On practical use of stagewise second-order backpropagation for multi-stage neural-network learning

Eiji Mizutani and Stuart Dreyfus

***Abstract*— We analyze the Hessian matrix $\mathbf{H}$ of the sum-squared-error measure for multilayer-perceptron (MLP) learning, showing the following intriguing results: At an early stage of learning, $\mathbf{H}$ is indefinite. The indefiniteness is related to the MLP structure, which also determines rank of $\mathbf{H}$ (per datum). Exploiting negative curvature leads to efficient learning algorithms. $\mathbf{H}$ can be much less ill-conditioned than the Gauss-Newton Hessian $\mathbf{J}^T\mathbf{J}$. All these new findings are obtained by our stagewise second-order backpropagation; the procedure exploits MLP's "layered symmetry" to evaluate $\mathbf{H}$ quickly, making exact Hessian evaluation feasible for fairly large practical problems. In fact, it works faster than rank-update methods that evaluate only $\mathbf{J}^T\mathbf{J}$. It also serves to appraise other existing learning algorithms that perform implicit Hessian-vector multiply.**

I. Introduction

Learning with multilayer-perceptron (MLP) neural networks can be viewed as a multiple N-stage (i.e., discrete-time) decision-making optimal control problem [1], [2]. Each stage s (or layer s) involves P_s-dimensional state vector $\mathbf{y}^s$ (of P_s node outputs) and n_s-dimensional control vector $\boldsymbol{\theta}^{s,s+1}$ (of weights including thresholds); hence, $n_s \equiv (1+P_s)P_{s+1}$. In standard MLP-learning, the objective function $E(.)$ is often the terminal cost only (at stage N) in the form of sum of squared residuals over D (training) data between terminal node outputs $\mathbf{y}^N(\boldsymbol{\theta})$ and target outputs $\mathbf{t}$; that is, $E(\mathbf{y}^N(\boldsymbol{\theta})) = \frac{1}{2}\mathbf{r}^T\mathbf{r}$, where $\mathbf{r} \equiv \mathbf{y}^N(\boldsymbol{\theta}) - \mathbf{t}$, the residual vector of length $m (\equiv P^N D)$, and $\boldsymbol{\theta}$ is the control (weight) vector of length $n (\equiv \sum_{s=1}^{N-1} n_s)$. We aim at optimizing $\boldsymbol{\theta}$ of MLP's weights while minimizing $E(.)$ usually with early stopping. Let $\mathbf{J}$ be $\left[\frac{\partial \mathbf{r}}{\partial \boldsymbol{\theta}}\right]$, the m-by-n Jacobian matrix of the residual vector; then, the n-dimensional gradient vector $\mathbf{g}$ of $E(.)$ and the n-by-n Hessian matrix $\mathbf{H}$ of $E(.)$ are given as $\mathbf{g} = \mathbf{J}^T\mathbf{r}$ and $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S}$, respectively, where $\mathbf{J}^T\mathbf{J}$ is called the *Gauss-Newton Hessian*, and $\mathbf{S} \equiv \sum_{i=1}^{m} r_i \left[\frac{\partial^2 r_i}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^T}\right]$ (e.g., see p.220 in [3]). The widely-employed first-order *backpropagation* (BP) first produces a vector $\mathbf{y}^s \equiv f^s(\mathbf{x}^s)$ of P_s after-node outputs with $f^s(.)$, a certain nonlinear node output function (e.g., tanh), at stage $s\,(>1)$, and then evaluates the gradient vector $\mathbf{g}$ efficiently by *backward pass* in a *stagewise* manner; this is a simplified Kelley-Bryson optimal-control gradient formula [1]. Similarly, $\mathbf{H}$ can be evaluated efficiently in a *stagewise* fashion [4], [5], as illustrated in Fig.1, where

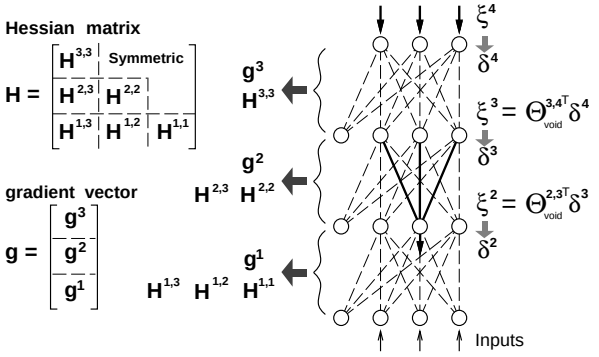


Fig. 1. Stagewise backpropagation procedures for two-hidden-layer ($N=4$) MLP-learning: By the backward pass, $\boldsymbol{\delta}^{s+1} \equiv \frac{\partial E}{\partial \mathbf{x}^{s+1}}$, the "before-node" sensitivities, are propagated back (using a weight matrix $\Theta_{\text{void}}^{s,s+1}$) to the previous stage as $\boldsymbol{\xi}^s \equiv \frac{\partial E}{\partial \mathbf{y}^s}$, the "after-node" sensitivities. During the backward process, the gradient vector $\mathbf{g}^s$ and the Hessian blocks ($\mathbf{H}^{s,s}$, $\mathbf{H}^{s,s+1}$, etc.) are evaluated at each stage s in a stagewise fashion. Each block $\mathbf{H}^{s,t}(1 \le s \le t \le 3)$ includes Hessian elements with respect to pairs of one parameter at stage s and another at stage t.

stagewise BP is applied to two-hidden-layer ($N=4$) MLP-learning for evaluating Hessian blocks in the *stagewise-partitioned* three-block-by-three-block $\mathbf{H}$.

Several non-stagewise algorithms are proposed for computing Hessian *"elements"* (e.g., pp.154-8 [6]; and pp.214-221 [7] only for $\mathbf{S}$); the superiority of *stagewise* procedures is self-evident [4], and arranging those elements into an arrow matrix is an important issue hitherto neglected. A haphazard organization makes it hard to deal with *sparsity*, which obviously arises in multiple-output problems (see Example in Sec.III-A). In what follows, we first outline our *stagewise second-order* BP, an extension of discrete-time optimal-control *stagewise Newton* of Dreyfus 1966 [8], [2], [5], showing how to exploit multi-stage structure in evaluating $\mathbf{H}$ as a block-partitioned (arrow) matrix with its arrowhead down to the right (p.84 [9]). We then show analyses of $\mathbf{H}$ that disclose several important findings on *rank*, *indefiniteness*, and *eigenvalues* for parameter optimization purposes.

II. Second-order stagewise backpropagation

We demonstrate our stagewise second-order BP for three-stage ($N \equiv 3$) (i.e., single-hidden-layer) MLP-learning with linear terminal outputs. This simple structure is commonly observed in separable (i.e., mixed linear & nonlinear) least squares problems; e.g., generalized linear discriminant functions [6], [10] with appropriately-defined *hidden-node* functions (e.g., radial basis and e^{-x} functions) at stage 2.

To illustrate the concrete structure of $\mathbf{H}$, we use a 1-2-2 MLP with a single input, two hidden-node functions, and two

Eiji Mizutani is with the Department of Industrial Management, National Taiwan University of Science and Technology, Taipei, 106, TAIWAN (email: eiji@mail.ntust.edu.tw).

