Supplementary Materials for “Distributed Newton Methods for Deep Neural Networks”

Chien-Chih Wang,[1] Kent Loong Tan,[1] Chun-Ting Chen,[1] Yu-Hsiang Lin,[2] S. Sathiya Keerthi,[3] Dhruv Mahajan,[4] S. Sundararajan,[3] Chih-Jen Lin[1]

[1]Department of Computer Science, National Taiwan University, Taipei 10617, Taiwan
[2]Department of Physics, National Taiwan University, Taipei 10617, Taiwan
[3]Microsoft
[4]Facebook Research

I List of Symbols

Notation	Description
$\boldsymbol{y}^i$	The label vector of the ith training instance.
$\boldsymbol{x}^i$	The feature vector of the ith training instance.
l	The number of training instances.
K	The number of classes.
$\boldsymbol{\theta}$	The model vector (weights and biases) of the neural network.
ξ	The loss function.
ξ_i	The training loss of the ith instance.
f	The objective function.
C	The regularization parameter.
L	The number of layers of the neural network.
n_m	The number of neurons in the mth layer.
n_0	The number of input neurons (the dimension of the feature vector).
n_L	The number of output neurons (the number of classes, except for binary classification one may use $n_L = 1$).
W^m	The weight matrix in the mth layer (with dimension $\Re^{n_{m-1}\times n_m}$).
w_{tj}^m	The weight between neuron t in the $(m-1)$th layer and neuron j in the mth layer.
$\boldsymbol{w}^m$	The vector obtained by concatenating the columns of W^m.
$\boldsymbol{b}^m$	The bias vector in the mth layer.
$\boldsymbol{s}^{m,i}$	The affine function $(W^m)^T\boldsymbol{z}^{m-1,i}+\boldsymbol{b}^m$ in the mth layer for the ith instance.
$\boldsymbol{z}^{m,i}$	The output vector (element-wise application of the activation function on $\boldsymbol{s}^{m,i}$) in the mth layer for the ith instance.
σ	The activation function.
n	The total number of weights and biases.
J^i	The Jacobian matrix of $\boldsymbol{z}^{L,i}$ with respect to $\boldsymbol{\theta}$.
J_p^i	The local component of the J^i in the partition p.

Notation	Description
B^i	The Hessian matrix of the loss function of the ith instance with respect to $z^{L,i}$ (the matrix element is $B^i_{ts} = \frac{\partial^2 \xi(\boldsymbol{z}^{L,i};\boldsymbol{y}^i)}{\partial z_t^{L,i} \partial z_s^{L,i}}$).
$\boldsymbol{\theta}^k$	The model vector $\boldsymbol{\theta}$ at the kth iteration.
H^k	The Hessian matrix $\nabla^2 f(\boldsymbol{\theta}^k)$ at the kth iteration.
G	The Gauss-Newton matrix of $f(\boldsymbol{\theta})$.
G^k	The Gauss-Newton matrix of $f(\boldsymbol{\theta}^k)$ at the kth iteration.
P	The number of partitions to the model variables.
T_m	A subset in $\{1, 2, \ldots, n_m\}$.
P_m	The set of T_m.
S	A subset in $\{1, 2, \ldots, l\}$.
S_k	A subset in $\{1, 2, \ldots, l\}$ chosen at the kth iteration.
G^S	The subsampled Gauss-Newton matrix of $f(\boldsymbol{\theta})$.
G^{S_k}	The subsampled Gauss-Newton matrix at the kth iteration.
$\boldsymbol{g}_p^k$	The local component of the gradient in the partition p at the kth iteration.
$\boldsymbol{d}^k$	The search direction at the kth iteration.
$\boldsymbol{v}$	An arbitrary vector in $\Re^n$.
$\boldsymbol{v}_p$	An arbitrary vector in $\Re^{\lvert T_{m-1}\rvert \times \lvert T_m \rvert}$ in the partition p.
$\mathcal{I}$	An identity matrix.
α_k	A step size at the kth iteration.
ρ_k	The ratio between the actual function reduction and the predicted reduction at the kth iteration.
λ_k	A parameter in the Levenberg-Marquardt method.
$\mathcal{N}(\mu, \sigma^2)$	A Gaussian distribution with mean μ and variance σ^2.

II Calculation of Hessian-vector Product Using $\mathcal{R}$ Operator

Here we demonstrate the calculation of the Hessian-vector product using $\mathcal{R}$ operator.

$$H^k \boldsymbol{v},$$

where

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}^1 \\ \bar{\boldsymbol{v}}^1 \\ \vdots \\ \boldsymbol{v}^L \\ \bar{\boldsymbol{v}}^L \end{bmatrix} \in \Re^n \tag{II.1}$$

is an iterate in the CG procedure. In the vector $\boldsymbol{v}$, sub-vectors $\boldsymbol{v}^m \in \Re^{n_{m-1} \times n_m}$ and $\bar{\boldsymbol{v}}^m \in \Re^{n_m}$ respectively correspond to variables $\boldsymbol{w}^m$ and $\boldsymbol{b}^m$ at the mth layer.

