linear equation

$$\nabla f(\boldsymbol{\theta}^j) + \nabla^2 f(\boldsymbol{\theta}^j)\boldsymbol{d}^j = 0. \tag{3}$$

Therefore, the Newton direction is obtained via

$$\boldsymbol{d}^j = -\nabla^2 f(\boldsymbol{\theta}^j)^{-1} \nabla f(\boldsymbol{\theta}^j),$$

and we can acquire $\boldsymbol{\theta}^*$ by iteratively applying the update rule

$$\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^j + \eta \boldsymbol{d}^j,$$

where η is the step size.

For non-convex problems, the solution to Eq. (2) reflects one of three possibilities: a local minimum $\boldsymbol{\theta}_{\text{min}}$, a local maximum $\boldsymbol{\theta}_{\text{max}}$ or a saddle point $\boldsymbol{\theta}_{\text{saddle}}$. Some previous works such as (Dauphin et al. 2014; Goldfarb et al. 2017; Mizutani and Dreyfus 2008) utilize negative curvature information to converge to a local minimum. Before illustrating how these previous works tackle the issue of negative curvature, we have to introduce a crucial concept that we can know the curvature information of f at a given point $\boldsymbol{\theta}$ by analyzing the Hessian matrix $\nabla^2 f(\boldsymbol{\theta})$. On the one hand, the Hessian matrix of f at any $\boldsymbol{\theta}_{\text{min}}$ is positive semi-definite. On the other hand, the Hessian matrices of f at any $\boldsymbol{\theta}_{\text{max}}$ and $\boldsymbol{\theta}_{\text{saddle}}$ are negative semi-definite and indefinite, respectively. After establishing the concept, we can use the following property to understand how to utilize the negative curvature information to resolve the issue of negative curvature.

Property 1. *Let f be a non-convex and twice-differentiable function. With a given point $\boldsymbol{\theta}^j$, we suppose that there exist some negative eigenvalues $\{\lambda_1, \ldots, \lambda_s\}$ for $\nabla^2 f(\boldsymbol{\theta}^j)$. Moreover, we take $V = span(\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_s\})$, which is the eigenspace corresponds to $\{\lambda_1, \ldots, \lambda_s\}$. If we take*

$$g(\boldsymbol{k}) = f(\boldsymbol{\theta}^j) + \nabla f(\boldsymbol{\theta}^j)^T \boldsymbol{v} + \frac{1}{2}\boldsymbol{v}^T \nabla^2 f(\boldsymbol{\theta}^j)\boldsymbol{v},$$

where $\boldsymbol{k} \in \mathbb{R}^s$ and $\boldsymbol{v} = k_1\boldsymbol{v}_1 + \ldots + k_s\boldsymbol{v}_s$, then $g(\boldsymbol{k})$ is a concave function.

According to **Property 1**, Eq. (3) may lead us to a local maximum or a saddle point if $\nabla^2 f(\boldsymbol{\theta}^j)$ has some negative eigenvalues. In order to converge to a local minimum, we substitute Pos-Eig$(\nabla^2 f(\boldsymbol{\theta}^j))$ for $\nabla^2 f(\boldsymbol{\theta}^j)$, where

Pos-Eig(A) is conceptually defined as replacing the negative eigenvalues of A with non-negative ones. That is,

$$\text{Pos-Eig}(A) = Q^T \begin{bmatrix} \gamma\lambda_1 & & & & & \\ & \ddots & & & & \\ & & \gamma\lambda_s & & & \\ & & & \lambda_{s+1} & & \\ & & & & \ddots & \\ & & & & & \lambda_n \end{bmatrix} Q,$$

where γ is a given scalar that is less than or equal to zero, and $\{\lambda_1, \ldots, \lambda_s\}$ and $\{\lambda_{s+1}, \ldots, \lambda_n\}$ are the negative and non-negative eigenvalues of A, respectively. This refinement implies that the point $\boldsymbol{\theta}^{j+1}$ escapes from either local maxima or saddle points if $\gamma < 0$. In case of $\gamma = 0$, this refinement means that the eigenspace of the negative eigenvalues is ignored. As a result, we do not converge to any saddle point or local maximum. In addition, every real symmetric matrix can be diagonalized according to the spectral theorem. Under our assumptions, $\nabla^2 f(\boldsymbol{\theta}^j)$ is a real symmetric matrix. Thus, $\nabla^2 f(\boldsymbol{\theta}^j)$ can be decomposed, and the function "Pos-Eig" can be realized easily.

When the number of variables in f is large, the Hessian matrix becomes intractable with respect to space complexity. Alternatively, we can utilize the CG method to solve Eq. (3). This alternative only requires calculating the Hessian-vector products rather than storing the whole Hessian matrix. Moreover, to save computation costs, it is desirable to restrict the iteration number of the CG method.

Computing the Hessian Matrix

For second-order methods we must compute the curvature information, and we utilize the Hessian matrix to capture the curvature information in our work. However, the Hessian matrix for training FCNNs is intrinsically complicated and intractable. Recently, Botev, Ritter, and Barber presented the idea of block Hessian recursion, in which the diagonal blocks of the Hessian matrix can be computed in a layer-wise manner. As the basis of our proposed PCH matrix, we first establish some notation for training FCNNs and reformulate the block Hessian recursion with our notation. Then, we present the steps to integrate the approximation concept proposed by (Martens and Grosse 2015) into our reformulation.

Fully-Connected Neural Networks

An FCNN with k layers takes an input vector $\boldsymbol{h}_i^0 = \boldsymbol{x}_i$, where $\boldsymbol{x}_i$ is the ith instance in the training set. For the ith instance, the activation values in the other layers can be recursively derived from: $\boldsymbol{h}_i^t = \sigma(\boldsymbol{W}^t \boldsymbol{h}_i^{t-1} + \boldsymbol{b}^t), t = 1, \ldots, k-1$, where σ is the activation function and can be any twice differentiable function, and $\boldsymbol{W}^t$ and $\boldsymbol{b}^t$ are the weights and biases in the tth layer, respectively. We further denote n_t as the number of the neurons in the tth layer, where $t = 0, \ldots, k$, and collect all the model parameters including all the weights and biases in each layer as $\boldsymbol{\theta} = (\text{Vec}(\boldsymbol{W}^1), \boldsymbol{b}^1, \ldots, \text{Vec}(\boldsymbol{W}^k), \boldsymbol{b}^k)$, where $\text{Vec}(A) = \begin{bmatrix}[A_{.1}]^T & [A_{.2}]^T & \cdots & [A_{.n}]^T\end{bmatrix}^T$. By following the notation mentioned above, we denote an FCNN output with k layers as $\boldsymbol{h}_i^k = F(\boldsymbol{\theta}|\boldsymbol{x}_i) = \boldsymbol{W}^k \boldsymbol{h}_i^{k-1} + \boldsymbol{b}^k$.