Stuart E. Dreyfus is with the Department of Industrial Engineering and Operations Research, University of California at Berkeley, CA 94720, USA (email: dreyfus@ieor.berkeley.edu).

terminal outputs ($N=3; P_1=1; P_2=P_3=2; C_2\equiv 1+P_2=3$), in which $y_1^1=x_1^1$ and $\mathbf{y}^3=\mathbf{x}^3$ (due to the linear functions at stages 1 and 3) and x_j^2, the net input to the jth hidden-node function $y_j^2=f_j^2(x_j^2)$ (at stage 2), is given on each datum as $x_j^2=\mathbf{y}_+^{1\mathrm{T}}\boldsymbol{\theta}_{\cdot,j}^{1,2}$, where $\mathbf{y}_+^1$ is a C_1-vector ($C_1\equiv 1+P_1$), and subscript + denotes the inclusion of a constant output ($y_0^1=1$) at bias node 0. Likewise, $\mathbf{y}_+^2$ is a C_2-vector on each datum (whereas $\mathbf{y}^2$ is a P_2-vector). There are totally ten weights ($n=10$) in the 1-2-2 MLP to be optimized: $n_1=C_1P_2=4; n_2=C_2P_3=6; n=n_1+n_2=10$. To form $\mathbf{H}$ of size 10×10, three blocks ($\mathbf{H}^{2,2}$, $\mathbf{H}^{1,1}$, and $\mathbf{H}^{1,2}$) are evaluated:

Algorithm: Stagewise second-order BP routine (per datum).

(*Step* 0) After forward pass up to (terminal) stage 3, stagewise-BP begins evaluating $\xi_k^3\equiv\frac{\partial E(.)}{\partial y_k^3}=r_k$ (kth residual), which is $\delta_k^3\equiv\frac{\partial E(.)}{\partial x_k^3}$ at node k ($k=1,...,P_3$) due to $y_k^3=x_k^3$.

(*Step* 1) Obtain the first Hessian block $\mathbf{H}^{2,2}$, which consists of P_3 identical blocks; hence, need to store triangular half of only one block $\mathbf{y}_+^2\mathbf{y}_+^{2\mathrm{T}}$, the C_2-by-C_2 rank-one matrix:

$$\underbrace{\mathbf{H}^{2,2}}_{\substack{n_2\times n_2\\=6\times 6}}=\underbrace{\left[\frac{\partial \mathbf{x}^3}{\partial \boldsymbol{\theta}^{2,3}}\right]^T}_{\substack{n_2\times P_3\\=6\times 2}}\underbrace{\left[\frac{\partial \mathbf{x}^3}{\partial \boldsymbol{\theta}^{2,3}}\right]}_{\substack{P_3\times n_2\\=2\times 6}}=\left[\begin{array}{c|c}\begin{bmatrix}y_0^2\\y_1^2\\y_2^2\end{bmatrix} & \\ \hline & \begin{bmatrix}y_0^2\\y_1^2\\y_2^2\end{bmatrix}\end{array}\right]\left[\begin{array}{c|c}\begin{bmatrix}y_0^2 & y_1^2 & y_2^2\end{bmatrix} & \\ \hline & \begin{bmatrix}y_0^2 & y_1^2 & y_2^2\end{bmatrix}\end{array}\right]. \quad (1)$$

(*Step* 2) Compute P_2-by-n_2 $\mathbf{F}^{2,2}$ and P_2-by-P_2 *symmetric* $\mathbf{Z}^2$:

$$\underbrace{\mathbf{F}^{2,2}}_{\substack{P_2\times n_2\\=2\times 6}}=\underbrace{\left[\frac{\partial \mathbf{y}^2}{\partial \mathbf{x}^2}\right]^{\mathrm{T}}}_{\substack{P_2\times P_2\\=2\times 2}}\left\{\underbrace{\boldsymbol{\Theta}_{\text{void}}^{2,3\mathrm{T}}}_{\substack{P_2\times P_3\\=2\times 2}}\underbrace{\left[\frac{\partial \mathbf{x}^3}{\partial \boldsymbol{\theta}^{2,3}}\right]}_{\substack{P_3\times n_2\\=2\times 6}}+\underbrace{\left\langle\left[\frac{\partial \boldsymbol{\theta}_{\text{void}}^{2,3}}{\partial \boldsymbol{\theta}^{2,3}}\right],\boldsymbol{\delta}^3\right\rangle}_{P_2\times n_2=2\times 6}\right\} \quad (2)$$

$$=\left[\begin{array}{c|c}f_1^{2\prime}(.) & \\ \hline & f_2^{2\prime}(.)\end{array}\right]\left\{\underbrace{\begin{bmatrix}\theta_{1,1}^{2,3} & \theta_{1,2}^{2,3}\\ \theta_{2,1}^{2,3} & \theta_{2,2}^{2,3}\end{bmatrix}\left[\begin{array}{c|c}\mathbf{y}_+^{2\mathrm{T}} & \\ \hline & \mathbf{y}_+^{2\mathrm{T}}\end{array}\right]}_{2\times 6}+\underbrace{\left[\begin{array}{cc|cc|cc}0 & \delta_1^3 & 0 & 0 & \delta_2^3 & 0\\ 0 & 0 & \delta_1^3 & 0 & 0 & \delta_2^3\end{array}\right]}_{2\times 6}\right\}$$

$$\underbrace{\mathbf{Z}^2}_{\substack{P_2\times P_2\\=2\times 2}}=\underbrace{\left[\frac{\partial \mathbf{y}^2}{\partial \mathbf{x}^2}\right]^{\mathrm{T}}}_{\substack{P_2\times P_2\\=2\times 2}}\underbrace{\boldsymbol{\Theta}_{\text{void}}^{2,3\mathrm{T}}}_{\substack{P_2\times P_3\\=2\times 2}}\underbrace{\boldsymbol{\Theta}_{\text{void}}^{2,3}}_{\substack{P_3\times P_2\\=2\times 2}}\underbrace{\left[\frac{\partial \mathbf{y}^2}{\partial \mathbf{x}^2}\right]}_{\substack{P_2\times P_2\\=2\times 2}}+\underbrace{\left\langle\left[\frac{\partial^2 \mathbf{y}^2}{\partial \mathbf{x}^2\partial \mathbf{x}^2}\right],\boldsymbol{\xi}^2\right\rangle}_{\substack{P_2\times P_2\\=2\times 2}} \quad (3)$$

$$=\underbrace{\sum_{k=1}^{P_3}\left[\mathbf{n}_{\cdot,k}^{2,3}\mathbf{n}_{\cdot,k}^{2,3\mathrm{T}}\right]}_{\substack{P_2\times P_2\\=2\times 2}}+\left[\begin{array}{c|c}f_1^{2\prime\prime}(.)\xi_1^2 & \\ \hline & f_2^{2\prime\prime}(.)\xi_2^2\end{array}\right]$$

where $\mathbf{n}_{\cdot,k}^{2,3}$ in Eq.(3) is a P_2-dimensional vector with its jth element $n_{j,k}^{2,3}=f_j^{2\prime}(.)\theta_{j,k}^{2,3}$, and $\boldsymbol{\theta}^{2,3}$ of length $n_2=C_2P_3$ includes P_3 threshold parameters $\boldsymbol{\theta}_{0,\cdot}^{2,3}$, whereas $\boldsymbol{\theta}_{\text{void}}^{2,3}$ of length P_2P_3 excludes the thresholds; two vectors, $\boldsymbol{\theta}^{2,3}$ and $\boldsymbol{\theta}_{\text{void}}^{2,3}$, can be reshaped to two matrices, C_2-by-P_3 $\boldsymbol{\Theta}^{2,3}$ and P_2-by-P_3 $\boldsymbol{\Theta}_{\text{void}}^{2,3}$, respectively. The (i,j)-element of $\langle .,.\rangle$-part in Eqs.(2) and (3) can be obtained by (a) with $j=(1+P_2)(k-1)+l+1$, and (b) with $\xi_l^2\equiv\frac{\partial E(.)}{\partial y_l^2}$ below:

$$\begin{aligned}&\text{(a) }\left\langle\left[\frac{\partial \boldsymbol{\theta}_{\text{void}}^{2,3}}{\partial \boldsymbol{\theta}^{2,3}}\right],\boldsymbol{\delta}^3\right\rangle_{ij}\overset{\text{def}}{=}\sum_{k=1}^{P_3}\sum_{i=1}^{P_2}\sum_{l=0}^{P_2}\delta_k^3\left[\frac{\partial \theta_{i,k}^{2,3}}{\partial \theta_{l,k}^{2,3}}\right];\\ &\text{(b) }\left\langle\left[\frac{\partial^2 \mathbf{y}^2}{\partial \mathbf{x}^2\partial \mathbf{x}^2}\right],\boldsymbol{\xi}^2\right\rangle_{ij}\overset{\text{def}}{=}\sum_{l=1}^{P_2}\sum_{j=1}^{P_2}\sum_{i=0}^{P_2}\xi_l^2\left[\frac{\partial^2 y_l^2}{\partial x_i^2\partial x_j^2}\right].\end{aligned} \quad (4)$$

(*Step* 3) Obtain Hessian blocks $\mathbf{H}^{1,1}$ and $\mathbf{H}^{1,2}$ at stage 1:

$$\underbrace{\mathbf{H}^{1,1}}_{\substack{n_1\times n_1\\=4\times 4}}=\underbrace{\left[\frac{\partial \mathbf{x}^2}{\partial \boldsymbol{\theta}^{1,2}}\right]^{\mathrm{T}}}_{\substack{n_1\times P_2\\=4\times 2}}\underbrace{\mathbf{Z}^2}_{\substack{P_2\times P_2\\=2\times 2}}\underbrace{\left[\frac{\partial \mathbf{x}^2}{\partial \boldsymbol{\theta}^{1,2}}\right]}_{\substack{P_2\times n_1\\=2\times 4}}=\left[\begin{array}{c|c}\begin{bmatrix}y_0^1\\y_1^1\end{bmatrix} & \\ \hline & \begin{bmatrix}y_0^1\\y_1^1\end{bmatrix}\end{array}\right]\left[\begin{array}{c|c}\begin{bmatrix}z_{11}^2 & z_{12}^2\\ z_{21}^2 & z_{22}^2\end{bmatrix} & \\ \hline & \begin{bmatrix}y_0^1 & y_1^1\end{bmatrix}\end{array}\right]\left[\begin{array}{c|c}\begin{bmatrix}y_0^1 & y_1^1\end{bmatrix} & \\ \hline & \begin{bmatrix}y_0^1 & y_1^1\end{bmatrix}\end{array}\right]. \quad (5)$$

$$\underbrace{\mathbf{H}^{1,2}}_{\substack{n_1\times n_2\\=4\times 6}}=\underbrace{\left[\frac{\partial \mathbf{x}^2}{\partial \boldsymbol{\theta}^{1,2}}\right]^{\mathrm{T}}}_{\substack{n_1\times P_2\\=4\times 2}}\underbrace{\mathbf{F}^{2,2}}_{\substack{P_2\times n_2\\=2\times 6}}=\left[\begin{array}{c|c}\begin{bmatrix}y_0^1\\y_1^1\end{bmatrix} & \\ \hline & \begin{bmatrix}y_0^1\\y_1^1\end{bmatrix}\end{array}\right]\left[\begin{array}{c|c}\begin{bmatrix}F_{11}^{2,2}F_{12}^{2,2}F_{13}^{2,2}\end{bmatrix} & \begin{bmatrix}F_{14}^{2,2}F_{15}^{2,2}F_{16}^{2,2}\end{bmatrix}\\ \hline \begin{bmatrix}F_{21}^{2,2}F_{22}^{2,2}F_{23}^{2,2}\end{bmatrix} & \begin{bmatrix}F_{24}^{2,2}F_{25}^{2,2}F_{26}^{2,2}\end{bmatrix}\end{array}\right]. \quad (6)$$

In Eq.(5), z_{ij}^2 is the (i,j)-element of $\mathbf{Z}^2$. In Eq.(6), P_2-by-n_2 $\mathbf{F}^{2,2}$ in Eq.(2) is actually part of $\mathbf{H}^{1,2}$ due to $y_0^1=1$. □

From Eqs.(1), (5), and (6), we get the following *outer-product* form of $\mathbf{H}$ per datum (for 1-2-2 MLP-learning):

$$\mathbf{H}=\left[\begin{array}{c|c}\underbrace{\mathbf{H}^{2,2}}_{n_2\times n_2} & \text{Symmetric}\\ \hline \underbrace{\mathbf{H}^{1,2}}_{n_1\times n_2} & \underbrace{\mathbf{H}^{1,1}}_{n_1\times n_1}\end{array}\right]=\left[\begin{array}{c|c}\mathbf{y}_+^2\mathbf{y}_+^{2\mathrm{T}} & \\ \hline & \mathbf{y}_+^2\mathbf{y}_+^{2\mathrm{T}}\end{array}\right.\left.\begin{array}{c|c}\multicolumn{2}{c}{\text{Symmetric}}\\ \hline \mathbf{y}_+^1\ \mathbf{F}_{1,\cdot}^{2,2\mathrm{T}} & z_{11}^2\mathbf{y}_+^1\mathbf{y}_+^{1\mathrm{T}} \quad \text{Symmetric}\\ \mathbf{y}_+^1\ \mathbf{F}_{2,\cdot}^{2,2\mathrm{T}} & z_{21}^2\mathbf{y}_+^1\mathbf{y}_+^{1\mathrm{T}} \quad z_{22}^2\mathbf{y}_+^1\mathbf{y}_+^{1\mathrm{T}}\end{array}\right]. \quad (7)$$

The posed stagewise procedure is easy to implement in software, as shown in Table I, a stylized algorithmic format of Step 2: $\mathbf{F}^{2,2}$ is split into P_3 partitions, where the kth partition is denoted by $\mathbf{F}^{2,2}[k]$ of size P_2-by-$(1+P_2)$.

III. Analyses on the Hessian matrix $\mathbf{H}$

The efficient evaluation of $\mathbf{H}$ by stagewise BP allows us to analyze learning algorithms in various aspects that follow.

A. The cost for the Hessian evaluations

In the literature of nonlinear least squares, it is often said that the difference in evaluation cost between $\mathbf{H}$ and $\mathbf{J}^T\mathbf{J}$ is $O(n^2)$ (e.g., see p.392 in [11]). This is why one may use quasi-Newton on $\mathbf{S}$ and then add it to $\mathbf{J}^T\mathbf{J}$ (see NL2SOL, pp.229-233 [3], [12]). In MLP-learning, however, the claim does not apply because $\mathbf{H}$ can be evaluated by our stagewise procedure faster than such approximation on $\mathbf{S}$.

