# Practical Quasi-Newton Methods for Training Deep Neural Networks

**Donald Goldfarb, Yi Ren, Achraf Bahamou**
Department of Industrial Engineering and Operations Research
Columbia University
New York, NY 10027
{goldfarb, yr2322, ab4689}@columbia.edu

## Abstract

We consider the development of practical stochastic quasi-Newton, and in particular Kronecker-factored block-diagonal BFGS and L-BFGS methods, for training deep neural networks (DNNs). In DNN training, the number of variables and components of the gradient $n$ is often of the order of tens of millions and the Hessian has $n^2$ elements. Consequently, computing and storing a full $n \times n$ BFGS approximation or storing a modest number of (step, change in gradient) vector pairs for use in an L-BFGS implementation is out of the question. In our proposed methods, we approximate the Hessian by a block-diagonal matrix and use the structure of the gradient and Hessian to further approximate these blocks, each of which corresponds to a layer, as the Kronecker product of two much smaller matrices. This is analogous to the approach in KFAC [30], which computes a Kronecker-factored block-diagonal approximation to the Fisher matrix in a stochastic natural gradient method. Because of the indefinite and highly variable nature of the Hessian in a DNN, we also propose a new damping approach to keep the upper as well as the lower bounds of the BFGS and L-BFGS approximations bounded. In tests on autoencoder feed-forward neural network models with either nine or thirteen layers applied to three datasets, our methods outperformed or performed comparably to KFAC and state-of-the-art first-order stochastic methods.

## 1 Introduction

We consider in this paper the development of practical stochastic quasi-Newton (QN), and in particular Kronecker-factored block-diagonal BFGS [6, 13, 16, 39] and L-BFGS [27], methods for training deep neural networks (DNNs). Recall that the BFGS method starts each iteration with a symmetric positive definite matrix $B$ (or $H = B^{-1}$) that approximates the current Hessian matrix (or its inverse), computes the gradient $\nabla \mathbf{f}$ of $f$ at the current iterate $\mathbf{x}$ and then takes a step $\mathbf{s} = -\alpha H \nabla \mathbf{f}$, where $\alpha$ is a step length (usually) determined by some inexact line-search procedure, such that $\mathbf{y}^\top \mathbf{s} > 0$, where $\mathbf{y} = \nabla \mathbf{f}^+ - \nabla \mathbf{f}$ and $\nabla \mathbf{f}^+$ is the gradient of $f$ at the new point $\mathbf{x}^+ = \mathbf{x} + \mathbf{s}$. The method then computes an updated approximation $B^+$ to $B$ (or $H^+$ to $H$) that remains symmetric and positive-definite and satisfies the so-called *quasi-Newton* (QN) condition $B^+\mathbf{s} = \mathbf{y}$ (or equivalently, $H^+\mathbf{y} = \mathbf{s}$). A consequence of this is that the matrix $B^+$ operates on the vector $\mathbf{s}$ in exactly the same way as the average of the Hessian matrix along the line segment between $\mathbf{x}$ and $\mathbf{x}^+$ operates on $\mathbf{s}$.

In DNN training, the number of variables and components of the gradient $n$ is often of the order of tens of millions and the Hessian has $n^2$ elements. Hence, computing and storing a full $n \times n$ BFGS approximation or storing $p$ $(\mathbf{s}, \mathbf{y})$ pairs, where $p$ is approximately 10 or larger for use in an L-BFGS implementation, is out of the question. Consequently, in our methods, we approximate the Hessian by a block-diagonal matrix, where each diagonal block corresponds to a layer, further approximating them as the Kronecker product of two much smaller matrices, as in [30, 5, 19, 10].

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

**Literature Review on Using Second-order Information for DNN Training.** For solving the stochastic optimization problems with high-dimensional data that arise in machine learning (ML), stochastic gradient descent (SGD) [36] and its variants are the methods that are most often used, especially for training DNNs. These variants include such methods as AdaGrad [12], RMSprop [21], and Adam [24], all of which scale the stochastic gradient by a diagonal matrix based on estimates of the first and second moments of the individual gradient components. Nonetheless, there has been a lot of effort to find ways to take advantage of second-order information in solving ML optimization problems. Approaches have run the gamut from the use of a diagonal re-scaling of the stochastic gradient, based on the secant condition associated with quasi-Newton (QN) methods [4], to sub-sampled Newton methods (e.g. see [43], and references therein), including those that solve the Newton system using the linear conjugate gradient method (see [8]).

In between these two extremes are stochastic methods that are based either on QN methods or generalized Gauss-Newton (GGN) and natural gradient [1] methods. For example, a stochastic L-BFGS method for solving strongly convex problems was proposed in [9] that uses sampled Hessian-vector products rather than gradient differences, which was proved in [33] to be linearly convergent by incorporating the variance reduction technique (SVRG [23]) to alleviate the effect of noisy gradients. A closely related variance reduced block L-BFGS method was proposed in [17]. A regularized stochastic BFGS method was proposed in [31], and an online L-BFGS method was proposed in [32] for strongly convex problems and extended in [28] to incorporate SVRG variance reduction. Stochastic BFGS and L-BFGS methods were also developed for online convex optimization in [38]. For nonconvex problems, a damped L-BFGS method which incorporated SVRG variance reduction was developed and its convergence properties was studied in [41].

GGN methods that approximate the Hessian have been proposed, including the Hessian-free method [29] and the Krylov subspace method [40]. Variants of the closely related natural gradient method that use block-diagonal approximations to the Fisher information matrix, where blocks correspond to layers, have been proposed in e.g. [20, 11, 30, 14]. Using further approximation of each of these (empirical) Fisher matrix and GGN blocks by the Kronecker product of two much smaller matrices, the efficient KFAC [30], KFRA [5], EKFAC [15], and Shampoo [19] methods were developed. See also [2] and [10], [37], which combine both Hessian and covariance (Fisher-like) matrix information in stochastic Newton type methods, Also, methods are given in [26, 42] that replace the Kullback-Leibler divergence by the Wasserstein distance to define the natural gradient, but with a greater computational cost.

**Our Contributions.** The main contributions of this paper can be summarized as follows:

1. New BFGS and limited-memory variants (i.e. L-BFGS) that take advantage of the structure of feed-forward DNN training problems;
2. Efficient non-diagonal second-order algorithms for deep learning that require a comparable amount of memory and computational cost per iteration as first-order methods;
3. A new damping scheme for BFGS and L-BFGS updating of an inverse Hessian approximation, that not only preserves its positive definiteness, but also limits the decrease (and increase) in its smallest (and largest) eigenvalues for non-convex problems;
4. A novel application of Hessian-action BFGS;
5. The first proof of convergence (to the best of our knowledge) of a stochastic Kronecker-factored quasi-Newton method.

## 2 Kronecker-factored Quasi-Newton Method for DNN

After reviewing the computations used in DNN training, we describe the Kronecker structures of the gradient and Hessian for a single data point, followed by their extension to approximate expectations of these quantities for multiple data-points and give a generic algorithm that employs BFGS (or L-BFGS) approximations for the Hessians.

**Deep Neural Networks.** We consider a feed-forward DNN with $L$ layers, defined by weight matrices $W_l$ (whose last columns are bias vectors $b_l$), activation functions $\phi_l$ for $l \in \{1 \dots L\}$ and loss function $\mathcal{L}$. For a data-point $(x, y)$, the loss $\mathcal{L}(a_L, y)$ between the output $a_L$ of the DNN and $y$ is a non-convex function of $\theta = \left[\text{vec}(W_1)^\top, \dots, \text{vec}(W_L)^\top\right]^\top$. The network's forward and backward pass for a single input data point $(x, y)$ is described in Algorithm 1.

**Algorithm 1** Forward and backward pass of DNN for a single data-point

1: Given input $(x, y)$, weights (and biases) $W_l$, and activations $\phi_l$ for $l \in [1, L]$
2: $\mathbf{a}_0 = x$; **for** $l = 1, .., L$ **do** $\bar{\mathbf{a}}_{l-1} = (\mathbf{a}_{l-1},\ 1)$; $\mathbf{h}_l = W_l \bar{\mathbf{a}}_{l-1}$; $\mathbf{a}_l = \phi_l(\mathbf{h}_l)$
3: $\mathcal{D}\mathbf{a}_L \leftarrow \frac{\partial \mathcal{L}(z,y)}{\partial z}\Big|_{z=\mathbf{a}_L}$
4: **for** $l = L, .., 1$ **do** $\mathbf{g}_l = \mathcal{D}\mathbf{a}_l \odot \phi_l'(\mathbf{h}_l)$; $\mathcal{D}W_l = \mathbf{g}_i \bar{\mathbf{a}}_{i-1}^\top$; $\mathcal{D}\mathbf{a}_{l-1} = W_l^\top \mathbf{g}_l$

For a training dataset that contains multiple data-points indexed by $i = 1, ..., I$, let $f(i; \theta)$ denote the loss for the $i$th data-point. Then, viewing the dataset as an empirical distribution, the total loss function $f(\theta)$ that we wish to minimize is

$$f(\theta) := \mathbb{E}_i[f(i;\theta)] := \frac{1}{I}\sum_{i=1}^{I} f(i;\theta).$$

**Single Data-point: Layer-wise Structure of the Gradient and Hessian.** Let $\nabla \mathbf{f}_l$ and $\nabla^2 f_l$ denote, respectively, the restriction of $\nabla \mathbf{f}$ and $\nabla^2 f$ to the weights $W_l$ in layer $l = 1, \ldots, L$. For a single data-point $\nabla \mathbf{f}_l$ and $\nabla^2 f_l$ have a tensor (Kronecker) structure, as shown in [30] and [5]. Specifically,

$$\nabla \mathbf{f}_l(i) = \mathbf{g}_l(i)(\mathbf{a}_{l-1}(i))^\top, \quad \text{equivalently,} \quad \text{vec}(\nabla \mathbf{f}_l(i)) = \mathbf{a}_{l-1}(i) \otimes \mathbf{g}_l(i), \tag{1}$$

$$\nabla^2 f_l(i) = (\mathbf{a}_{l-1}(i)(\mathbf{a}_{l-1}(i))^\top) \otimes G_l(i), \tag{2}$$

where the pre-activation gradient $\mathbf{g}_l(i) = \frac{\partial f(i)}{\partial \mathbf{h}_l(i)}$, and the pre-activation Hessian $G_l(i) = \frac{\partial^2 f(i)}{\partial \mathbf{h}_l(i)^2}$. Our algorithm uses an approximation to $(G_l(i))^{-1}$, which is updated via the BFGS updating formulas based upon a secant condition that relates the change in $\mathbf{g}_l(i)$ with the change in $\mathbf{h}_l(i)$.

Although we focus on fully-connected layers in this paper, the idea of Kronecker-factored approximations to the diagonal blocks $\nabla^2 f_l$, $l = 1, \ldots, L$ of the Hessian can be extended to other layers used in deep learning, such as convolutional and recurrent layers.

**Multiple Data-points: Kronecker-factored QN Approach.** Now consider the case where we have a dataset of $I$ data-points indexed by $i = 1, \ldots, I$. By (2), we have

$$\mathbb{E}_i[\nabla^2 f_l(i)] \approx \mathbb{E}_i\left[\mathbf{a}_{l-1}(i)(\mathbf{a}_{l-1}(i))^\top\right] \otimes \mathbb{E}_i\left[G_l(i)\right] := A_l \otimes G_l \tag{3}$$

Note that the approximation in (3) that the expectation of the Kronecker product of two matrices equals the Kronecker product of their expectations is the same as the one used by KFAC [30]. Now, based on this structural approximation, we use $H^l = H_a^l \otimes H_g^l$ as our QN approximation to $\left(\mathbb{E}_i[\nabla^2 f_l(i)]\right)^{-1}$, where $H_a^l$ and $H_g^l$ are positive definite approximations to $A_l^{-1}$ and $G_l^{-1}$, respectively. Hence, using our layer-wise block-diagonal approximation to the Hessian, a step in our algorithm for each layer $l$ is computed as

$$\text{vec}(W_l^+) - \text{vec}(W_l) = -\alpha H^l \text{vec}\left(\widehat{\nabla \mathbf{f}_l}\right) = -\alpha (H_a^l \otimes H_g^l)\text{vec}\left(\widehat{\nabla \mathbf{f}_l}\right) = -\alpha \text{vec}\left(H_g^l \widehat{\nabla \mathbf{f}_l} H_a^l\right), \tag{4}$$

where $\widehat{\nabla \mathbf{f}_l}$ denotes the estimate to $\mathbb{E}_i[\nabla \mathbf{f}_l(i)]$ and $\alpha$ is the learning rate. After computing $W_l^+$ and performing another forward/backward pass, our method computes or updates $H_a^l$ and $H_g^l$ as follows:

1. For $H_g^l$, we use a damped version of BFGS (or L-BFGS) (See Section 3) based on the $(\mathbf{s}, \mathbf{y})$ pairs corresponding to the average change in $\mathbf{h}_l(i)$ and in the gradient with respect to $\mathbf{h}_l(i)$; i.e.,

$$\mathbf{s}_g^l = \mathbb{E}_i[\mathbf{h}_l^+(i)] - \mathbb{E}_i[\mathbf{h}_l(i)], \qquad \mathbf{y}_g^l = \mathbb{E}_i[\mathbf{g}_l^+(i)] - \mathbb{E}_i[\mathbf{g}_l(i)]. \tag{5}$$

2. For $H_a^l$ we use the "Hessian-action" BFGS method described in Section 4. The issue of possible singularity of the positive semi-definite matrix $A_l$ approximated by $(H_a^l)^{-1}$ is also addressed there by incorporating a **Levenberg-Marquardt (LM)** damping term.