The definition of the $\mathcal{R}$ operator on any function is as follows

$$\mathcal{R}_{\boldsymbol{v}}\{f(\boldsymbol{\theta})\} = \frac{\partial}{\partial r} f(\boldsymbol{\theta} + r\boldsymbol{v}) \Big|_{r=0}. \tag{II.2}$$

Because

$$\begin{aligned}\mathcal{R}_{\boldsymbol{v}}\{\nabla f(\boldsymbol{\theta})\} &= \frac{\partial}{\partial r}\nabla f(\boldsymbol{\theta}+r\boldsymbol{v})\Big|_{r=0}\\ &= \nabla^2 f(\boldsymbol{\theta}+r\boldsymbol{v})\boldsymbol{v}\big|_{r=0}\\ &= \nabla^2 f(\boldsymbol{\theta})\boldsymbol{v},\end{aligned}$$

we can apply the $\mathcal{R}$ operator to (11)-(14) to get the Hessian-vector product.

$$\begin{aligned}\mathcal{R}_{\boldsymbol{v}}\{\frac{\partial \xi}{\partial s_j^{m,i}}\} &= \sigma'(s_j^{m,i})\mathcal{R}_{\boldsymbol{v}}\{\frac{\partial \xi}{\partial z_j^{m,i}}\} + \frac{\partial \xi}{\partial z_j^{m,i}}\sigma''(s_j^{m,i})\mathcal{R}_{\boldsymbol{v}}\{s_j^{m,i}\},\\ \mathcal{R}_{\boldsymbol{v}}\{\frac{\partial \xi}{\partial z_t^{m-1,i}}\} &= \sum_{j=1}^{n_m}\left(w_{tj}^m\mathcal{R}_{\boldsymbol{v}}\{\frac{\partial \xi}{\partial s_j^{m,i}}\} + \mathcal{R}_{\boldsymbol{v}}\{w_{tj}^m\}\frac{\partial \xi}{\partial s_j^{m,i}}\right), \qquad \text{(II.3)}\\ \mathcal{R}_{\boldsymbol{v}}\{\frac{\partial f}{\partial w_{tj}^m}\} &= \frac{1}{C}\,\mathcal{R}_{\boldsymbol{v}}\{w_{tj}^m\} + \frac{1}{l}\sum_{i=1}^{l}\left(z_t^{m-1,i}\mathcal{R}_{\boldsymbol{v}}\{\frac{\partial \xi}{\partial s_j^{m,i}}\} + \mathcal{R}_{\boldsymbol{v}}\{z_t^{m-1,i}\}\frac{\partial \xi}{\partial s_j^{m,i}}\right),\\ \mathcal{R}_{\boldsymbol{v}}\{\frac{\partial f}{\partial b_j^m}\} &= \frac{1}{C}\mathcal{R}_{\boldsymbol{v}}\{b_j^m\} + \frac{1}{l}\sum_{i=1}^{l}\mathcal{R}_{\boldsymbol{v}}\{\frac{\partial \xi}{\partial s_j^{m,i}}\}.\end{aligned}$$

In this work, instead of Hessian we consider the Gauss-Newton matrix, so (II.3) must be modified, and details are in Section III.

III Details of Using $\mathcal{R}$ Operators for Gauss-Newton Matrix Vector Products

The matrix-vector product can be obtained by following the settings in Schraudolph [2002], Martens and Sutskever [2012]. We consider the first-order approximation of $\boldsymbol{z}^{L,i}(\boldsymbol{\theta})$ to define

$$\hat{\boldsymbol{z}}^{L,i}(\boldsymbol{\theta}) = \boldsymbol{z}^{L,i}(\boldsymbol{\theta}^k) + \hat{J}^i(\boldsymbol{\theta}-\boldsymbol{\theta}^k) \approx \boldsymbol{z}^{L,i}(\boldsymbol{\theta}),\ i=1,\ldots,l,$$

where

$$\hat{J}^i = J^i\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}.$$

Next we define

$$\hat{f}(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l}\xi(\hat{\boldsymbol{z}}^{L,i};\boldsymbol{y}^i).$$

The gradient vector of $\hat{f}(\boldsymbol{\theta})$ is

$$\nabla\hat{f}(\boldsymbol{\theta}) = \frac{1}{C}\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l}(\hat{J}^i)^T\frac{\partial\xi(\hat{\boldsymbol{z}}^{L,i};\boldsymbol{y}^i)}{\partial\hat{\boldsymbol{z}}_j^{L,i}} \qquad \text{(III.4)}$$

and the Hessian matrix of $\hat{f}(\boldsymbol{\theta})$ is

$$\frac{1}{C}\mathcal{I} + \frac{1}{l}\sum_{i=1}^{l}(\hat{J}^i)^T\hat{B}^i\hat{J}^i$$

where

$$\hat{B}^i_{ts} = \frac{\partial^2 \xi(\hat{\boldsymbol{z}}^{L,i}; \boldsymbol{y}^i)}{\partial \hat{z}^{L,i}_t \partial \hat{z}^{L,i}_s},\ t = 1, \ldots, n_L,\ s = 1, \ldots, n_L.$$

Therefore, from (16) we can derive that

$$\nabla \hat{f}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k} = \nabla f(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k} \text{ and } \nabla^2 \hat{f}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k} = G|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}. \tag{III.5}$$

We can apply the $\mathcal{R}$ operator to $\nabla \hat{f}(\boldsymbol{\theta})$ and derive the Gauss-Newton matrix-vector product. That is,

$$\mathcal{R}_{\boldsymbol{v}}\{\nabla \hat{f}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}\} = \nabla^2 \hat{f}(\boldsymbol{\theta}^k)\boldsymbol{v} = G\boldsymbol{v}. \tag{III.6}$$

From (III.5), instead of using

$$\mathcal{R}\{\nabla \hat{f}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}\}$$

we can use

$$\mathcal{R}\{\nabla f(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}\}.$$

Then in (11)-(14), $z^{m-1,i}_t$, $s^{m,i}_j$, and w^m_{ta} can be considered as constant values after substituting $\boldsymbol{\theta}$ with $\boldsymbol{\theta}^k$. Therefore, the following results can be derived.

$$\mathcal{R}\{z^{m-1,i}_t\} = 0,\ \mathcal{R}\{s^{m,i}_j\} = 0, \text{ and } \mathcal{R}\{w^m_{ta}\} = 0.$$