Theorem III-A: *The cost required for evaluating* $\mathbf{H}$ *and that for* $\mathbf{J}^T\mathbf{J}$ *are essentially the same in MLP-learning.*

Proof: In stagewise second-order BP for MLP-learning with N layers, the elements for $\mathbf{S}$ in the n-by-n $\mathbf{H}$ are computed in $O(n)\approx O(P_NP_{N-1}+...+P_3P_2)$ during the first-order backward pass; e.g., for $N=3$, see lines 20 (shallow two-nested loops) and 23 in Table I. All the other procedures cost $O(n^2)\approx O\left((P_NP_{N-1}+...+P_3P_2+P_2P_1)^2\right)$ common to calculating $\mathbf{J}^T\mathbf{J}$. Since the overall cost for evaluating the triangular half of $\mathbf{H}$ is $O(n^2)$, the additional cost in $O(n)$ due to $\mathbf{S}$ is negligibly small. □ *QED* □

Example: Stagewise algorithm can evaluate $\mathbf{H}$ faster than popular methods that get $\mathbf{J}^T\mathbf{J}$ with *sparsity* ignored [13] by rank updates $\mathbf{u}_i\mathbf{u}_i^T$, where $\mathbf{u}_i$, the ith row of $\mathbf{J}$, is obtainable by first-order backward pass at each terminal node per datum; e.g., see calcjejj.m (in MATLAB Neural-Network Toolbox). In the letter recognition benchmark problem (16,000 data) by a 16-26-26 MLP (1,144 weights) on a Pentium-4 3.4 GHz PC, such a method took 22 minutes for $\mathbf{J}^T\mathbf{J}$ alone, whereas stagewise BP took 41 seconds for either $\mathbf{H}$ or $\mathbf{J}^T\mathbf{J}$ (averaged over 10 trials). Note also that stagewise BP works with *any number of hidden layers* (see [2], [5] and references therein).

TABLE I

Algorithm: TWO ROUTINES FOR EVALUATING $\mathbf{Z}^2$ AND $\mathbf{F}^{2,2}$ MATRICES FOR THE STAGEWISE SECOND-ORDER BACKPROPAGATION.

• Second-order backward-pass routine:

1 **for** $k=1$ **to** P_3
2 **for** $j=1$ **to** P_2
3 $n_j \leftarrow f_j^{2'}(.)\theta_{j,k}^{2,3}$
4 $\beta \leftarrow n_j$
5 **for** $i=0,1,...,P_2$ (i=0 for a bias node)
6 $F_{j,i+1}^{2,2}[k] \leftarrow \beta y_i^2$
7 **end for** i.
8 **end for** j.
9 **for** $i=1$ **to** P_2
10 **for** $j=1$ **to** i (for lower-triangular half)
11 **if** $k=1$ **then** $z_{i,j}^2 \leftarrow n_i n_j$
12 **otherwise** $z_{i,j}^2 \leftarrow z_{i,j}^2 + n_i n_j$
13 **end for** j.
14 **end for** i.
15 **end for** k. □

• First-order backward-pass routine:

16 **for** $j=1$ **to** P_2
17 $\xi \leftarrow 0$
18 **for** $k=1$ **to** P_3
19 $\xi \leftarrow \xi + \theta_{j,k}^{2,3}\delta_k^3$
20 $F_{j,j+1}^{2,2}[k] \leftarrow F_{j,j+1}^{2,2}[k] + f_j^{2'}(.)\delta_k^3$
21 **end for** k.
22 $\delta_j^2 \leftarrow f_j^{2'}(.)\xi$
23 $z_{j,j}^2 \leftarrow z_{j,j}^2 + f_j^{2''}(.)\xi$
24 **end for** j. □

Note: When the terminal-node function is *non-linear*, line 4 should be $\beta = n_j z_k^3$, where $z_k^3 \equiv \left[\left\{f_k^{3'}(.)\right\}^2 + f_k^{3''}(.)r_k\right]$ evaluated at terminal layer 3 together with residuals r_k.

B. Sparse matrices can be pulled out of $\mathbf{S}$

In single-hidden-layer MLPs, the m-by-n residual Jacobian matrix of $\mathbf{r}$ is given by $\mathbf{J}=[\widetilde{\mathbf{A}}|\widetilde{\mathbf{B}}]$ with $\widetilde{\mathbf{A}} \equiv \left[\frac{\partial \mathbf{r}}{\partial \boldsymbol{\theta}^{2,3}}\right]$ and $\widetilde{\mathbf{B}} \equiv \left[\frac{\partial \mathbf{r}}{\partial \boldsymbol{\theta}^{1,2}}\right]$. Accordingly, $\mathbf{H}=\mathbf{J}^T\mathbf{J}+\mathbf{S}$ can be expressed as

$$\underbrace{\mathbf{H}}_{n\times n} = \begin{bmatrix} \underbrace{\widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}}}_{n_2\times n_2} & \underbrace{\widetilde{\mathbf{A}}^T\widetilde{\mathbf{B}}}_{n_2\times n_1} \\ \underbrace{\widetilde{\mathbf{B}}^T\widetilde{\mathbf{A}}}_{n_1\times n_2} & \underbrace{\widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}}}_{n_1\times n_1} \end{bmatrix} + \begin{bmatrix} & \underbrace{\boldsymbol{\Gamma}^{1,2^T}}_{n_2\times n_1} \\ \underbrace{\boldsymbol{\Gamma}^{1,2}}_{n_1\times n_2} & \end{bmatrix} + \begin{bmatrix} & \\ & \underbrace{\mathbf{V}^{1,1}}_{n_1\times n_1} \end{bmatrix}, \tag{8}$$

where $\mathbf{S}$ is separated into two types of blocks $\boldsymbol{\Gamma}^{s,t}$ and $\mathbf{V}^{s,s}$; in general for $1\le s<t\le N-1$, these blocks are defined as

$$\underbrace{\boldsymbol{\Gamma}^{s,t}}_{n_s\times n_t} = \underbrace{\left[\frac{\partial \mathbf{y}^t}{\partial \boldsymbol{\theta}^{s,s+1}}\right]^{\mathrm{T}}}_{n_s\times P_t} \underbrace{\left\langle \left[\frac{\partial \boldsymbol{\theta}_{(\text{void})}^{t,t+1}}{\partial \boldsymbol{\theta}^{t,t+1}}\right], \boldsymbol{\delta}^{t+1} \right\rangle}_{P_t\times n_t}; \tag{9}$$

$$\underbrace{\mathbf{V}^{s,s}}_{n_s\times n_s} = \underbrace{\left[\frac{\partial \mathbf{x}^{s+1}}{\partial \boldsymbol{\theta}^{s,s+1}}\right]^{\mathrm{T}}}_{n_s\times P_{s+1}} \underbrace{\left\langle \left[\frac{\partial^2 \mathbf{y}^{s+1}}{\partial \mathbf{x}^{s+1}\mathbf{x}^{s+1}}\right], \boldsymbol{\xi}^{s+1} \right\rangle}_{P_{s+1}\times P_{s+1}} \underbrace{\left[\frac{\partial \mathbf{x}^{s+1}}{\partial \boldsymbol{\theta}^{s,s+1}}\right]}_{P_{s+1}\times n_s}. \tag{10}$$

Here, $\langle .,. \rangle$-terms are obtainable from Eq.(4), where (b) gives a $P_{s+1}\times P_{s+1}$ *diagonal* matrix [e.g., see the last term in Eq.(3)], and (a) yields a P_t-by-n_t *sparse* matrix, whose column space has P_{t+1} partitions: Each partition has $(1+P_t)$ columns in the form of $[\mathbf{0}\,|\,\delta_k^{t+1}\mathbf{I}\,]$, a P_t-vector of *zeros* (in the first column) plus the P_t-by-P_t identity matrix $\mathbf{I}$ multiplied by scalar δ_k^{t+1}; e.g., see in Eq.(2) the last "sparse" 2-by-6 matrix that has two such partitions. As a result, each $\mathbf{V}^{s,s}$ is a (sparse) *block-diagonal* matrix that has P_{s+1} square blocks of size $(1+P_s)$-by-$(1+P_s)$, and $\boldsymbol{\Gamma}^{s,t}$ is a *sparse* matrix of *only first derivatives* (hence, easy to evaluate). See Example in Sec.III-D.