Algorithm 2 gives a high-level summary of our **K-BFGS** or **K-BFGS(L)** algorithms (which use BFGS or L-BFGS to update $H_g^l$, respectively). See Algoirthm 4 in the appendix for a detailed pseudocode. The use of mini-batches is described in Section 6. Note that, an additional forward-backward pass is used in Algorithm 2 because the quantities in (5) need to be estimated using the same mini-batch.

**Algorithm 2** High-level summary of K-BFGS / K-BFGS(L)

**Require:** Given initial weights $\theta$, batch size $m$, learning rate $\alpha$

1: **for** $k = 1, 2, \ldots$ **do**
2: Sample mini-batch of size $m$: $M_k = \{\xi_{k,i}, i = 1, \ldots, m\}$
3: Perform a forward-backward pass over the current mini-batch $M_k$ (see Algorithm 1)
4: **for** $l = 1, \ldots, L$ **do** $p_l = H_g^l \widehat{\nabla \mathbf{f}}_l H_a^l$; $W_l = W_l - \alpha \cdot p_l$
5: Perform another forward-backward pass over $M_k$ to get $(\mathbf{s}_g^l, \mathbf{y}_g^l)$
6: Use damped BFGS or L-BFGS to update $H_g^l$ $(l = 1, ..., L)$ (see Section 3, in particular Algorithm 3)
7: Use Hessian-action BFGS to update $H_a^l$ $(l = 1, ..., L)$ (see Section 4 )

## 3 BFGS and L-BFGS for $G_l$

**Damped BFGS Updating.** It is well-known that training a DNN is a non-convex optimization problem. As (2) and (3) show, this non-convexity manifests in the fact that $G_l \succ 0$ often does not hold. Thus, for the BFGS update of $H_g^l$, the approximation to $G_l^{-1}$, to remain positive definite, we have to ensure that $(\mathbf{s}_g^l)^\top \mathbf{y}_g^l > 0$. Due to the stochastic setting, ensuring this condition by line-search, as is done in deterministic settings, is impractical. In addition, due to the large changes in curvature in DNN models that occur as the parameters are varied, we also need to suppress large changes to $H_g^l$ as it is updated. To deal with both of these issues, we propose a **double damping (DD)** procedure (Algorithm 3), which is based upon **Powell's damped-BFGS** approach [35], for modifying the $(\mathbf{s}_g^l, \mathbf{y}_g^l)$ pair. To motivate Algorithm 3, consider the formulas used for BFGS updating of $B$ and $H$:

$$B^+ = B - \frac{B\mathbf{s}\mathbf{s}^\top B}{\mathbf{s}^\top B\mathbf{s}} + \rho \mathbf{y}\mathbf{y}^\top, \quad H^+ = (I - \rho \mathbf{s}\mathbf{y}^\top) H (I - \rho \mathbf{y}\mathbf{s}^\top) + \rho \mathbf{s}\mathbf{s}^\top, \tag{6}$$

where $\rho = \frac{1}{\mathbf{s}^\top \mathbf{y}} > 0$. If we can ensure that $0 < \frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{1}{\mu_1}$ and $0 < \frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{1}{\mu_2}$ , then we can obtain the following bounds:

$$\|B^+\| \leq \|B - \frac{B\mathbf{s}\mathbf{s}^\top B}{\mathbf{s}^\top B\mathbf{s}}\| + \|\rho \mathbf{y}\mathbf{y}^\top\| \leq \|B\| + \|\frac{B^{1/2} H^{1/2} \mathbf{y}\mathbf{y}^\top H^{1/2} B^{1/2}}{\mathbf{s}^\top \mathbf{y}}\| \tag{7}$$

$$\leq \|B\| + \|B\| \frac{\|H^{1/2}\mathbf{y}\|^2}{\mathbf{s}^\top \mathbf{y}} \leq \|B\| \left(1 + \frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}}\right) \leq \|B\| \left(1 + \frac{1}{\mu_1}\right) \tag{8}$$

and

$$\|H^+\| \leq \|H^{1/2} - \frac{\mathbf{s}\mathbf{y}^\top H^{1/2}}{\mathbf{s}^\top \mathbf{y}}\|^2 + \|\frac{\mathbf{s}\mathbf{s}^\top}{\mathbf{s}^\top \mathbf{y}}\| \leq \left(\|H^{1/2}\| + \frac{\|\mathbf{s}\| \|H^{1/2}\mathbf{y}\|}{\mathbf{s}^\top \mathbf{y}}\right)^2 + \frac{\|\mathbf{s}\|^2}{\mathbf{s}^\top \mathbf{y}} \tag{9}$$

$$\leq \left(\|H^{1/2}\| + (\frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}})^{1/2} (\frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}})^{1/2}\right)^2 + \frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}} \leq \left(\|H^{1/2}\| + \frac{1}{\sqrt{\mu_1 \mu_2}}\right)^2 + \frac{1}{\mu_2}. \tag{10}$$

Thus, the change in $B$ (and $H$) is controlled if $\frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{1}{\mu_1}$ and $\frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{1}{\mu_2}$. Our DD approach is a two-step procedure, where the first step (i.e. Powell's damping of $H$) guarantees that $\frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{1}{\mu_1}$ and the second step (i.e., Powell's damping with $B = I$) guarantees that $\frac{\mathbf{s}^\top \mathbf{s}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{1}{\mu_2}$. Note that there is no guarantee of $\frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{1}{\mu_1}$ after the second step. However, we can skip updating $H$ in this case so that the bounds on these matrices hold. In our implementation, we always do the update, since in empirical testing, we observed that at least 90% of the pairs satisfy $\frac{\mathbf{y}^\top H \mathbf{y}}{\mathbf{s}^\top \mathbf{y}} \leq \frac{2}{\mu_1}$. See Section C in the appendix for more details on damping.

**Algorithm 3** Double Damping (DD)

1: **Input:** $\mathbf{s}$, $\mathbf{y}$; **Output:** $\tilde{\mathbf{s}}$, $\tilde{\mathbf{y}}$; **Given:** $H, \mu_1, \mu_2$
2: **if** $\mathbf{s}^\top\mathbf{y} < \mu_1\mathbf{y}^\top H\mathbf{y}$ **then** $\theta_1 = \frac{(1-\mu_1)\mathbf{y}^\top H\mathbf{y}}{\mathbf{y}^\top H\mathbf{y} - \mathbf{s}^\top\mathbf{y}}$ **else** $\theta_1 = 1$
3: $\tilde{\mathbf{s}} = \theta_1\mathbf{s} + (1-\theta_1)H\mathbf{y}$ {Powell's damping on $H$}
4: **if** $\tilde{\mathbf{s}}^\top\mathbf{y} < \mu_2\tilde{\mathbf{s}}^\top\tilde{\mathbf{s}}$ **then** $\theta_2 = \frac{(1-\mu_2)\tilde{\mathbf{s}}^\top\tilde{\mathbf{s}}}{\tilde{\mathbf{s}}^\top\tilde{\mathbf{s}} - \tilde{\mathbf{s}}^\top\mathbf{y}}$ **else** $\theta_2 = 1$
5: $\tilde{\mathbf{y}} = \theta_2\mathbf{y} + (1-\theta_2)\tilde{\mathbf{s}}$ {Powell's damping with $B = I$}
6: **return** $\tilde{s}, \tilde{y}$

**L-BFGS Implementation.** L-BFGS can also be used to update $H_g^l$. However, implementing L-BFGS using the standard "two-loop recursion" (see Algorithm 7.4 in [34]) is not efficient. This is because the main work in computing $H_g^l\widehat{\nabla\mathbf{f}_l}H_a^l$ in line 4 of Algorithm 2 would require $4p$ matrix-vector multiplications, each requiring $O(d_i d_o)$ operations, where $p$ denotes the number of $(\mathbf{s}, \mathbf{y})$ pairs stored by L-BFGS. (Recall that $\widehat{\nabla\mathbf{f}_l} \in R^{d_o \times d_i}$.) Instead, we use a "non-loop" implementation [7] of L-BFGS, whose main work involves 2 matrix-matrix multiplications, each requiring $O(pd_i d_o)$ operations. When $p$ is not small (we used $p = 100$ in our tests), and $d_i$ and $d_o$ are large, this is much more efficient, especially on GPUs.

## 4 "Hessian action" BFGS for $A_l$

In addition to approximating $G_l^{-1}$ by $H_g^l$ using BFGS, we also propose approximating $A_l^{-1}$ by $H_a^l$ using BFGS. Note that $A_l$ does not correspond to some Hessian of the objective function. However, we can generate $(\mathbf{s}, \mathbf{y})$ pairs for it by "Hessian action" (see e.g. [9, 17, 18]).

**Connection between Hessian-action BFGS and Matrix Inversion.** In our methods, we choose $\mathbf{s} = H_a^l \cdot \mathbb{E}_i[\mathbf{a}_{l-1}(i)]$ and $\mathbf{y} = A_l\mathbf{s}$, which as we now show, is closely connected to using the Sherman-Morrison modification formula to invert $A_l$. In particular, suppose that $A^+ = A + c{\cdot}\mathbf{a}\mathbf{a}^\top$; i.e., only a rank-one update is made to $A$. This corresponds to the case where the information of $A$ is accumulated from iteration to iteration, and the size of the mini-batch is 1 or $\mathbf{a}$ represents the average of the vectors $\mathbf{a}(i)$ from multiple data-points.

**Theorem 1.** *Suppose that $A$ and $H$ are symmetric and positive definite, and that $H = A^{-1}$. If we choose $\mathbf{s} = H\mathbf{a}$ and $\mathbf{y} = A^+\mathbf{s}$, where $A^+ = A + c\cdot\mathbf{a}\mathbf{a}^\top (c > 0)$. Then, the $H^+$ generated by any QN update in the **Broyden family***

$$H^+ = H - \sigma H\mathbf{y}\mathbf{y}^\top H + \rho\mathbf{s}\mathbf{s}^\top + \phi(\mathbf{y}^\top H\mathbf{y})\mathbf{h}\mathbf{h}^\top, \tag{11}$$

*where $\rho = 1/\mathbf{s}^\top\mathbf{y}$, $\sigma = 1/\mathbf{y}^\top H\mathbf{y}$, $\mathbf{h} = \rho\mathbf{s} - \sigma H\mathbf{y}$ and $\phi$ is a scalar parameter in $[0, 1]$, equals $(A^+)^{-1}$. Note that $\phi = 1$ yields the BFGS update (6) and $\phi = 0$ yields the DFP update.*

*Proof.* If $\mathbf{s} = H\mathbf{a}$ and $\mathbf{y} = A^+\mathbf{s}$, then $\mathbf{h} = 0$ , so all choices of $\phi$ yield the same matrix $H^+$. Since $H^+A^+\mathbf{s} = H^+\mathbf{y} = \mathbf{s}$ and for any vector $\mathbf{v}$ that is orthogonal to $\mathbf{a}$, $H^+A^+\mathbf{v} = H^+A\mathbf{v} = \mathbf{v}$, since $\mathbf{s}^\top A\mathbf{v} = 0$ and $\mathbf{y}^\top HA\mathbf{v} = 0$, it follows that $H^+A^+ = I$, using the fact that $\mathbf{s}$ together with any linearly independent set of $n-1$ vectors orthogonal to $\mathbf{a}$ spans $R^n$. (Note that $\mathbf{s}^\top\mathbf{a} = \mathbf{a}^\top H\mathbf{a} > 0$, since $H \succ 0 \Rightarrow$ that $\mathbf{s}$ is not orthogonal to $\mathbf{a}$.) □

In fact, all updates in the Broyden family are equivalent to applying the Sherman-Morrison modification formula to $A^+ = A + c{\cdot}\mathbf{a}\mathbf{a}^\top$, given $H = A^{-1}$, since after substituting for $\mathbf{s}$ and $\mathbf{y}$ in (11) and simplifying, one obtains

$$H^+ = H - H\mathbf{a}(c^{-1} + \mathbf{a}^\top H\mathbf{a})^{-1}\mathbf{a}^\top H.$$

When using momentum, $A^+ = \beta A + (1-\beta)\mathbf{a}\mathbf{a}^\top$ $(0 < \beta < 1)$. Hence, if we still want Theorem 1 to hold, we have to scale $H$ by $1/\beta$ before updating it. This, however, turns out to be unstable. Hence, in practice, we use the non-scaled version of "Hessian action" BFGS.