To train this FCNN, we must decide a loss function ξ that can be any twice differentiable function. Training this FCNN can therefore be interpreted as solving the following minimization problem:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{l} \xi(\boldsymbol{h}_i^k \mid \boldsymbol{y}_i) \equiv \min_{\boldsymbol{\theta}} \sum_{i=1}^{l} C(\hat{\boldsymbol{y}}_i \mid \boldsymbol{y}_i),$$

where l is the number of the instances in the training set, $\boldsymbol{y}_i$ is the label of the ith instance, $\hat{\boldsymbol{y}}_i$ is softmax$(\boldsymbol{h}_i^k)$, and C is the criterion function.

Layer-wise Equations for the Hessian Matrix

For lucid exposition of the block Hessian recursion, we start by reformulating the equations of Backpropagation according to the notation defined in the previous subsection. Please note that we separate the bias terms ($\boldsymbol{b}^t$) from the weight terms ($\boldsymbol{W}^t$) and treat each of them individually during backward propagation of gradients. The gradients of ξ with respect to the bias and weight terms can be derived from our reformulated equations in a layer-wise manner, similar to the original Backpropagation method. For the ith instance, our reformulated equations are as follows:

$$\begin{aligned} \nabla_{\boldsymbol{b}^k} \xi_i &= \nabla_{\boldsymbol{h}_i^k} \xi_i \\ \nabla_{\boldsymbol{b}^{t-1}} \xi_i &= \text{diag}(\boldsymbol{h}_i^{(t-1)'}) \boldsymbol{W}^{tT} \nabla_{\boldsymbol{b}^t} \xi_i \\ \nabla_{\boldsymbol{W}^t} \xi_i &= \nabla_{\boldsymbol{b}^t} \xi_i \otimes \boldsymbol{h}_i^{(t-1)T} \end{aligned}$$

where $\xi_i = \xi(\boldsymbol{h}_i^k \mid \boldsymbol{y}_i)$, $\otimes$ is the Kronecker product, and $\boldsymbol{h}_i^{(t-1)'} = \nabla_{\boldsymbol{z}} \sigma(\boldsymbol{z})|_{\boldsymbol{z} = \boldsymbol{W}^{t-1} \boldsymbol{h}_i^{t-2} + \boldsymbol{b}^{t-1}}$. Likewise, we strive to propagate the Hessian matrix of ξ with respect to the bias and weight terms backward in a layer-wise manner. This can be achieved by utilizing the Kronecker product and following the similar fashion above. The resultant equations for the ith instance are as follows:

$$\nabla^2_{\boldsymbol{b}^k} \xi_i = \nabla^2_{\boldsymbol{h}_i^k} \xi_i \tag{4a}$$

$$\begin{aligned} \nabla^2_{\boldsymbol{b}^{t-1}} \xi_i = &\text{diag}(\boldsymbol{h}_i^{(t-1)'}) \boldsymbol{W}^{tT} \nabla^2_{\boldsymbol{b}^t} \xi_i \boldsymbol{W}^t \text{diag}(\boldsymbol{h}_i^{(t-1)'}) \\ &+ \text{diag}(\boldsymbol{h}_i^{(t-1)''} \odot (\boldsymbol{W}^{tT} \nabla_{\boldsymbol{b}^t} \xi_i)) \end{aligned} \tag{4b}$$

$$\nabla^2_{\boldsymbol{W}^t} \xi_i = (\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}) \otimes \nabla^2_{\boldsymbol{b}^t} \xi_i, \tag{4c}$$

where $\odot$ is the element-wise product, $\left[\boldsymbol{h}_i^{(t-1)''}\right]_s = \left[\nabla^2_{\boldsymbol{z}} \sigma(\boldsymbol{z})\big|_{\boldsymbol{z} = \boldsymbol{W}^{t-1} \boldsymbol{h}_i^{t-2} + \boldsymbol{b}^{t-1}}\right]_{ss}$, and the derivative order of $\nabla^2_{\boldsymbol{W}^t} \xi_i$ is column-wise traversal of $\boldsymbol{W}^t$. Moreover, it is worth noting that the original block Hessian recursion unified the bias and weight terms, which is distinct from our separate treatment of these terms.

Expectation Approximation

Martens and Grosse propose one approximation concept that is referred as *expectation approximation* in (Botev, Ritter,

and Barber 2017). The idea behind expectation approximation is that the covariance between $[\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}]_{uv}$ and $[\nabla_{\boldsymbol{b}^t}\xi_i \otimes \nabla_{\boldsymbol{b}^t}\xi_i{}^T]_{\mu\nu}$ with given indices (u, v) and (μ, ν) is shown to be tiny and thus ignored due to computational efficiency, i.e.,

$$\begin{aligned}&\mathbb{E}_i[[\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}]_{uv} \cdot [\nabla_{\boldsymbol{b}^t}\xi_i \otimes \nabla_{\boldsymbol{b}^t}\xi_i{}^T]_{\mu\nu}]\\ \approx&\mathbb{E}_i[[\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}]_{uv}] \cdot \mathbb{E}_i[[\nabla_{\boldsymbol{b}^t}\xi_i \otimes \nabla_{\boldsymbol{b}^t}\xi_i{}^T]_{\mu\nu}].\end{aligned}$$

To explain this concept on our formulations, we define cov-t as Ele-Cov$((\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}) \otimes \mathbb{1}_{n_t,n_t}, \mathbb{1}_{n_{t-1},n_{t-1}} \otimes \nabla^2_{\boldsymbol{b}^t}\xi_i)$, where "Ele-Cov" is denoted as element-wise covariance, and $\mathbb{1}_{u,v}$ is the matrix whose elements are 1 in $\mathbb{R}^{u\times v}$, $t = 1, \ldots, k$. With the definition of cov-t and our devised equations in the previous subsection, the approximation can be interpreted as follows:

$$\begin{aligned}\mathbb{E}_i[\nabla^2_{\boldsymbol{W}^t}\xi_i] =&\text{EhhT}^{t-1} \otimes \mathbb{E}_i[\nabla^2_{\boldsymbol{b}^t}\xi_i] + \text{cov-}t\\ \approx&\text{EhhT}^{t-1} \otimes \mathbb{E}_i[\nabla^2_{\boldsymbol{b}^t}\xi_i],\end{aligned} \tag{5}$$

where $\text{EhhT}^{t-1} = \mathbb{E}_i[\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}]$.

