Making L-BFGS Work with Industrial-Strength Nets

Abhay Yadav
jaiabhay@cs.umd.edu
Tom Goldstein
tomg@cs.umd.edu
David Jacobs
djacobs@cs.umd.edu

University of Maryland
College Park,
Maryland, USA

Abstract

L-BFGS has been one of the most popular methods for convex optimization, but good performance by L-BFGS in deep learning has been elusive. Recent work has modified L-BFGS for deep networks for classification tasks and been able to show performance competitive with SGD and Adam (the most popular current algorithms) when batch normalization is not used. However, this work cannot be applied with batch normalization. Since batch normalization is a defacto standard and important to good performance in deep networks, this still limits the use of L-BFGS. In this paper, we address this issue. Our proposed method can be used as a drop-in replacement without changing existing code. The proposed method performs consistently better than Adam and existing L-BFGS approaches, and comparable to carefully tuned SGD. We show results on three datasets, CIFAR-10, CIFAR-100, and STL-10 using three different popular deep networks ResNet, DenseNet and Wide ResNet. This work marks another significant step towards making L-BFGS competitive in the deep learning community.

1 Introduction

Deep learning has become an indispensable tool in computer vision and emerged as a clear winner in many important applications such as image classification [16, 18, 40] and object detection [14]. Despite its enormous success, training a deep network presents many computational and optimization challenges. For example, (SGD) stochastic gradient descent [1, 6, 31] is the standard algorithm used to train deep networks, but it requires an **expert** to find the optimal learning rate and decay schedule [34, 36]. This usually requires an expensive grid search over possible parameter values, especially costly since training a single instance of a deep network takes considerable resources.

Second order methods like L-BFGS [26] (perhaps the most commonly used second-order method in machine learning) have a proven track record of handling these issues for simple classifiers. They automatically select learning rates and provide fast convergence, along with several other advantages over SGD [35]. Recently, several attempts have been made to explore L-BFGS for deep networks as well [3, 5, 29]. For example, Bollapragada et al. [5] achieved performance comparable to SGD. One would expect that the advantages of

L-BFGS should have popularized the use (or at-least promoted further exploration) of the method for training deep networks. However, the applicability of L-BFGS has been limited because it does not play well with state of the art networks [16, 18, 40] that rely on Batch Normalization [9, 20, 32].

Batch Normalization (BatchNorm) is an integral component of almost all modern deep networks (Ioffe and Szegedy [20] has more than 15,000 citations according to Google Scholar). To put things in perspective, without BatchNorm the classification accuracy of deep networks drops to almost 70% [5] from 92% [17] on CIFAR-10 using ResNet [16], and one can notice a similar drop in accuracy for many other popular deep networks and datasets. This discourages researchers from trying L-BFGS on deep networks without BatchNorm.

In this work we show how to get L-BFGS to work well with BatchNorm, obtaining performance comparable to SGD. Our method is general enough to apply to a wide range of L-BFGS variants [12, 13, 42], assuming only that curvature pair updates are done using finite gradient differencing. These updates consist of estimating a component of curvature by taking the difference between the gradient at different locations. Hopefully, our results will encourage more use of L-BFGS in deep learning, and more research on how to further improve its performance.

Contributions: We describe a novel scheme for stable curvature pair updates in the stochastic L-BFGS method for deep networks, in a way that is robust to the presence of BatchNorm. The scheme is generic enough to be applied to any variant of L-BFGS that employs a finite gradient differencing approach. This makes our method even more appealing, as it opens the possibility of borrowing other existing L-BFGS variants and their tricks. An additional advantage is that the method needs almost no parameter tuning, which is one of the benefits of using L-BFGS, and very important while training deep networks. Our numerical experiments show that the method proposed in this paper – which we call the Frozen-Batch L-BFGS (FbLBFGS) method – outperforms existing L-BFGS approaches by a large margin (more than 10% in generalization accuracy for some problems) and also achieves performance comparable to SGD on standard large scale networks such as ResNet [16], DenseNet [18], and Wide ResNet [40] as demonstrated on standard datasets including CIFAR-10 [25], STL-10 [8], and CIFAR-100 [24].

