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

Algorithm 4

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

1: \( \mu_1 = 0.2, \beta = 0.9 \) \{set default hyper-parameter values\}
2: \( \lambda_A = \lambda_G = \sqrt{\lambda} \) \{split the damping into \( A \) and \( G \)\}
3: \( \nabla f_l = 0, A_l = E_\theta [a_{l-1}(i)a_{l-1}(i)^\top] \) by forward pass, \( H_a^l = H_g^l = I \) \((l = 1, \ldots, L)\) \{Initialization\}
4: for \( k = 1, 2, \ldots \) do
5: Sample mini-batch of size \( m \): \( M_k = \{\xi_{k,i}, i = 1, \ldots, m\} \)
6: Perform a forward-backward pass over the current mini-batch \( M_k \) to compute \( \nabla f_l, a_l, h_l, \) and \( g_l(l = 1, \ldots, L) \) \{see Algorithm 1\}
7: for \( l = 1, \ldots, L \) do
8: \( \nabla f_l = \beta \nabla f_l + (1 - \beta) \nabla f_l \)
9: \( p_l = H_g^l \nabla f_l H_a^l \)
10: \{In K-BFGS(L), when computing \( H_g^l \left( \nabla f_l H_a^l \right) \), L-BFGS is initialized with an identity matrix\}
11: \( W_l = W_l - \alpha \cdot p_l \)
12: Perform another forward-backward pass over \( M_k \) to compute \( h_l^+, g_l^+ \) \((l = 1, \ldots, L)\)
13: for \( l = 1, \ldots, L \) do
14: \{Use damped BFGS or L-BFGS to update \( H_g^l \) \{see Section 3\}\}
15: \( s_g^l = \beta \cdot s_g^l + (1 - \beta) \cdot \left( h_l^+ - h_l \right), y_g^l = \beta \cdot y_g^l + (1 - \beta) \cdot \left( g_l^+ - g_l \right) \)
16: \( (s_g^l, y_g^l) = \text{DD}(s_g^l, y_g^l) \) with \( H = H_g^l, \mu_1 = \mu_2 = \lambda_G \) \{See Algorithm 3\}
17: Use BFGS or L-BFGS with \((s_g^l, y_g^l)\) to update \( H_g^l \)
18: \{Use Hessian-action BFGS to update \( H_g^l \) \{see Section 4\}\}
19: \( A_l = \beta \cdot A_l + (1 - \beta) \cdot a_{l-1} a_{l-1}^\top \)
20: \( A_l^{LM} = A_l + \lambda_A I_A \)
21: \( s_g^l = H_g^l a_{l-1}, y_g^l = A_l^{LM} s_g^l \)
22: Use BFGS with \((s_g^l, y_g^l)\) to update \( H_g^l \)

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 \((s_g^l)^\top \hat{y}_g^l < \mu_1(\hat{y}_g^l)^\top H_g^l \hat{y}_g^l \) \{see Line 16\};
- we set \( H_a^l \) to the exact inverse of \( A_l^{LM} \) \{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

\textbf{AS. 4.} There exist two positive constants \( \xi, \bar{\kappa} \), such that \( \xi I \preceq H_k \preceq \bar{\kappa} I, \forall k; \) 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 $A^2_{LM}$