More generally, in two-hidden-layer ($N=4$) MLP-learning, $\mathbf{S}$ consists of totally nine sparse blocks below:

$$\mathbf{S} = \begin{bmatrix} & \boldsymbol{\Gamma}^{2,3^T} & \boldsymbol{\Gamma}^{1,3^T} \\ \underbrace{\boldsymbol{\Gamma}^{2,3}}_{n_2\times n_3} & & \boldsymbol{\Gamma}^{1,2^T} \\ \underbrace{\boldsymbol{\Gamma}^{1,3}}_{n_1\times n_3} & \underbrace{\boldsymbol{\Gamma}^{1,2}}_{n_1\times n_2} & \end{bmatrix} + \begin{bmatrix} \underbrace{\mathbf{V}^{3,3}}_{n_3\times n_3} & & \\ & \underbrace{\mathbf{V}^{2,2}}_{n_2\times n_2} & \\ & & \underbrace{\mathbf{V}^{1,1}}_{n_1\times n_1} \end{bmatrix}. \tag{11}$$

If terminal node functions are *linear*, then $\mathbf{V}^{3,3}=\mathbf{0}$.

C. The rank analysis on Hessian blocks

The sparsity structures of $\boldsymbol{\Gamma}^{s,t}$ and $\mathbf{V}^{s,s}$ in $\mathbf{S}$ play an important role in *determining the rank* of Hessian blocks $\mathbf{H}^{s,t}$; Table II(left) summarizes "possible maximum" rank of each Hessian block when $N=4$, where we used a well-known rank inequality between two matrices $\mathbf{A}$ (of size $a\times b$) and $\mathbf{B}$ ($b\times c$): $\mathsf{rank}(\mathbf{AB}) \le \min\{\mathsf{rank}(\mathbf{A}), \mathsf{rank}(\mathbf{B})\} \le \min\{a,b,c\}$. In the single-output case (i.e., $P_4=1$), the *Gauss-Newton Hessian* $\mathbf{J}^T\mathbf{J}$ becomes just rank-one per datum; see the second column in Table II(left), whereas the full Hessian $\mathbf{H}$ has a greater rank, which is greatly affected by $\boldsymbol{\Gamma}^{s,t}$ and $\mathbf{V}^{s,s}$. During MLP-learning, when all the hidden-node sigmoidal tanh-functions are driven to their limits: i.e., *saturation* ($+1$ or -1), all Hessian blocks became virtually zeros except $\mathbf{H}^{3,3}$.

With a P_1-P_2-P_3 single-hidden-layer MLP (with *linear* terminal outputs), $\mathbf{H}^{2,2}$ [see Eq.(1)] is block-diagonal with P_3 diagonal blocks [$P_3=2$ in Eq.(7)]; each block is identical to $\mathbf{y}_+^2\mathbf{y}_+^{2\,\mathrm{T}}$ of rank 1; hence, $\mathbf{H}^{2,2}$ is a matrix of rank P_3 per datum. Over all D data, D rank-one updates lead to C_2-by-C_2 $\mathbf{A}^T\mathbf{A}$, one diagonal block of $\mathbf{H}^{2,2}=\widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}}$ in Eq.(8). In Eq.(5), $\mathbf{H}^{1,1}$ is a P_2-by-P_2-partitioned matrix of rank P_2 per datum. Each partitioned block is a *symmetric* rank-one matrix $\mathbf{y}_+^1\mathbf{y}_+^{1\,\mathrm{T}}$ (of size C_1-by-C_1) multiplied by a scalar z_{ij}^2. Notice here that even "off-diagonal" blocks in $\mathbf{H}^{1,1}$ are also *symmetric*; so, need to evaluate only the triangular half. In Eq.(6), $\mathbf{H}^{1,2}$ is produced by only P_2 outer product operations; hence, a matrix of rank P_2.

With a 1-2-2 MLP, each Hessian block in Eq.(7) is of rank 2 (at most) per datum; so, $\mathsf{rank}(\mathbf{J}^T\mathbf{J}) \le 2$; $\mathsf{rank}(\mathbf{S}) \le 4$; $\mathsf{rank}(\mathbf{H}) \le 6$ per datum. Table II(right) shows another example on how the rank of $\mathbf{H}=\mathbf{J}^T\mathbf{J}+\mathbf{S}$ changes differently depending on the number of data processed in two distinct models: 1-2-1 MLP and 2-1-2 MLP. In the single-output case with the 1-2-1 MLP ($n=7$; $P_3=1; P_2=2$), $\mathsf{rank}(\mathbf{H}) \le 5$ per datum, although $\mathsf{rank}(\mathbf{J}^T\mathbf{J})=1$. This implies that only with two data (accumulated), $\mathbf{H}$ can be *full rank* (i.e., rank 7), while $\mathbf{J}^T\mathbf{J}$ is of rank 2; hence, *rank deficient* (so is the *Fisher information* matrix for natural-gradient learning). However, $\mathbf{H}$ may be *indefinite*, which we shall discuss in Sec. IV.

TABLE II

THE LEFT TABLE SHOWS THE (POSSIBLE MAXIMUM) RANK *per datum* OF SIX HESSIAN BLOCKS IN THE STANDARD TWO-HIDDEN-LAYER MLP-LEARNING (I.E., $N=4$) WITH MULTIPLE OUTPUTS ($P_4 > 1$). THE RIGHT TABLE DESCRIBES HOW THE RANK OF THREE 7-BY-7 MATRICES ($\mathbf{J}^T\mathbf{J}$, $\mathbf{S}$, $\mathbf{H}$) CHANGES AS DATA ARE ACCUMULATED (IN BATCH) IN TWO SINGLE-HIDDEN-LAYER MODELS: 2-1-2 MLP AND 1-2-1 MLP.

	$\mathbf{J}^T\mathbf{J}^{s,t}$	$\mathbf{\Gamma}^{s,t}$	$\mathbf{V}^{s,s}$	$\mathbf{S}^{s,t}-\mathbf{\Gamma}^{s,t}$	$\mathbf{S}^{s,s}-\mathbf{V}^{s,s}$	rank$(\mathbf{H}^{s,t})$
$\mathbf{H}^{3,3}$	P_4	–	P_4	–	0	P_4
$\mathbf{H}^{2,2}$	$\min\{P_4, P_3\}$	–	P_3	–	$\min\{P_4, P_3\}$	P_3
$\mathbf{H}^{2,3}$	$\min\{P_4, P_3\}$	P_3	–	$\min\{P_4, P_3\}$	–	P_3
$\mathbf{H}^{1,1}$	$\min\{P_4,P_3,P_2\}$	–	P_2	–	$\min\{P_3, P_2\}$	P_2
$\mathbf{H}^{1,2}$	$\min\{P_4,P_3,P_2\}$	P_2	–	$\min\{P_3, P_2\}$	–	P_2
$\mathbf{H}^{1,3}$	$\min\{P_4,P_3,P_2\}$	$\min\{P_3, P_2\}$	–	$\min\{P_4,P_3,P_2\}$	–	$\min\{P_3, P_2\}$

# of data	2-1-2 MLP $\mathbf{J}^T\mathbf{J}$	2-1-2 MLP $\mathbf{S}$	2-1-2 MLP $\mathbf{H}$	1-2-1 MLP $\mathbf{J}^T\mathbf{J}$	1-2-1 MLP $\mathbf{S}$	1-2-1 MLP $\mathbf{H}$
1	2	2	4	1	4	5
2	4	4	6	2	6	7
3	5	5	7	3	6	7
4	6	5	7	4	6	7
5	7	5	7	5	6	7
6	7	5	7	6	6	7
7	7	5	7	7	6	7

D. Indefinite $\mathbf{S}$ *and saddle points in the weight space*

Recall in Eq.(11) that if the terminal node functions at stage 4 are *linear* [i.e., $y_k^4 = f_k^4(x_k^4) = x_k^4$], then $\mathbf{V}^{3,3}$ disappears due to $f_k^{4''}(x_k^4)=0$, so does $\mathbf{V}^{2,2}$ in Eq.(8); this is true in any *separable nonlinear least squares* learning.