Frozen-Batch L-BFGS (FbLBFGS) Method: In this paper, we study how to design a stable curvature pair update in a stochastic setting when BatchNorm is used. We first note that it is crucial to take two gradient steps with the same training batch to obtain a consistent curvature estimate to use in updating the inverse Hessian. If two different batches are used, the gradient noise may dominate the curvature computation, destabilizing the update. In the frozen batch method, each time we select a new batch, we freeze it and take two gradient steps before making the Hessian update. This is a simple trick, and has been used before in a closely related method for online L-BFGS [33], namely oLBFGS, which also computes the finite difference of gradients using the same batch. The key difference is that oLBFGS always uses a fresh batch to take the actual descent gradient step (the recycled batch is only used for the Hessian update), whereas the proposed method uses the recycled batch for both the Hessian update and gradient step. This subtle difference has a huge impact on performance when BatchNorm is used.

2 Background and Related Work

Second-order methods have been studied in both convex and non-convex optimization, for more details see [3, 4, 5, 7, 10, 11, 21, 28, 33, 37, 39, 42]. Schraudolph et al. [33] proposed an online L-BFGS (oLBFGS) method to ensure a stable quasi-Newton curvature pair update by computing gradients on the same batch at the beginning and end of the iteration. Since this results in an extra computation of the gradients, Berahas et al. [4] proposed to use overlapping batches that share a subset of their data samples. In this case, the Hessian is updated using only samples that are shared between two adjacent batches, while the graident descent step uses all samples in a batch. This idea was further explored for large scale machine learning problems by Berahas and Takáč [3] and Bollapragada et al. [5]. Wang et al. [39] further extended oLBFGS and proposed a damped version (SdLBFGS) of it to maintain stable convergence in stochastic setting. Other approaches approximate curvature using the Fisher information matrix [2, 15, 27]. Krishnan et al. [23] approximately computes the inverse Hessian by expanding the matrices as the Neumann power series.

Multi-Batch L-BFGS

Let us consider the problem

$$\min_{x\in\mathbb{R}^d} F(x) \triangleq \frac{1}{N}\sum_{i=1}^{N} F_i(x) = \frac{1}{N}\sum_{i=1}^{N} f(x;z^i), \tag{1}$$

where $F_i(x) = f(x;z^i)$, f is a function (parametrized by x), and (z^i) is a collection of data drawn from an unknown probability distribution $P(z)$. A stochastic quasi-Newton method is given by

$$x_{k+1} = x_k - \alpha_k H_k g_k^{S_k}, \tag{2}$$

where the batch gradient is

$$g_k^{S_k} = \nabla F_{S_k}(x_k) \triangleq \frac{1}{|S_k|}\sum_{i\in S_k} \nabla F_i(x_k), \tag{3}$$

the set $S_k \subset \{1,2,\cdots\}$ indexes data points $\{z^i\}$ sampled from the distribution P, and H_k is a positive definite approximation to the inverse Hessian.

Stable Quasi-Newton Updates

In the L-BFGS methods, the inverse Hessian approximation is updated using the following recursive formula,

$$\begin{aligned} H_{k+1} &= V_k^T H_k V_k + \rho_k \theta_k \theta_k^T \\ \rho_k &= (y_k^T \theta_k)^{-1} \\ V_k &= I - \rho_k y_k \theta_k^T \end{aligned} \tag{4}$$

where $\theta_k = x_{k+1} - x_k$ and $y_k = \nabla F_{S_{k+1}}(x_{k+1}) - \nabla F_{S_k}(x_k)$ is the difference in the gradients at x_{k+1} and x_k. When the batch changes from one iteration to the next ($S_{k+1} \neq S_k$), y_k is computed using different samples and the updating process (which is very sensitive to noise) may be unstable. To fix this, one approach is to repeat the same batch twice [33, 39] to compute gradient at both the iterates (x_k and x_{k+1}), given by

$$y_k = g_{k+1}^{S_k} - g_k^{S_k}. \tag{5}$$

However, this comes at the additional cost of wasted gradient computation. To avoid this, Berahas et al. [4] and Bollapragada et al. [5] propose to use overlapping batches, and compute Hessian updates using only the overlapping samples using the formula

$$y_k = g_{k+1}^{O_k} - g_k^{O_k}, \tag{6}$$

where $O_k = S_k \cap S_{k+1}$. This needs no extra computation since the two gradients in this case are subsets of the gradients corresponding to the samples S_k and S_{k+1}.