Botev, Ritter, and Barber also adopted expectation approximation in their proposed method. Similarly, we integrate this approximation into our proposed layer-wise Hessian matrix equations, thereby resulting in the following approximation equation:

$$\begin{aligned}&\mathbb{E}_i[\nabla^2_{\boldsymbol{b}^{t-1}}\xi_i]\\ \approx&\mathbb{E}_i[\text{diag}(\boldsymbol{h}_i^{(t-1)'})\boldsymbol{W}^{tT}\mathbb{E}_i[\nabla^2_{\boldsymbol{b}^t}\xi_i]\boldsymbol{W}^t\text{diag}(\boldsymbol{h}_i^{(t-1)'})\\ &+\text{diag}(\boldsymbol{h}_i^{(t-1)''} \odot (\boldsymbol{W}^{tT}\nabla_{\boldsymbol{b}^t}\xi_i))]\\ =&(\boldsymbol{W}^{tT}\mathbb{E}_i[\nabla^2_{\boldsymbol{b}^t}\xi_i]\boldsymbol{W}^t) \odot \text{EhhT}^{(t-1)'}\\ &+\mathbb{E}_i[\text{diag}(\boldsymbol{h}_i^{(t-1)''} \odot (\boldsymbol{W}^{tT}\nabla_{\boldsymbol{b}^t}\xi_i))].\end{aligned} \tag{6}$$

where $\text{EhhT}^{(t-1)'} = \mathbb{E}_i[\boldsymbol{h}_i^{(t-1)'} \otimes \boldsymbol{h}_i^{(t-1)'T}]$. The difference between the original and the approximate Hessian matrices in Eq. (6) is bounded by

$$\begin{aligned}&\left\|\text{Ele-Cov}(\boldsymbol{W}^{tT}\nabla^2_{\boldsymbol{b}^t}\xi_i\boldsymbol{W}^t, \boldsymbol{h}_i^{(t-1)'} \otimes \boldsymbol{h}_i^{(t-1)'T})\right\|_F^2\\ \leq&L^4\sum_{\mu,\nu}\text{Var}([\boldsymbol{W}^{tT}\nabla^2_{\boldsymbol{b}^t}\xi_i\boldsymbol{W}^t]_{\mu\nu}),\end{aligned} \tag{7}$$

where L is the Lipschitz constant of activation functions. For example, L_{ReLU} and L_{sigmoid} are 1 and 0.25, respectively.

Deriving the Newton Direction

Now we present a computationally feasible method to train FCNNs with Newton directions. First, we explain the ways to construct a PCH matrix. Based on PCH matrices, we propose an efficient CG-based method incorporating the expectation approximation to derive Newton directions for multiple training instances, which we call EA-CG. Finally, we provide an analysis of the space and time complexity for EA-CG.

PCH Matrix

Based on our layer-wise equations in the previous section and the integration of expectation approximation, we can construct block matrices that vary in size and are located at the diagonal of the Hessian matrix. We denote this block-diagonal matrix $\mathbb{E}_i[\overline{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$ as $\text{diag}(\mathbb{E}_i[\nabla^2_{\boldsymbol{W}^1}\xi_i], \mathbb{E}_i[\nabla^2_{\boldsymbol{b}^1}\xi_i], \ldots, \mathbb{E}_i[\nabla^2_{\boldsymbol{W}^k}\xi_i], \mathbb{E}_i[\nabla^2_{\boldsymbol{b}^k}\xi_i])$. Please note that $\mathbb{E}_i[\overline{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$ is a block-diagonal Hessian matrix and not the complete Hessian matrix. According to the explanation for the three possibilities of update directions in aforementioned section, $\mathbb{E}_i[\overline{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$ is required to be modified. Thus, we replace $\mathbb{E}_i[\overline{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$ with $\text{diag}(\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^1}}\xi_i], \mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^1}}\xi_i], \ldots, \mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^k}}\xi_i], \mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^k}}\xi_i])$ and denote the modified result as $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$, where

$$\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^k}}\xi_i] =\text{Pos-Eig}(\mathbb{E}_i[\nabla^2_{\boldsymbol{h}_i^k}\xi_i]) \tag{8a}$$

$$\begin{aligned}\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^{t-1}}}\xi_i] =&(\boldsymbol{W}^{tT}\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]\boldsymbol{W}^t) \odot \text{EhhT}^{(t-1)'}\\ &+\text{Pos-Eig}(\text{diag}(\mathbb{E}_i[\boldsymbol{h}_i^{(t-1)''} \odot (\boldsymbol{W}^{tT}\nabla_{\boldsymbol{b}^t}\xi_i)]))\end{aligned} \tag{8b}$$

$$\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i] =\text{EhhT}^{t-1} \otimes \mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]. \tag{8c}$$

We call $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$ the block-diagonal approximation of positive curvature Hessian (PCH) matrix. Any PCH matrix can be guaranteed to be PSD, which we explain in the following.

In order to show $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$ is PSD, we have to show that for any t, both $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]$ and $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i]$ are PSD. First, we consider the block matrix $\mathbb{E}_i[\nabla^2_{\boldsymbol{h}_i^k}\xi_i]$ that is a n_k by n_k square matrix in Eq. (8a). If the criterion function $C(\hat{\boldsymbol{y}}_i \mid \boldsymbol{y}_i)$ is convex, $\mathbb{E}_i[\nabla^2_{\boldsymbol{h}_i^k}\xi_i]$ is a PSD matrix. Otherwise, we decompose the matrix and replace the negative eigenvalues. Fortunately n_k is usually not very large, so $\mathbb{E}_i[\nabla^2_{\boldsymbol{h}_i^k}\xi_i]$ can be decomposed quickly[1] and modified to a PSD matrix $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{h}_i^k}}\xi_i]$. Second, suppose that $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]$ is a PSD matrix, then $(\boldsymbol{W}^{tT}\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]\boldsymbol{W}^t) \odot \text{EhhT}^{(t-1)'}$ is PSD. Consequently, the negative eigenvalues of $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]$ stems form the diagonal part $\text{diag}(\mathbb{E}_i[\boldsymbol{h}_i^{(t-1)''} \odot (\boldsymbol{W}^{tT}\nabla_{\boldsymbol{b}^t}\xi_i)])$, so we take the Pos-Eig function for this diagonal part in Eq. (8b). Third, because the Kronecker product of two PSD matrices is PSD, it implies $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i]$ is PSD.

Solving Linear Equation via EA-CG

After obtaining a PCH matrix $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$, we derive the update direction by solving the linear equation

$$((1-\alpha)\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{\theta}}}\xi_i] + \alpha I)\boldsymbol{d}_{\boldsymbol{\theta}} = -\mathbb{E}_i[\nabla_{\boldsymbol{\theta}}\xi_i], \tag{9}$$

where $0 < \alpha < 1$ and $\boldsymbol{d}_{\boldsymbol{\theta}} = [\boldsymbol{d}_{\boldsymbol{W}^1}^T\ \boldsymbol{d}_{\boldsymbol{b}^1}^T \cdots \boldsymbol{d}_{\boldsymbol{W}^k}^T\ \boldsymbol{d}_{\boldsymbol{b}^k}^T]^T$. Here, we use the weighted average of $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$ and an identity matrix I because this average turns the coefficient matrix

[1] In our experience, the decomposition of an 1000×1000 matrix can be done within a few seconds in PyTorch.