Lemma 1: *If the terminal node functions at stage N are linear in N-layer MLP-learning, then $\mathbf{S}$ is always indefinite.*
Proof: $\mathbf{S}^{N-1,\,N-1}=\mathbf{0}$. Owing to the Cauchy interlace theorem (pp.202-205[14]), there always exist sign-different eigenvalues; hence, $\mathbf{S}$ is indefinite with any number of data. □ *QED* □

Lemma 2: *Given a 1-H-1 MLP that has a linear terminal output with no thresholds ($2H$ weights), the $2H$-by-$2H$ matrix $\mathbf{S}$ is of "full rank" but "indefinite," and easy to determine the eigen-spectrum regardless of the number of data.*
Proof: $\mathbf{S}$ can be stagewise-partitioned as in Eq.(8) above

$$\mathbf{S}=\left[\begin{array}{c|c}\mathbf{V}^{2,2} & \text{Symmetric}\\ \hline \mathbf{\Gamma}^{1,2} & \mathbf{V}^{1,1}\end{array}\right];\ \mathbf{\Gamma}^{1,2}=\begin{bmatrix}\beta_1 & & \\ & \beta_2 & \\ & & \ddots\end{bmatrix};\mathbf{V}^{1,1}=\begin{bmatrix}\alpha_1 & & \\ & \alpha_2 & \\ & & \ddots\end{bmatrix}. \quad (12)$$

Here, $\mathbf{V}^{2,2}$,$\mathbf{\Gamma}^{1,2}$, and$\mathbf{V}^{1,1}$ are H-by-H; $\mathbf{V}^{2,2}=\mathbf{0}$; and both $\mathbf{\Gamma}^{1,2}$ and $\mathbf{V}^{1,1}$ are *diagonal* due to the node connections between layers in standard MLPs. The characteristic polynomial of $\mathbf{S}$ is given by $\det(\lambda\mathbf{I}-\mathbf{S})=g_1(\lambda)g_2(\lambda)...g_i(\lambda)...g_H(\lambda)$, where each $g_i(\lambda)$ for $i=1,...,H$ is quadratic of the form $\lambda^2-\alpha_i\lambda-\beta_i^2$ with α_i (the ith diagonal element of $\mathbf{V}^{1,1}$) and β_i (the ith diagonal element of $\mathbf{\Gamma}^{1,2}$). Clearly, each $g_i(\lambda)=0$ yields a pair of positive & negative eigenvalues: $\lambda=\frac{1}{2}(\alpha_i\pm\sqrt{\alpha_i^2+4\beta_i^2})$. Hence, $\mathbf{S}$ has H such pairs as long as $\beta_i\neq 0$ (usually true; see Example below) with any number of data. □ *QED* □

Example: When $H=2$ with a 1-2-1 MLP ($2H=4$ weights), the entire Hessian $(\mathbf{H}=)\,\mathbf{J}^T\mathbf{J}+\mathbf{S}$ per datum is given by

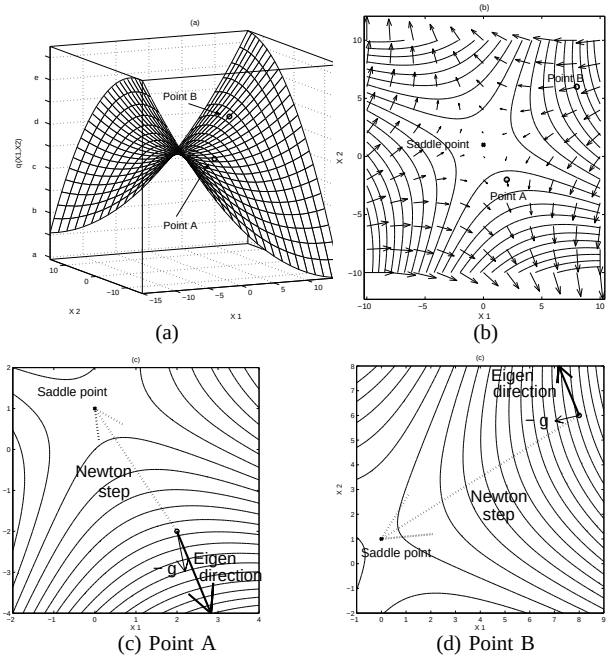
$$\begin{bmatrix}\{y_1\}^2 & y_1y_2 & xpf_1'(.)y_1 & xqf_2'(.)y_1\\ y_1y_2 & \{y_2\}^2 & xpf_1'(.)y_2 & xqf_2'(.)y_2\\ xpf_1'(.)y_1 & xpf_1'(.)y_2 & \left\{xpf_1'(.)\right\}^2 & \{x\}^2pqf_1'(.)f_2'(.)\\ xqf_2'(.)y_1 & xqf_2'(.)y_2 & \{x\}^2pqf_1'(.)f_2'(.) & \left\{xqf_2'(.)\right\}^2\end{bmatrix}+\begin{bmatrix}0 & 0 & xf_1'(.)r & 0\\ 0 & 0 & 0 & xf_2'(.)r\\ xf_1'(.)r & 0 & \{x\}^2pf_1''(.)r & 0\\ 0 & xf_2'(.)r & 0 & \{x\}^2qf_2''(.)r\end{bmatrix},$$

where p and q are $\theta_{1,1}^{2,3}$ and $\theta_{2,1}^{2,3}$, respectively; r a residual; and $y_i=f_i(x)(i=1,2)$ two hidden outputs. On a single datum, rank$(\mathbf{J}^T\mathbf{J})=1$ but rank$(\mathbf{S})=4$, "full" rank (but indefinite); so is $\mathbf{H}$ provided that $\beta_i\equiv xf_i'(.)r\neq 0$ (usually holds) in $\mathbf{S}$.

Lemma 3: *In the weight space of 1-H-1 MLP in Lemma 2, the points having any zero weight(s) are a "saddle" point.*
Proof by *interlacing* as in Lemma 1 (see Lemma 8 in [15]). The claim is obvious when p (or q) is zero in Example above.

IV. ON EXPLOITING NEGATIVE CURVATURE

In neural networks nonlinear least squares learning, the initial weights are often randomly generated uniformly in a small range; thus, the residuals tend to be very large. We show experimentally that $\mathbf{H}$ is *indefinite* (because $\mathbf{S}$ becomes dominant in $\mathbf{H}$ when residuals are large and/or highly nonlinear; see Sec.III-D), and $\mathbf{H}$ can be much "less" *ill-conditioned* than $\mathbf{J}^T\mathbf{J}$ unlike the report in [16]. These were observed in the former half of the learning phase. When $\mathbf{H}$ is indefinite, the Newton step always moves the current point to the nearest *saddle* point. In order to move away efficiently from the saddle point, it is better to exploit *negative curvature*. The figures (a) to (d) below illustrate a simple situation, where Newton, steepest descent (denoted by $-\mathbf{g}$), and eigen-directions are compared at two points: Points A and B. Clearly, from figures (c) and (d), the eigen-direction (thick solid arrow) paves efficiently the descent path for avoiding locking onto the nearest saddle point.

