On Improving Trust-Region Variable Projection Algorithms for Separable Nonlinear Least Squares Learning

Eiji Mizutani and James Demmel

***Abstract*— In numerical linear algebra, the variable projection (VP) algorithm has been a standard approach to separable "mixed" linear and nonlinear least squares problems since early 1970s. Such a separable case often arises in diverse contexts of machine learning (e.g., with generalized linear discriminant functions); yet VP is not fully investigated in the literature. We thus describe in detail its implementation issues, highlighting an economical trust-region implementation of VP in the framework of a so-called block-arrow least squares (BA) algorithm for a general multiple-response nonlinear model. We then present numerical results using an exponential-mixture benchmark, seven-bit parity, and color reproduction problems; in some situations, VP enjoys quick convergence and attains high classification rates, while in some others VP works poorly. This observation motivates us to investigate original VP's strengths and weaknesses compared with other (full-functional) approaches. To overcome the limitation of VP, we suggests how VP can be modified to be a Hessian matrix-based approach that exploits negative curvature when it arises. For this purpose, our economical BA algorithm is very useful in implementing such a modified VP especially when a given model is expressed in a multi-layer (neural) network for efficient Hessian evaluation by the so-called second-order stagewise backpropagation.**

I. Introduction

Given a set of D (training) data, we assume a general multiple F-response nonlinear model. Then, in the standard nonlinear least squares problem, m, the number of residuals, is given by $m \equiv FD$. The associated objective function E (in 2-norm) can be expressed below with three m-length vectors, $\mathbf{r}$, $\mathbf{y}$, and $\mathbf{t}$, of residuals, model's outputs, and desired outputs, respectively:

$$E(\boldsymbol{\theta}) = \tfrac{1}{2}\|\mathbf{r}(\boldsymbol{\theta})\|_2^2 = \tfrac{1}{2}\|\mathbf{y}(\boldsymbol{\theta}) - \mathbf{t}\|_2^2. \quad (1)$$

Here, $\boldsymbol{\theta}$ denotes an n-vector of parameters of our model to be optimized. In a variety of machine learning models, linear parameters (denoted by $\boldsymbol{\theta}^A$) and nonlinear ones ($\boldsymbol{\theta}^B$) often co-exist (hence, $\boldsymbol{\theta}^T \equiv [\boldsymbol{\theta}^{A^T} | \boldsymbol{\theta}^{B^T}]$), leading to so-called *separable* nonlinear least squares problems. In the literature, a great number of learning algorithms have been proposed; they could be classified into four groups below, depending on how such parameter separability is treated (see Fig.1):

- Group (1): *Full functional* approach by updating both $\boldsymbol{\theta}^A$ and $\boldsymbol{\theta}^B$ simultaneously (i.e., treating $\boldsymbol{\theta}$, all parameters, as if they were nonlinear parameters);
- Group (2): Independent updating approach (ignoring some second-order correlations between $\boldsymbol{\theta}^A$ and $\boldsymbol{\theta}^B$);

Eiji Mizutani is with the Department of Industrial Management, National Taiwan University of Science and Technology, Taipei, Taiwan (email: eiji@mail.ntust.edu.tw), and James Demmel is with Mathematics Department and Computer Science Division, University of California at Berkeley, CA, USA (email: demmel@cs.berkeley.edu).

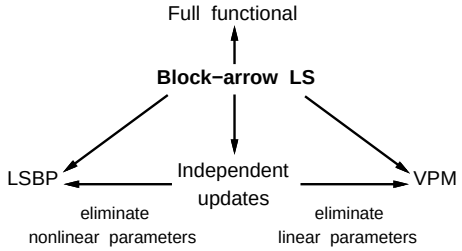

Fig. 1. A relation between several nonlinear least squares methods: (1) full-functional approach (e.g., NL2SOL); (2) LSBP (least squares backpropagation); (3) sequential independent update methods (layer-by-layer BP, ANFIS, etc); (4) VPM (variable projection method). Those could be implemented in a so-called block-arrow least squares, Algorithm BA.

- Group (3): Approach by eliminating $\boldsymbol{\theta}^B$;
- Group (4): *Reduced functional* method by eliminating $\boldsymbol{\theta}^A$.

Group (1) includes natural gradient learning, a well-known classical Levenberg-Marquardt method (e.g., trainlm.m [18]), and its much more robust *trust-region* version (see NL2SOL, a standard nonlinear-least-squares solver [6]). Group (2) uses a first-order gradient method for $\boldsymbol{\theta}^B$ and a linear least squares solution for $\boldsymbol{\theta}^A$, including a layer-by-layer backpropagation for multilayer-perceptron (MLP) learning [12], and ANFIS hybrid learning [17], which allows uphill movements (hence, may not converge). Group (3) eliminates $\boldsymbol{\theta}^B$ first (by using the inverse of nonlinear basis functions), then optimizes $\boldsymbol{\theta}^A$: In MLP-learning, for instance, hidden-node sigmoidal functions are inverted; see LSBP (least squares backpropagation) [1] and GIL (generalized inverse learning), pp. 144–149 in [2]. These methods tend to be very sensitive to initial parameters, resulting in slow learning, as reported on LSBP; this may be observed with other group methods as well (see Sec.V). Group (4) is represented by the so-called *Variable Projection* (VP) method [9], [10]. In what follows, we describe practical *trust-region* VP methods as Group (4) (see Algorithms TR-GPK & BA-VP later), and a full-functional method in Group (1) (see Algorithm BA) that resembles VP in implementation, being designed to exploit parameter separability in multiple-response separable problems. We then compare their performances to fathom out the strengths and weaknesses of VP.