The following equations are therefore used in a backward process to get the Gauss-Newton matrix-vector product.

$$\mathcal{R}\{\frac{\partial \xi}{\partial s^{m,i}_j}\} = \sigma'(s^{m,i}_j)\mathcal{R}\{\frac{\partial \xi}{\partial z^{m,i}_j}\}, \tag{III.7}$$

$$\mathcal{R}\{\frac{\partial \xi}{\partial z^{m-1,i}_t}\} = \sum_{j=1}^{n_m} w^m_{tj}\mathcal{R}\{\frac{\partial \xi}{\partial s^{m,i}_j}\},$$

$$\mathcal{R}\{\frac{\partial \hat{f}}{\partial w^m_{tj}}\} = \sum_{i=1}^{l} z^{m-1,i}_t \mathcal{R}\{\frac{\partial \xi}{\partial s^{m,i}_j}\},$$

$$\mathcal{R}\{\frac{\partial \hat{f}}{\partial b^m_j}\} = \sum_{i=1}^{l} \mathcal{R}\{\frac{\partial \xi}{\partial s^{m,i}_j}\}. \tag{III.8}$$

However, because

$$\mathcal{R}\{\frac{\partial \xi}{\partial z^{L,i}_j}\} = \frac{\partial^2 \xi}{\partial (z^{L,i}_j)^2}\mathcal{R}\{z^{L,i}_j\},$$

$\mathcal{R}\{z^{L,i}_j\}$ are also needed. They are not computed in the backward process. Instead, we can pre-calculate them in the following forward process.

$$\mathcal{R}\{s^{m,i}_j\} = \mathcal{R}\{\sum_{t=1}^{n_{m-1}} w^m_{tj} z^{m-1,i}_t + b^m_j\} = \sum_{t=1}^{n_{m-1}} (w^m_{tj}\mathcal{R}\{z^{m-1,i}_t\} + v^m_{tj} z^{m-1,i}_t) + \mathcal{R}\{b^m_j\} \tag{III.9}$$

$$\mathcal{R}\{z^{m,i}_j\} = \mathcal{R}\{\sigma(s^{m,i}_j)\} = \mathcal{R}\{s^{m,i}_j\}\sigma'(s^{m,i}_j)$$

where $m = 1, \ldots, L$. Notice that $\mathcal{R}\{z^{0,i}_t\} = 0$ because $z^{0,i}_t$ is a constant. For more details, see Section 6.1 of Martens and Sutskever [2012].

$$\begin{tikzcd}
 & (A_0, A_1) \arrow[dl, shift left] \arrow[dl, leftarrow, shift right] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & & & (A_0, B_1) \arrow[dl, shift left] \arrow[dl, leftarrow, shift right] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & \\
(B_0, A_1) & & (C_0, A_1) & (B_0, B_1) & & (C_0, B_1)
\end{tikzcd}$$

Figure IV.1: An example of using binary trees to implement *allreduce* operations for obtaining $s_j^{1,i}$, $\forall j \in \{1, \dots, n_1\}$, $i \in \{1, \dots, l\}$. The left tree corresponds to the calculation in (23).

$$\begin{tikzcd}
 & & (A_0, A_1) \arrow[dl] \arrow[dr] & & & & (A_0, B_1) \arrow[dl] \arrow[dr] & \\
 & (A_1, A_2) \arrow[dl] & & (A_1, B_2) & & (B_1, A_2) \arrow[dl] & & (B_1, B_2) \\
(A_1, C_2) & & & & (B_1, C_2) & & &
\end{tikzcd}$$

Figure IV.2: An example of using binary trees to broadcast $s_j^{1,i}$ to partitions between layers 1 and 2.

IV Implementation Details of Allreduce and Broadcast Operations

We follow Agarwal et al. [2014] to use a binary-tree based implementation. For the network in Figure 2, we give an illustration in Figure IV.1 for summing up the three local values respectively in partitions (A_0, A_1), (B_0, A_1), and (C_0, A_1), and then broadcast the resulting

$$s_j^{1,i},\ i = 1, \dots, l,\ j \in A_1 \tag{IV.10}$$

back. After the *allreduce* operation, we broadcast the same values in (IV.10) to partitions between layers 1 and 2 by choosing (A_0, A_1) as the root of another binary tree shown in Figure IV.2. For values

$$s_j^{1,i},\ i = 1, \dots, l,\ j \in B_1,$$

a similar *allreduce* operation is applied to (A_0, B_1), (B_0, B_1) and (C_0, B_1). The result is broadcasted from (A_0, B_1) to (B_1, A_2), (B_1, B_2) and (B_1, C_2); see binary trees on the right-hand side of Figures IV.1-IV.2.

Next we discuss details of implementing *reduce* and *broadcast* operations in the Jacobian operation. We again take Figure 2 as an example. To sum up the three values in (38), we consider a binary tree in Figure IV.3. Partitions (A_1, B_2) and (A_1, C_2) send the second and the third terms in (38) to (A_1, A_2), respectively. Then, (A_1, A_2) broadcasts the resulting

$$\frac{\partial z_u^{L,i}}{\partial z_t^{1,i}},\ t \in A_1,\ u = 1, \dots, n_L,\ i = 1, \dots, l$$

to nodes (A_0, A_1), (B_0, A_1), and (C_0, A_1) by being the root of another binary tree shown in Figure IV.4. A similar reduce operation is applied to (B_1, A_2), (B_1, B_2),

$$\begin{tikzcd}
& (A_1,A_2) & & & & (B_1,A_2) & \\
(A_1,B_2) \arrow[ur] & & (A_1,C_2) \arrow[ul] & & (B_1,B_2) \arrow[ur] & & (B_1,C_2) \arrow[ul]
\end{tikzcd}$$