Stochastic Line Search

Historically, L-BFGS is combined with a line search method that automatically selects a stepsize by checking that the objective decreases sufficiently on each iteration, and cutting the stepsize if not. Bollapragada et al. [5] propose to perform a backtracking line search that aims to satisfy the Armijo condition

$$F_{S_k}(x_k - \alpha_k H_k g_k^{S_k}) \leq F_{S_k}(x_k) - c_1 \alpha_k (g_k^{S_k})^T H_k g_k^{S_k}, \tag{7}$$

where $0 < c_1 < 1$. This condition checks whether the observed decrease in the loss function is at least c_1 times the decrease predicted by a local linear approximation. This condition guarantees convergence in the deterministic setting, but not in the stochastic setting. The initial value of α_k is given by,

$$\alpha_k = \left(1 + \frac{\mathrm{Var}_{i \in S_k^v}\{g_k^i\}}{|S_k| \left\| g_k^{S_k} \right\|^2} \right)^{-1}, \tag{8}$$

where $\mathrm{Var}_{i \in S_k^v}\{g_k^i\} = \frac{1}{|S_k^v|-1} \sum_{i \in S_k^v} \left\| g_k^i - g_k^{S_k} \right\|^2$, and $S_k^v \subseteq S_k$.

Other authors suggest using a decaying learning rate such as $1/\sqrt{k}$ [33, 39]. This decaying learning rate is more theoretically justified in that convergence is guaranteed if the Hessian approximation is constant, but in practice this may be slow.

Batch Normalization

The batch normalization[1] (BatchNorm) normalizes the activation output $f_l(x_k; z^i)$ of a given layer l as follows:

$$\begin{aligned} \mu^{S_k} &\leftarrow \frac{1}{|S_k|}\sum_{i=1}^{|S_k|} f_l(x_k; z^i) \\ \mathrm{Var}^{S_k} &\leftarrow \frac{1}{|S_k|}\sum_{i=1}^{|S_k|} (f_l(x_k; z^i) - \mu^{S_k})^2 \\ f_l(x_k; z^i; \mu^{S_k}, \mathrm{Var}^{S_k}) &\leftarrow \frac{f_l(x_k; z^i) - \mu^{S_k}}{\sqrt{\mathrm{Var}^{S_k} + \varepsilon}} \\ &\equiv \mathrm{BN}^{S_k}(f_l(x_k; z^i)) \end{aligned} \tag{9}$$

where $z^i \in S_k$, S_k is the batch at iteration k, ε is a small number used for numerical stability and f_l is the transformation function of the layer l. From Eq 9, it is evident that the BN transform does not independently process each training example. Rather, $\mathrm{BN}^{S_k}(f_l(x_k; z^i))$ is a function of both the training example and the other examples in that batch. For more details refer to [19].

3 ProposedMethod

3.1 Stable Quasi-Newton Updates With BatchNorm:

We now explain the challenges BatchNorm poses for existing L-BFGS approaches and then we discuss our proposed solution. As per the overlapping batch approach suggested by Berahas et al. [4], Bollapragada et al. [5], the stochastic gradient difference y_k with BatchNorm can be written as:

$$\begin{aligned} y_k = &\frac{1}{|O_k|}\sum_{i\in O_k} \nabla F_i(x_{k+1}; \mathrm{BN}^{S_{k+1}}) \\ &- \frac{1}{|O_k|}\sum_{i\in O_k} \nabla F_i(x_k; \mathrm{BN}^{S_k}), \end{aligned} \tag{10}$$

where BN^{S_k} represents the batch normalization statistics for the batch S_k. From (10), it is clear that because of the different BatchNorm parameters, the gradients used to compute y_k are not consistent; even though only overlapping samples are used, the batch norm statistics depend on the non-overlapping samples. This breaks the gradient consistency for the overlapping approach.