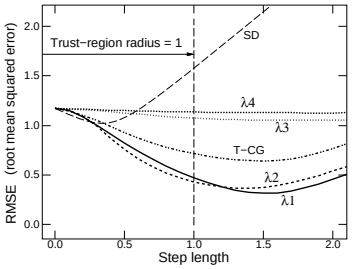


(a) (b) (c) Point A (d) Point B

A. Directions of Negative Curvature

Consider a 1-2-1 MLP (seven weights; $n=7$): When only one block of two data are processed, $\mathbf{H}$ becomes of "full rank" [see Table II(right)] but *indefinite*. The following figure shows how much six different descent directions can reduce

RMSE (root mean squared error) when four negative eigenvalues existed: $\lambda_1 = -8.09$ (the most negative eigenvalue); $\lambda_2 = -6.85$; $\lambda_3 = -0.33$; and $\lambda_4 = -0.18$.



Those six descent directions included four directions of negative curvature (λ_1 through λ_4) plus the steepest-descent (SD) and a truncated conjugate-gradient (T-CG) directions. Here, T-CG uses only Hessian-vector multiply, which is suitable for a very large-scale problem, but picks up "arbitrary" negative curvature in Krylov subspaces (see pp.202-218 in [17], [18]).

If a step is taken in the eigen-direction associated with the least negative eigenvalue λ_4, then it is not very efficient (even less effective than the small steepest-descent (SD) step in this example). The fundamental concept is to choose a step using the direction of negative curvature associated with strongly negative eigenvalues if any (e.g., λ_1 and λ_2), so as to move away effectively from non-minimizing stationary *saddle* points. Here, a greedy weight update is not recommendable because it may cause *hidden-node saturations* (on the next data block), resulting in *rank-deficient* $\mathbf{H}$ (see Sec.III-C); therefore, we consider steps subject to *trust-region regularization*, proceeding only within a trust-region radius. In classical Levenberg-Marquardt methods (e.g., with $\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}$), only scalar $\mu(\geq 0)$ is controlled (e.g., see [19]), but modern trust-region methods must control the trust-region radius; for detailed mechanisms, refer to [17].

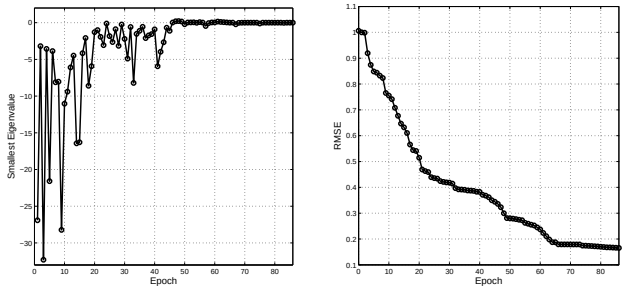
In all the simulations that follow, we employ a second-order trust-region method [17] that *exploits negative curvature* (unlike the methods in [16], [19]), and MLP's weights are randomly-generated per trial uniformly in $[-0.2, +0.2]$, a range small enough to avoid initial hidden-node saturations.

B. Classification Problems with ON/Off (± 1) Binary Data

We first considered the two-class parity classification problems. In the two-bit XOR problem ($m = D = 4$ data) with a 2-2-1 MLP ($n = 9$ weights); from Table II, $\mathsf{rank}(\mathbf{J}^T\mathbf{J}) = 4$ and $\mathsf{rank}(\mathbf{H}) = 9$ (in batch mode). Our trust-region method took only about three steps (averaged over 400 trials) for solution in the direction of negative curvature, as shown next:

Required epoch	1	2	3	4	≥ 5	Avg.
tanh terminal node	21	65	158	106	50	3.36
linear terminal node	14	60	184	81	61	3.38

We next show a result on the seven-bit parity ($m = D = 128$ data). We applied *weight-sharing* [15] to a 7-4-1 MLP in such a way that the seven-input nodes to each of four hidden nodes are merged into one weight; hence, a 1-4-1 MLP ($n = 13$ weights), to which the input is the sum of seven-bit signals. In this setting, 128 data reduced to 8 distinct patterns; so, $\mathsf{rank}(\mathbf{J}^T\mathbf{J}) = 8$ and $\mathsf{rank}(\mathbf{H}) = 13$ (in batch mode), and the



(a) Smallest eigenvalue (b) RMSE

Fig. 2. The seven-bit parity classification problem (128 data) attacked by a "weight-sharing" 1-4-1 MLP (13 weights): (a) The smallest eigenvalue of $\mathbf{H}$, and (b) RMSE (root-mean-square-error) learning curve. The eigenvalues were obtained by using dsyevx in CLAPACK (www.netlib.org).

*condition number** of $\mathbf{H}$ was $O(10^2)$ or even smaller at the beginning. Fig. 2 shows a typical learning behavior; remarkably, at the early phase of learning, the smallest eigenvalue tended to be strongly negative. Note in some cases with a first-order gradient method that MLP-learning may be stuck at RMSE $= 1$, making all the weights eventually converge to zero (i.e, the *origin* of the weight space). Such a situation can be circumvented by using the negative curvature, which indicates that the origin corresponds to the saddle point. Parity is a small (intrinsically *zero*) residual problem.

C. A Nonlinear Regression Problem

Fig.3 shows an example of curve-fitting ($D = 105$ data) by a 1-3-1 MLP ($n = 10$). Even with 105 different data points, $\mathbf{J}^T\mathbf{J}$ was *rank-deficient* ($\mathsf{rank}(\mathbf{J}^T\mathbf{J}) < 10$) at the start of learning, whereas $\mathbf{H}$ was of full-rank and its condition number was $O(10^2)$ or even smaller on average. With such a small MLP, the posed problem is not a small-residual problem; hence, $\mathbf{S}$ in $\mathbf{H}$ must be important to efficiency.

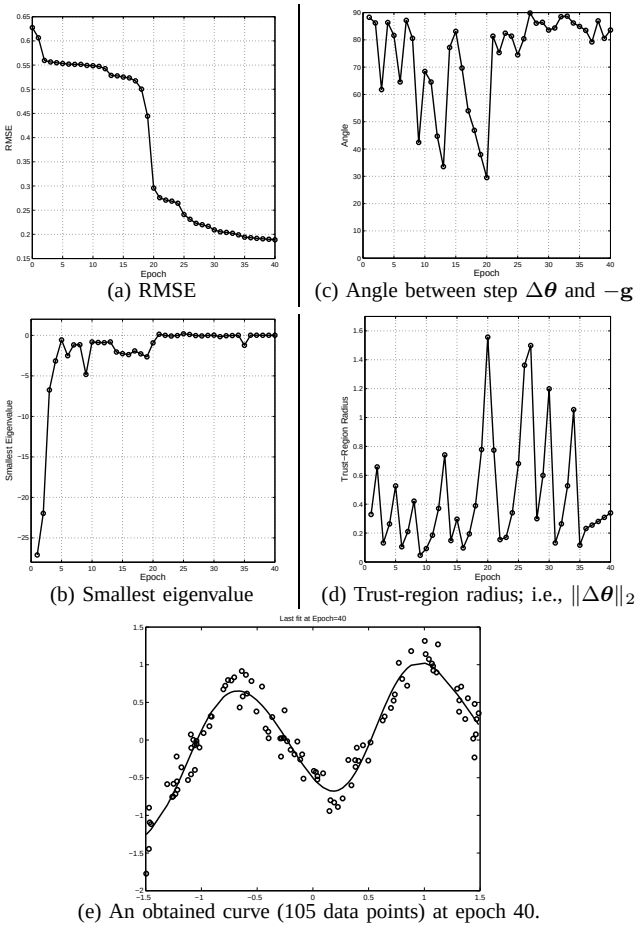
In any event, an algorithm that exploits the negative curvature appears to be very useful for MLP-learning. For all these ventures, an efficient algorithm that evaluates $\mathbf{H}$ is required because the Hessian evaluation cost is *dominant* when $D > n$, a usual case. Here, stagewise second-order BP comes into play as an efficient method of choice.