II. Separable Jacobian Systems

From Eq.(1), the n-length gradient vector $\mathbf{g}$ and the $n \times n$ *symmetric* Hessian matrix $\mathbf{H}$ of $E(.)$ have such a special form as $\mathbf{g} = \mathbf{J}^T\mathbf{r}$ and $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S}$, where $\mathbf{J}$ is the $m \times n$ Jacobian matrix of the m-vector $\mathbf{r}$ of residuals, and $\mathbf{S} \equiv \sum_{i=1}^{m} r_i \nabla^2 r_i$, the residual Hessian matrix involving

second-derivatives of $\mathbf{r}$. The cross-product matrix $\mathbf{J}^T\mathbf{J}$ is called the *Gauss-Newton Hessian*, and the associated (modified) Gauss-Newton step $\Delta\boldsymbol{\theta}_{\text{GN}}$ is given as the solution to the Jacobian system $\mathbf{J}\Delta\boldsymbol{\theta}_{\text{GN}} = -\mathbf{r}$, or to the normal-equation system $\mathbf{J}^T\mathbf{J}\Delta\boldsymbol{\theta}_{\text{GN}} = -\mathbf{J}^T\mathbf{r}$. When $\mathbf{S}$ is omitted from $\mathbf{H}$, the *trust-region methods* [6], [3] pursue a regularized trust-region step $\Delta\boldsymbol{\theta}$ (for parameter update: $\boldsymbol{\theta}_{\text{next}} = \boldsymbol{\theta}_{\text{now}} + \Delta\boldsymbol{\theta}$) by solving the so-called *trust-region subproblem* below per epoch

$$\min_{\Delta\boldsymbol{\theta}}\left\{\|\mathbf{r}+\mathbf{J}\Delta\boldsymbol{\theta}\|_2^2 + \mu\|\Delta\boldsymbol{\theta}\|_2^2\right\} \iff \min_{\Delta\boldsymbol{\theta}}\left\{\|\mathbf{r}+\mathbf{J}\Delta\boldsymbol{\theta}\|_2^2\right\} \quad \text{subject to } \|\Delta\boldsymbol{\theta}\|_2^2 < R_{tr}^2 \tag{2}$$

where μ and R_{tr} are some scalar. The solution step $\Delta\boldsymbol{\theta}$ satisfies $(\mathbf{J}^T\mathbf{J} + \mu^2\mathbf{I})\Delta\boldsymbol{\theta} = -\mathbf{J}^T\mathbf{r}$, the Levenberg-Marquardt formula, corresponding to an *intermediate* step between the Gauss-Newton step $\Delta\boldsymbol{\theta}_{\text{GN}}$ and the *steepest descent* (or Cauchy) step. In reality, we often seek an approximate solution $\Delta\boldsymbol{\theta}$; e.g., see a so-called dogleg step in Eq.(13) later.

In separable nonlinear least squares, $\boldsymbol{\theta}$, the n-vector of parameters of our model, separates into $\boldsymbol{\theta}^A$, n_A linear parameters, and $\boldsymbol{\theta}^B$, n_B nonlinear ones ($n = n_A + n_B$). Since the model is assumed to produce multiple F outputs, $\boldsymbol{\theta}^A$ is further split into F groups; hence, the following separability:

$$\boldsymbol{\theta}^T = \left[\boldsymbol{\theta}^{A^T} \,\middle|\, \boldsymbol{\theta}^{B^T}\right] = \left[\boldsymbol{\theta}_1^{A^T}, \cdots, \boldsymbol{\theta}_k^{A^T}, \cdots, \boldsymbol{\theta}_F^{A^T} \,\middle|\, \boldsymbol{\theta}^{B^T}\right]. \tag{3}$$

Here, we assume that n_A *linear* parameters $\boldsymbol{\theta}^A$ directly link to F terminal outputs, each of which is associated with a subgroup of $C_A \equiv \frac{n_A}{F}$ linear parameters. This is the usual situation in multilayer perceptron (MLP) learning. Given D data, the final output vector of the model can be expressed as an m-vector below ($m \equiv FD$):

$$\underbrace{\mathbf{y}(\boldsymbol{\theta})}_{m\times 1} = \underbrace{\widetilde{\mathbf{A}}(\boldsymbol{\theta}^B)}_{m\times n_A}\underbrace{\boldsymbol{\theta}^A}_{n_A\times 1}. \tag{4}$$

Accordingly, the $m \times n$ residual Jacobian matrix $\mathbf{J}$ has a so-called *block angular* form $\mathbf{J} \equiv [\widetilde{\mathbf{A}}|\widetilde{\mathbf{B}}]$ (cf. [11]), where

$$\underbrace{\widetilde{\mathbf{A}}}_{m\times n_A} \overset{\text{def}}{=} \begin{bmatrix} \mathbf{A} & & \\ & \mathbf{A} & \\ & & \ddots \\ & & & \mathbf{A} \end{bmatrix}, \text{ and } \underbrace{\widetilde{\mathbf{B}}}_{m\times n_B} \overset{\text{def}}{=} \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \vdots \\ \mathbf{B}_F \end{bmatrix}. \tag{5}$$