Table 1: The comparisons of second-order methods.

	(Martens and Grosse 2015)	(Botev, Ritter, and Barber 2017)	Ours
Criterion Func.	Non-Convex	Convex	Non-Convex
Curvature Info.	Fisher	Gauss-Newton	PCH
Time Required	$O(\lvert\text{Batch}\rvert \times \sum_{t=1}^{k} n_t^2)$	$O(\sum_{t=2}^{k} n_{t-1}^2 n_t)$	$O(\sum_{t=2}^{k} n_{t-1}^2 n_t)$
Solving Scheme	KFI	KFI	EA-CG
Space Required	$O(\sum_{t=0}^{k-1} n_t^2 + \sum_{t=1}^{k} n_t^2)$	$O(\sum_{t=0}^{k-1} n_t^2 + \sum_{t=1}^{k} n_t^2)$	$O(\sum_{t=0}^{k-1} n_t + \sum_{t=1}^{k} n_t^2)$
Time Required	$O(\sum_{t=1}^{k}[n_t^3 + n_{t-1}^3 + n_{t-1}^2 n_t + n_{t-1} n_t^2])$	$O(\sum_{t=1}^{k}[n_t^3 + n_{t-1}^3 + n_{t-1}^2 n_t + n_{t-1} n_t^2])$	$O(\lvert\text{CG}\rvert \times \sum_{t=1}^{k}[n_t^2 + 2n_t n_{t-1}])$

of Eq. (9) that is PSD to positive definite and thus makes the solutions more stable. Due to the essence of diagonal blocks, Eq. (9) can be decomposed as

$$((1-\alpha)\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i] + \alpha I)\boldsymbol{d}_{\boldsymbol{b}^t} = -\,\mathbb{E}_i[\nabla_{\boldsymbol{b}^t}\xi_i] \tag{10a}$$

$$((1-\alpha)\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i] + \alpha I)\boldsymbol{d}_{\boldsymbol{W}^t} = -\,\text{Vec}(\mathbb{E}_i[\nabla_{\boldsymbol{W}^t}\xi_i]), \tag{10b}$$

for $t = 1, \ldots, k$. To solve Eq. (10a), we attain the solutions by using the CG method directly. For Eq. (10b), storing $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i]$ is not an efficient way, so we apply the equation $(C^T \otimes A)\text{Vec}(B) = \text{Vec}(ABC)$ and Eq. (5) to have the Hessian-vector product with a given vector $\text{Vec}(P)$:

$$\begin{aligned}
&\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i]\text{Vec}(P) \\
=&\text{Vec}(\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i] \cdot P \cdot \mathbb{E}_i[\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}]) && \text{(11a)} \\
\approx&\text{Vec}(\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i] \cdot P \cdot \mathbb{E}_i[\boldsymbol{h}_i^{t-1}] \otimes \mathbb{E}_i[\boldsymbol{h}_i^{(t-1)T}]). && \text{(11b)}
\end{aligned}$$

Based on Eq. (11a), we derive the Hessian-vector products of $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i]$ via $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]$, thereby reducing the space complexity of storing the curvature information. Furthermore, applying the CG method becomes much more efficient with Eq. (11b), which we call EA-CG. The details are elaborated in the following subsections.

Analysis of Space Complexity

By considering the devised method in aforementioned subsection, we analyze the required space complexity to store the distinct types of the curvature information. The original Newton's method requires space to store $\mathbb{E}_i[\nabla^2_{\boldsymbol{\theta}}\xi_i]$, so that its space complexity is $O([\sum_{t=1}^{k} n_{t-1}(n_t+1)]^2)$. If we consider the PCH matrix $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{\theta}}}\xi_i]$, the space complexity turns out to be $O(\sum_{t=1}^{k}[(n_{t-1}n_t)^2 + n_t^2])$. According to Eq. (11a), it is not necessary to store $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{W}^t}}\xi_i]$ anymore because $\mathbb{E}_i[\widehat{\nabla^2_{\boldsymbol{b}^t}}\xi_i]$ is sufficient to derive the solution to Eq. (9) with the CG method. Thus, the space complexity is reduced to $O(\sum_{t=1}^{k} n_t^2)$.

Analysis of Additional Time Complexity

In this subsection, we elaborate on the additional time complexity introduced in our proposed method. In contrast to the SGD, our method contributes more computation to the training process of FCNNs. The extra computation mainly originates from two portions of our method: the propagation of the curvature information and the computation of EA-CG method. To propagate the curvature information, we are required to perform the matrix-to-matrix products twice, which is more computationally expensive than propagating gradients. The time complexity of propagating the curvature information with Eq. (6) can be estimated as $O(\sum_{t=1}^{k}[n_{t-1}n_t(n_{t-1}+n_t)])$. We also have the extra cost involved in applying EA-CG method. The complexity of EA-CG method mainly stems from the Hessian-vector products Eq. (11b). Regarding Eq. (11b), we must conduct the matrix-vector products twice and the vector-vector outer products once in order to acquire the Hessian-vector product. Thus, the time complexity pertaining to the truncated-Newton method is $O(\lvert\text{CG}\rvert \times \sum_{t=1}^{k}[n_t(2n_{t-1}+n_t)])$, where $\lvert\text{CG}\rvert$ is the number of iterations of EA-CG method.

Related Works

In this section, we elaborate on the differences between our work and two closely related works using the notation established in the previous sections.

Martens and Grosse developed KFAC by considering the FCNNs with the convex criterion and non-convex activation functions. KFAC utilized $(\mathbb{E}_i[\tilde{F}_i] + \alpha I)$ where $\tilde{F}_i$ is the Fisher matrix (for any training instance $\boldsymbol{x}_i$) to measure the curvature and used the Khatri-Rao product to rewrite $\tilde{F}_i$, which yields the following equation:

$$[\tilde{F}_i]_{\mu\nu} = (\boldsymbol{h}_i^{\mu-1} \otimes \boldsymbol{h}_i^{(\mu-1)T}) \otimes (\nabla_{\boldsymbol{b}^\nu}\xi_i \otimes \nabla_{\boldsymbol{b}^\nu}\xi_i^T).$$