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

1: $\mu_1 = 0.2$, $\beta = 0.9$ (set default hyper-parameter values)
2: $\lambda_A = \lambda_G = \sqrt{\lambda}$ (split the damping into $A$ and $G$)
3: $A_t(0) = E \left[ a_{t-1}(i) a_{t-1}(i)^T \right]$ by forward pass, $H_a^g(0) = H_*^g(0) = I$ ($l = 1, \ldots, L$) (Initialization)
4: for $k = 1, 2, \ldots$ do
5: Sample mini-batch of size $m$: $M_k = \{\xi_{k,i}, i = 1, \ldots, m\}$
6: Perform a forward-backward pass over the current mini-batch $M_k$ to compute $\nabla f_l, a_l, h_l$, and $g_l$ ($l = 1, \ldots, L$) (see Algorithm 1)
7: for $l = 1, \ldots, L$ do
8: $p_l = H^g_l(k-1) \nabla f_l H^g_l(k-1)$, where $\nabla f_l = \nabla f_l$
9: (When computing $H^g_l \left( \nabla f_l, H^g_l \right)$, L-BFGS is initialized with an identity matrix)
10: $W_l = W_l - \alpha_k \cdot p_l$
11: Perform another forward-backward pass over $M_k$ to compute $h_l^+, g_l^+$ ($l = 1, \ldots, L$)
12: for $l = 1, \ldots, L$ do
13: (Use damped L-BFGS with skip to update $H^g_*$ (see Section 3))
14: $s_l^g = \beta \cdot s_l^g + (1 - \beta) \cdot (h_l^g - h_l^g)$, $y_l^g = \beta \cdot y_l^g + (1 - \beta) \cdot (g_l^g - g_l^g)$
15: $(s_l^g, y_l^g) = \text{DD}(s_l^g, y_l^g)$ with $H = H^g_l(k-1)$, $\mu_1 = \mu_1, \mu_2 = \lambda_G$ (See Algorithm 3)
16: if $(s_l^g)^T y_l^g \geq \mu_1 (s_l^g)^T H^g_l s_l^g$ then
17: Use L-BFGS with $(s_l^g, y_l^g)$ to update $H^g_*(k)$
18: (Use exact inversion to compute $H^g_*$)
19: $A_t(k) = \beta \cdot A_t(k-1) + (1 - \beta) \cdot a_{t-1} a_{t-1}$
20: $A^2_{LM}(k) = A_t(k) + \lambda_A I_A$
21: $H^g_*(k) = \left( A^2_{LM}(k) \right)^{-1}$

Algorithm 6 SQN method for nonconvex stochastic optimization.

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

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

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

**Proof of Lemma 1:**

**Proof.** Because $A^2_{LM}(k) \geq \lambda_A I_A$, we have that $H^g_*(k) \preceq \tilde{\kappa}_A I_A$, where $\tilde{\kappa}_A = \frac{1}{\sqrt{\lambda}}$.

On the other hand, for any $x \in \mathbb{R}^d$, by Cauchy-Schwarz, $\langle a_{l-1}(i), x \rangle^2 \leq \|x\|^2 \|a_{l-1}(i)\|^2 \leq \|x\|^2 (1 + \varphi^2 d_i)$. Hence, $\left\| a_{l-1} a_{l-1}^\top \right\| \leq 1 + \varphi^2 d_i$; similarly, $\|A_t(0)\| \leq 1 + \varphi^2 d_i$. Because $\|A_t(k)\| \leq \beta \|A_t(k-1)\| + (1 - \beta) \left\| a_{l-1} a_{l-1}^\top \right\|$, by induction, $\|A_t(k)\| \leq 1 + \varphi^2 d_i$ for any $k$ and $l$. Thus, $\|A^2_{LM}(k)\| \leq 1 + \varphi^2 d_i + \lambda_A$. Hence, $H^g_*(k) \preceq \tilde{\kappa}_A I_A$, where $\tilde{\kappa}_A = \frac{1}{1 + \varphi^2 d_i + \lambda_A}$.

**Proof of Lemma 2:**
Algorithm 5 is positive definite, which is true since $H_h^i$ depends on $H$, where $H_h^i$ denotes the matrix $H_h^i$ used at the iteration in which $s_i$ and $y_i$ 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 $(G_0^T(\theta_k)^{-1}$, the L-BFGS method updates $H_i$ recursively as

$$H_i = \left( I - \rho_i s_i y_i^T \right) H_{i-1} \left( I - \rho_i s_i y_i^T \right) + \rho_i s_i s_i^T,$$

where $\rho_i = (s_i^T y_i)^{-1}$, and equivalently,

$$B_i = B_{i-1} + \frac{y_i y_i^T}{s_i^T s_i} - \frac{B_{i-1} s_i s_i^T B_{i-1}}{s_i^T B_{i-1} s_i}, \quad i = 1, \ldots, p,$$