To address this issue, one obvious solution is to repeat the same batch twice [33, 39]. However, this requires that the gradient be evaluated twice for every batch S_k at x_k and x_{k+1}. We make use of the extra gradient computation by actually taking another gradient step. We update the Hessian only using gradients from the same batch. Specifically, we propose to freeze the batch for two consecutive iterations, take two gradient steps, and then update the curvature pair (y_k, θ_k). The newly updated Hessian is applied to a gradient from the

[1]For clarity, we omit the learnable parameters γ and β, which produce an affine transformation applied on top of the batch-norm layer. This is just another trainable layer, not affecting the proposed analysis. See [19] for more details.

same batch. Then in the next iteration, when changing the batch from S_k to S_{k+1}, we do not update the curvature pair. Put another way, we compute two gradients for each batch, and we take a gradient descent step using both of these gradients. We find that this approach works significantly better than the existing approach SdLBFGS [39], improving L-BFGS performance by a large margin. We believe this is due to the fact that that the gradient direction in our method gets pre-conditioned by a Hessian that was updated on the same batch. While with Wang et al. [39], one takes a (consistent and probably stable) Hessian update on one batch, and then use it to pre-condition on another.

Algorithm 1 Frozen-Batch L-BFGS (FbLBFGS)

Input: x_0 (initial iterate), $D = \{(z^i, t^i), \text{for } i = 1, \ldots, n\}$ (training data), m (memory parameter), U = False (Flag to control curvature update).

```
 1: Create initial batch $S_1$
 2: for $k = 1, 2, \ldots$ do
 3:     if $k == 1$ then
 4:         Set the search direction $p_k = -g_k^{S_k}$
 5:     else
 6:         Calculate the search direction $p_k = -H_k g_k^{S_k}$ {Using L-BFGS Two-Loop Recursion (Procedure 3.1 in [39])}
 7:     end if
 8:     Normalize the search direction $p_k = \frac{p_k}{||p_k||_2}$
 9:     Set $\alpha_k = 1$
10:     while the Armijo condition (7) not satisfied do
11:         Set $\alpha_k = \alpha_k / 2$
12:     end while
13:     Compute $x_{k+1} = x_k + \alpha_k p_k$
14:     if $U$ is True then
15:         Compute the curvature pairs $\theta_k = x_{k+1} - x_k$ and $y_k = g_{k+1}^{S_k} - g_k^{S_k}$
16:         Replace the oldest pair $(\theta_i, y_i)$ by $\theta_k, y_k$
17:         Create the next batch $S_{k+1}$
18:         Set $U$ = False {Do not update curvature in next iteration}
19:     else
20:         Set $U$ = True {Update curvature in next iteration}
21:         Set $S_{k+1} = S_k$ {Freeze the sample in next iteration}
22:     end if
23: end for
```

3.2 Line Search:

The stochastic line search (eq 7) proposed by Bollapragada et al. [5] requires computation of the initial value of α_k, which in turn requires computation of the variance of the gradient for each example. This is not possible when using BatchNorm, since estimation of the variance of the gradient requires a forward pass for each example one by one, which will change the batch statistics.

In the stochastic case, sometimes the norm of the search direction can be too large, causing the algorithm to be very unstable. To cater to this, we propose a heuristic: we normalize the