Since it is difficult to find the inverse of $(\mathbb{E}_i[\tilde{F}_i] + \alpha I)$, KFAC substitutes a block-diagonal matrix $\hat{F}_i$ for $\tilde{F}_i$ and thus has the formulation:

$$\begin{aligned}
\mathbb{E}_i[\hat{F}_i^t] =& \mathbb{E}_i[(\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}) \otimes (\nabla_{\boldsymbol{b}^t}\xi_i \otimes \nabla_{\boldsymbol{b}^t}\xi_i^T)] \\
\approx& \mathbb{E}_i[\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}] \otimes \mathbb{E}_i[\nabla_{\boldsymbol{b}^t}\xi_i \otimes \nabla_{\boldsymbol{b}^t}\xi_i^T],
\end{aligned}$$

where $t = 1, \ldots, k$. To derive $(\mathbb{E}_i[\hat{F}_i^t] + \alpha I)^{-1}$ efficiently, KFAC comes up with the approximation

$$(\mathbb{E}_i[\hat{F}_i^t] + \alpha I)^{-1} \approx (\mathcal{H}^t)^{-1} \otimes (\mathcal{G}^t)^{-1}, \tag{12}$$

where $(\mathcal{H}^t)^{-1} = (\mathbb{E}_i[\boldsymbol{h}_i^{t-1} \otimes \boldsymbol{h}_i^{(t-1)T}] + \pi_t\sqrt{\alpha}I)^{-1}$, $(\mathcal{G}^t)^{-1} = (\mathbb{E}_i[\nabla_{\boldsymbol{b}^t}\xi_i \otimes \nabla_{\boldsymbol{b}^t}\xi_i^T] + (\sqrt{\alpha}/\pi_t)I)^{-1}$. Therefore,

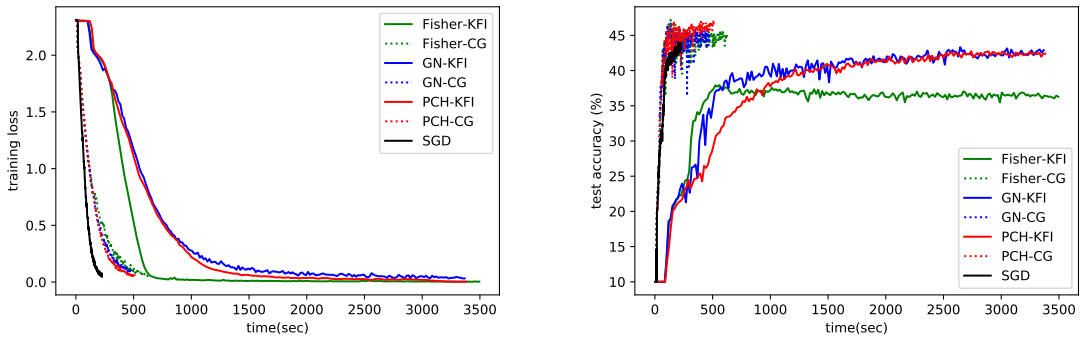

Figure 1: Comparison of different curvature information and solving methods for the convex criterion function "cross-entropy" on "Cifar-10".

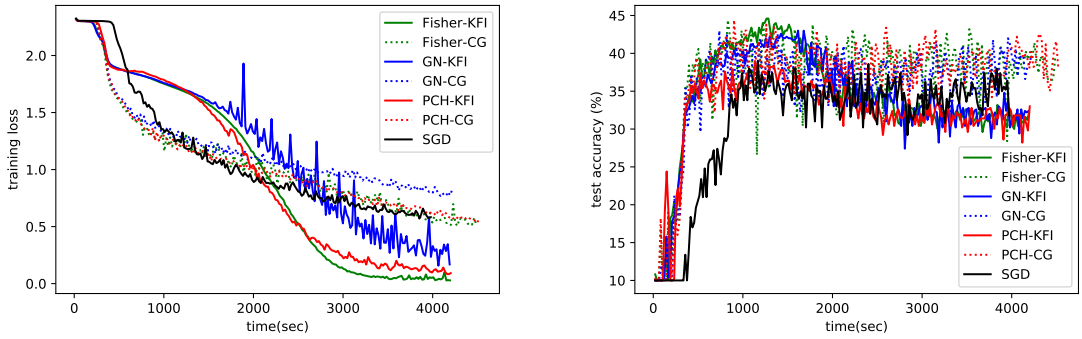

Figure 2: Comparison of different curvature information and solving methods for the convex criterion function "cross-entropy" on "ImageNet-10".

the update directions of KFAC can be acquired via the following equation:

$$\begin{aligned}\hat{\boldsymbol{d}}_{\boldsymbol{W}^t} &= -\left((\mathcal{H}^t)^{-1} \otimes (\mathcal{G}^t)^{-1}\right) \cdot \mathrm{Vec}(\mathbb{E}_i[\nabla_{\boldsymbol{W}^t}\xi_i]) \\ &= -\mathrm{Vec}((\mathcal{G}^t)^{-1}\mathbb{E}_i[\nabla_{\boldsymbol{W}^t}\xi_i](\mathcal{H}^t)^{-1}),\end{aligned}$$

and we refer this type of inverse methods as Kronecker-Factored Inverse (KFI) methods in this paper. In contrast, we derive the update directions by applying EA-CG method. Moreover, KFAC uses the Fisher matrix, while we use the PCH matrix.

Based on the discussion of the FCNNs with convex criterion functions and piecewise linear activation functions, Botev, Ritter, and Barber developed KFRA. As a result, the diagonal term disappears in Eq. (4b). Thus, the Gauss-Newton matrix becomes no different from the Hessian matrix. KFRA also employs KFI method, and hence the update direction of KFRA can be derived from

$$\tilde{\boldsymbol{d}}_{\boldsymbol{W}^t} = -\mathrm{Vec}((\tilde{\mathcal{G}}^t)^{-1}\mathbb{E}_i[\nabla_{\boldsymbol{W}^t}\xi_i](\mathcal{H}^t)^{-1}),$$

where $(\tilde{\mathcal{G}}^t)^{-1} = (\mathbb{E}_i[\mathrm{GN}(\nabla^2_{\boldsymbol{b}^t}\xi_i)] + (\sqrt{\alpha}/\pi_t)I)^{-1}$, and GN stands for Gauss-Newton matrix. The method of deriving the update directions is again different between KFRA and our work. In addition, KFRA still works under the non-convex activation functions, but it exists the difference between Gauss-Newton matrix and Hessian matrix.

Table 1 highlights the three main components of these two related works and our method. As shown in Table 1, KFRA cannot handle non-convex criterion functions due to the curvature information they utilized. The Gauss-Newton matrix becomes indefinite if the criterion function is non-convex. Note that the difference between the original and approximated matrices always exists in Eq. (12) if $\mathbb{E}_i[\hat{F}_i^t]$ is not diagonal. Moreover, the time complexity of KFI method is similar to EA-CG method.

Experimental Evaluation

Our empirical studies aim to examine not only three different types of curvature information including the Fisher, Gauss-Newton and PCH matrices, but also the two solving methods, i.e., KFI and EA-CG methods, in terms of training loss and testing accuracy. We encompassed SGD with momentum as the baseline and fixed its momentum to 0.9. For better comparison, we considered FCNNs with either convex or non-convex criterion functions and conducted our experiments[2] on the image datasets that comprise Cifar-10

[2]Experiments were implemented by using PyTorch libraries and run on a GTX-1080Ti GPU.