where $B_i = H_i^{-1}$. Since we use DD with skipping, we have that $s_{i+1}^T y_{i+1} / s_i^T y_i \leq 1 / \mu_2$, so we cannot directly apply (10). Hence, by (8), we have that $\|B_i\| \leq \|B_{i-1}\| \left( 1 + \frac{1}{\mu_1} \right)$. Hence, $\|B\| = \|B_{i-1}\| \leq \|B_0\| \left( 1 + \frac{1}{\mu_1} \right)^p = \left( 1 + \frac{1}{\mu_1} \right)^p I$, where $H \geq \left( 1 + \frac{1}{\mu_1} \right)^{-p} I = \kappa_0 I$.

On the other hand, since $\kappa_0$ is a uniform lower bound for $H_i^j(k)$ for any $k$ and $l$, $\bar{H}_i^j(k) \geq \kappa_0 I$. Thus,

$$\frac{1}{\mu_1} \geq \frac{y_i^T H_i^j y_i}{s_i^T s_i} \geq \frac{\kappa_0}{s_i^T s_i} \Rightarrow \frac{y_i^T y_i}{s_i^T s_i} \leq \frac{1}{\mu_1 \kappa_0}.$$

Hence, using the fact that $\|uv^\top\| = \|u\| \cdot \|v\|$ for any vectors $u, v$, $\|\rho_i s_i s_i^T\| = \|s_i\| \cdot \|s_i\| = s_i^T s_i \leq 1 / \mu_2$.

$$\|H_i\| = \| \left( I - \rho_i s_i y_i^T \right) H_{i-1} \left( I - \rho_i s_i y_i^T \right) + \rho_i s_i s_i^T \|$$

$$= \|H_{i-1} + \rho_i^2 (y_i^T H_{i-1} y_i) s_i s_i^T - \rho_i s_i y_i^T H_{i-1} + \rho_i H_{i-1} y_i s_i^T + \rho_i s_i s_i^T \|$$

$$\leq \|H_{i-1}\| + \|\rho_i^2 (y_i^T H_{i-1} y_i) s_i s_i^T\| + \|\rho_i s_i y_i^T H_{i-1}\| + \|\rho_i H_{i-1} y_i s_i^T\| + \|\rho_i s_i s_i^T\|$$

$$\leq \|H_{i-1}\| + \|H_{i-1}\| \cdot \|\rho_i y_i^T y_i s_i s_i^T\| + 2 \rho_i \|s_i\| \cdot \|y_i^T H_{i-1}\| + \frac{1}{\mu_1}$$

$$\leq \|H_{i-1}\| + \|H_{i-1}\| \cdot \frac{1}{\mu_1 \kappa_0} + 2 \rho_i \|s_i\| \cdot \|y_i^T H_{i-1}\| + \frac{1}{\mu_2}$$

$$\leq \|H_{i-1}\| \left( 1 + \frac{1}{\mu_1 \kappa_0} + 2 \sqrt{\frac{1}{\mu_1 \mu_2 \kappa_0}} \right) + \frac{1}{\mu_2}$$

$$= \bar{\mu} ||H_{i-1}|| + \frac{1}{\mu_2}, \quad \text{where} \quad \bar{\mu} = \left( 1 + \frac{1}{\sqrt{\mu_1 \mu_2 \kappa_0}} \right)^2.$$

From the fact that $H_0 = I$, and induction, we have that $\|H\| \leq \bar{\mu}^p + \frac{\bar{\mu}^p - 1}{\mu_1} \frac{1}{\mu_2} \equiv \bar{r}_g$.

Proof of Lemma 3:

Proof. By Lemma 1, 2 and the fact that $H_k = \text{diag}\{ H_h^k(k-1) \otimes H_s^k(k-1), \ldots, H_h^k(k-1) \otimes H_s^k(k-1) \}$.