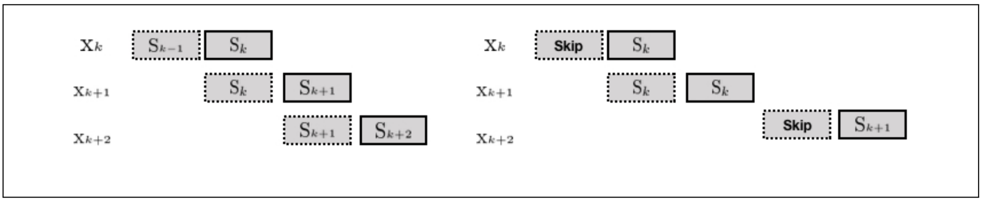

Figure 1: A side-by-side schematic depiction of curvature pair (y_k, θ_k) update scheme in SdLBFGS [39] and FbLBFGS (ours). Here, the dotted box represents the batch used for a curvature pair update and the solid box represents the batch used for the gradient step. The SdLBFGS method computes an auxiliary stochastic gradient at x_k using the sample S_{k-1} from the $(k-1)$-st iteration, which is used for gradient differencing only (to update y_k). Then a gradient step is taken on a new batch S_k and the process repeats. On the other hand, FbLBFGS (proposed) does not compute any auxiliary gradients but skips the first curvature pair update (k-th iteration), takes a gradient step, and in the next iteration ($(k+1)$-st) the same frozen batch S_k is used both to update the curvature pair and to take another gradient step. In the following next iteration (x_{k+2}), when the batch changes to S_{k+1}, the Hessian update is skipped, and the process repeats.

search direction (-$H_k g_k^{S_k}$) before doing the standard Armijo line search. This heuristic works surprisingly well in practice for all the datasets and all the models we tried.

3.3 On Convergence

In this section, we discuss the convergence of the proposed mehtod (FbLBFGS). The main challenge in designing a stochastic L-BFGS method for non-convex problem lies in the difficulty in preserving the positive-definiteness of the inverse Hessian approximation H_k, due to the non-convexity of the problem and the noise in gradient estimation. Wang et al. [39] proposed to address this issue by using damped curvature pair update. We leverage the convergence proof by Wang et al. [39] and show that we do not break their convergence conditions.

When frozen batch is used, we update the inverse Hessian approximation following [39]:

$$\begin{aligned} H_{k+1} &= V_k^T H_k V_k + \rho_k \theta_k \theta_k^T \\ \hat{y}_k &= \lambda_k y_k + (1-\lambda_k) H_k^{-1} \theta_k \\ \rho_k &= (\hat{y}_k{}^T \theta_k)^{-1} \\ V_k &= I - \rho_k \hat{y}_k \theta_k^T , \end{aligned} \tag{11}$$

where $\theta_k = x_{k+1} - x_k$ and $y_k = \nabla F_{S_k}(x_{k+1}) - \nabla F_{S_k}(x_k)$ is the difference in the gradients at x_{k+1} and x_k. The damping factor λ_k is give by,

$$\lambda_k = \begin{cases} \frac{0.75\theta_k^\top H_k^{-1}\theta_k}{\theta_k^\top H_k^{-1}\theta_k - \theta_k^\top y_k}, & \text{if } \theta_k^\top y_k < 0.25\theta_k^\top H_k^{-1}\theta_k, \\ 1, & \text{otherwise.} \end{cases} \tag{12}$$

When the batch is changed, we do not update the inverse Hessian approximation, i.e., we skip the update and set $H_{k+1} = H_k$. So, this ensures that the inverse Hessian approximation always remains positive-definite.

4 Experiments

In this section, we empirically demonstrate the proposed method's effectiveness on three benchmark datasets: STL-10 [8], CIFAR-10 [25], and CIFAR-100 [24]. Our results show three main points. First, without fine tuning, FbLBFGS can obtain generalization performance competitive with carefully tuned SGD using practical state-of-the-art architectures that use BatchNorm [19]. Second, FbLBFGS outperforms existing adaptive optimizers that automatically set the learning rate with minimal or no user supervision. Third, we show that our simple approach substantially outperforms existing L-BFGS methods.

It is impressive that FbLBFGS's performance approaches that of highly tuned SGD. For each model and dataset, untold grad student hours have been spent on grid search for good hyperparameters. Often these are chosen based on performance on the test set, instead of a separate held out dataset, overfitting the data [30]. To get a better sense of this, we also test SGD with a standard method of automatic parameter tuning. S1 and S2 are defined as follows: the learning rate starts with 0.1 for S1 or 1 for S2 and is reduced by $1/10$ after every $100, 200, 300$ epochs. FbLBFGS often outperforms these approaches.

We also compare our method against Adam [22]. Adam is the most popular and effective adaptive optimizer. We find that other adaptive optimizers [38, 41] perform similarly. FbLBFGS always significantly outperforms Adam.