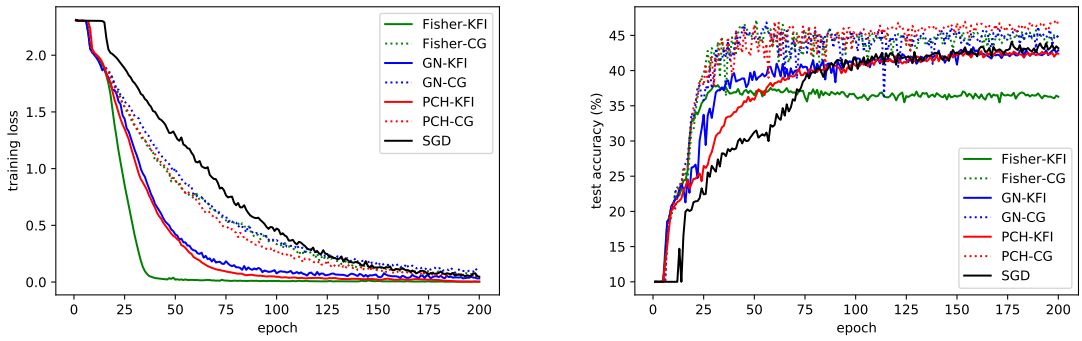

Figure 3: Comparison of different curvature information and solving methods for the convex criterion function "cross-entropy" on "Cifar-10".

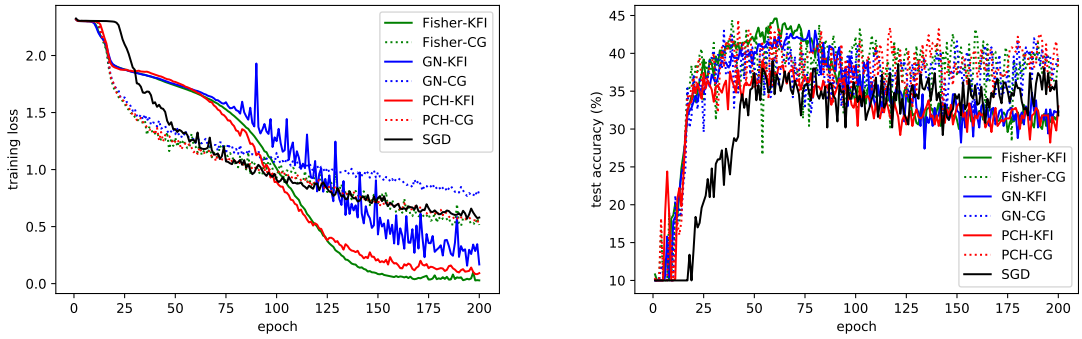

Figure 4: Comparison of different curvature information and solving methods for the convex criterion function "cross-entropy" on "ImageNet-10".

(Krizhevsky and Hinton 2009) and ImageNet-10[3]. The network structures are "3072-1024-512-256-128-64-32-16-10" and "150528-1024-512-256-128-64-32-16-10" for Cifar-10 and ImageNet-10, respectively. For the PCH matrices, we explored two possible scenarios of Eq.(8b): 1) taking the absolute values of the diagonal part, i.e., $\gamma = -1$ and 2) applying $\max(x, 0)$ function to the diagonal part, i.e., $\gamma = 0$. The first and second scenarios of the PCH matrices are separately dubbed PCH-1 and PCH-2. Since PCH-1 and PCH-2 produced the similar results of training loss and testing accuracy, we only reported PCH-1 in the corresponding figures below for the sake of simplicity. In all of our experiments, we used sigmoid as the non-convex activation function of FCNNs and trained the networks with 200 epochs, i.e., seeing the entire training samples 200 times. Furthermore, we utilized "Xavier" initialization method (Glorot and Bengio 2010) and performed the grid search on learning rate $= [0.05, 0.1, 0.2]$, $|\text{Batch}| = [100, 500, 1000]$ and $\alpha = [0.01, 0.02, 0.05, 0.1]$. Regarding EA-CG method, we have to determine two hyper-parameters that control its stopping conditions. One is the maximal iteration number, $\max|\text{CG}|$; the other is the constant of the related error bound, ϵ_{CG}. Thus, we also performed the grid search on $\max|\text{CG}| = [5, 10, 20, 50]$ and $\epsilon_{\text{CG}} = [10^{-10}, 10^{-5}, 10^{-2}, 10^{-1}]$. Finally, to further investigate the different types of curvature information, we calculate the errors caused by approximating the true Hessian in a layer-wise fashion.

[3]We randomly choose ten classes from the ImageNet (Deng et al. 2009) dataset.

Convex Criterion

In the first type of experiments, we examined the performance of different second-order methods. According to the prior section, the differences between these second-order methods originate from two parts. The first part is the curvature information, e.g., the Fisher matrix $\hat{F}_i$ in KFAC, and the other is the solving method for the linear equations. In these experiments, we used cross-entropy as the convex criterion function.

It is noteworthy that the original KFI method runs out of GPU memory for ImageNet-10 because the dimension of the features (n_0) in ImageNet-10 is too high, which conforms with our analysis in Table 1. Thus, we utilized

$$\mathcal{H}^1 \approx \mathbb{E}_i[\boldsymbol{h}_i^0] \otimes \mathbb{E}_i[\boldsymbol{h}_i^{0T}] + \pi_1 \sqrt{\alpha} I \tag{13}$$

and Sherman-Morrison formula to derive the update directions. Albeit this approximation worked in our ImageNet-10 experiments, we observed that this approximation is not stable for other datasets and models that we have explored.

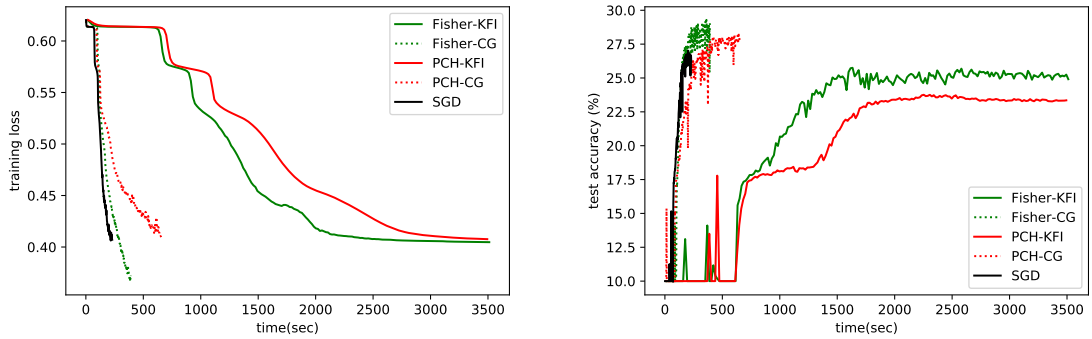

Figure 5: Comparison of different curvature information and solving methods for the non-convex criterion function "Eq. (14)" on "Cifar-10". The Gauss-Newton matrix is removed from this figure since it is not PSD in this type of experiments.