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 $H_k = \text{diag}\{ H_h^k(k-1) \otimes H_s^k(k-1), \ldots, H_h^k(k-1) \otimes H_s^k(k-1) \}$ and $H_h^j(k)$ and $H_s^j(k)$ are positive definite for all $k$ and $l$. Then by Lemma 3, and the fact that $H_k$ depends on $H_h^j(k)$ and $H_s^j(k)$, and $H_h^j(k)$ and $H_s^j(k)$ do not depend on random samplings in the $j$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 $y^\top s > 0$. However, when used to update $H^l_\beta$, there is no guarantee that $(y^l_\theta)^\top s^l > 0$ 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 $s^\top y > 0$, 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 $y$ in the BFGS update, by $\tilde{y} = \theta y + (1 - \theta)Bs$, where

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

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

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

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

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

**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 $\frac{s^\top B s}{s^\top y}$ 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 $s^\top \tilde{y} \geq \mu s^\top s$.

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 $\mu$ would lead to $B \geq \mu I$. Then, the secant condition $y = Bs$ implies

$$y^\top s = s^\top Bs \geq \mu s^\top s,$$

which is the same inequality as we get using Powell’s damping with $B = I$.

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 $\frac{s^\top H y}{s^\top y} \leq \frac{2}{\mu^2}$ holds after DD. While we skip using pairs that do not satisfy this inequality, when updating $H^l_\beta$, in proving the convergence of the K-BFGS(L) variant Algorithm 5, we use all $(s, y)$ pairs to update $H^l_\beta$ 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 $\frac{s^\top H y}{s^\top y} \leq \frac{2}{\mu^2}$, 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.

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

D.1.3 Initialization of Algorithms

We now describe how each algorithm is initialized. For all algorithms, \( \widehat{\nabla} f \) 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;
- Not applicable to Adam because of the bias correction.

Lastly, for K-BFGS and K-BFGS(L), \( H^l_0 \) is always initially set to an identity matrix. \( H^l_0 \) is also initially set to an identity matrix in K-BFGS; for K-BFGS(L), when updating \( H^l_0 \) using L-BFGS, the incorporation of the information from the \( p(s,y) \) 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).
Algorithm 7 KFAC

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

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 \( l = 1, 2, \ldots, L \) do
5: \( p_l = H_l^{H_l} \hat{f}_l H_l \)
6: \( W_l = W_l - \alpha \cdot p_l \).
7: Perform another pass over \( M_k \) with \( y \) sampled from the predictive distribution
8: Update \( A_l = \beta \cdot A_l + (1 - \beta) \cdot a_{l-1} a_{l-1}^\top \), \( G_l = \beta \cdot G_l + (1 - \beta) \cdot g_{l} g_{l}^\top \)
9: if \( k \leq T \) or \( k \equiv 0 \) (mod \( T \)) then
10: Recompute \( H_l^{H_l} = (A_l + \sqrt{\lambda} I)^{-1} \), \( H_l^{H_l} = (G_l + \sqrt{\lambda} I)^{-1} \)

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 \( L(a_l, y) = \sum n [y_n \log a_{L,n} + (1 - y_n) \log(1 - a_{L,n})] \), MSE \( L(a_l, y) = \frac{1}{2} \sum_n (a_{L,n} - y_n)^2 \). 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>
<thead>
<tr>
<th>Dataset</th>
<th># data points</th>
<th># training examples</th>
<th># testing examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>70,000</td>
<td>60,000</td>
<td>10,000</td>
</tr>
<tr>
<td>FACES</td>
<td>165,600</td>
<td>103,500</td>
<td>62,100</td>
</tr>
<tr>
<td>CURVES</td>
<td>30,000</td>
<td>20,000</td>
<td>10,000</td>
</tr>
</tbody>
</table>