Figure IV.3: An example to implement *reduce* operations for obtaining $\partial z_u^{2,i}/\partial z_t^{1,i}$, $\forall t \in \{1,\ldots,n_1\}$, $i \in \{1,\ldots,l\}$. The left tree corresponds to the calculation in (38).

$$\begin{tikzcd}
& (A_1,A_2) \arrow[dl] \arrow[dr] & & & & (B_1,A_2) \arrow[dl] \arrow[dr] & \\
(A_0,A_1) \arrow[d] & & (B_0,A_1) & & (A_0,B_1) \arrow[d] & & (B_0,B_1) \\
(C_0,A_1) & & & & (C_0,B_1) & &
\end{tikzcd}$$

Figure IV.4: An example to *broadcast* $\partial z_u^{2,i}/\partial z_t^{1,i}$ to partitions between layers 0 and 1.

and (B_1, C_2), and then the result is broadcasted from (B_1, A_2) to (A_0, B_1), (B_0, B_1), and (C_0, B_1); see binary trees on the right-hand side of Figures IV.3-IV.4.

V Cost for Solving (61)

In Section 5.3, we omit discussing the cost of solving (61) after the CG procedure. Here we provide details.

To begin we check the memory consumption. The extra space needed to store the linear system (62) is negligible. For the computational and the communication cost, we analyze the following steps in constructing (61).

1. For $-\nabla f(\boldsymbol{\theta}^k)^T \boldsymbol{d}^k$ and $-\nabla f(\boldsymbol{\theta}^k)^T \bar{\boldsymbol{d}}^k$, we sum up local inner products of sub-vectors (i.e., $-\boldsymbol{g}_p \boldsymbol{d}_p^k$ and $-\boldsymbol{g}_p \bar{\boldsymbol{d}}_p^k$ at the pth node) among all partitions.
2. For the 2 by 2 matrix in (62), we must calculate

$$\begin{aligned}
(\boldsymbol{d}^k)^T G^{S_k} \boldsymbol{d}^k &= \frac{1}{C}(\boldsymbol{d}^k)^T \boldsymbol{d}^k + \frac{1}{|S_k|}(\sum_{p=1}^{P} J_p^{S_k} \boldsymbol{d}_p^k)^T B^{S_k} (\sum_{p=1}^{P} J_p^{S_k} \boldsymbol{d}_p^k),\\
(\boldsymbol{d}^k)^T G^{S_k} \bar{\boldsymbol{d}}^k &= \frac{1}{C}(\boldsymbol{d}^k)^T \bar{\boldsymbol{d}}^k + \frac{1}{|S_k|}(\sum_{p=1}^{P} J_p^{S_k} \boldsymbol{d}_p^k)^T B^{S_k} (\sum_{p=1}^{P} J_p^{S_k} \bar{\boldsymbol{d}}_p^k),\\
(\bar{\boldsymbol{d}}^k)^T G^{S_k} \bar{\boldsymbol{d}}^k &= \frac{1}{C}(\bar{\boldsymbol{d}}^k)^T \bar{\boldsymbol{d}}^k + \frac{1}{|S_k|}(\sum_{p=1}^{P} J_p^{S_k} \bar{\boldsymbol{d}}_p^k)^T B^{S_k} (\sum_{p=1}^{P} J_p^{S_k} \bar{\boldsymbol{d}}_p^k),
\end{aligned} \tag{V.11}$$

where

$$J_p^{S_k}\boldsymbol{d}_p^k = \begin{bmatrix} \vdots \\ J_p^i \\ \vdots \end{bmatrix} \boldsymbol{d}_p^k, \quad J_p^{S_k}\bar{\boldsymbol{d}}_p^k = \begin{bmatrix} \vdots \\ J_p^i \\ \vdots \end{bmatrix} \bar{\boldsymbol{d}}_p^k, \quad i \in S_k.$$

From (V.11), $J_p^{S_k}\boldsymbol{d}_p^k$ and $J_p^{S_k}\bar{\boldsymbol{d}}_p^k$, $\forall p$ must be summed up. We reduce all these $\mathcal{O}(n_L)$ vectors to one node and obtain

$$\sum_{p=1}^{P} J_p^{S_k}\boldsymbol{d}_p^k \text{ and } \sum_{p=1}^{P} J_p^{S_k}\bar{\boldsymbol{d}}_p^k.$$

For $(\boldsymbol{d}^k)^T\boldsymbol{d}^k$, $(\boldsymbol{d}^k)^T\bar{\boldsymbol{d}}^k$, and $(\bar{\boldsymbol{d}}^k)^T\bar{\boldsymbol{d}}^k$, we also sum up local inner products in all partitions by a reduce operation. Then, the selected partition can compute the three values in (V.11) and broadcast them to all other partitions.

The computation at each node for $J_p^{S_k}\boldsymbol{d}_p^k$ and $J_p^{S_k}\bar{\boldsymbol{d}}_p^k$ is comparable to two matrix-vector products in the CG procedure. This cost is relatively cheap because the CG procedure often needs more matrix-vector products.

For the reduce and broadcast operations, by the analysis in Section 5.3, for the binary tree implementation, a rough cost estimation of the reduce operation is

$$\mathcal{O}(\alpha + 2 \times (\beta + \gamma) \times (|S_k| \times n_L) \times \lceil \log_2(\sum_{m=1}^{L} \frac{n_{m-1}}{|T_{m-1}|} \times \frac{n_m}{|T_m|})\rceil). \qquad \text{(V.12)}$$

Note that one reduce operation is enough because we can concatenate all local values as a vector. The broadcast operation is cheap because only three scalars in (V.11) are involved.