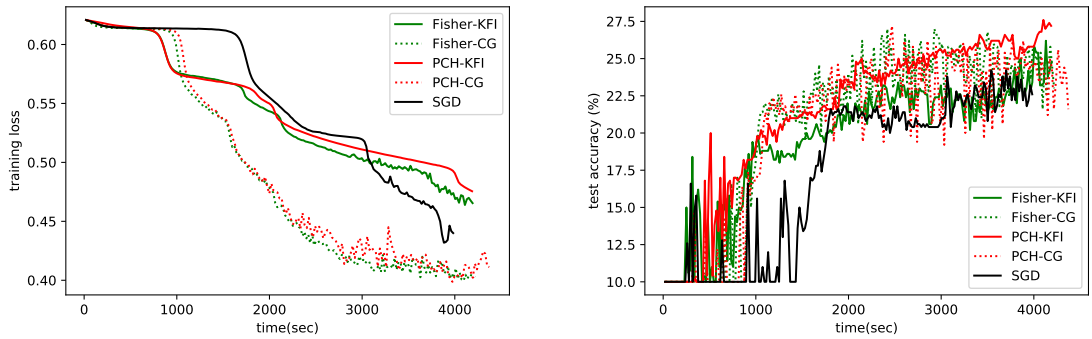

Figure 6: Comparison of different curvature information and solving methods for the non-convex criterion function "Eq. (14)" on "ImageNet-10". The Gauss-Newton matrix is removed from this figure since it is not PSD in this type of experiments.

Wall Clock Time As shown in Figure 1, EA-CG method converges faster with respect to wall clock time and has better performance of testing accuracy than KFI method for Cifar-10. The runtime behavior adheres to Table 1. We also notice that the different types of curvature information with EA-CG method have the similar performance. In contrast, KFI method using the Fisher matrix converges faster than using the other two types of matrices. For ImageNet-10, Figure 2 exhibits that EA-CG and KFI methods take almost the same amount of time for 200 epochs. Besides, as shown in Figure 2, KFI method has lower training loss but may suffer from the overfitting issue. Please note that we applied Eq (13) to the first layer of the neural network for KFI method. Otherwise, the original KFI method ran out of GPU memory for ImageNet-10. However, applying this approximation reduces the memory usage of KFI method but expedites KFI method accordingly.

It is worth noting that second-order methods performed comparable to SGD in Figure 2, but the phenomenon is completely different in Figure 1. This is because the image sizes of Cifar-10 and ImageNet-10 are varied. The image size of ImageNet-10 is $224 \times 224 \times 3$, and hence propagating the gradients of the weights backward is much more expensive than it in Cifar-10 whose image size is $32 \times 32 \times 3$. Based on this premise, the extra cost of propagating our proposed PCH where only the Hessian of the bias terms is required to propagate is comparatively low. Similarly, the cost of solving the update directions is comparatively low.

Epoch As shown in Figure 3, KFI method provides a precise descent direction of each epoch, so the training loss decreases faster in terms of epochs. In contrast, Figure 1 shows KFI method converges slower in terms of wall clock time, which varied with different implementation. We argue that we have already done our best effort to implement KFI method by using the inverse function in PyTorch. As shown in either Figure 1 or Figure 3, EA-CG method has the better results of testing accuracy for Cifar-10. For ImageNet-10, the results of Figure 4 are consistent with the results of Figure 2. Thus, we do not repeat the same narrative here.

Non-Convex Criterion

In order to demonstrate our capability of handling non-convex criterion functions, we designed the second type of experiments. In these experiments, we considered the following criterion function:

$$C(\hat{\boldsymbol{y}}_i \mid \boldsymbol{y}_i) = \frac{1}{1 + e^{\delta(\boldsymbol{y}_i^T \hat{\boldsymbol{y}}_i - \epsilon)}}, \tag{14}$$

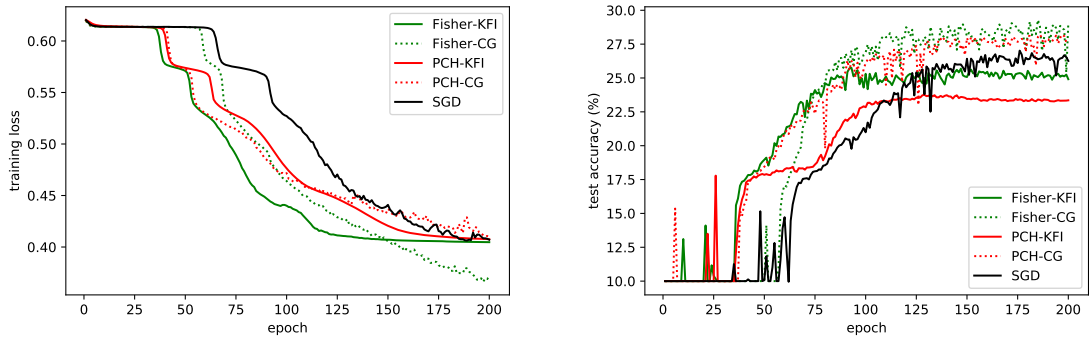

Figure 7: Comparison of different curvature information and solving methods for the non-convex criterion function "Eq. (14)" on "Cifar-10". The Gauss-Newton matrix is removed from this figure since it is not PSD in this type of experiments.

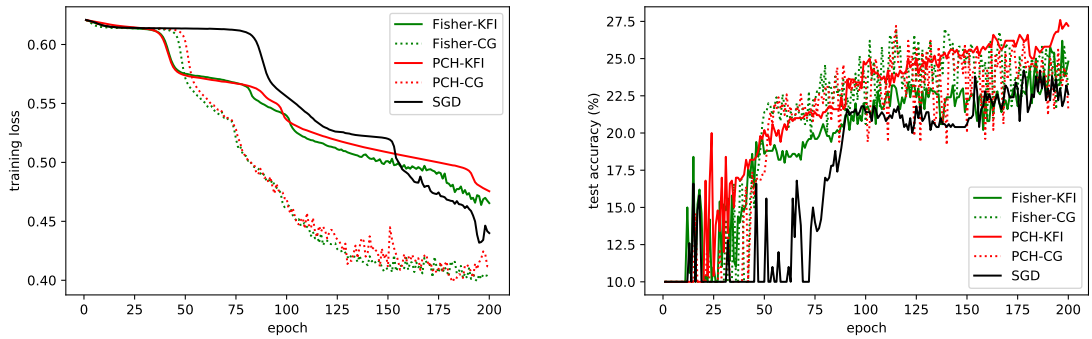

Figure 8: Comparison of different curvature information and solving methods for the non-convex criterion function "Eq. (14)" on "ImageNet-10". The Gauss-Newton matrix is removed from this figure since it is not PSD in this type of experiments.

where we fix $\delta = 5$ and $\epsilon = 0.2$. This criterion function implies that the upper bound of the loss function exists. Apart from the criterion function, we followed the same settings as those used in the first type of experiments. Note that the Gauss-Newton matrix is not PSD if the criterion function of training FCNNs is non-convex. Thus, we excluded the Gauss-Newton matrix from this comparison and focused on comparing the other two types of matrices with either EA-CG or KFI methods.