**Levenberg-Marquardt Damping for $A_l$.** Since $A_l = \mathbb{E}_i\left[(\mathbf{a}_{l-1}(i)(\mathbf{a}_{l-1}(i))^\top)\right] \succeq 0$ may not be positive definite, or may have very small positive eigenvalues, we add an **Levenberg-Marquardt (LM) damping** term to make our "Hessian-action" BFGS stable; i.e., we use $A_l + \lambda_A I_A$ instead of $A_l$, when we update $H_a^l$. Specifically, "Hessian action" BFGS for $A_l$ is performed as

1. $A_l = \beta \cdot A_l + (1-\beta) \cdot \mathbb{E}_i \left[ \mathbf{a}_{l-1}(i) \mathbf{a}_{l-1}(i)^\top \right]$; $A_l^{\text{LM}} = A_l + \lambda_A I_A$.
2. $\mathbf{s}_a^l = H_a^l \cdot \mathbb{E}_i[\mathbf{a}_{l-1}(i)]$, $\mathbf{y}_a^l = A_l^{\text{LM}} \mathbf{s}_a^l$; use BFGS with $(\mathbf{s}_a^l, \mathbf{y}_a^l)$ to update $H_a^l$.

## 5 Convergence Analysis

Following the framework for stochastic quasi-Newton methods (SQN) established in [41] for solving nonconvex stochastic optimization problems (see Section B in the appendix for this framework), we prove that, under fairly standard assumptions, for our K-BFGS(L) algorithm with skipping DD and exact inversion on $A_l$ (see Algorithm 5 in Section B), the number of iterations $N$ needed to obtain $\frac{1}{N}\sum_{k=1}^{N} \mathbb{E}[\|\nabla \mathbf{f}(\theta_k)\|^2] \leq \epsilon$ is $N = O(\epsilon^{-\frac{1}{1-\beta}})$, for step size $\alpha_k$ chosen proportional to $k^{-\beta}$, where $\beta \in (0.5, 1)$ is a constant. Our proofs, which are delayed until Section B, make use of the following assumptions, the first two of which, were made in [41].

**AS. 1.** *$f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. $f(\theta) \geq f^{low} > -\infty$, for any $\theta \in \mathbb{R}^n$. $\nabla \mathbf{f}$ is globally $L$-Lipschitz continuous; namely for any $x, y \in \mathbb{R}^n$, $\|\nabla \mathbf{f}(x) - \nabla \mathbf{f}(y)\| \leq L\|x - y\|$.*

**AS. 2.** *For any iteration $k$, the stochastic gradient $\widehat{\nabla \mathbf{f}}_k = \widehat{\nabla \mathbf{f}}(\theta_k, \xi_k)$ satisfies: a) $\mathbb{E}_{\xi_k}\left[\widehat{\nabla \mathbf{f}}(\theta_k, \xi_k)\right] = \nabla \mathbf{f}(\theta_k)$, b) $\mathbb{E}_{\xi_k}\left[\left\|\widehat{\nabla \mathbf{f}}(\theta_k, \xi_k) - \nabla \mathbf{f}(\theta_k)\right\|^2\right] \leq \sigma^2$, where $\sigma > 0$, and $\xi_k, k = 1, 2, \ldots$ are independent samples that are independent of $\{\theta_j\}_{j=1}^k$.*

**AS. 3.** *The activation functions $\phi_l$ have bounded values: $\exists \varphi > 0$ s.t. $\forall l, \forall h, |\phi_l(h)| \leq \varphi$.*

To use the convergence analysis in [41], we need to show that the block-diagonal approximation of the inverse Hessian used in Algorithm 5 satisfies the assumption that it is bounded above and below by positive-definite matrices. Given the Kronecker structure of our Hessian inverse approximation, it suffices to prove boundness of both $H_a^l(k)$ and $H_g^l(k)$ for all iterations $k$. Making the additional assumption AS.3, we are able to prove Lemma 1, and hence Lemma 3, below. Note that many popular activation functions satisfy AS.3, such as sigmoid and tanh.

**Lemma 1.** *Suppose that AS.3 holds. There exist two positive constants $\underline{\kappa}_a$, $\bar{\kappa}_a$ such that $\underline{\kappa}_a I \preceq H_a^l(k) \preceq \bar{\kappa}_a I, \forall k, l$.*

**Lemma 2.** *There exist two positive constants $\underline{\kappa}_g$ and $\bar{\kappa}_g$, such that $\underline{\kappa}_g I \preceq H_g^l(k) \preceq \bar{\kappa}_g I, \forall k, l$.*

**Lemma 3.** *Suppose that AS.3 holds. Let $\theta_{k+1} = \theta_k - \alpha_k H_k \widehat{\nabla \mathbf{f}}_k$ be the step taken in Algorithm 5. There exists two positive constants $\underline{\kappa}$, $\bar{\kappa}$ such that $\underline{\kappa} I \preceq H_k \preceq \bar{\kappa} I, \forall k$.*

Using Lemma 3, we can now apply Theorem 2.8 in [41] to prove the convergence of Algorithm 5:

**Theorem 2.** *Suppose that assumptions AS.1-3 hold for $\{\theta_k\}$ generated by Algorithm 5 with mini-batch size $m_k = m$ for all $k$, and $\alpha_k$ is chosen as $\alpha_k = \frac{\underline{\kappa}}{L\bar{\kappa}^2} k^{-\beta}$, with $\beta \in (0.5, 1)$. Then*

$$\frac{1}{N}\sum_{k=1}^{N} \mathbb{E}\left[\|\nabla \mathbf{f}(\theta_k)\|^2\right] \leq \frac{2L\left(M_f - f^{low}\right)\bar{\kappa}^2}{\underline{\kappa}^2} N^{\beta-1} + \frac{\sigma^2}{(1-\beta)m}\left(N^{-\beta} - N^{-1}\right)$$

*where $N$ denotes the iteration number and $M_f > 0$ depends only on $f$. Moreover, for a given $\epsilon \in (0, 1)$, to guarantee that $\frac{1}{N}\sum_{k=1}^{N} \mathbb{E}\left[\|\nabla \mathbf{f}(\theta_k)\|^2\right] < \epsilon$, the number of iterations $N$ needed is at most $O\left(\epsilon^{-\frac{1}{1-\beta}}\right)$.*

Note: other theorems in [41], namely Theorems 2.5 and 2.6, also apply here under our assumptions.

## 6 Experiments

Before we present some experimental results, we address the use of moving averages, and the computational and storage requirements of the algorithms that we tested.

**Mini-batch and Moving Average.** Clearly, using the whole dataset at each iteration is inefficient; hence, we use a mini-batch to estimate desired quantities. We use $\overline{X}$ to denote the averaged value of $X$ across the mini-batch for any quantity $X$. To incorporate information from the past as well as reducing the variability, we use an exponentially decaying moving average to estimate desired quantities with decay parameter $\beta \in (0, 1)$:

1. To estimate the gradient $\mathbb{E}_i[\nabla \mathbf{f}(i)]$, at each iteration, we update $\widehat{\nabla \mathbf{f}} = \beta \cdot \widehat{\nabla \mathbf{f}} + (1-\beta) \cdot \overline{\nabla \mathbf{f}}$.
2. $H_a^l$: To estimate $A_l$, at each iteration we update $\widehat{A_l} = \beta \cdot \widehat{A_l} + (1-\beta) \cdot \overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}$. Note that although we compute $\mathbf{s}_a^l$ as $H_a^l \cdot \overline{\mathbf{a}_{l-1}}$, we update $\widehat{A_l}$ with $\overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}$ (i.e. the average $\mathbf{a}_{l-1}(i)\mathbf{a}_{l-1}(i)^\top$ over the minibatch, not $\overline{\mathbf{a}_{l-1}} \cdot (\overline{\mathbf{a}_{l-1}})^\top$).
3. $H_g^l$: BFGS "uses" momentum implicitly incorporated in the matrices $H_g^l$. To further stabilize the BFGS update, we also use a moving-averaged $(\mathbf{s}_g^l, \mathbf{y}_g^l)$ (before damping); i.e., We update $\mathbf{s}_g^l = \beta \cdot \mathbf{s}_g^l + (1-\beta) \cdot \left(\overline{\mathbf{h}_l^+} - \overline{\mathbf{h}_l}\right)$, and $\mathbf{y}_g^l = \beta \cdot \mathbf{y}_g^l + (1-\beta) \cdot \left(\overline{\mathbf{g}_l^+} - \overline{\mathbf{g}_l}\right)$.

Finally, when computing $\overline{\mathbf{h}_l^+}$ and $\overline{\mathbf{g}_l^+}$, we use the same mini-batch as was used to compute $\overline{\mathbf{h}_l}$ and $\overline{\mathbf{g}_l}$. This doubles the number of forward-backward passes at each iteration.

**Storage and Computational Complexity.** Tables 1 and 2 compare the storage and computational requirements, respectively, for a layer with $d_i$ inputs and $d_o$ outputs for K-BFGS, K-BFGS(L), KFAC, and Adam/RMSprop. We denote the size of mini-batch by $m$, the number of $(\mathbf{s}, \mathbf{y})$ pairs stored for L-BFGS by $p$, and the frequency of matrix inversion in KFAC by $T$. Besides the requirements listed in Table 1, all algorithms need storage for the parameters $W_l$ and the estimate of the gradient, $\widehat{\nabla \mathbf{f}_l}$, (i.e. $O(d_i d_o)$). Besides the work listed in Table 2, all algorithms also need to do a forward-backward pass to compute $\nabla \mathbf{f}_l$ as well as updating $W_l$, (i.e. $O(m d_i d_o)$). Also note that, even though we use big-$O$ notation in these tables, the constants for all of the terms in each of the rows are roughly at the same level and relatively small.

In Table 2, for K-BFGS and K-BFGS(L), "Additional pass" refers to Line 5 of Algorithm 2; under "Curvature", $O(md_i^2)$ arises from "Hessian action" BFGS to update $H_a^l$ (see the algorithm at the end of Section 4), $O(md_o)$ arises from (5), $O(d_o^2)$ arises from updating $H_g^l$ (only for K-BFGS); and "Step $\Delta W_l$" refers to (4). For KFAC, referring to Algorithm 7 (in the appendix), "Additional pass" refers to Line 7; under "Curvature", $O(md_i^2 + md_o^2)$ refers to Line 8, and $O(\frac{1}{T}d_i^3 + \frac{1}{T}d_o^3)$ refers to Line 10; and "Step $\Delta W_l$" refers to Line 5.

From Table 1, we see that the Kronecker property enables K-BFGS and K-BFGS(L) (as well as KFAC) to have storage requirements comparable to those of first-order methods. Moreover, from Table 2, we see that K-BFGS and K-BFGS(L) require less computation per iteration than KFAC, since they only involve matrix multiplications, whereas KFAC requires matrix inversions which depend cubically on both $d_i$ and $d_o$. The cost of matrix inversion in KFAC (and singular value decomposition in [19]) is amortized by performing these operations only once every $T$ iterations; nonetheless, these amortized operations usually become much slower than matrix multiplication as models scale up.

Table 1: Storage

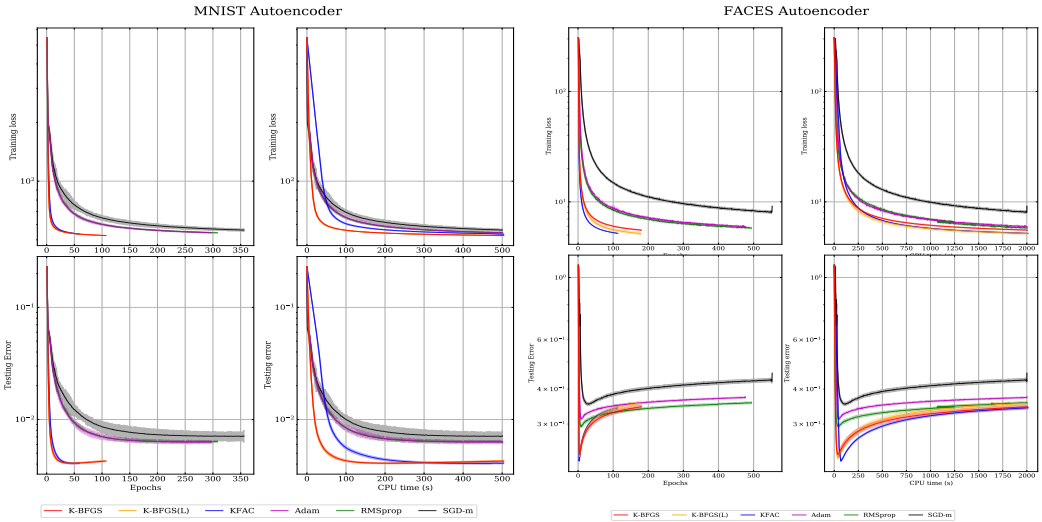
| Algorithm | $\nabla f_l \odot \nabla f_l$ | $A$ | $G$ | Total |
|---|---|---|---|---|
| K-BFGS | — | $O(d_i^2)$ | $O(d_o^2)$ | $O(d_i^2 + d_o^2 + d_i d_o)$ |
| K-BFGS(L) | — | $O(d_i^2)$ | $O(pd_o)$ | $O(d_i^2 + d_i d_o + pd_o)$ |
| KFAC | — | $O(d_i^2)$ | $O(d_o^2)$ | $O(d_i^2 + d_o^2 + d_i d_o)$ |
| Adam/RMSprop | $O(d_i d_o)$ | — | — | $O(d_i d_o)$ |