In $\widetilde{\mathbf{A}}$, there are F *identical* sub-blocks; hence, need to store only one sub-block $\mathbf{A}$ of size D-by-C_A (with $C_A \equiv \frac{n_A}{F}$). In this separable context, $E(.)$ in Eq.(1) can be rewritten as

$$E(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{r}(\boldsymbol{\theta})\|_2^2 = \frac{1}{2}\|\mathbf{y}(\boldsymbol{\theta}) - \mathbf{t}\|_2^2 = \frac{1}{2}\|\widetilde{\mathbf{A}}\boldsymbol{\theta}^A - \mathbf{t}\|_2^2, \tag{6}$$

where both $\mathbf{r}$ and $\mathbf{t}$ encapsulate all the multiple F-output information in an m-length single-column vector form; namely, $\mathbf{r} \equiv [\mathbf{r}_1^T, \mathbf{r}_2^T, ..., \mathbf{r}_F^T]^T$ and $\mathbf{t} \equiv [\mathbf{t}_1^T, \mathbf{t}_2^T, ..., \mathbf{t}_F^T]^T$. Alternatively, $E(.)$ in Eq.(1) can be expressed in a multiple F-column *matrix* form using the Frobenius norm of matrix (e.g., $\|\mathbf{X}\|_F^2 = \sum_{i,j} x_{i,j}^2$) as

$$\begin{aligned} E(\boldsymbol{\theta}) &= \tfrac{1}{2}\|\mathbf{R}(\boldsymbol{\theta})\|_F^2 \\ &= \tfrac{1}{2}\|\mathbf{Y}(\boldsymbol{\theta}) - \mathbf{T}\|_F^2 = \tfrac{1}{2}\|\mathbf{A}(\boldsymbol{\theta}^B)\boldsymbol{\Theta}^A - \mathbf{T}\|_F^2, \end{aligned} \tag{7}$$

where $\mathbf{R}(\boldsymbol{\theta})$ is D-by-F; $\mathbf{Y}(\boldsymbol{\theta})$ D-by-F; $\mathbf{A}(\boldsymbol{\theta}^B)$ D-by-C_A (with $C_A \equiv \frac{n_A}{F}$); $\boldsymbol{\Theta}^A$ C_A-by-F; and $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_F]$, D-by-F, comprising multiple F sets of D-length *desired output* vectors in columns; that is, column k ($k = 1, \cdots, F$) is given by a D-length vector $\mathbf{t}_k$. Each target vector $\mathbf{t}_k$ is to be fit by our nonlinear model in the sense that each set of linear parameters $\boldsymbol{\theta}_k^A$ (i.e., the kth column of $\boldsymbol{\Theta}^A$) linked to the kth output are allowed to be different for each $\mathbf{t}_k$ but all nonlinear parameters $\boldsymbol{\theta}^B$ must be optimized to target those $\mathbf{t}_k (k = 1, \cdots, F)$ altogether. To this end, we may solve a multiple right-hand-side *Jacobian system* for optimizing $\boldsymbol{\Theta}^A$ but always solve a single right-hand-side *Jacobian system* for $\boldsymbol{\theta}^B$. Depending on the context, the separable Jacobian system can be represented in various forms below

$$\begin{cases} \text{(a) } \widetilde{\mathbf{A}}\Delta\boldsymbol{\theta}_{\text{GN}}^A + \widetilde{\mathbf{B}}\Delta\boldsymbol{\theta}_{\text{GN}}^B = -\mathbf{r}, \\ \text{(b) } \widetilde{\mathbf{A}}\Delta\boldsymbol{\theta}_{\text{GN}}^A + \widetilde{\mathbf{B}}\Delta\boldsymbol{\theta}_{\text{GN}}^B = -\left(\widetilde{\mathbf{A}}\boldsymbol{\theta}_{\text{now}}^A - \mathbf{t}\right), \\ \text{(c) } \widetilde{\mathbf{A}}(\boldsymbol{\theta}_{\text{now}}^A + \Delta\boldsymbol{\theta}_{\text{GN}}^A) + \widetilde{\mathbf{B}}\Delta\boldsymbol{\theta}_{\text{GN}}^B = \mathbf{t}, \\ \text{(d) } \widetilde{\mathbf{A}}\boldsymbol{\theta}_{\text{next}}^A + \widetilde{\mathbf{B}}\Delta\boldsymbol{\theta}_{\text{GN}}^B = \mathbf{t}, \\ \text{(e) } \widetilde{\mathbf{B}}\Delta\boldsymbol{\theta}_{\text{GN}}^B = -\left(\widetilde{\mathbf{A}}\boldsymbol{\theta}_{\text{next}}^A - \mathbf{t}\right) = -\mathbf{r}_{\text{next}}, \end{cases} \tag{8}$$