Finally, we also compare the proposed method with existing L-BFGS methods. Specifically, we compare against the overlapping batch approach by Bollapragada et al. [5] and the stochastic damped L-BFGS (SdLBFGS) method by Wang et al. [39]. We use our line search with both of these so that we can compare our approach of freezing batches to prior approaches that address the same issue.

In all experiments, the batch size used is 128, history size is 5, and default parameters for Adam and other L-BFGS methods are used. We ran all the methods for 350 epochs. It is observed that as the optimizer reaches towards the solution, the gradients might be too noisy, and to get high fidelity gradients its recommended to increase the batch size [5, 11]. To address this issue, we set the batch size to 25% of the dataset towards the end of the training, for all the experiments. The deep networks used are WRN-28-10 [40], ResNet-18 [16], and DenseNet-40-12 [18]. The computational cost of FbLBFGS is increased because it makes two passes from the same frozen batch twice for each epoch. For a fair comparisons, we ran all other methods with a similar frozen batch as well (but did not notice any change in the performance as compared to using the batch only once per epoch). We report results over 5 different random seeds, as shown in Table 1, Table 2, and Table 3, for STL-10, CIFAR-10 and CIFAR-100, respectively. We also show the training and test accuracy (Figure 2) for STL-10, CIFAR-10, and CIFAR-100 on the deep networks Wide ResNet, DenseNet, and ResNet, respectively[2].

From the results, one can observe that our approach (FbLBFGS) always outperforms all the adaptive methods. Its performance compared with tuned SGD is comparable; sometimes it performs better than tuned SGD, sometimes about the same, and sometimes a bit worse. This shows the effectiveness of our approach to BatchNorm and also the practical usefulness of L-BFGS when applying deep networks to problems in which SGD has not been carefully tuned, since FbLBFGS has no free parameters.

The Figure 3 shows the number of function evaluations per line search for each L-BFGS method. The overlap and SdLBFGS methods take nearly 10 times more backtracking steps

[2]All the training curves are in the supplementary material.

than our method, which is likely due to a better Hessian approximation by our method. We also observed that the Armijo condition almost always accepts the step-length 1, which means there is little overhead for using the line search. We believe normalizing the search direction before doing a line search is an effective heuristic. Without normalizing the search direction, all L-BFGS methods perform poorly. Hence, we ignored those results.

Method	ResNet Train	ResNet Test	DenseNet Train	DenseNet Test	Wide ResNet Train	Wide ResNet Test
SGD*	99.9	73.7 ±0.3	99.9	**73.5** ±0.2	99.9	**77.1**±0.5
SGD(S1)	99.8	68.0 ±0.2	99.9	63.9 ±0.2	99.9	71.1 ±0.2
SGD(S2)	99.9	72.5 ±0.3	99.9	71.9 ±0.3	99.9	71.8 ±0.6
Adam	99.9	69.5 ±0.1	99.9	70.2 ±0.2	99.9	69.2 ±0.2
Bollapragada et al. [5]	95.8	65.8 ±0.1	98.2	67.2 ±0.4	93.2	62.5 ±0.7
SdLBFGS [39]	96.7	66.8 ±0.3	96.7	69.8 ±0.2	94.8	65.1 ±0.8
FbLBFGS (ours)	99.8	**75.1** ±0.1	99.9	73.4 ±0.5	99.9	76.4 ±0.4

Table 1: Comparison of train/test accuracy for STL-10 on ResNet, DenseNet, and Wide ResNet respectively. The results are shown in the format of 'mean ±*std*' computed over 5 random seeds. Higher are better. $\star$ denotes highly tuned SGD with grid search and potentially over-fitting the test set for a particular model. We report these results for complete perspective.