Table 2: Computation per iteration

| Algorithm | Additional pass | Curvature | Step $\Delta W_l$ |
|---|---|---|---|
| K-BFGS | $O(md_i d_o)$ | $O(md_i^2 + md_o + d_o^2)$ | $O(d_i^2 d_o + d_o^2 d_i)$ |
| K-BFGS(L) | $O(md_i d_o)$ | $O(md_i^2 + md_o)$ | $O(d_i^2 d_o + pd_i d_o)$ |
| KFAC | $O(md_i d_o)$ | $O(md_i^2 + md_o^2 + \frac{1}{T}d_i^3 + \frac{1}{T}d_o^3)$ | $O(d_i^2 d_o + d_o^2 d_i)$ |
| Adam/RMSprop | — | $O(d_i d_o)$ | $O(d_i d_o)$ |

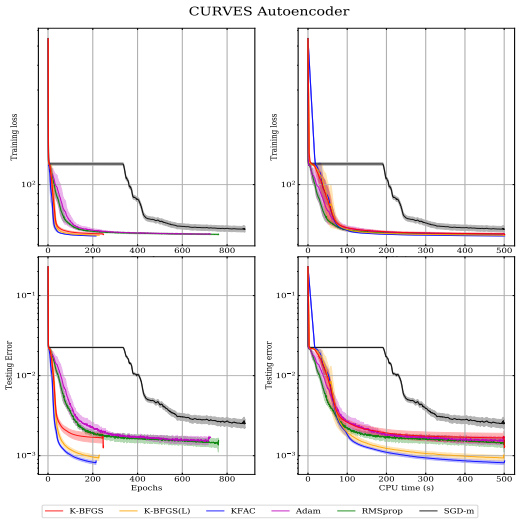
**Experimental Results.** We tested K-BFGS and K-BFGS(L), as well as KFAC, Adam/RMSprop and SGD-m (SGD with momentum) on three autoencoder problems, namely, MNIST [25], FACES, and CURVES, which are used in e.g. [22, 29, 30], except that we replaced the sigmoid activation with ReLU. See Section D in the appendix for a complete description of these problems and the competing algorithms.

Since one can view Powell's damping with $B = I$ as LM damping, we write $\mu_2 = \lambda_G$, where $\lambda_G$ denotes the LM damping parameter for $G_l$. We then define $\lambda = \lambda_A \lambda_G$ as the overall damping term of our QN approximation. To simplify matters, we chose $\lambda_A = \lambda_G = \sqrt{\lambda}$, so that we needed to tune only one hyper-parameter (HP) $\lambda$.




(a) MNIST

(b) FACES




(c) CURVES

Figure 1: Comparison between algorithms on (a) MNIST, (b) FACES, (c) CURVES. For each problem, the upper (lower) rower depicts training loss (testing (mean square) error), whereas the left (right) column depicts training/test progress versus epoch (CPU time), respectively. After each epoch, the training loss/testing error from the whole training/testing set is reported (the time for computing this loss is not included in the plots). For each problem, algorithms are terminated after the same amount of CPU time.

To obtain the results in Figure 1, we first did a grid-search on (learning rate, damping) pairs for all algorithms (except for SGD-m, whose grid-search was only on learning rate), where damping refers to $\lambda$ for K-BFGS/K-BFGS(L)/KFAC, and $\epsilon$ for RMSprop/Adam. We then selected the best (learning rate, damping) pairs with the lowest training loss encountered. The range for the grid-search and the best HP values (as well as other fixed HP values) are listed in Section D in the appendix. Using the

best HP values that we found, we then made 20 runs employing different random seeds, and plotted the mean value of the 20 runs as the solid line and the standard deviation as the shaded area.[1]

From the training loss plots in Figure 1, our algorithms clearly outperformed the first-order methods, except for RMSprop/Adam on CURVES, with respect to CPU time, and performed comparably to KFAC in terms of both CPU time and number of epochs taken. The testing error plots in Figure 1 show that our K-BFGS(L) method and KFAC behave very similarly and substantially outperform all of the first-order methods in terms of both of these measures. This suggests that our algorithms not only optimize well, but also generalize well.

To further demonstrate the robustness of our algorithms, we examined the loss under various HP settings, which showed that our algorithms are stable under a fairly wide range for the HPs (see Section D.4 in the appendix).

We also repeated our experiments using mini-batches of size 100 for all algorithms (see Figures 4, 5, and 6 in the appendix, where HPs are optimally tuned for batch sizes of 100) and our proposed methods continue to demonstrate advantageous performance, both in training and testing. For these experiments, we did not experiment with 20 random seeds. These results show that our approach works as well for relatively small mini-batch sizes of 100, as those of size 1000, which are less noisy, and hence is robust in the stochastic setting employed to train DNNs.

Compared with other methods mentioned in this paper, our K-BFGS and K-BFGS(L) have the extra advantage of being able to double the size of minibatch for computing the stochastic gradient with almost no extra cost, which might be of particular interest in a highly stochastic setting. See Section D.6 in the appendix for more discussion on this and some preliminary experimental results.

## 7 Conclusion

We proposed Kronecker-factored QN methods, namely, K-BFGS and K-BFGS(L), for training multi-layer feed-forward neural network models, that use layer-wise second-order information and require modest memory and computational resources. Experimental results indicate that our methods outperform or perform comparably to the state-of-the-art first-order and second-order methods. Our methods can also be extended to convolutional and recurrent NNs.

[1] Results were obtained on a machine with 8 x Intel(R) Xeon(R) CPU @ 2.30GHz and 1 x NVIDIA Tesla P100. Code is available at https://github.com/renyiryry/kbfgs_neurips2020_public.

## Broader Impact

The research presented in this paper provides a new method for training DNNs that our experimental testing has shown to be more efficient in several cases than current state-of-the-art optimization methods for this task. Consequently, because of the wide use of DNNs in machine learning, this should help save a substantial amount of energy. Our new algorithms simply attempt to minimize the loss function that are given to it and the computations that it performs are all transparent. In machine learning, DNNs can be trained to address many types of problems, some of which should lead to positive societal outcomes, such as ones in medicine (e.g., diagnostics and drug effectiveness), autonomous vehicle development, voice recognition and climate change. Of course, optimization algorithms for training DNNs can be used to train models that may have negative consequences, such as those intended to develop psychological profiles, invade privacy and justify biases. The misuse of any efficient optimization algorithm for machine learning, and in particular our algorithms, is beyond the control of the work presented here.

## Acknowledgments and Disclosure of Funding

DG and YR were supported in part by NSF Grant IIS-1838061. DG acknowledges the computational support provided by Google Cloud Platform Education Grant, Q81G-U4X3-5AG5-F5CG.

## References

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

[2] J. Ba, R. B. Grosse, and J. Martens. Distributed second-order optimization using kronecker-factored approximations. In *ICLR*, 2017.

[3] H. Badreddine, S. Vandewalle, and J. Meyers. Sequential quadratic programming (sqp) for optimal control in direct numerical simulation of turbulent flow. *Journal of Computational Physics*, 256:1–16, 2014.

[4] A. Bordes, L. Bottou, and P. Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10(Jul):1737–1754, 2009.

[5] A. Botev, H. Ritter, and D. Barber. Practical gauss-newton optimisation for deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 557–565. JMLR. org, 2017.

[6] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[7] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1-3):129–156, 1994.

[8] R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

[9] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[10] F. Dangel, P. Hennig, and S. Harmeling. Modular block-diagonal curvature approximations for feedforward architectures. *arXiv preprint arXiv:1902.01813*, 2019.

[11] G. Desjardins, K. Simonyan, R. Pascanu, et al. Natural neural networks. In *Advances in Neural Information Processing Systems*, pages 2071–2079, 2015.

[12] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[13] R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[14] Y. Fujimoto and T. Ohira. A neural network model with bidirectional whitening. In *International Conference on Artificial Intelligence and Soft Computing*, pages 47–57. Springer, 2018.

[15] T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, pages 9550–9560, 2018.

[16] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[17] R. Gower, D. Goldfarb, and P. Richtárik. Stochastic block bfgs: Squeezing more curvature out of data. In *International Conference on Machine Learning*, pages 1869–1878, 2016.

[18] R. M. Gower and P. Richtárik. Randomized quasi-newton updates are linearly convergent matrix inversion algorithms. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1380–1409, 2017.

[19] V. Gupta, T. Koren, and Y. Singer. Shampoo: Preconditioned stochastic tensor optimization. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1842–1850. PMLR, 2018.

[20] T. Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12, 01 2000. doi: 10.1162/089976600300015637.

[21] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8), 2012.

[22] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[23] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[24] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] W. Li and G. Montúfar. Natural gradient via optimal transport. *Information Geometry*, 1(2):181–214, 2018.

[27] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[28] A. Lucchi, B. McWilliams, and T. Hofmann. A variance reduced stochastic newton method. *arXiv preprint arXiv:1503.08316*, 2015.

[29] J. Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

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

[31] A. Mokhtari and A. Ribeiro. Res: Regularized stochastic bfgs algorithm. *IEEE Transactions on Signal Processing*, 62(23):6089–6104, 2014.

[32] A. Mokhtari and A. Ribeiro. Global convergence of online limited memory bfgs. *The Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

[33] P. Moritz, R. Nishihara, and M. Jordan. A linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, pages 249–258, 2016.

[34] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[35] M. J. Powell. Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, 14(1):224–248, 1978.

[36] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[37] N. L. Roux and A. W. Fitzgibbon. A fast natural newton method. In *ICML*, 2010.

[38] N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-newton method for online convex optimization. In *Artificial intelligence and statistics*, pages 436–443, 2007.

[39] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[40] O. Vinyals and D. Povey. Krylov subspace descent for deep learning. In *Artificial Intelligence and Statistics*, pages 1261–1268, 2012.

[41] X. Wang, S. Ma, D. Goldfarb, and W. Liu. Stochastic quasi-newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, 27(2):927–956, 2017.

[42] Y. Wang and W. Li. Information newton's flow: second-order optimization method in probability space. *arXiv preprint arXiv:2001.04341*, 2020.

[43] P. Xu, F. Roosta, and M. W. Mahoney. Newton-type methods for non-convex optimization under inexact hessian information. *Mathematical Programming*, pages 1–36, 2019.

# A Pseudocode for K-BFGS/K-BFGS(L)

Algorithm 4 gives pseudocode for K-BFGS/K-BFGS(L), which is implemented in the experiments. For details see Sections 3, 4, and Section C in the Appendix.

**Algorithm 4** Pseudocode for K-BFGS / K-BFGS(L)

**Require:** Given initial weights $\theta = \left[\text{vec}\left(W_1\right)^\top, \ldots, \text{vec}\left(W_L\right)^\top\right]^\top$, batch size $m$, learning rate $\alpha$, damping value $\lambda$, and for K-BFGS(L), the number of $(\mathbf{s}, \mathbf{y})$ pairs $p$ that are stored and used to compute $H_g^l$ at each iteration

$\mu_1 = 0.2$, $\beta = 0.9$ {set default hyper-parameter values}

$\lambda_A = \lambda_G = \sqrt{\lambda}$ {split the damping into $A$ and $G$}

$\widehat{\nabla \mathbf{f}}_l = 0$, $A_l = \mathbb{E}_i\left[\mathbf{a}_{l-1}(i)\mathbf{a}_{l-1}(i)^\top\right]$ by forward pass, $H_a^l = (A_l + \lambda_A I_A)^{-1}$, $H_g^l = I$ $(l = 1, ..., L)$ {Initialization}

**for** $k = 1, 2, \ldots$ **do**

  Sample mini-batch of size $m$: $M_k = \{\xi_{k,i}, i = 1, \ldots, m\}$

  Perform a forward-backward pass over the current mini-batch $M_k$ to compute $\overline{\nabla \mathbf{f}}_l$, $\mathbf{a}_l$, $\mathbf{h}_l$, and $\mathbf{g}_l$ $(l = 1, \ldots, L)$ (see Algorithm 1)

  **for** $l = 1, \ldots, L$ **do**

    $\widehat{\nabla \mathbf{f}}_l = \beta \widehat{\nabla \mathbf{f}}_l + (1 - \beta)\overline{\nabla \mathbf{f}}_l$

    $p_l = H_g^l \widehat{\nabla \mathbf{f}}_l H_a^l$

    {In K-BFGS(L), when computing $H_g^l\left(\widehat{\nabla \mathbf{f}}_l H_a^l\right)$, L-BFGS is initialized with an identity matrix}

    $W_l = W_l - \alpha \cdot p_l$

  Perform another forward-backward pass over $M_k$ to compute $\mathbf{h}_l^+$, $\mathbf{g}_l^+$ $(l = 1, \ldots, L)$

  **for** $l = 1, ..., L$ **do**

    {Use damped BFGS or L-BFGS to update $H_g^l$ (see Section 3)}

    $\mathbf{s}_g^l = \beta \cdot \mathbf{s}_g^l + (1 - \beta) \cdot \left(\overline{\mathbf{h}_l^+} - \overline{\mathbf{h}_l}\right)$, $\mathbf{y}_g^l = \beta \cdot \mathbf{y}_g^l + (1 - \beta) \cdot \left(\overline{\mathbf{g}_l^+} - \overline{\mathbf{g}_l}\right)$

    $(\tilde{\mathbf{s}}_g^l, \tilde{\mathbf{y}}_g^l) = \text{DD}(\mathbf{s}_g^l, \mathbf{y}_g^l)$ with $H = H_g^l$, $\mu_1 = \mu_1$, $\mu_2 = \lambda_G$ {See Algorithm 3}

    Use BFGS or L-BFGS with $(\tilde{\mathbf{s}}_g^l, \tilde{\mathbf{y}}_g^l)$ to update $H_g^l$

    {Use Hessian-action BFGS to update $H_a^l$ (see Section 4)}

    $A_l = \beta \cdot A_l + (1 - \beta) \cdot \overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}$

    $A_l^{\text{LM}} = A_l + \lambda_A I_A$

    $\mathbf{s}_a^l = H_a^l \cdot \overline{\mathbf{a}_{l-1}}$, $\mathbf{y}_a^l = A_l^{\text{LM}} \mathbf{s}_a^l$

    Use BFGS with $(\mathbf{s}_a^l, \mathbf{y}_a^l)$ to update $H_a^l$

# B Convergence: Proofs of Lemmas 1-3 and Theorem 2

In this section, we prove the convergence of Algorithm 5, a variant of K-BFGS(L). Algorithm 5 is very similar to our actual implementation of K-BFGS(L) (i.e. Algorithm 4), except that

- we skip updating $H_g^l$ if $(\tilde{\mathbf{s}}_g^l)^\top \tilde{\mathbf{y}}_g^l < \mu_1 (\tilde{\mathbf{y}}_g^l)^\top H_g^l \tilde{\mathbf{y}}_g^l$ (see Line 16);
- we set $H_a^l$ to the exact inverse of $A_l^{\text{LM}}$ (see Line 21);
- we use decreasing step sizes $\{\alpha_k\}$ as specified in Theorem 2;
- we use the mini-batch gradient instead of the momentum gradient (see Line 8).