where $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{B}}$ are defined in Eq.(5). From (c) to (d), we take $\Delta\boldsymbol{\theta}^A = \Delta\boldsymbol{\theta}_{\text{GN}}^A$, and then update the n_A-vector $\boldsymbol{\theta}^A$ of linear parameters by $\boldsymbol{\theta}_{\text{next}}^A = \boldsymbol{\theta}_{\text{now}}^A + \Delta\boldsymbol{\theta}_{\text{GN}}^A$, which is obtainable directly from linear-equation solving on either $\widetilde{\mathbf{A}}\boldsymbol{\theta}_{\text{next}}^A = \mathbf{t}$ or $\mathbf{A}\boldsymbol{\Theta}_{\text{next}}^A = \mathbf{T}$; this means that there is no need to provide the *initial* values for $\boldsymbol{\theta}^A$, the linear parameters (hence, *automatic initialization* of $\boldsymbol{\theta}^A$). As a result, we get (e), where $\mathbf{r}_{\text{next}}$ is the new residual vector resulting from updating the linear parameters (from $\boldsymbol{\theta}_{\text{now}}^A$ to $\boldsymbol{\theta}_{\text{next}}^A$). Here the question is whether $\widetilde{\mathbf{B}}$ is computed *before* $\boldsymbol{\theta}^A$ is updated or *after*. The former is in the spirit of the full functional approach [i.e., Group (1)], whereas the latter is the essence of VP methods to be described next.

III. Variable Projection (VP) Algorithms

VP methods eliminate linear parameters $\boldsymbol{\theta}^A$ to optimize nonlinear parameters $\boldsymbol{\theta}^B$ alone in the reduced parameter space (from $\Re^n$ down to $\Re^{n_B}$) by minimizing $E_B(\boldsymbol{\theta}^B)$, the residuals that are projected onto $\mathsf{null}(\mathbf{A}^T)$, the null space of $\mathbf{A}^T$. The resulting objective function $E_B(\boldsymbol{\theta}^B)$ is obtainable from the linear least squares (LLS) subproblem (see pp. 108-109 in [4])

$$\begin{aligned} E_B(\boldsymbol{\theta}^B) &\overset{\text{def}}{=} \min_{\boldsymbol{\Theta}^A} \tfrac{1}{2}\|\mathbf{A}\boldsymbol{\Theta}^A - \mathbf{T}\|_F^2 \\ &= \min_{\boldsymbol{\Theta}^A} \tfrac{1}{2}\left\{\|\mathbf{U}\boldsymbol{\Theta}^A - \mathbf{Q}_1^T\mathbf{T}\|_F^2 + \|\mathbf{Q}_2^T\mathbf{T}\|_F^2\right\} \\ &= \tfrac{1}{2}\|\mathbf{Q}_2^T\mathbf{T}\|_F^2, \end{aligned} \tag{9}$$

which yields $E_B(\boldsymbol{\theta}^B) = \frac{1}{2}\|\mathbf{Q}_2^T\mathbf{T}\|_F^2$, called *the VP functional*; here, $\mathbf{Q}$, $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are orthogonal matrices, and $\mathbf{U}$ is a C_A-by-C_A upper triangular matrix (with positive diagonal elements when $\mathbf{A}$ has full rank), which is the Cholesky

factor of $\mathbf{A}^T\mathbf{A}$, resulting from the full QR decomposition of $\mathbf{A}(\boldsymbol{\theta}^B)$ below:

$$\underbrace{\mathbf{A}}_{D\times C_A}=\left[\begin{array}{c} \\ \underbrace{\mathbf{Q}}_{D\times D} \\ \\ \end{array}\right]\left[\begin{array}{c}\mathbf{U}\\ \hline \mathbf{0}\end{array}\right]=\left[\begin{array}{c|c} \underbrace{\mathbf{Q}_1}_{D\times C_A} & \underbrace{\mathbf{Q}_2}_{D\times(D-C_A)}\end{array}\right]\left[\begin{array}{c}\mathbf{U}\\ \hline \mathbf{0}\end{array}\right]. \quad (10)$$

where $\mathsf{range}(\mathbf{Q}_1)=\mathsf{range}(\mathbf{A})$; $\mathsf{range}(\mathbf{Q}_2)=\mathsf{range}(\mathbf{A})^{\perp}=\mathsf{null}(\mathbf{A}^T)$; e.g., see pp. 108-109 in [4].

VP methods first eliminate $\boldsymbol{\theta}^A$ using the full LLS-step, the solution to $\mathbf{Q}^T\mathbf{A}\boldsymbol{\Theta}^A_{\text{next}}=\mathbf{Q}^T\mathbf{T}$, a multiple right-hand-side LLS, then seek $\Delta\boldsymbol{\theta}^B$ for updating $\boldsymbol{\theta}^B$, often using the aforementioned Gauss-Newton step $\Delta\boldsymbol{\theta}^B_{\text{GN}}$, the solution to the Jacobian system $\widetilde{\mathbf{B}}_{Q2}\Delta\boldsymbol{\theta}^B_{\text{GN}}=\widetilde{\mathbf{t}}_{Q2}$, which signifies

$$\underbrace{\left[\begin{array}{c}\mathbf{Q}_2^T\mathbf{B}^1\\ \mathbf{Q}_2^T\mathbf{B}^2\\ \vdots\\ \mathbf{Q}_2^T\mathbf{B}^F\end{array}\right]}_{F(D-C_A)\times n_B\ \ \widetilde{\mathbf{B}}_{Q2}}\Delta\boldsymbol{\theta}^B_{\text{GN}}\ =\ \underbrace{\left[\begin{array}{c}\mathbf{Q}_2^T\mathbf{t}_1\\ \mathbf{Q}_2^T\mathbf{t}_2\\ \vdots\\ \mathbf{Q}_2^T\mathbf{t}_F\end{array}\right]}_{F(D-C_A)\times 1\ \ \widetilde{\mathbf{t}}_{Q2}}. \quad (11)$$