Method	ResNet Train	ResNet Test	DenseNet Train	DenseNet Test	Wide ResNet Train	Wide ResNet Test
SGD*	99.9	92.1 ±0.2	99.6	91.2 ±0.3	99.9	**95.2** ±0.5
SGD(S1)	99.8	90.6 ±0.3	99.5	89.9 ±0.4	99.9	93.2 ±0.4
SGD(S2)	90.6	89.2 ±0.2	99.9	**93.1** ±0.1	99.9	94.8 ±0.2
Adam	97.1	91.4 ±0.5	95.2	89.2 ±0.1	95.1	90.1 ±0.3
Bollapragada et al. [5]	93.7	84.6±0.6	95.0	85.6 ±0.4	95.8	90.1 ±0.3
SdLBFGS [39]	94.5	86.2 ±0.2	95.5	87.4 ±0.3	96.1	88.1 ±0.2
FbLBFGS (ours)	99.9	**92.9** ±0.4	99.5	91.2 ±0.1	99.8	94.2 ±0.3

Table 2: Comparison of train/test accuracy for CIFAR-10 on ResNet, DenseNet, and Wide ResNet respectively. The results are shown in the format of 'mean ±*std*' computed over 5 random seeds. Higher are better. $\star$ denotes highly tuned SGD with grid search and potentially over-fitting the test set for a particular model. We report these results for complete perspective.

Method	ResNet Train	ResNet Test	DenseNet Train	DenseNet Test	Wide ResNet Train	Wide ResNet Test
SGD*	96.5	**66.6** ±0.2	97.1	**68.4**±0.1	99.9	**79.4** ±0.3
SGD(S1)	93.1	63.5 ±0.3	97.9	67.5 ±0.3	99.9	76.7 ±0.4
SGD(S2)	96.0	66.2 ±0.4	96.9	67.0 ±0.2	99.9	79.4 ±0.3
Adam	94.9	63.2 ±0.5	97.5	63.4 ±0.2	97.4	67.0 ±0.4
Bollapragada et al. [5]	89.6	58.8 ±0.3	72.1	62.2 ±0.5	91.2	66.4 ±0.3
SdLBFGS [39]	89.1	59.5 ±0.3	75.3	64.8 ±0.3	95.8	66.6 ±0.1
FbLBFGS (ours)	93.8	65.2 ±0.3	95.2	67.9 ±0.3	99.5	75.4±0.2

Table 3: Comparison of train/test accuracy for CIFAR-100 on ResNet, DenseNet, and Wide ResNet respectively. The results are shown in the format of 'mean ±*std*' computed over 5 random seeds. Higher are better. $\star$ denotes highly tuned SGD with grid search and potentially over-fitting the test set for a particular model. We report these results for complete perspective.

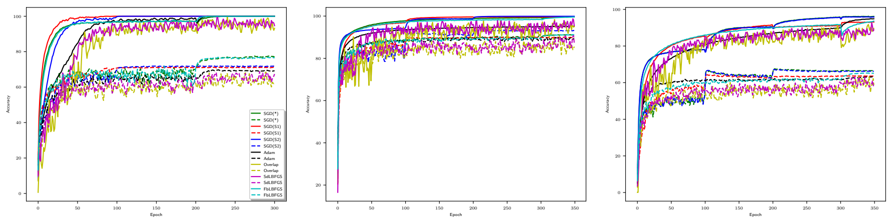

Figure 2: Overview of the performance of STL-10, CIFAR-10 and CIFAR-100 on Wide ResNet, DenseNet, and ResNet respectively. The solid lines represent train accuracy and dashed lines represent test accuracy, respectively.

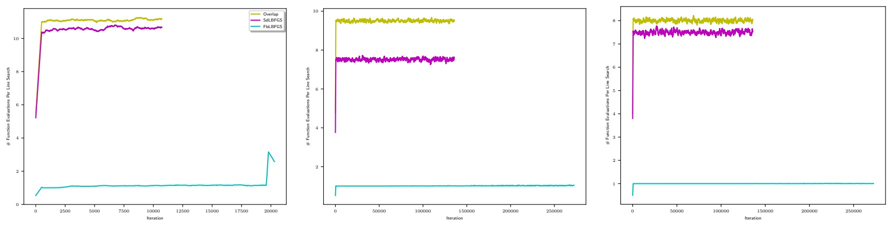

Figure 3: The number of function evaluations per line search for STL-10, CIFAR-10, and CIFAR-100 on Wide ResNet, DenseNet, and ResNet, respectively. The average of every consecutive 100 iteration is plotted for display purpose.