Table 4: Architecture of 3 auto-encoder problems

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Layer width &amp; activation</th>
<th>Loss function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>[784, 1000, 500, 250, 30, 250, 500, 1000, 784] [ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, sigmoid]</td>
<td>binary entropy</td>
</tr>
<tr>
<td>FACES</td>
<td>[625, 2000, 1000, 500, 30, 500, 1000, 2000, 625] [ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, linear]</td>
<td>MSE</td>
</tr>
<tr>
<td>CURVES</td>
<td>[784, 400, 200, 100, 50, 25, 6, 25, 50, 100, 200, 400, 784] [ReLU, ReLU, ReLU, ReLU, ReLU, ReLU, linear, ReLU, ReLU, ReLU, ReLU, ReLU, sigmoid]</td>
<td>binary entropy</td>
</tr>
</tbody>
</table>

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 \( \alpha_k = \alpha \in \{ 1e-5, 3e-5, 1e-4, 3e-4, 1e-3, 3e-3, 1e-2, 3e-2, 1e-1, 3e-1, 1, 3, 10 \} \).
- 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.

\(^3\)Downloadable at [www.cs.toronto.edu/~jmartens/newfaces_rot_single.mat](www.cs.toronto.edu/~jmartens/newfaces_rot_single.mat)
\(^4\)Downloadable at [www.cs.toronto.edu/~jmartens/digs3pts_1.mat](www.cs.toronto.edu/~jmartens/digs3pts_1.mat)
Table 5: Best (learning rate, damping) for Figure 1

<table>
<thead>
<tr>
<th></th>
<th>K-BFGS</th>
<th>K-BFGS(L)</th>
<th>KFAC</th>
<th>Adam</th>
<th>RMSprop</th>
<th>SGD-m</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>(0.3, 0.3)</td>
<td>(0.3, 0.3)</td>
<td>(1, 3)</td>
<td>(1e-4, 1e-4)</td>
<td>(1e-4, 1e-4)</td>
<td>(0.03, -)</td>
</tr>
<tr>
<td>FACES</td>
<td>(0.1, 0.1)</td>
<td>(0.1, 0.1)</td>
<td>(0.1, 0.1)</td>
<td>(1e-4, 1e-4)</td>
<td>(1e-4, 1e-4)</td>
<td>(0.01, -)</td>
</tr>
<tr>
<td>CURVES</td>
<td>(0.1, 0.01)</td>
<td>(0.3, 0.3)</td>
<td>(0.3, 0.3)</td>
<td>(1e-3, 1e-3)</td>
<td>(1e-3, 1e-3)</td>
<td>(0.1, -)</td>
</tr>
</tbody>
</table>

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 $(s, y)$ 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.

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.](image)

\(^5\)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 $\beta_2 = 0.9$. 

18
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) row 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

<table>
<thead>
<tr>
<th></th>
<th>K-BFGS</th>
<th>K-BFGS(L)</th>
<th>KFAC</th>
<th>Adam</th>
<th>RMSprop</th>
<th>SGD-m</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>(0.1, 0.3)</td>
<td>(0.1, 0.3)</td>
<td>(0.1, 0.3)</td>
<td>(1e^{-4}, 1e^{-4})</td>
<td>(1e^{-4}, 1e^{-4})</td>
<td>(0.03, -)</td>
</tr>
<tr>
<td>FACES</td>
<td>(0.03, 0.03)</td>
<td>(0.03, 0.3)</td>
<td>(0.03, 0.3)</td>
<td>(3e^{-5}, 1e^{-4})</td>
<td>(3e^{-5}, 1e^{-4})</td>
<td>(0.01, -)</td>
</tr>
<tr>
<td>CURVES</td>
<td>(0.3, 1)</td>
<td>(0.3, 0.3)</td>
<td>(0.03, 0.1)</td>
<td>(3e^{-4}, 1e^{-4})</td>
<td>(3e^{-3}, 1e^{-4})</td>
<td>(0.03, -)</td>
</tr>
</tbody>
</table>
Figure 4: Comparison between algorithms on MNIST with batch size 100
Figure 5: Comparison between algorithms on FACES 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 \( \nabla f \) computed in the second pass of the previous iteration that is needed for computing the \((s, y)\) pair for applying the BFGS or L-BFGS updates, and average it with the stochastic gradient \( \nabla f \) 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 \( H_n \), 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