The associated normal-equation system is given by

$$\widetilde{\mathbf{B}}^T_{Q2}\widetilde{\mathbf{B}}_{Q2}\Delta\boldsymbol{\theta}^B_{\text{GN}}=\widetilde{\mathbf{B}}^T_{Q2}\widetilde{\mathbf{t}}_{Q2}. \quad (12)$$

Here, only $\mathbf{Q}_2^T\mathbf{B}^k(k=1,...,F)$ (known as the Kaufman Jacobian matrix [7]) are used with $\mathbf{Q}_1^T\mathbf{B}^k$ omitted, leading to what we called Golub-Pereyra-Kaufman's VP method: Their original version employed a classical Levenberg-Marquardt method (e.g., see varpro.f at www.netlib.org). Its more robust *trust-region* variant [7] that uses NL2SOL [6] is also available on the web[‡]. Yet it would be an arduous task to adapt such Fortrun codes (even by using f2c[*]) for attacking machine learning problems. For this reason, we next present algorithmic recipes of a *trust-region* VP algorithm:

Algorithm TR-GPK: *A trust-region safeguarded Golub-Pereyra-Kaufman's VP algorithm.*

(0) Initialize $\boldsymbol{\theta}^B_{\text{next}}=\boldsymbol{\theta}^B_{\text{now}}\leftarrow\boldsymbol{\theta}^B_{\text{init}}$ and $\Delta\boldsymbol{\theta}^B\leftarrow\mathbf{0}$.
(1) Compute a D-by-C_A matrix $\mathbf{A}\equiv\mathbf{A}(\boldsymbol{\theta}^B_{\text{next}})$ of basis-function outputs by forward pass.
(2) Perform *full* QR on $\mathbf{A}$ [see Eq. (10)].
(3) Obtain $\boldsymbol{\Theta}^A_{\text{next}}$ by the LLS (linear least squares) step, solving the triangular system $\mathbf{U}\boldsymbol{\Theta}^A_{\text{next}}=\mathbf{Q}_1^T\mathbf{T}$.
(4) Evaluate residuals: $\mathbf{R}_{\text{next}}=\mathbf{Q}_2^T\mathbf{T}$, which yields a D-by-F matrix of residuals of $E_B(.)$ in Eq.(9).
(5) If stopping criteria (e.g., use $\mathbf{R}_{\text{next}}$ to check) are met, then **terminate**; else continue:
  – If the trust-region step $\Delta\boldsymbol{\theta}^B$ is *satisfactory*, then
    (a) **accept** the step: $\boldsymbol{\theta}^B_{\text{now}}\leftarrow\boldsymbol{\theta}^B_{\text{next}}$,
    (b) set $\mathbf{R}_{\text{now}}\leftarrow\mathbf{R}_{\text{next}}$ and $E_B(\boldsymbol{\theta}^B_{\text{now}})=\frac{1}{2}\|\mathbf{R}_{\text{now}}\|^2_F$;
    (c) adapt the trust-region radius R_{tr};
  – Otherwise,
    (d) **reject** the step: $\boldsymbol{\theta}^B_{\text{now}}=\boldsymbol{\theta}^B_{\text{next}}-\Delta\boldsymbol{\theta}^B$,
    (e) shrink the trust-region radius R_{tr};
    (f) recompute a trust-region step $\Delta\boldsymbol{\theta}^B$ by Eq.(13);
    (g) go back to Step (1) with $\boldsymbol{\theta}^B_{\text{next}}=\boldsymbol{\theta}^B_{\text{now}}+\Delta\boldsymbol{\theta}^B$.
(6) Evaluate the Jacobian matrix $\widetilde{\mathbf{B}}$.
(7) Obtain $\widetilde{\mathbf{B}}_{Q2}$ by $\mathbf{Q}_2^T\mathbf{B}^k$ $(k=1,...,F)$; see Eq.(11).
(8) Compute the *Cauchy step* $\Delta\boldsymbol{\theta}^B_C\stackrel{\text{def}}{=}-\left(\frac{\mathbf{g}^T\mathbf{g}}{\mathbf{g}^T\mathbf{H}\mathbf{g}}\right)\mathbf{g}$, where $\mathbf{H}=\widetilde{\mathbf{B}}^T_{Q2}\widetilde{\mathbf{B}}_{Q2}$ and $\mathbf{g}=-\widetilde{\mathbf{B}}^T_{Q2}\widetilde{\mathbf{t}}_{Q2}$; see Eq.(12).
(9) Determine a trust-region step $\Delta\boldsymbol{\theta}^B$ of length n_B by:

$$\Delta\boldsymbol{\theta}^B\leftarrow\begin{cases}(a)\ \Delta\boldsymbol{\theta}^B_{\text{RC}}\stackrel{\text{def}}{=}-\frac{R_{tr}}{\|\mathbf{g}\|}\mathbf{g},\ \text{if } R_{tr}\le\|\Delta\boldsymbol{\theta}^B_C\|;\\ (b)\ \Delta\boldsymbol{\theta}^B_{\text{GN}}\text{ in Eq.(11), if } R_{tr}\ge\|\Delta\boldsymbol{\theta}^B_{\text{GN}}\|;\\ (c)\ \Delta\boldsymbol{\theta}^B_D\stackrel{\text{def}}{=}(1-\beta)\Delta\boldsymbol{\theta}^B_C+\beta\Delta\boldsymbol{\theta}^B_{\text{GN}};\end{cases} \quad (13)$$

where scalar β is the positive root to $\|\mathbf{s}+\beta\mathbf{p}\|=R_{tr}$:

$$\beta\stackrel{\text{def}}{=}\frac{-\mathbf{s}^T\mathbf{p}+\sqrt{(\mathbf{s}^T\mathbf{p})^2+\mathbf{p}^T\mathbf{p}(R^2_{tr}-\mathbf{s}^T\mathbf{s})}}{\mathbf{p}^T\mathbf{p}} \quad (14)$$

with $\mathbf{s}\equiv\Delta\boldsymbol{\theta}^B_C$ and $\mathbf{p}\equiv\Delta\boldsymbol{\theta}^B_{\text{GN}}-\Delta\boldsymbol{\theta}^B_C$.
(10) Set $\boldsymbol{\theta}^B_{\text{next}}=\boldsymbol{\theta}^B_{\text{now}}+\Delta\boldsymbol{\theta}^B$, and go back to Step (1). □

Remarks: Initially at epoch 1, when Step (5) is "first" entered, no trust-region step is computed [i.e., $\Delta\boldsymbol{\theta}^B=\mathbf{0}$ given by Step (0)]; hence, $\Delta\boldsymbol{\theta}^B$ is always *accepted* and Step (6) follows Steps (5-a) and (5-b). Otherwise, $\Delta\boldsymbol{\theta}^B$ is *accepted* when ρ is greater than a small constant (e.g., 0.01); here,

$$\rho\stackrel{\text{def}}{=}\frac{\text{Actual Error Reduction}}{\text{Predicted Error Reduction}}=\frac{E_B(\boldsymbol{\theta}^B_{\text{next}})-E_B(\boldsymbol{\theta}^B_{\text{now}})}{q(\Delta\boldsymbol{\theta}^B)} \quad (15)$$

measures how well the actual error reduction is predicted by $q(\Delta\boldsymbol{\theta}^B)\stackrel{\text{def}}{=}\mathbf{g}^T\Delta\boldsymbol{\theta}^B+\frac{1}{2}\Delta\boldsymbol{\theta}^{B^T}\mathbf{H}\Delta\boldsymbol{\theta}^B$, the local quadratic error model of $E_B(.)$. If pass the test (i.e., $\Delta\boldsymbol{\theta}^B$ is *satisfactory*), then follow Steps (5-a) to (5-c); otherwise, reject $\Delta\boldsymbol{\theta}^B$ following Steps (5-d) to (5-g). At Step (5-c), R_{tr} is kept or changed, depending on ρ, but if $R_{tr}<R^{\min}_{tr}$, then $R_{tr}\leftarrow R^{\min}_{tr}$ (e.g., $R^{\min}_{tr}=0.01$) for the next epoch, although R_{tr} can be arbitrarily small at Step (5-e) within each epoch. Step (9) is known as a *dogleg* trust-region globalization strategy, and Eq.(13)(c) gives a dogleg step, an intermediate step between the Cauchy (steepest descent) step $\Delta\boldsymbol{\theta}^B_C$ and the Gauss-Newton step $\Delta\boldsymbol{\theta}^B_{\text{GN}}$, for which Eq.(11) is solved only when $R_{tr}>\|\Delta\boldsymbol{\theta}^B_C\|$. If Step (8) is "first" entered (only at epoch 1), the trust-region radius can be initialized by $R_{tr}\leftarrow\mathsf{minimum}\{R^{\max}_{tr},\|\Delta\boldsymbol{\theta}^B_C\|\}$; hence, either $\Delta\boldsymbol{\theta}^B=\Delta\boldsymbol{\theta}^B_{\text{RC}}$ or $\Delta\boldsymbol{\theta}^B=\Delta\boldsymbol{\theta}^B_C$ in Eq.(13) at Step (9). See Chap. 7 [3] also.

The posed algorithm uses full QR-decomposition in Eq.(10) and efficient multiply by $\mathbf{Q}_2^T$ at Step (4) for $\mathbf{Q}_2^T\mathbf{T}$, which is reshaped as $\widetilde{\mathbf{t}}_{Q2}$ at Step (8) [see Eq.(11)], and at Step (8) for $\widetilde{\mathbf{B}}_{Q2}$ (i.e., $\mathbf{Q}_2^T\mathbf{B}^k;k=1,...,F$). The fastest way to perform this operation is problem-oriented, depending on the relative number of columns in $\mathbf{Q}_1$ and $\mathbf{Q}_2$ as well as the number of columns in $\mathbf{T}$ and $\mathbf{B}^k$. These may not be troublesome on Matlab, but for C or C++ programming, here we show two simplest implementations with CLAPACK (see www.netlib.org/clapack): (1) Form $\mathbf{Q}$ or $\mathbf{Q}^T$ explicitly by calling DORMQR with the D-by-D identity matrix plugged in, and only use $\mathbf{Q}_2$ in DGEMM (no copying needed, just pointing to the right starting location); and (2)

[‡]Consult "Port-3 Manual" (e.g., see usnsfb.pdf for a VP method, nsg.f) available at www-out.bell-labs.com/project/PORT/doc/port3docpdf.tar.