To accomplish this, we prove Lemmas 1-3, which in addition to Assumptions AS.1-2, ensure that all of the assumptions in Theorem 2.8 in [41] are satisfied, and hence that the generic stochastic quasi-Newton (SQN) method, i.e. Algorithm 6, below converges. Specifically, Theorem 2.8 in [41] requires, in addition to Assumptions AS.1-2, the assumption

**AS. 4.** *There exist two positive constants $\underline{\kappa}, \bar{\kappa}$, such that $\underline{\kappa} I \preceq H_k \preceq \bar{\kappa} I, \forall k$; for any $k \geq 2$, the random variable $H_k$ depends only on $\xi_{[k-1]}$.*

**Algorithm 5** K-BFGS(L) with DD-skip and exact inversion of $A_l^{\text{LM}}$

**Require:** Given initial weights $\theta = \left[\text{vec}\,(W_1)^\top, \ldots, \text{vec}\,(W_L)^\top\right]^\top$, batch size $m$, learning rate $\{\alpha_k\}$, damping value $\lambda$, and the number of $(\mathbf{s}, \mathbf{y})$ pairs $p$ that are stored and used to compute $H_g^l$ at each iteration

$\mu_1 = 0.2$, $\beta = 0.9$ {set default hyper-parameter values}
$\lambda_A = \lambda_G = \sqrt{\lambda}$ {split the damping into $A$ and $G$}
$A_l(0) = \mathbb{E}_i\left[\mathbf{a}_{l-1}(i)\mathbf{a}_{l-1}(i)^\top\right]$ by forward pass, $H_a^l(0) = (A_l(0) + \lambda_A I_A)^{-1}$, $H_g^l(0) = I$ $(l = 1, ..., L)$ {Initialization}
**for** $k = 1, 2, \ldots$ **do**
  Sample mini-batch of size $m$: $M_k = \{\xi_{k,i}, i = 1, \ldots, m\}$
  Perform a forward-backward pass over the current mini-batch $M_k$ to compute $\overline{\nabla \mathbf{f}}_l$, $\mathbf{a}_l$, $\mathbf{h}_l$, and $\mathbf{g}_l$ $(l = 1, \ldots, L)$ (see Algorithm 1)
  **for** $l = 1, \ldots, L$ **do**
    $p_l = H_g^l(k-1)\widehat{\nabla \mathbf{f}}_l H_a^l(k-1)$, where $\widehat{\nabla \mathbf{f}}_l = \overline{\nabla \mathbf{f}}_l$
    {When computing $H_g^l\left(\widehat{\nabla \mathbf{f}}_l H_a^l\right)$, L-BFGS is initialized with an identity matrix}
    $W_l = W_l - \alpha_k \cdot p_l$
  Perform another forward-backward pass over $M_k$ to compute $\mathbf{h}_l^+$, $\mathbf{g}_l^+$ $(l = 1, \ldots, L)$
  **for** $l = 1, ..., L$ **do**
    {Use damped L-BFGS with skip to update $H_g^l$ (see Section 3)}
    $\mathbf{s}_g^l = \beta \cdot \mathbf{s}_g^l + (1-\beta) \cdot \left(\overline{\mathbf{h}_l^+} - \overline{\mathbf{h}_l}\right)$, $\mathbf{y}_g^l = \beta \cdot \mathbf{y}_g^l + (1-\beta) \cdot \left(\overline{\mathbf{g}_l^+} - \overline{\mathbf{g}_l}\right)$
    $(\tilde{\mathbf{s}}_g^l, \tilde{\mathbf{y}}_g^l) = \text{DD}(\mathbf{s}_g^l, \mathbf{y}_g^l)$ with $H = H_g^l(k-1)$, $\mu_1 = \mu_1$, $\mu_2 = \lambda_G$ {See Algorithm 3}
    **if** $(\tilde{\mathbf{s}}_g^l)^\top \tilde{\mathbf{y}}_g^l \geq \mu_1 (\tilde{\mathbf{y}}_g^l)^\top H_g^l \tilde{\mathbf{y}}_g^l$ **then**
      Use L-BFGS with $(\tilde{\mathbf{s}}_g^l, \tilde{\mathbf{y}}_g^l)$ to update $H_g^l(k)$
    {Use exact inversion to compute $H_a^l$}
    $A_l(k) = \beta \cdot A_l(k-1) + (1-\beta) \cdot \overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}$
    $A_l^{\text{LM}}(k) = A_l(k) + \lambda_A I_A$
    $H_a^l(k) = \left(A_l^{\text{LM}}(k)\right)^{-1}$

**Algorithm 6** SQN method for nonconvex stochastic optimization.

**Require:** Given $\theta_1 \in \mathbb{R}^n$, batch sizes $\{m_k\}_{k\geq 1}$, and step sizes $\{\alpha_k\}_{k\geq 1}$

**for** $k = 1, 2, \ldots$ **do**
  Calculate $\widehat{\nabla \mathbf{f}}_k = \frac{1}{m_k}\sum_{i=1}^{m_k} \nabla \mathbf{f}(\theta_k, \xi_{k,i})$
  Generate a positive definite Hessian inverse approximation $H_k$
  Calculate $\theta_{k+1} = \theta_k - \alpha_k H_k \widehat{\nabla \mathbf{f}}_k$

In the following proofs, $\|\cdot\|$ denotes the 2-norm for vectors, and the spectral norm for matrices.

**Proof of Lemma 1**:

*Proof.* Because $A_l^{\text{LM}}(k) \succeq \lambda_A I_A$, we have that $H_a^l(k) \preceq \bar{\kappa}_a I_A$, where $\bar{\kappa}_a = \frac{1}{\lambda_A}$.

On the other hand, for any $\mathbf{x} \in \mathbb{R}^{d_l}$, by Cauchy-Schwarz, $\langle \mathbf{a}_{l-1}(i), \mathbf{x}\rangle^2 \leq \|\mathbf{x}\|^2 \|\mathbf{a}_{l-1}(i)\|^2 \leq \|\mathbf{x}\|^2(1+\varphi^2 d_l)$. Hence, $\left\|\overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}\right\| \leq 1 + \varphi^2 d_l$; similarly, $\|A_l(0)\| \leq 1 + \varphi^2 d_l$. Because $\|A_l(k)\| \leq \beta\|A_l(k-1)\| + (1-\beta)\left\|\overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}\right\|$, by induction, $\|A_l(k)\| \leq 1 + \varphi^2 d_l$ for any $k$ and $l$. Thus, $\|A_l^{\text{LM}}(k)\| \leq 1 + \varphi^2 d_l + \lambda_A$. Hence, $H_a^l(k) \succeq \underline{\kappa}_a I_A$, where $\underline{\kappa}_a = \frac{1}{1+\varphi^2 d_l + \lambda_A}$.

□

**Proof of Lemma 2:**

*Proof.* To simplify notation, we omit the subscript $g$, superscript $l$ and the iteration index $k$ in the proof. Hence, our goal is to prove $\underline{\kappa}_g I \preceq H = H_g^l(k) \preceq \bar{\kappa}_g I$, for any $l$ and $k$. Let $(\mathbf{s}_i, \mathbf{y}_i)$ $(i = 1, ..., p)$ denote the pairs used in an L-BFGS computation of $H$. Since $(\mathbf{s}_i, \mathbf{y}_i)$ was **not skipped**, $\frac{\mathbf{y}_i^\top \bar{H}^{(i)} \mathbf{y}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \leq \frac{1}{\mu_1}$, where $\bar{H}^{(i)}$ denotes the matrix $H_g^l$ used at the iteration in which $\mathbf{s}_i$ and $\mathbf{y}_i$ were computed. Note that this is not the matrix $H_i$ used in the recursive computation of $H$ at the current iterate $\theta_k$.

Given an initial estimate $H_0 = B_0^{-1} = I$ of $(G_g^l(\theta_k))^{-1}$, the L-BFGS method updates $H_i$ recursively as

$$H_i = \left(I - \rho_i \mathbf{s}_i \mathbf{y}_i^\top\right) H_{i-1} \left(I - \rho_i \mathbf{y}_i \mathbf{s}_i^\top\right) + \rho_i \mathbf{s}_i \mathbf{s}_i^\top, \quad i = 1, \ldots, p, \tag{12}$$

where $\rho_i = (\mathbf{s}_i^\top \mathbf{y}_i)^{-1}$, and equivalently,

$$B_i = B_{i-1} + \frac{\mathbf{y}_i \mathbf{y}_i^\top}{\mathbf{s}_i^\top \mathbf{y}_i} - \frac{B_{i-1} \mathbf{s}_i \mathbf{s}_i^\top B_{i-1}}{\mathbf{s}_i^\top B_{i-1} \mathbf{s}_i}, \quad i = 1, \ldots, p,$$

where $B_i = H_i^{-1}$. Since we use DD with skipping, we have that $\frac{\mathbf{s}_i^\top \mathbf{s}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \leq \frac{1}{\mu_2}$ and $\frac{\mathbf{y}_i^\top \bar{H}^{(i)} \mathbf{y}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \leq \frac{1}{\mu_1}$. Note that we don't have $\frac{\mathbf{y}_i^\top H_{i-1} \mathbf{y}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \leq \frac{1}{\mu_1}$, so we cannot direct apply (10). Hence, by (8), we have that $||B_i|| \leq ||B_{i-1}|| \left(1 + \frac{1}{\mu_1}\right)$. Hence, $||B|| = ||B_p|| \leq ||B_0|| \left(1 + \frac{1}{\mu_1}\right)^p = \left(1 + \frac{1}{\mu_1}\right)^p$. Thus, $B \preceq \left(1 + \frac{1}{\mu_1}\right)^p I$, $H \succeq \left(1 + \frac{1}{\mu_1}\right)^{-p} I := \underline{\kappa}_g I$.