By comparing (V.12) with the communication cost for function/gradient evaluations in (74)-(75), the cost here is not significant. Note that here we need only one *reduce/broadcast* operation, but in each function/gradient evaluation, multiple operations are needed because of the forward/backward process.

VI Effectiveness of Using Levenberg-Marquardt Methods

We mentioned in Section 4.5 that the Levenberg-Marquardt method is often not applied together with line search. Here we conduct a preliminary investigation by considering the following two settings.

1. *diag* + *sync* 50%: the proposed method in Section 8.1.
2. *noLM* + *diag* + *sync* 50%: it is the same as *diag* + *sync* 50% except that the LM method is not considered.

The experimental results are shown in Figure VI.6. We can make the following observations.

1. For testing accuracy/AUC versus number of iterations, the setting without applying LM converges faster. The possible explanation is that for *noLM* + *diag* + *sync* 50%, by solving

$$G^{S_k}\boldsymbol{d} = -\nabla f(\boldsymbol{\theta}^k) \tag{VI.13}$$

   without the term $\lambda_k\mathcal{I}$, the direction leads to a better second-order approximation of the function value.

2. For testing accuracy/AUC versus training time, we observe the opposite result. The setting with the Levenberg-Marquardt method is faster for all problems except HIGGS1M. It is faster because of fewer CG steps in the CG procedure. A possible explanation is that with the term $\lambda_k\mathcal{I}$, the linear system becomes better conditioned and therefore can be solved by a smaller number of CG steps.

VII Effectiveness of Combining Two Directions by Solving (61)

In Section 4.3 we discussed a technique from Wang et al. [2015] by combining the direction $\boldsymbol{d}^k$ from the CG procedure and the direction $\boldsymbol{d}^{k-1}$ from the previous iteration. Here we preliminarily investigate the effectiveness of this technique.

We take the approach *diag* + *sync* 50% in Section 8.1 and compare the results with/without solving (61).

The results are shown in Figure VII.8. From the figures of test accuracy versus the number of iterations, we see that the method of combining two directions after the CG procedure effectively improves the convergence speed. Further, the figures of showing training time are almost the same as those of iterations. This result confirms our analysis in Section V that the extra cost at each iteration is not significant.

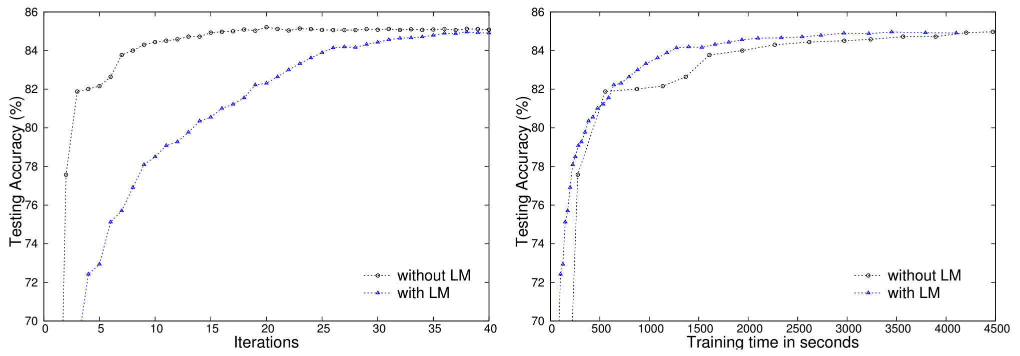
References

A. Agarwal, O. Chapelle, M. Dudik, and J. Langford. A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 15:1111–1133, 2014.

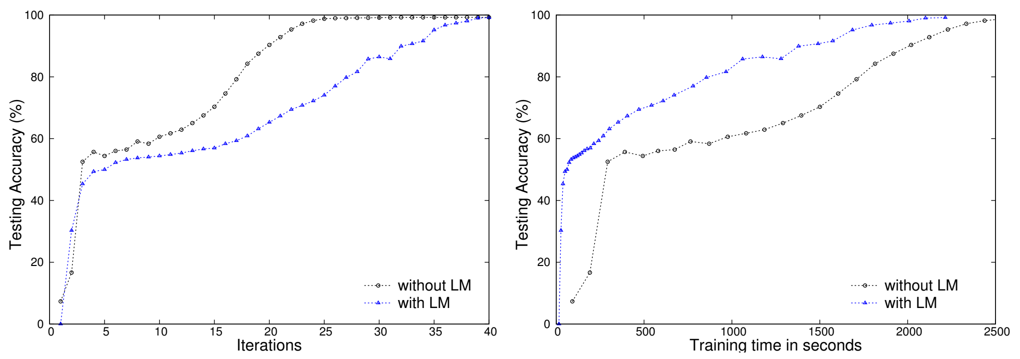
J. Martens and I. Sutskever. Training deep and recurrent networks with Hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pages 479–535. Springer, 2012.

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

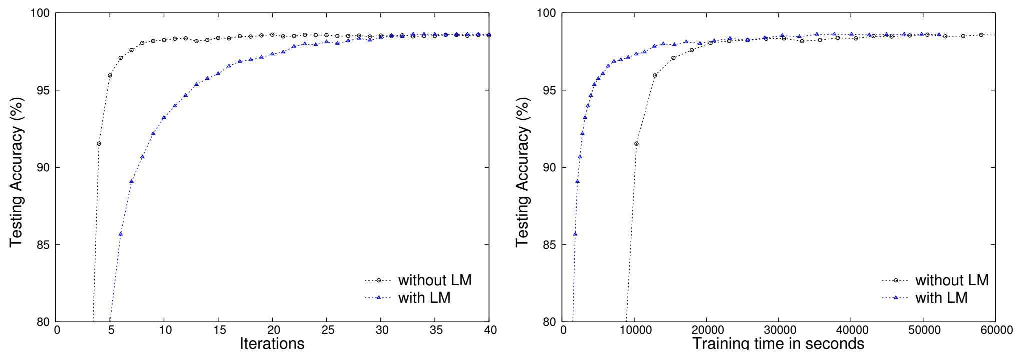
C.-C. Wang, C.-H. Huang, and C.-J. Lin. Subsampled Hessian Newton methods for supervised learning. *Neural Computation*, 27:1766–1795, 2015. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/sub_hessian/sample_hessian.pdf`.