[*]www.llnl.gov/casc/Overture/henshaw/install/node6.html

Multiply by $\mathbf{Q}^T$ using DORMQR and only keep the trailing $D-C_A$ rows of the answer, where $D-C_A$ is the number of columns in $\mathbf{Q}_2$. Hereafter, we occasionally quote CLAPACK subroutines for readers' convenience.

IV. Separable Normal Equations Approach

In the normal-equation approaches, the cross-product (Gauss-Newton Hessian) matrix $\mathbf{J}^T\mathbf{J}$ [see $\mathbf{J}$ in Eq.(5)] is an n-by-n (block) *arrow matrix* (due to its appearance) below

$$\underbrace{\mathbf{J}^T\mathbf{J}}_{n\times n} = \left[\begin{array}{c|c} \underbrace{\widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}}}_{\substack{n_A\times n_A \\ (n_A\equiv FC_A)}} & \widetilde{\mathbf{A}}^T\widetilde{\mathbf{B}} \\ \hline \widetilde{\mathbf{B}}^T\widetilde{\mathbf{A}} & \widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}} \end{array}\right] = \left[\begin{array}{ccccc|c} \mathbf{A}^T\mathbf{A} & & & & & \mathbf{A}^T\mathbf{B}_1 \\ & \mathbf{A}^T\mathbf{A} & & & & \mathbf{A}^T\mathbf{B}_2 \\ & & \ddots & & & \vdots \\ & & & \mathbf{A}^T\mathbf{A} & & \mathbf{A}^T\mathbf{B}_F \\ \hline \mathbf{B}_1^T\mathbf{A} & \mathbf{B}_2^T\mathbf{A} & \cdots & \mathbf{B}_F^T\mathbf{A} & & \widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}} \end{array}\right], \quad (16)$$

where we only need to evaluate half of only one (symmetric) block $\mathbf{A}^T\mathbf{A}$ of size $C_A\times C_A$; half of $\widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}}=\sum_{k=1}^{F}\mathbf{B}_k^T\mathbf{B}_k$ of size $n_B\times n_B$; and F blocks of $\mathbf{B}_k^T\mathbf{A}$ of size $n_B\times C_A$.

In Algorithm TR-GPK, we exploit sparsity of $\mathbf{J}$ [see Eq.(5)] by employing QR block by block, factoring $\mathbf{A}$ only once as in Eq.(10). In the normal-equation approach with (modified) Cholesky decomposition, we also exploit sparsity; that is, an arrow matrix $\mathbf{J}^T\mathbf{J}$ should be formed † with *its arrowhead pointing downward to the right* (see pp. 83–90 in [4]), as shown in Eq.(16), in order to avoid any *fill-ins*, creation of *new* non-zeros, which can occur in *forming* and *factoring* $\mathbf{J}^T\mathbf{J}$. Then, the resultant lower-triangular Cholesky factor $\mathbf{L}$ requires the same memory space for *dense* blocks in the lower-half of the original matrix $\mathbf{J}^T\mathbf{J}$ [see Eqs.(16) and (17)]. All manipulations are performed in place with no extra space. We next describe such a normal-equation approach that belongs to Group (1) in Sec.I, and then relate it to a different implementation of Algorithm TR-GPK.

A. Block-Arrow Least Squares Algorithms

By the full functional approaches of Group (1), all the n parameters are updated *simultaneously* per epoch, at which we determine a step $\Delta\boldsymbol{\theta}$ subject to *trust-region regularization* by using Eq.(13), where $\Delta\boldsymbol{\theta}^B$ of length n_B should be replaced with $\Delta\boldsymbol{\theta}$ of length n in this context. For instance, suppose that we seek the Gauss-Newton step $\Delta\boldsymbol{\theta}_{GN}$ of length n as the solution to the normal equations $\mathbf{J}^T\mathbf{J}\Delta\boldsymbol{\theta}_{GN}=-\mathbf{J}^T\mathbf{r}$. Then, we reduce it by Cholesky to the two-triangular systems: $\mathbf{L}\mathbf{L}^T\Delta\boldsymbol{\theta}_{GN}=-\mathbf{J}^T\mathbf{r}$, where $\mathbf{L}$ is the lower-triangular Cholesky factor. Recall the dogleg trust-region strategy in Eq.(13); the posed linear-equation solving for $\Delta\boldsymbol{\theta}_{GN}$ is performed only when the current trust region radius R_{tr} is greater than the length of the steepest-descent (or Cauchy) step $\Delta\boldsymbol{\theta}_C\equiv-(\frac{\mathbf{g}^T\mathbf{g}}{\mathbf{g}^T\mathbf{H}\mathbf{g}})\mathbf{g}$, for which $\mathbf{H}=\mathbf{J}^T\mathbf{J}$ in Eq.(16) and $\mathbf{g}=\mathbf{J}^T\mathbf{r}$ (the n-length gradient vector) are evaluated. The following summarizes the normal-equation solving with Cholesky on the arrow matrix in Eq.(16), when $R_{tr}>\|\Delta\boldsymbol{\theta}_C\|$ holds (after $\Delta\boldsymbol{\theta}_C$ is evaluated).