5 Conclusion

It has proven challenging to effectively apply L-BFGS training methods to neural networks with batch normalization. We have shown how to make a simple and extremely effective modification to L-BFGS that makes it competitive with well-tuned SGD on classification tasks with BatchNorm. Our approach uses a frozen batch to ensure that all elements of the Hessian update are based on the same batch statistics. Along with a new approach to line search that reduces the number of expensive backtracking steps, we achieve results that considerably improve on previous L-BFGS implementations for training neural networks with batch normalization.

We find it a bit surprising to see L-BFGS compete with tuned SGD; the community often assumes that L-BFGS is overly aggressive, and line search methods get stuck in local minima. Interestingly, when applied to high-performance networks with BatchNorm (with the proposed modifications to maintain stability), L-BFGS does not suffer from these problems. This further supports the intuition that BatchNorm, while not understood theoretically, promotes more well behaved loss functions.

Acknowledgements

The work of A. Yadav and D. Jacobs was supported by the National Science Foundation, grant no. IIS-1910132 and the Guaranteeing AI Robustness Against Deception (GARD) program from DARPA. The work of PI Goldstein was supported by the ONR and AFOSR MURI programs, and the National Science Foundation Directorate of Mathematical Sciences.

References

[1] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5 (4-5):185–196, 1993.

[2] Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. 2016.

[3] Albert S Berahas and Martin Takáč. A robust multi-batch l-bfgs method for machine learning. *arXiv preprint arXiv:1707.08552*, 2017.

[4] Albert S Berahas, Jorge Nocedal, and Martin Takác. A multi-batch l-bfgs method for machine learning. In *Advances in Neural Information Processing Systems*, pages 1055–1063, 2016.

[5] Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. A progressive batching l-bfgs method for machine learning. In *International Conference on Machine Learning*, pages 619–628, 2018.

[6] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[7] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[8] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[9] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[10] Frank Curtis. A self-correcting variable-metric algorithm for stochastic optimization. In *International Conference on Machine Learning*, pages 632–641, 2016.

[11] Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Automated inference with adaptive batches. In *Artificial Intelligence and Statistics*, pages 1504–1513, 2017.

[12] Yun Fei, Guodong Rong, Bin Wang, and Wenping Wang. Parallel l-bfgs-b algorithm on gpu. *Computers & Graphics*, 40:1–9, 2014.

[13] Wenbo Gao and Donald Goldfarb. Quasi-newton methods: superlinear convergence without line searches for self-concordant functions. *Optimization Methods and Software*, 34(1):194–217, 2019.

[14] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[15] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[21] Nitish Shirish Keskar and Albert S Berahas. adaqn: An adaptive quasi-newton algorithm for training rnns. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 1–16. Springer, 2016.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Shankar Krishnan, Ying Xiao, and Rif A Saurous. Neumann optimizer: A practical optimization algorithm for deep neural networks. 2018.

[24] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6, 2009.

[25] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

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

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

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

[29] Jacob Rafati and Roummel F Marcia. Improving l-bfgs initialization for trust-region methods in deep learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 501–508. IEEE, 2018.

[30] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

[31] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[32] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

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

[34] Andrew Senior, Georg Heigold, Ke Yang, et al. An empirical study of learning rates in deep neural networks for speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6724–6728. IEEE, 2013.

[35] Adrian J Shepherd. *Second-order methods for neural networks: Fast and reliable training methods for multi-layer perceptrons*. Springer Science & Business Media, 2012.

[36] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[37] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *International Conference on Machine Learning*, pages 604–612, 2014.

[38] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

[39] Xiao Wang, Shiqian Ma, Donald Goldfarb, and Wei Liu. Stochastic quasi-newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, 27(2):927–956, 2017.

[40] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[41] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[42] Chaoxu Zhou, Wenbo Gao, and Donald Goldfarb. Stochastic adaptive quasi-newton methods for minimizing expected values. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4150–4159. JMLR. org, 2017.