Wall Clock Time For Cifar-10, Figure 5 shows that EA-CG method outperforms KFI method in terms of both training loss and testing accuracy. But the Fisher matrix with EA-CG method has the best performance. For ImageNet-10, Figure 6 shows that EA-CG method surpasses KFI method regarding training loss. But the performance of EA-CG and KFI methods is hard to distinguish by the testing accuracy.

Epoch As shown in Figure 7, for Cifar-10, the performance of EA-CG and KFI methods is hard to distinguish by the convergence speed of training loss in terms of epochs. But the Fisher matrix with EA-CG method has the best training loss in Figure 7. Regarding testing accuracy, EA-CG method has the better performance than KFI method for this type of experiments. For ImageNet-10, Figure 8 shows that EA-CG method surpasses KFI method regarding training loss, which is consistent with Figure 6. However, the performance of EA-CG and KFI methods is hard to distinguish by the testing accuracy, and the PCH matrix using KFI method has the best testing accuracy.

Table 2: Layer-wise errors between each of the approximate Hessian matrices and the true Hessian for the convex criterion function "cross-entropy" on "Cifar-10".

	Fisher	GN	PCH-1	PCH-2
Layer-1	0.0071	0.0057	**0.0036**	0.0043
Layer-2	0.0470	**0.0212**	0.0237	0.0228
Layer-3	0.1140	0.0220	0.0238	**0.0165**
Layer-4	0.0726	0.0207	0.0119	**0.0085**
Layer-5	0.0397	0.0130	0.0086	**0.0066**
Layer-6	0.0219	0.0107	0.0093	**0.0071**
Layer-7	0.0185	0.0176	0.0132	**0.0106**
Layer-8	0.0251	**0.0000**	**0.0000**	**0.0000**
Total	0.1535	0.0446	0.0402	**0.0330**

Comparison with the True Hessian

In addition to the comparison of training loss and testing accuracy on different criterion functions and datasets, we fur-

ther examine the difference between the true Hessian and the approximate Hessian matrices such as the Fisher matrix. Here, we measure the difference in the errors that is elaborated as follows. Given the tth layer and the model parameters $\boldsymbol{\theta}$, the error is defined as

$$\left\| \mathbb{E}_i[\widetilde{\nabla^2_{\boldsymbol{b}^t}} \xi_i] - |\mathbb{E}_i[\nabla^2_{\boldsymbol{b}^t} \xi_i]| \right\|_F ,$$

where $\widetilde{\nabla^2_{\boldsymbol{b}^t}} \xi_i$ stands for the approximate Hessian matrix, and $|\mathbb{E}_i[\nabla^2_{\boldsymbol{b}^t} \xi_i]|$ is to take the absolute values of the eigenvalues of $\mathbb{E}_i[\nabla^2_{\boldsymbol{b}^t} \xi_i]$, which we follow (Dauphin et al. 2014).

The results are shown in Table 2 where each value is derived from averaging the errors of the initial parameters $\boldsymbol{\theta}^0$ to the parameters $\boldsymbol{\theta}^{s-1}$ that are updated s times. Table 2 reflects that the errors are not accumulated with layers for any approximate Hessian matrix. This observation is important for KFRA and ours since both methods derive the curvature information by approximating the Hessian layer-by-layer recursively. The "Total" row of Table 2 also indicates that our proposed PCH-1 and PCH-2 are closer to the true Hessian than the Fisher and Gauss-Newton matrices.

Concluding Remarks

To achieve more computationally feasible second-order methods for training FCNNs, we developed a practical approach, including our proposed PCH matrix and devised EA-CG method. Our proposed PCH matrix overcomes the problem of training FCNNs with non-convex criterion functions. Besides, EA-CG provides another alternative to efficiently derive update directions in this context. Our empirical studies show that our proposed PCH matrix can compete with the state-of-the-art curvature approximation, and EA-CG does converge faster and enjoys better testing accuracy than KFI method. Specially, the performance of our proposed approach is competitive with SGD. As future work, we will extend the idea to work with convolutional nets.

References

Amari, S.-I.; Park, H.; and Fukumizu, K. 2000. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation* 12(6):1399–1409.

Belagiannis, V.; Rupprecht, C.; Carneiro, G.; and Navab, N. 2015. Robust optimization for deep regression. In *Proceedings of the IEEE International Conference on Computer Vision*, 2830–2838.

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

Chang, E. Y.; Wu, M.-H.; Tang, K.-F. T.; Kao, H.-C.; and Chou, C.-N. 2017. Artificial intelligence in xprize deepq tricorder. In *Proceedings of the 2Nd International Workshop on Multimedia for Personal Health and Health Care*, MMHealth '17, 11–18. New York, NY, USA: ACM.

Dauphin, Y. N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; and Bengio, Y. 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, 2933–2941.

Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W., and Titterington, M., eds., *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, 249–256. Chia Laguna Resort, Sardinia, Italy: PMLR.

Goldfarb, D.; Mu, C.; Wright, J.; and Zhou, C. 2017. Using negative curvature in solving nonlinear programs. *Computational Optimization and Applications* 68(3):479–502.

Grosse, R., and Martens, J. 2016. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, 573–582.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

LeCun, Y.; Bottou, L.; Orr, G. B.; and Müller, K.-R. 1998. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer. 9–50.

Martens, J., and Grosse, R. 2015. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, 2408–2417.

Martens, J. 2010. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 735–742.

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

Pearlmutter, B. A. 1994. Fast exact multiplication by the hessian. *Neural computation* 6(1):147–160.

Polyak, B. T., and Juditsky, A. B. 1992. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization* 30(4):838–855.

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

Stewart, C. V. 1999. Robust parameter estimation in computer vision. *SIAM review* 41(3):513–537.

Zhang, H.; Xiong, C.; Bradbury, J.; and Socher, R. 2017. Block-diagonal hessian-free optimization for training neural networks. *arXiv preprint arXiv:1712.07296*.