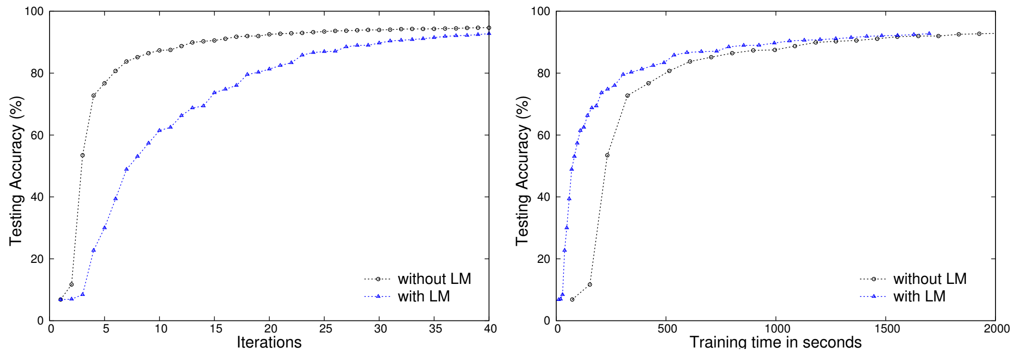
(a) SensIT Vehicle



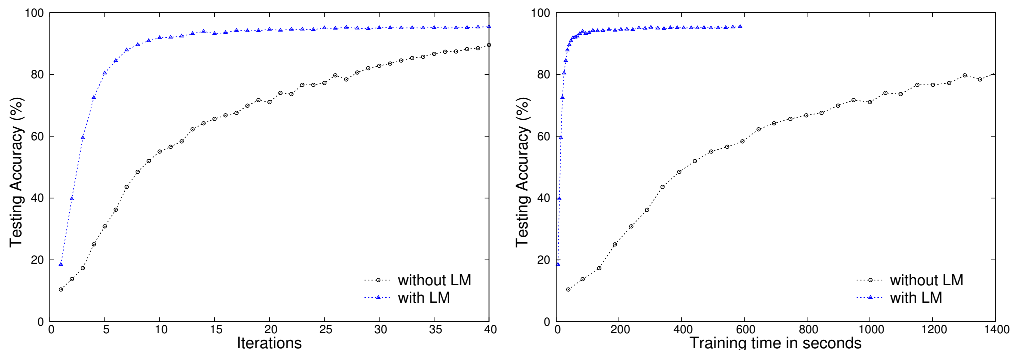
(b) Poker



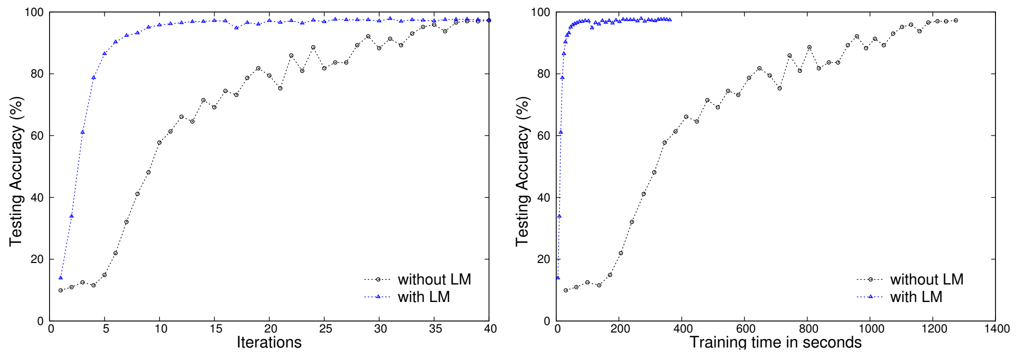
(c) MNIST



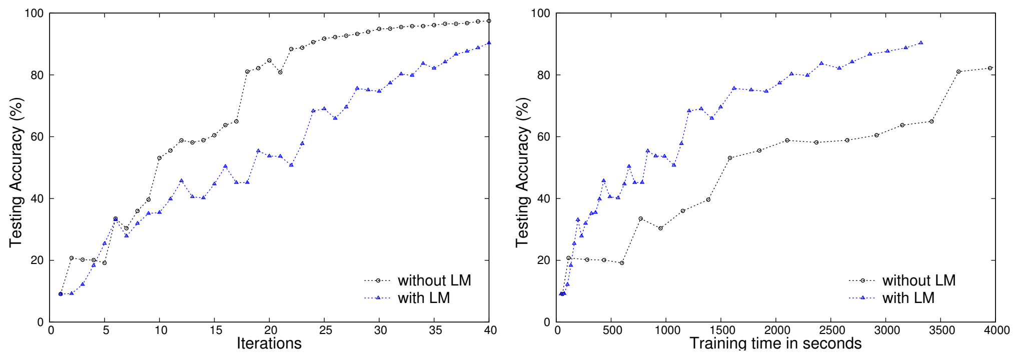
(d) Letter



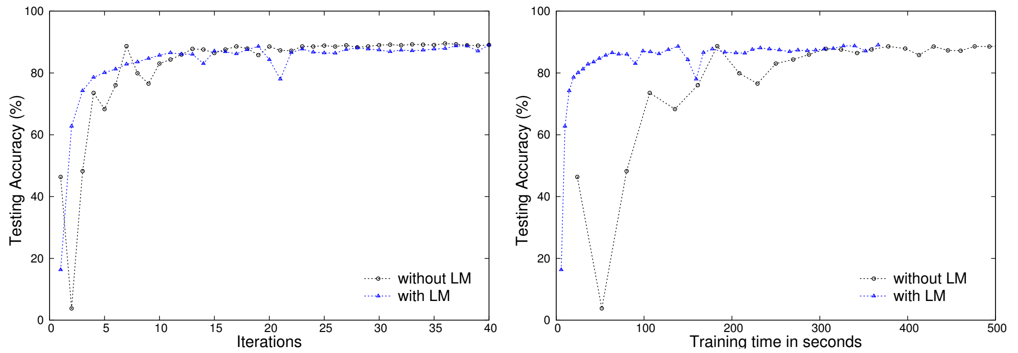
(a) USPS



(b) Pendigits