V. SUMMARY

The stagewise second-order backpropagation exploits MLP's symmetric and layered structure, evaluating the Hessian $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S}$ of the sum-square-error measure at the essentially same cost of the Gauss-Newton Hessian $\mathbf{J}^T\mathbf{J}$ (see Theorem III-A). The procedure works faster than popular rank-update methods that compute only $\mathbf{J}^T\mathbf{J}$, having made the exact evaluation of $\mathbf{H}$ feasible in practice (see numerical evidence in Sec.III-A). Furthermore, it computes $\mathbf{H}$ as stagewise-partitioned blocks in *outer-product form* [see Eq.(7)], which allows us to disclose intriguing structure of $\mathbf{S}$, leading to rank and eigenvalue analysis of $\mathbf{H}$.

When the residuals are still large and/or highly nonlinear, such as in the early learning phase, $\mathbf{H}$ is prone to be *indefinite*

*In double precision, if the condition number of a matrix is $O(10^{16})$ or greater, then no significant digits are correct; e.g., see Chapter 1 in [9].



(a) RMSE (c) Angle between step $\Delta\boldsymbol{\theta}$ and $-\mathbf{g}$

(b) Smallest eigenvalue (d) Trust-region radius; i.e., $\|\Delta\boldsymbol{\theta}\|_2$

(e) An obtained curve (105 data points) at epoch 40.

Fig. 3. A nonlinear regression (data fitting) problem by a 1-3-1 MLP (10 weights) with a second-order trust-region method that exploits negative curvature: (a) learning curve in RMSE; (b) the smallest eigenvalue of $\mathbf{H}$; (c) angle between the trust-region step $\Delta\boldsymbol{\theta}$ and the steepest descent direction $-\mathbf{g}$; (d) trust-region radius (i.e., step length); (e) a resulting curve.

(see Lemmas 1, 2 and 3); hence, $\mathbf{S}$ becomes important to efficiency for parameter optimization. $\mathbf{H}$ is expected to be positive definite at the solution, but on the way there, it may not be. The issue of $\mathbf{H}$ being less positive definite than $\mathbf{J}^T\mathbf{J}$ is certainly worth investigating because when it is a problem, the Gauss-Newton-type method (including natural-gradient learning) will degenerate into a slow first-order gradient-type approach. We have confirmed experimentally that $\mathbf{H}$ tends to be much better-conditioned than $\mathbf{J}^T\mathbf{J}$ at an early stage of learning. Indefiniteness of $\mathbf{H}$ is a welcome event in that it signals the direction of negative curvature to proceed. Since the theory of trust-region methods has thrived on *negative curvature*, we have examined a structure-exploiting trust-region algorithm; the simulation results were significant.

For a very large-scale problem, one may use Krylov-subspace methods; e.g., Lanczos algorithms for eigen-direction, which requires only implicit Hessian-vector multiply, but for evaluating the performance and analyzing the behaviors, the stagewise second-order BP would prove very useful (e.g., see T-CG in Sec.IV-A using a small example on purpose). In addition, some nonlinear least squares benchmark problems actually have stagewise structure (e.g., see p.394 [11]), for which $\mathbf{H}$ can be evaluated efficiently. In this way, our stagewise-BP could apply broadly to learning machines in yet unexplored domains and therefore have enormous potential for diverse future extensions.

ACKNOWLEDGMENT

The authors would like to thank James Demmel (UC Berkeley) and John Dennis (Rice University) for useful discussions.

REFERENCES

[1] Eiji Mizutani, Stuart Dreyfus, and Ken-ichi Nishio. On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application. In *Proceedings of the IEEE International Conference on Neural Networks (vol.2)*, pages 167–172, Como, Italy, July 2000.

[2] Eiji Mizutani and Stuart E. Dreyfus. Stagewise Newton, differential dynamic programming, and neighboring optimum control for neural-network learning. In *Proceedings of the 24th American Control Conference (ACC 2005)*, pages 1331–1336, Portland, Oregon, USA, June 2005.

[3] John E. Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, New Jersey, 1983.

[4] Eiji Mizutani, Stuart E. Dreyfus, and James W. Demmel. Second-order backpropagation algorithms for a stagewise-partitioned separable Hessian matrix. In *Proceedings of the 2005 International Joint Conference on Neural Networks (INNS-IEEE IJCNN'05)*, Montreal Quebec, CANADA, July 31-August 4 2005.

[5] Eiji Mizutani and Stuart E. Dreyfus. On derivation of stagewise second-order backpropagation by invariant imbedding for multi-stage neural-network learning. In *Proceedings of the IEEE-INNS International Joint Conf. on Neural Networks (IJCNN'06)*, Vancouver, Canada, July 2006.

[6] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford Press, 1995.

[7] Raúl Rojas. *Neural Networks – A Systematic Introduction*. Springer-Verlag, Berlin, New-York, 1996.

[8] Stuart E. Dreyfus. The numerical solution of non-linear optimal control problems. In D. Greenspan, editor, *Numerical Solutions of Nonlinear Differential Equations: Proceedings of an Advanced Symposium*, pages 97–113. John Wiley & Sons, Inc., 1966.

[9] James W. Demmel. *Applied Applied Numerical Linear Algebra*. SIAM, 1997.

[10] Richard O. Duda, Peter .E. Hart, and David G. Stork. *Pattern Classifi cation*. John Wiley & Sons, Inc., New York, 2001 (second ed.).

[11] K.M. Brown and J.E. Dennis. A new algorithm for nonlinear least squares curve fitting. In J.R. Rice, editor, *Mathematical Software*, pages 391–396. Academic Press, 1970.

[12] Eiji Mizutani. Powell's dogleg trust-region steps with the quasi-Newton augmented Hessian for neural nonlinear least-squares learning. In *Proceedings of the IEEE International Joint Conference on Neural Networks (vol.2)*, pages 1239–1244, Washington, D.C., USA, July 1999.

[13] Eiji Mizutani. On computing the Gauss-Newton Hessian matrix for neural-network learning. In *Proceedings of the 12th International Conference on Neural Information Processing (ICONIP 2005)*, pages 43–48, Taipei, TAIWAN, Oct. 30 – Nov. 2 2005.

[14] Beresford Parlett. *The Symmetric Eigenvalue Problem*. SIAM, 1997.

[15] Eiji Mizutani and Jing-Yun Carey Fan. On exploiting symmetry for multilayer perceptron learning. To appear in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. Orlando, Florida USA, August, 2007.

[16] S. Saarinen, R. Bramley, and G. Cybenko. Ill-conditioning in neural network training problems. *SIAM Sci. Stat. Comp*, 14(3):693–714, 1993.

[17] Andrew R. Conn, Nicholas IM Gould, and Philippe L. Toint. *Trust-Region Methods*. SIAM MPS/SIAM Series on Optimization, 2000.

[18] Eiji Mizutani and James W. Demmel. Iterative scaled trust-region learning in Krylov subspaces via Pearlmutter's implicit sparse Hessian-vector multiply. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, volume 16, pages 209–216. MIT Press, 2004.

[19] Guian Zhou and Jennie Si. Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency. *IEEE Trans. on Neural Networks*, 9(3):448–453, 1998.