On the other hand, since $\underline{\kappa}_g$ is a uniform lower bound for $H_g^l(k)$ for any $k$ and $l$, $\bar{H}^{(i)} \succeq \underline{\kappa}_g I$. Thus,

$$\frac{1}{\mu_1} \geq \frac{\mathbf{y}_i^\top \bar{H}^{(i)} \mathbf{y}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \geq \underline{\kappa}_g \frac{\mathbf{y}_i^\top \mathbf{y}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \Rightarrow \frac{\mathbf{y}_i^\top \mathbf{y}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \leq \frac{1}{\mu_1 \underline{\kappa}_g}.$$

Hence, using the fact that $||uv^\top|| = ||u|| \cdot ||v||$ for any vectors $u, v$, $||\rho_i \mathbf{s}_i \mathbf{s}_i^\top|| = \rho_i ||\mathbf{s}_i|| ||\mathbf{s}_i|| = \frac{\mathbf{s}_i^\top \mathbf{s}_i}{\mathbf{s}_i^\top \mathbf{y}_i} \leq \frac{1}{\mu_2}$,

$$\begin{aligned}
||H_i|| &= || \left(I - \rho_i \mathbf{s}_i \mathbf{y}_i^\top\right) H_{i-1} \left(I - \rho_i \mathbf{y}_i \mathbf{s}_i^\top\right) + \rho_i \mathbf{s}_i \mathbf{s}_i^\top || \\
&= ||H_{i-1} + \rho_i^2 (\mathbf{y}_i^\top H_{i-1} \mathbf{y}_i) \mathbf{s}_i \mathbf{s}_i^\top - \rho_i \mathbf{s}_i \mathbf{y}_i^\top H_{i-1} - \rho_i H_{i-1} \mathbf{y}_i \mathbf{s}_i^\top + \rho_i \mathbf{s}_i \mathbf{s}_i^\top || \\
&\leq ||H_{i-1}|| + ||\rho_i^2 (\mathbf{y}_i^\top H_{i-1} \mathbf{y}_i) \mathbf{s}_i \mathbf{s}_i^\top || + ||\rho_i \mathbf{s}_i \mathbf{y}_i^\top H_{i-1}|| + ||\rho_i H_{i-1} \mathbf{y}_i \mathbf{s}_i^\top || + ||\rho_i \mathbf{s}_i \mathbf{s}_i^\top || \\
&\leq ||H_{i-1}|| + ||H_{i-1}|| \cdot ||\rho_i^2 (\mathbf{y}_i^\top \mathbf{y}_i) \mathbf{s}_i \mathbf{s}_i^\top || + 2\rho_i ||\mathbf{s}_i|| \cdot ||\mathbf{y}_i^\top H_{i-1}|| + \frac{1}{\mu_2} \\
&\leq ||H_{i-1}|| + ||H_{i-1}|| \cdot \frac{1}{\mu_1 \underline{\kappa}_g} \frac{1}{\mu_2} + 2\rho_i ||\mathbf{s}_i|| \cdot ||\mathbf{y}_i^\top|| \cdot ||H_{i-1}|| + \frac{1}{\mu_2} \\
&\leq ||H_{i-1}|| \left(1 + \frac{1}{\mu_1 \underline{\kappa}_g} \frac{1}{\mu_2} + 2 \frac{1}{\sqrt{\mu_1 \mu_2 \underline{\kappa}_g}}\right) + \frac{1}{\mu_2} \\
&= \hat{\mu} ||H_{i-1}|| + \frac{1}{\mu_2}, \quad \text{where} \quad \hat{\mu} = \left(1 + \frac{1}{\sqrt{\mu_1 \mu_2 \underline{\kappa}_g}}\right)^2.
\end{aligned}$$

From the fact that $H_0 = I$, and induction, we have that $||H|| \leq \hat{\mu}^p + \frac{\hat{\mu}^p - 1}{\hat{\mu} - 1} \frac{1}{\mu_2} \equiv \bar{\kappa}_g$.

□

**Proof of Lemma 3:**

*Proof.* By Lemma 1, 2 and the fact that $H_k = \text{diag}\{H_a^1(k-1) \otimes H_g^1(k-1), ..., H_a^L(k-1) \otimes H_g^L(k-1)\}$. □

**Proof of Theorem 2:**

*Proof.* To show that Algorithm 5 lies in the framework of Algorithm 6, it suffices to show that $H_k$ generated by Algorithm 5 is positive definite, which is true since $H_k = \text{diag}\{H_a^1(k-1) \otimes H_g^1(k-1), ..., H_a^L(k-1) \otimes H_g^L(k-1)\}$ and $H_a^l(k)$ and $H_g^l(k)$ are positive definite for all $k$ and $l$. Then by Lemma 3, and the fact that $H_k$ depends on $H_a^l(k-1)$ and $H_g^l(k-1)$, and $H_a^l(k-1)$ and $H_g^l(k-1)$ does not depend on random samplings in the $k$th iteration, AS.4 holds. Hence, Theorem 2.8 of [41] applies to Algorithm 5, proving Theorem 2. □

# C Powell's Damped BFGS Updating

For BFGS and L-BFGS, one needs $\mathbf{y}^\top \mathbf{s} > 0$. However, when used to update $H_g^l$, there is no guarantee that $(\mathbf{y}_g^l)^\top \mathbf{s}_g^l > 0$ for any layer $l = 1, \ldots, L$. In deterministic optimization, positive definiteness of the QN Hessian approximation $B$ (or its inverse) is maintained by performing an inexact line search that ensures that $\mathbf{s}^T\mathbf{y} > 0$, which is always possible as long as the function being minimized is bounded below. However, this would be expensive to do for DNN. Thus, we propose the following heuristic based on Powell's damped-BFGS approach [35].

**Powell's Damping on $B$.** Powell's damping on $B$, proposed in [35], replaces $\mathbf{y}$ in the BFGS update, by $\tilde{\mathbf{y}} = \theta\mathbf{y} + (1-\theta)B\mathbf{s}$, where

$$\theta = \begin{cases} \frac{(1-\mu)\mathbf{s}^\top B\mathbf{s}}{\mathbf{s}^\top B\mathbf{s} - \mathbf{s}^\top\mathbf{y}}, & \text{if } \mathbf{s}^\top\mathbf{y} < \mu\mathbf{s}^\top B\mathbf{s}, \\ 1, & \text{otherwise.} \end{cases}$$

It is easy to verify that $\mathbf{s}^\top\tilde{\mathbf{y}} \geq \mu\mathbf{s}^\top B\mathbf{s}$.

**Powell's Damping on $H$.** In Powell's damping on $H$ (see e.g. [3]), $\tilde{\mathbf{s}} = \theta\mathbf{s} + (1-\theta)H\mathbf{y}$ replaces $\mathbf{s}$, where

$$\theta = \begin{cases} \frac{(1-\mu_1)\mathbf{y}^\top H\mathbf{y}}{\mathbf{y}^\top H\mathbf{y} - \mathbf{s}^\top\mathbf{y}}, & \text{if } \mathbf{s}^\top\mathbf{y} < \mu_1\mathbf{y}^\top H\mathbf{y}, \\ 1, & \text{otherwise.} \end{cases}$$

This is used in lines 2 and 3 of the DD (Algorithm 3). It is also easy to verify that $\tilde{\mathbf{s}}^\top\mathbf{y} \geq \mu_1\mathbf{y}^\top H\mathbf{y}$.

**Powell's Damping with $B = I$.** Powell's damping on $B$ is not suitable for our algorithms because we do not keep track of $B$. Moreover, it does not provide a simple bound on $\frac{\tilde{\mathbf{s}}^\top\tilde{\mathbf{s}}}{\tilde{\mathbf{s}}^\top\tilde{\mathbf{y}}}$ that is independent of $\|B\|$. Therefore, we use Powell's damping with $B = I$, in lines 4 and 5 of the DD (Algorithm 3). It is easy to verify that it ensures that $\tilde{\mathbf{s}}^\top\tilde{\mathbf{y}} \geq \mu_2\tilde{\mathbf{s}}^\top\tilde{\mathbf{s}}$.

Powell's damping with $B = I$ can be interpreted as adding an Levenberg-Marquardt (LM) damping term to $B$. Note that an LM damping term $\mu_2$ would lead to $B \succeq \mu_2 I$. Then, the secant condition $\tilde{\mathbf{y}} = B\tilde{\mathbf{s}}$ implies

$$\tilde{\mathbf{y}}^\top\tilde{\mathbf{s}} = \tilde{\mathbf{s}}^\top B\tilde{\mathbf{s}} \geq \mu_2\tilde{\mathbf{s}}^\top\tilde{\mathbf{s}},$$

which is the same inequality as we get using Powell's damping with $B = I$. Note that although the $\mu_2$ parameter in Powell's damping with $B = I$ can be interpreted as an LM damping, we recommend setting the value of $\mu_2$ within $(0, 1]$ so that $\tilde{\mathbf{y}}$ is a convex combination of $\mathbf{y}$ and $\tilde{\mathbf{s}}$. In all of our experimental tests, we found that the best value for the hyperparameter $\lambda$ for both K-BFGS and K-BFGS(L) was less than or equal to 1, and hence that $\mu_2 = \lambda_G = \sqrt{\lambda}$ was in the interval $(0, 1]$.

## C.1 Double Damping (DD)

Our double damping (Algorithm 3) is a two-step damping procedure, where the first step (i.e. Powell's damping on $H$) can be viewed as an interpolation between the current curvature and the previous ones, and the second step (i.e. Powell's damping with $B = I$) can be viewed as an LM damping.

Recall that there is no guarantee that $\frac{\mathbf{y}^\top H\mathbf{y}}{\mathbf{s}^\top\mathbf{y}} \leq \frac{2}{\mu_1}$ holds after DD. While we skip using pairs that do not satisfy this inequality, when updating $H_g^l$ in proving the convergence of the K-BFGS(L) variant Algorithm 5 , we use all $(\mathbf{s}, \mathbf{y})$ pairs to update $H_g^l$ in our implementations of both K-BFGS and K-BFGS(L) . However, whether one skips or not makes only slight difference in the performance of these algorithms, because as our empirical testing has shown, at least 90% of the iterations satisfy $\frac{\mathbf{y}^\top H\mathbf{y}}{\mathbf{s}^\top\mathbf{y}} \leq \frac{2}{\mu_1}$, even if we don't skip. See Figure 2 which reports results on this for K-BFGS(L) when tested on the MNIST, FACES and CURVES datasets.

# D Implementation Details and More Experiments

## D.1 Description of Competing Algorithms

### D.1.1 KFAC

We first describe KFAC in Algorithm 7. Note that $G_l$ in KFAC refers to the $G$ matrices in [30], which is different from the $G_l$ in K-BFGS.

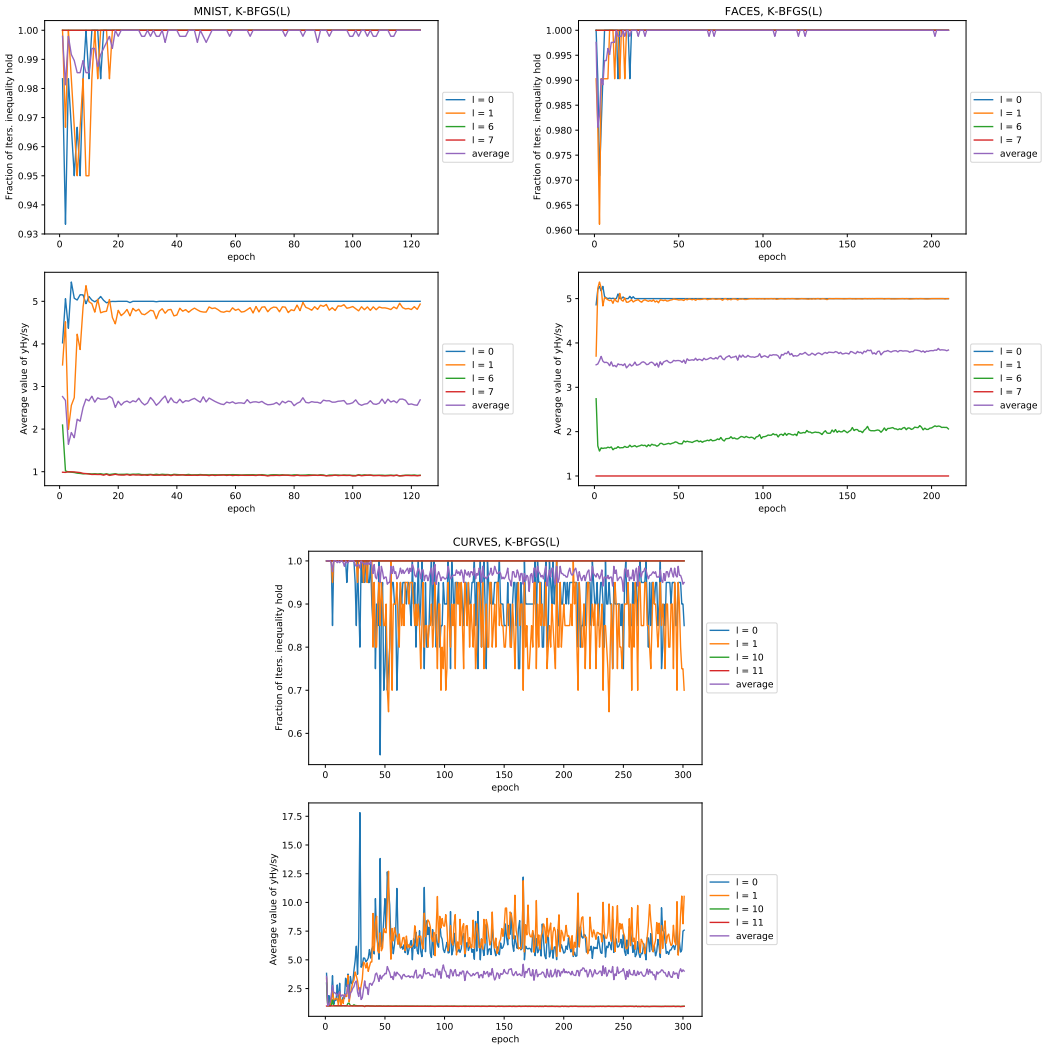



Figure 2: Fraction of the number of iterations in each epoch, in which the inequality $\frac{\mathbf{y}^\top H\mathbf{y}}{\mathbf{s}^\top\mathbf{y}} \leq \frac{2}{\mu_1}$ holds (upper plots), and the average value of $\frac{\mathbf{y}^T H\mathbf{y}}{\mathbf{s}^\top\mathbf{y}}$ (lower plots) in each epoch. Legends in each plot assign different colors to represent each layer $l$.

### D.1.2 Adam/RMSprop

We implement Adam and RMSprop exactly as in [24] and [21], respectively. Note that the only difference between them is that Adam does bias correction for the 1st and 2nd moments of gradient while RMSprop does not.