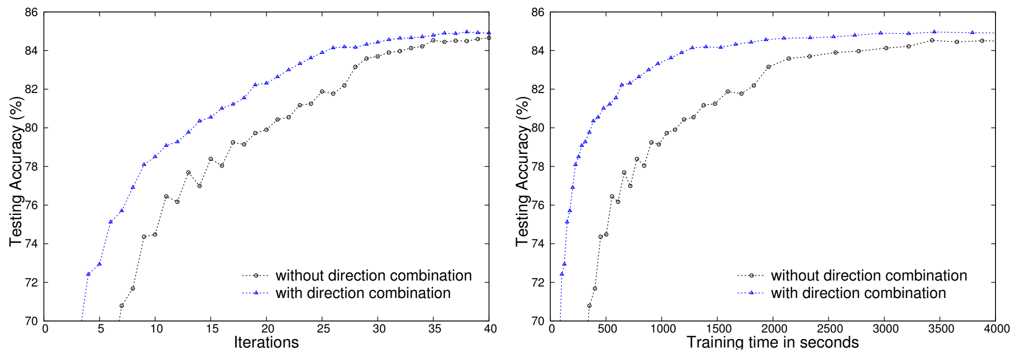


(c) Sensorless

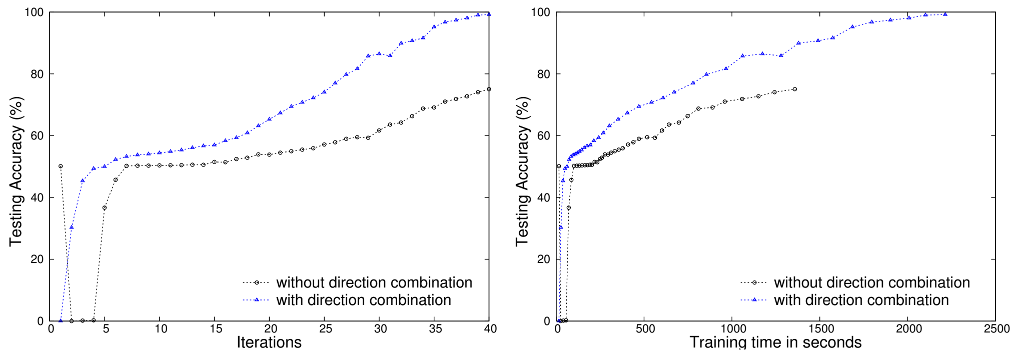


(d) Satimage

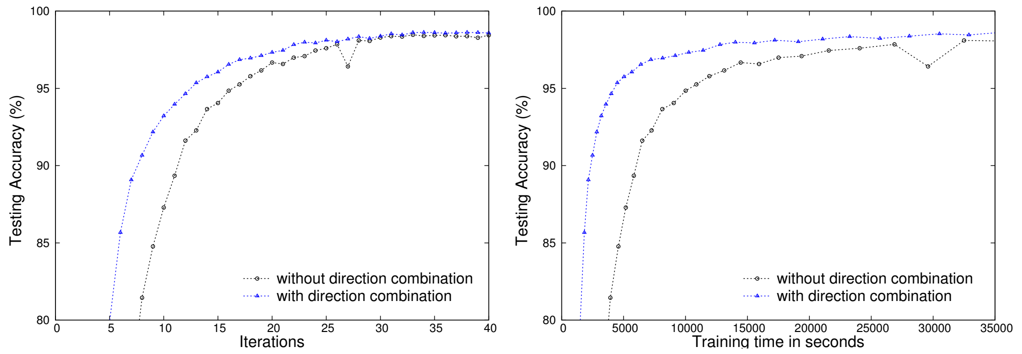
Figure VI.6: A comparison of using LM methods and without using LM methods. Left: testing accuracy versus number of iterations. Right: testing accuracy versus training time.