### D.1.3 Initialization of Algorithms

We now describe how each algorithm is initialized. For all algorithms, $\widehat{\nabla \mathbf{f}}$ is always initialized as zero.

For second-order information, we use a "warm start" to estimate the curvature when applicable. In particular, we estimate the following curvature information using the entire training set before we start updating parameters. The information gathered is

- $A_l$ for K-BFGS and K-BFGS(L);
- $A_l$ and $G_l$ for KFAC;
- $\nabla f \odot \nabla f$ for RMSprop;

**Algorithm 7 KFAC**

**Require:** Given $\theta_0$, batch size $m$, and learning rate $\alpha$, damping value $\lambda$, inversion frequency $T$

**for** $k = 1, 2, \ldots$ **do**
  Sample mini-batch of size $m$: $M_k = \{\xi_{k,i}, i = 1, \ldots, m\}$
  Perform a forward-backward pass over the current mini-batch $M_k$ (see Algorithm 1)
  **for** $l = 1, 2, \ldots L$ **do**
    $p_l = H_g^l \widehat{\nabla \mathbf{f}}_l H_a^l$
    $W_l = W_l - \alpha \cdot p_l$.
  Perform another pass over $M_k$ with $y$ sampled from the predictive distribution
  Update $A_l = \beta \cdot A_l + (1-\beta) \cdot \overline{\mathbf{a}_{l-1}\mathbf{a}_{l-1}^\top}$, $G_l = \beta \cdot G_l + (1-\beta) \cdot \overline{\mathbf{g}_l \mathbf{g}_l^\top}$
  **if** $k \leq T$ or $k \equiv 0 \pmod{T}$ **then**
    Recompute $H_a^l = (A_l + \sqrt{\lambda} I)^{-1}$, $H_g^l = (G_l + \sqrt{\lambda} I)^{-1}$

- Not applicable to Adam because of the bias correction.

Lastly, for K-BFGS and K-BFGS(L), $H_a^l$ is always initially set to an identity matrix. $H_g^l$ is also initially set to an identity matrix in K-BFGS; for K-BFGS(L), when updating $H_g^l$ using L-BFGS, the incorporation of the information from the $p\,(\mathbf{s}, \mathbf{y})$ pairs is applied to an initial matrix that is set to an identity matrix. Hence, the above initialization/warm start costs are roughly twice as large for KFAC as they are for K-BFGS and K-BFGS(L).

### D.2 Autoencoder Problems

Table 3 lists information about the three datasets, namely, MNIST[2], FACES[3], and CURVES[4]. Table 4 specifies the architecture of the 3 problems, where binary entropy $\mathcal{L}\,(a_L, y) = \sum_n [y_n \log a_{L,n} + (1-y_n) \log(1-a_{L,n})]$, MSE $\mathcal{L}\,(a_L, y) = \frac{1}{2} \sum_n (a_{L,n} - y_n)^2$. Besides the loss function in Table 4, we further add a regularization term $\frac{\eta}{2}||\theta||^2$ to the loss function, where $\eta = 10^{-5}$.

Table 3: Info for 3 datasets

| Dataset | # data points | # training examples | # testing examples |
|---|---|---|---|
| MNIST | 70,000 | 60,000 | 10,000 |
| FACES | 165,600 | 103,500 | 62,100 |
| CURVES | 30,000 | 20,000 | 10,000 |

Table 4: Architecture of 3 auto-encoder problems

| Dataset | Layer width & activation | Loss function |
|---|---|---|
| MNIST | [784, 1000, 500, 250, 30, 250, 500, 1000, 784]<br>[ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, sigmoid] | binary entropy |
| FACES | [625, 2000, 1000, 500, 30, 500, 1000, 2000, 625]<br>[ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, linear] | MSE |
| CURVES | [784, 400, 200, 100, 50, 25, 6, 25, 50, 100, 200, 400, 784]<br>[ReLU, ReLU, ReLU, ReLU, ReLU, linear,<br>ReLU, ReLU, ReLU, ReLU, ReLU, sigmoid] | binary entropy |

### D.3 Specification of Hyper-parameters

In our experiments, we focus our tuning effort onto learning rate and damping. The range of the tuning values is listed below:

- learning rate $\alpha_k = \alpha \in \{$ 1e-5, 3e-5, 1e-4, 3e-4, 1e-3, 3e-3, 1e-2, 3e-2, 1e-1, 3e-1, 1, 3, 10 $\}$.

[2]Downloadable at http://yann.lecun.com/exdb/mnist/

[3]Downloadable at www.cs.toronto.edu/~jmartens/newfaces_rot_single.mat

[4]Downloadable at www.cs.toronto.edu/~jmartens/digs3pts_1.mat

- damping:
  - $\lambda$ for K-BFGS, K-BFGS(L) and KFAC: $\lambda \in \{$ 3e-3, 1e-2, 3e-2, 1e-1, 3e-1, 1, 3 $\}$.
  - $\epsilon$ for RMSprop and Adam: $\epsilon \in \{$ 1e-10, 1e-8, 1e-6, 1e-4, 1e-3, 1e-2, 1e-1 $\}$.
  - Not applicable for SGD with momentum.

Table 5: Best (learning rate, damping) for Figure 1

| | K-BFGS | K-BFGS(L) | KFAC | Adam | RMSprop | SGD-m |
|---|---|---|---|---|---|---|
| MNIST | (0.3, 0.3) | (0.3, 0.3) | (1, 3) | (1e-4, 1e-4) | (1e-4, 1e-4) | (0.03, -) |
| FACES | (0.1, 0.1) | (0.1, 0.1) | (0.1, 0.1) | (1e-4, 1e-4) | (1e-4, 1e-4) | (0.01, -) |
| CURVES | (0.1, 0.01) | (0.3, 0.3) | (0.3, 0.3) | (1e-3, 1e-3) | (1e-3, 1e-3) | (0.1, -) |

The best hyper-parameters were those that produced the lowest value of the deterministic loss function encountered at the end of every epoch until the algorithm was terminated. These values were used in Figure 1 and are listed in Table 5. Besides the tuning hyper-parameters, we also list other fixed hyper-parameters with their values:

- Size of minibatch $m$ = 1000, which is also suggested in [5].
- Decay parameter:
  - K-BFGS, K-BFGS(L): $\beta = 0.9$;
  - KFAC: $\beta = 0.9$;
  - RMSprop, Adam: Following the notation in [24], we use $\beta_1 = \beta_2 = 0.9$;[5]
  - SGD with momentum: $\beta = 0.9$.
- Other:
  - $\mu_1 = 0.2$ in double damping (DD):

    We recommend to leave the value as default because $\mu_1$ represents the "ratio" between current and past, which is scaling invariant;
  - Number of $(\mathbf{s}, \mathbf{y})$ pairs stored for K-BFGS(L) $p = 100$:

    It might be more efficient to use a smaller $p$ for the narrow layers. We didn't investigate this for simplicity and consistency;
  - Inverse frequency $T = 20$ in KFAC.

[5]The default value of $\beta_2$ recommended in [24] is 0.999. Hence, we also tested $\beta_2 = 0.999$, and obtained results that were similar to those presented in Figure 1 (i.e., with $\beta_2 = 0.9$). For the sake of fair comparison, we chose to report the results with $\beta_2 = 0.9$.

### D.4 Sensitivity to Hyper-parameters

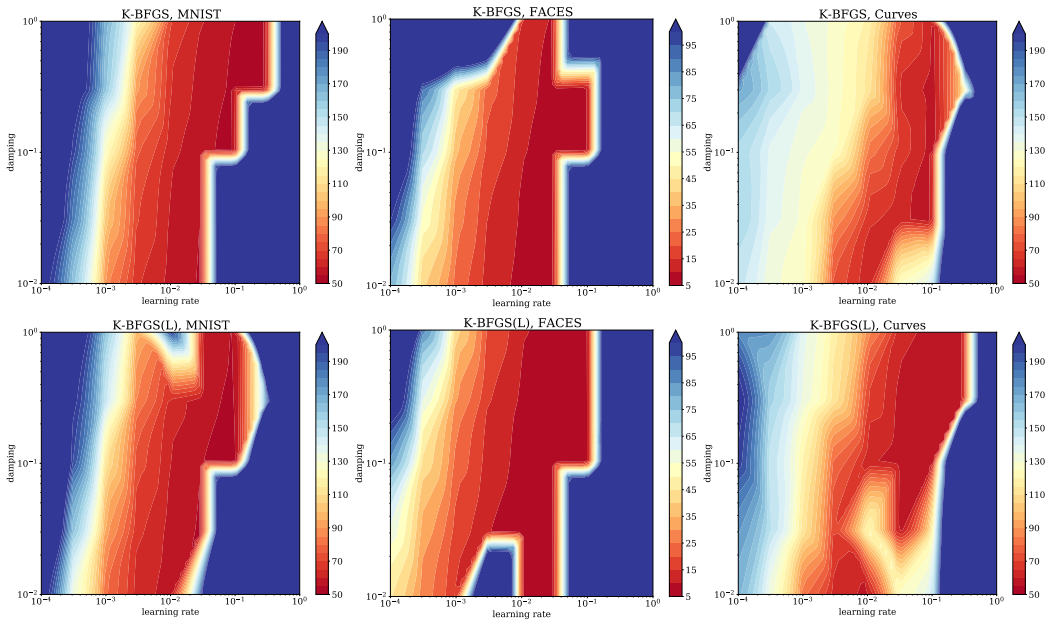



Figure 3: Landscape of loss w.r.t hyper-parameters (i.e. learning rate and damping). The left, middle, right columns depict results for MNIST, FACES, CURVES, which are terminated after 500, 2000, 500 seconds (CPU time), respectively, for K-BFGS (upper) and K-BFGS(L) (lower) row.

Figure 3 shows the sensitivity of K-BFGS and K-BFGS(L) to hyper-parameter values (i.e. learning rate and damping). The $x$-axis corresponds to the learning rate $\alpha$, while the $y$-axis correspond to the damping value $\lambda$. Color corresponds to the loss after a certain amount of CPU time. We can see that both K-BFGS and K-BFGS(L) are robust within a fairly wide range of hyper-parameters.

To get the plot, we first obtained training loss with $\alpha \in \{$1e-4, 3e-4, 1e-3, 3e-3, 1e-2, 3e-2, 1e-1, 3e-1, 1$\}$ and $\lambda \in \{$1e-2, 3e-2, 1e-1, 3e-1, 1$\}$, and then drew contour lines of the loss within the above ranges.

### D.5 Experimental Results Using Mini-batches of Size 100

We repeated our experiments using mini-batches of size 100 for all algorithms (see Figures 4, 5, and 6). For each figure, the upper (lower) rower depict training loss (testing (mean square) error), whereas the left (right) column depicts training/test progress versus epoch (CPU time), respectively.

The best hyper-parameters were those that produce the lowest value of the deterministic loss function encountered at the end of every epoch until the algorithm was terminated. These values were used in Figures 4, 5, 6 and are listed in Table 6.

Our proposed methods continue to demonstrate advantageous performance, both in training and testing. It is interesting to note that, whereas for a minibatch size of 1000, KFAC slightly outperformed K-BFGS(L), for a minibatch size of 100, K-BFGS(L) clearly outperformed KFAC in training on CURVES.

Table 6: Best (learning rate, damping) for Figures 4, 5, 6

| | K-BFGS | K-BFGS(L) | KFAC | Adam | RMSprop | SGD-m |
|---|---|---|---|---|---|---|
| MNIST | (0.1, 0.3) | (0.1, 0.3) | (0.1, 0.3) | (1e-4, 1e-4) | (1e-4, 1e-4) | (0.03, -) |
| FACES | (0.03, 0.03) | (0.03, 0.3) | (0.03, 0.3) | (3e-5, 1e-4) | (3e-5, 1e-4) | (0.01, -) |
| CURVES | (0.3, 1) | (0.3, 0.3) | (0.03, 0.1) | (3e-4, 1e-4) | (3e-3, 1e-4) | (0.03, -) |

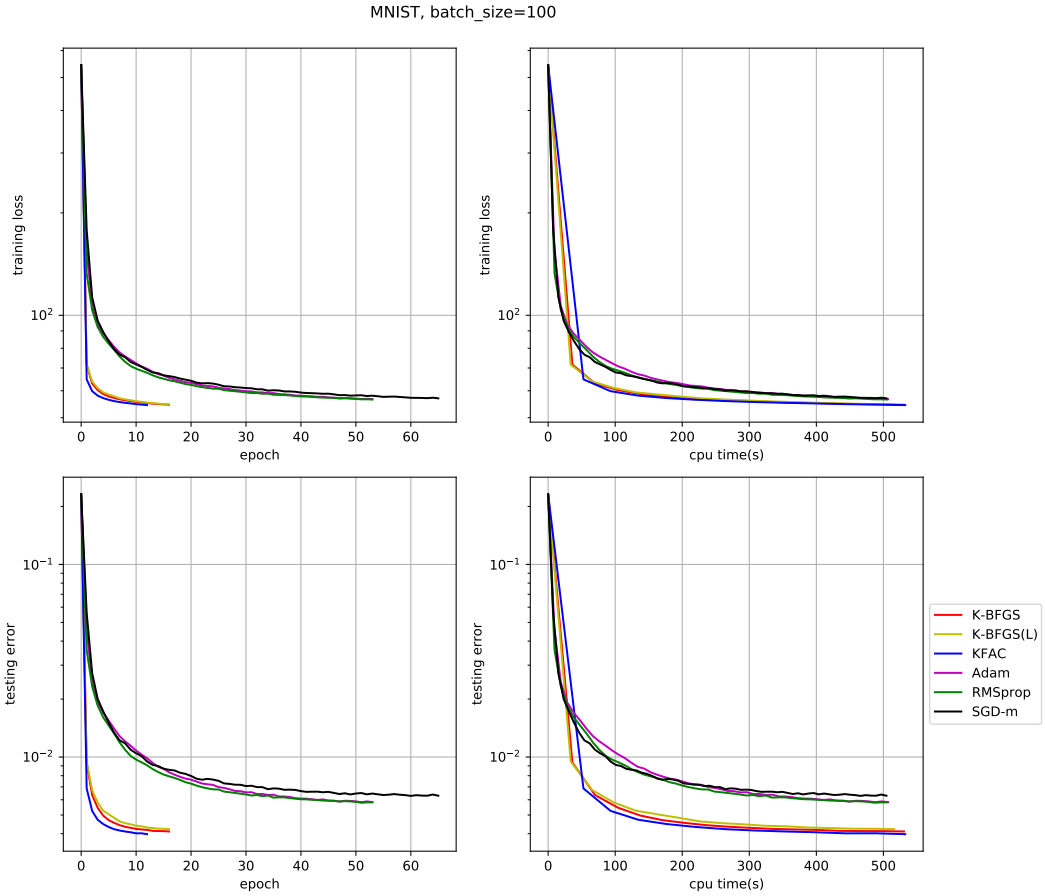



Figure 4: Comparison between algorithms on MNIST with batch size 100

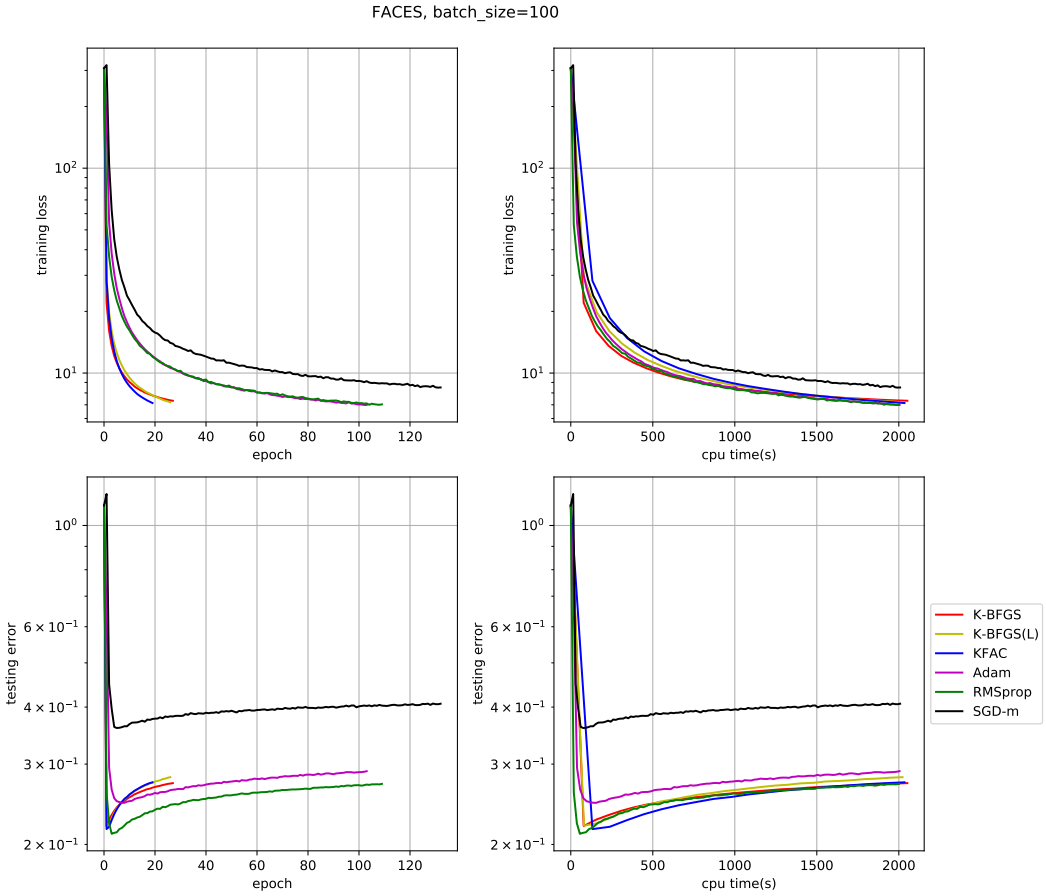



Figure 5: Comparison between algorithms on FACES with batch size 100

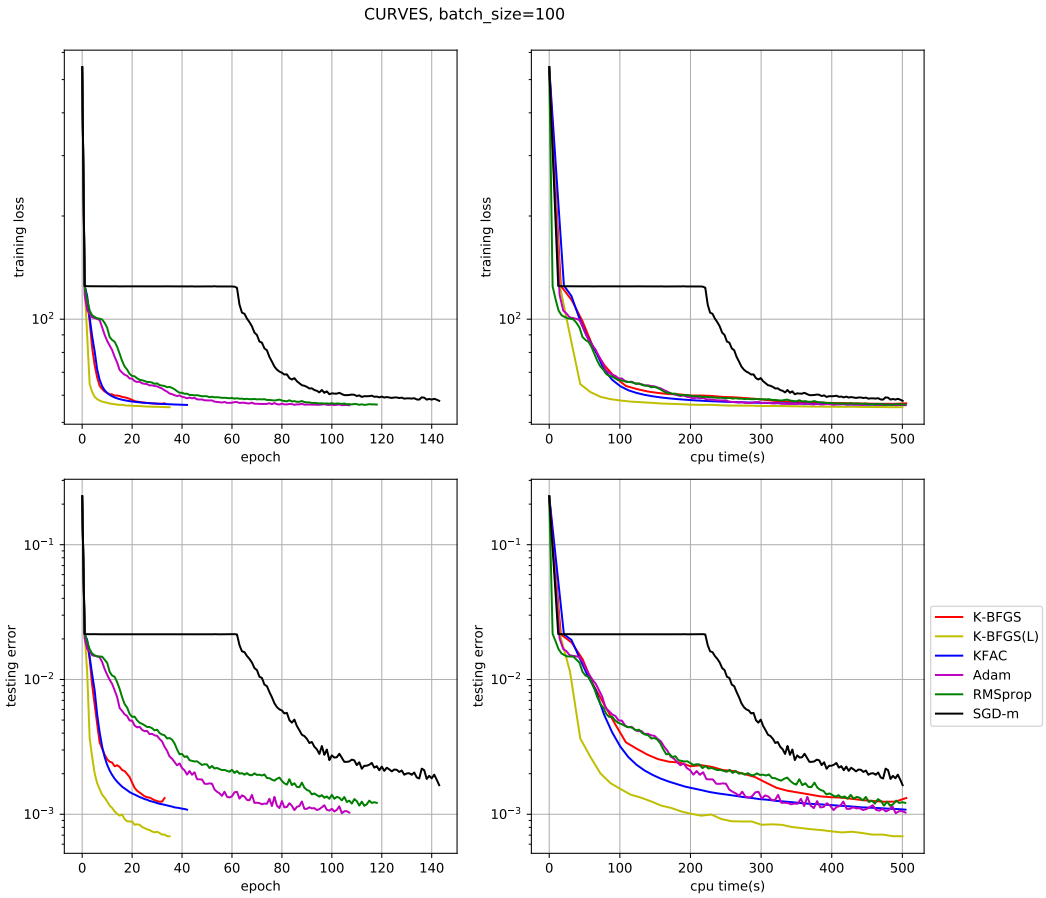



Figure 6: Comparison between algorithms on CURVES with batch size 100

### D.6 Doubling the Mini-batch for the Gradient at Almost No Cost

Compared with other methods mentioned in this paper, our K-BFGS and K-BFGS(L) methods have the extra advantage of being able to double the size of the minibatch used to compute the stochastic gradient with almost no extra cost, which might be of particular interest in a highly stochastic setting. To accomplish this, we can make use of the stochastic gradient $\overline{\nabla \mathbf{f}}^+$ computed in the **second** pass of the previous iteration that is needed for computing the $(\bar{\mathbf{s}}, \bar{\mathbf{y}})$ pair for applying the BFGS or L-BFGS updates, and average it with the stochastic gradient $\overline{\nabla \mathbf{f}}$ of the current iteration. For example if the size of minibatch is $m = 1000$, the above "double-grad-minibatch" method computes a stochastic gradient from 2000 data points at each iteration, except at the very first iteration.

The results of some preliminary experiments are depicted in Figure 7, where we compare an earlier version of the K-BFGS algorithm (Algorithm 4), which uses a slightly different variant of Hessian-action to update $H_a^l$, using a size of $m = 1000$ for mini-batches, with its "double-grad-minibatch" variants for $m = 500$ and $1000$. Even though "double-grad" does not help much in these experiments, our K-BFGS algorithm performs stably across these different variants. These results indicate that there is a potential for further improvements; e.g., a finer grid search might identify hyper-parameter values that result in better performing algorithms.

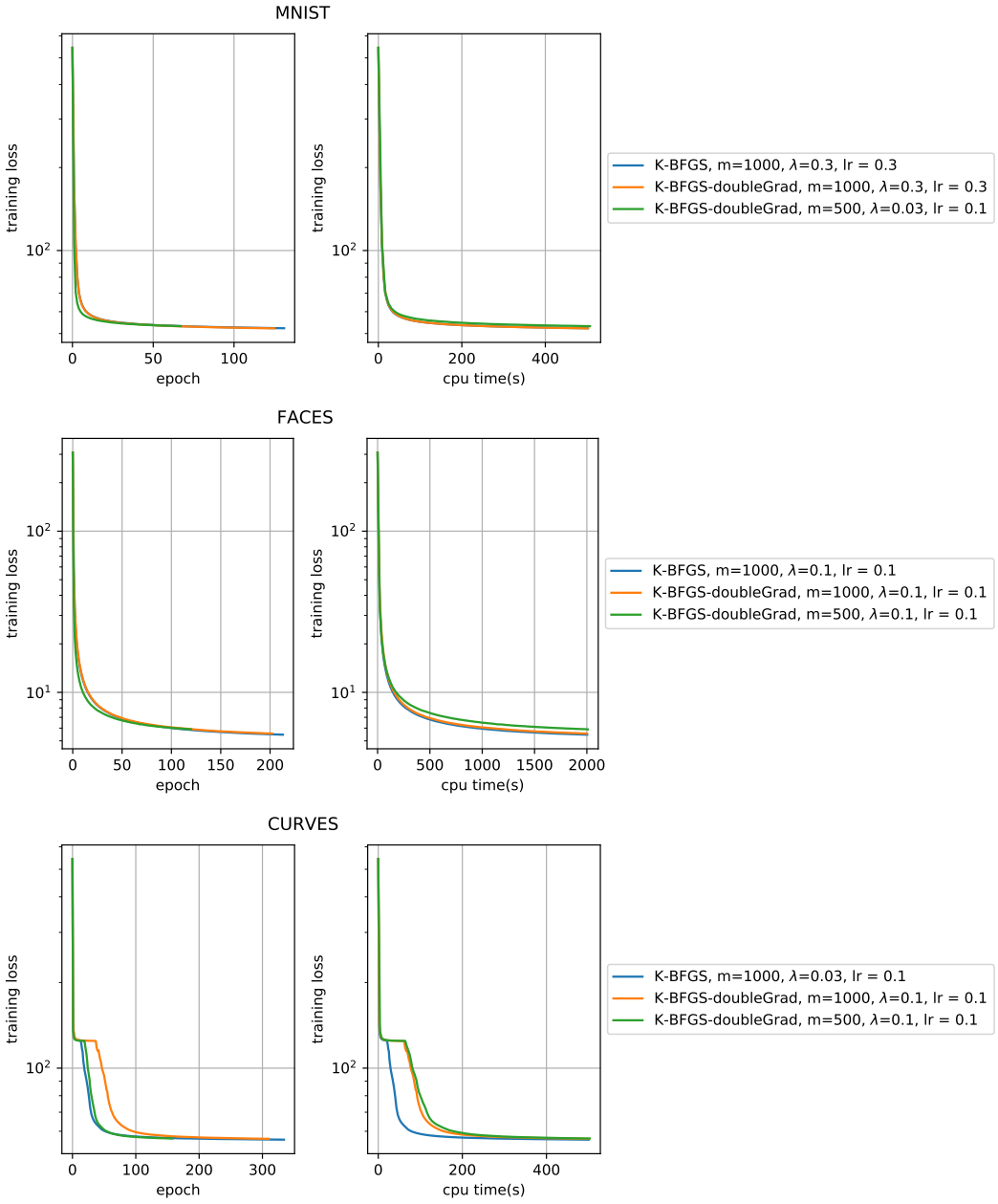



Figure 7: Comparison between K-BFGS and its "double-grad" variants