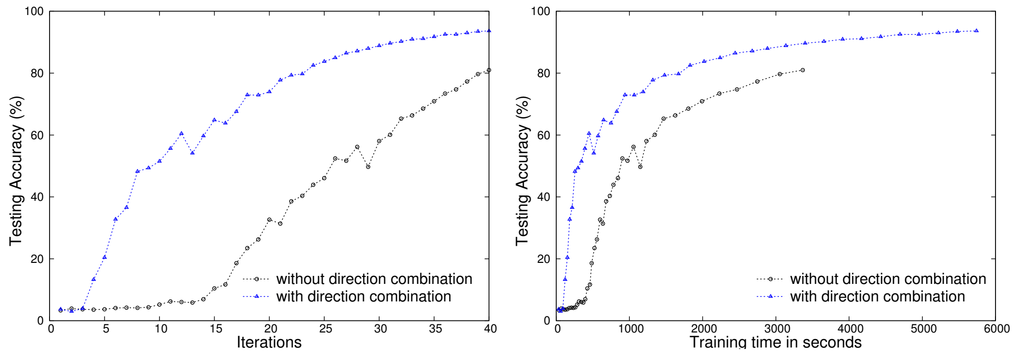
(a) SensIT Vehicle



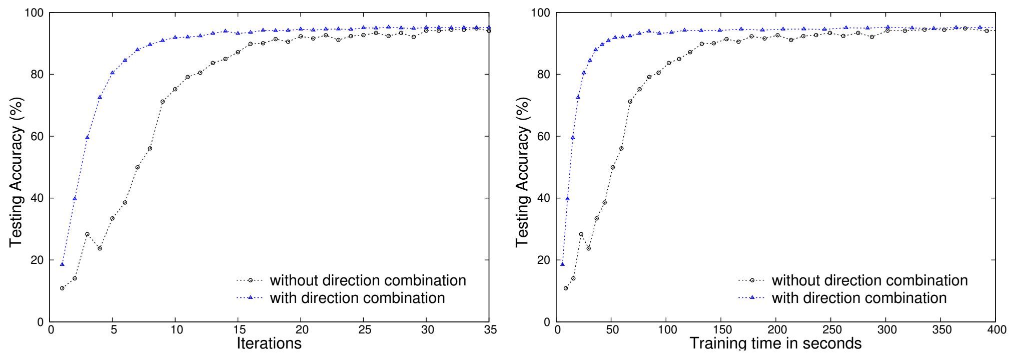
(b) poker



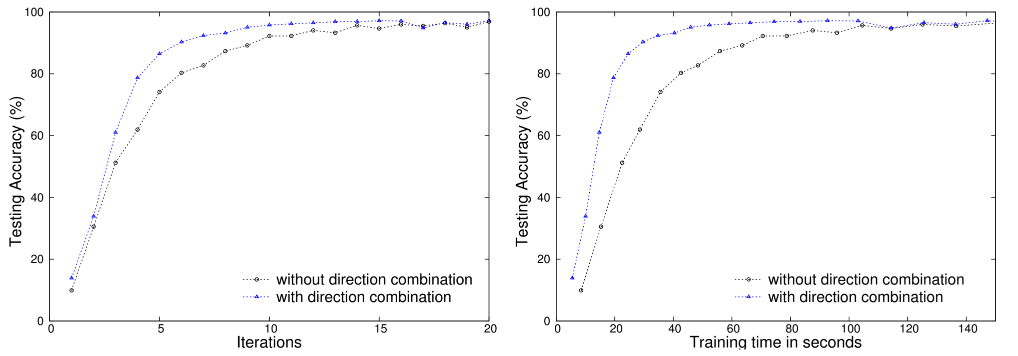
(c) MNIST



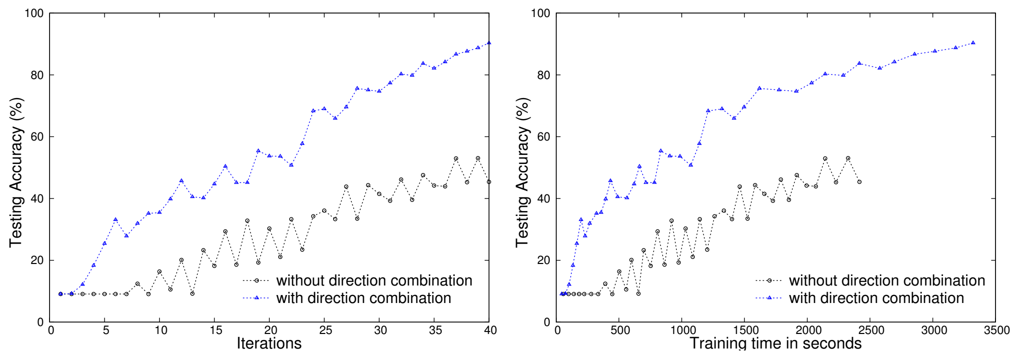
(d) Letter



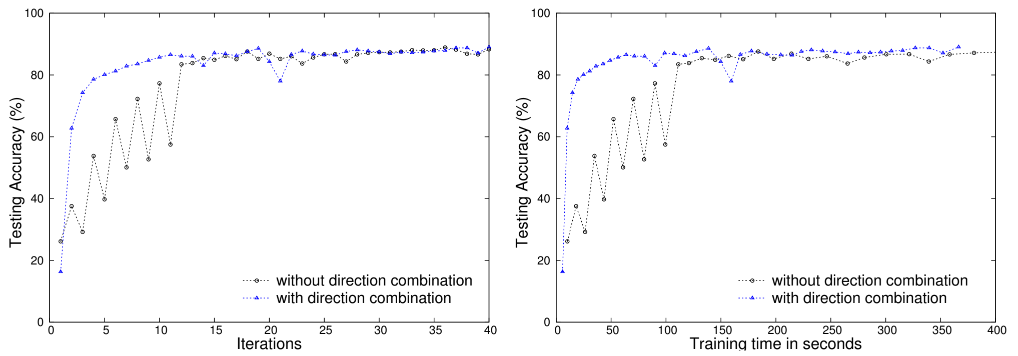
(a) USPS



(b) Pendigits



(c) Sensorless



(d) Satimage

Figure VII.8: A comparison between with/without implementing the combination of two directions by solving (61). Left: testing accuracy versus number of iterations. Right: testing accuracy versus training time.