Algorithm BA: *Block-arrow least squares algorithm [15] with (modified) Cholesky* (when $R_{tr}>\|\Delta\boldsymbol{\theta}_C\|$).

(1) Perform Cholesky on $\mathbf{A}^T\mathbf{A}$ (i.e., $\mathbf{A}^T\mathbf{A}=\mathbf{U}^T\mathbf{U}$), where $\mathbf{U}$ is a $C_A\times C_A$ upper-triangular matrix.

(2) Obtain F blocks $\mathbf{Z}_k=(\mathbf{B}_k^T\mathbf{A})\mathbf{U}^{-1}$ of size $n_B\times C_A$ by solving a transposed (multiple right-hand side) triangular system (here, no need to invert $\mathbf{U}$).

(3) Compute the n_B-by-n_B *Schur complement* matrix $\mathbf{V}$ of $\widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}}$ in $\mathbf{J}^T\mathbf{J}$ as $\mathbf{V}=\widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}}-\sum_{k=1}^{F}\mathbf{Z}_k\mathbf{Z}_k^T$; e.g., by calling DSYRK, a BLAS (www.netlib.org/blas) routine, for symmetric rank updates.

(4) Perform (modified) Cholesky on $\mathbf{V}$ to get $\mathbf{V}$=$\mathbf{X}\mathbf{X}^T$.

(5) Obtain $\Delta\boldsymbol{\theta}_{GN}$ of length n by solving the resulting two block-structured triangular systems, $\mathbf{L}\mathbf{L}^T\Delta\boldsymbol{\theta}_{GN}=-\mathbf{g}$, where $\mathbf{L}$, a block lower-triangular Cholesky factor below, is evaluated block by block in the preceding steps:

$$\underbrace{\mathbf{L}}_{n\times n} \stackrel{\text{def}}{=} \left[\begin{array}{ccccc} \mathbf{U}^T & & & & \\ & \mathbf{U}^T & & & \\ & & \ddots & & \\ & & & \mathbf{U}^T & \\ \hline \mathbf{Z}_1 & \mathbf{Z}_2 & \cdots & \mathbf{Z}_F & \mathbf{X} \end{array}\right]. \quad (17)$$

- Determine $\Delta\boldsymbol{\theta}$ (of length n), an *effective* (regularized) trust-region step, using the obtained $\Delta\boldsymbol{\theta}_{GN}$; e.g., by a *dogleg* mechanism in Eq.(13). □

Remarks: One may employ the Gill-Murray-Wright modified Cholesky decomposition (see pp.108-115 in [8]) that automatically renders $\mathbf{J}^T\mathbf{J}$ sufficiently positive-definite during the procedure (leading to a modified Gauss-Newton step). In place of Steps (1) and (2), rather than with (modified) Cholesky, we could proceed with *reduced* Householder QR-factorization (e.g., see p. 121 in [4]) in the following way:

(1′) Perform *reduced* QR on $\mathbf{A}$ as $\mathbf{A}=\mathbf{Q}_1\mathbf{U}$ [i.e., Eq.(10) with no $\mathbf{Q}_2$ involved], which is done in place, with a factored form of $\mathbf{Q}_1$ and $\mathbf{U}$ overwriting $\mathbf{A}$ by calling CLAPACK routine DGEQRF.

(2′) Construct $\mathbf{Z}_k=\mathbf{B}_k^T\mathbf{Q}_1$ by calling CLAPACK routine DORMQR, without forming $\mathbf{Q}_1$ explicitly.

- The rest of the procedures remain the same; so, Step (3) in Algorithm BA follows, yielding the Schur complement matrix $\mathbf{V}=\widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}}-\sum_{k=1}^{F}\mathbf{Z}_k\mathbf{Z}_k^T$, which can be expressed as below (owing to a fact that $\mathbf{Q}\mathbf{Q}^T=\mathbf{I}$)

$$\mathbf{V}=\sum_{k=1}^{F}\Big(\mathbf{B}_k^T\mathbf{Q}\mathbf{Q}^T\mathbf{B}_k-\mathbf{B}_k^T\mathbf{Q}_1\mathbf{Q}_1^T\mathbf{B}_k\Big)=\sum_{k=1}^{F}\mathbf{B}_k^T\mathbf{Q}_2\mathbf{Q}_2^T\mathbf{B}_k. \quad (18)$$

Since D, the number of data, is usually large in machine learning, the Cholesky-driven Algorithm BA may be preferred, although it may impair the accuracy when $\mathbf{A}$ is ill-conditioned (due to squaring the condition number of matrix $\mathbf{A}$). This is common to the normal-equation approaches.

†One could observe "complex" sparse patterns in $\mathbf{J}$ (accordingly in $\mathbf{J}^T\mathbf{J}$) obtainable by calcjx.m and calcjejj.m (used in trainlm.m, a "classical" Levenberg-Marquardt method) in the MATLAB Neural Network Toolbox. For parameter optimization purposes, it is not recommendable to use or approximate the inverse of the posed block-arrow (Gauss-Newton) Hessian matrix $\widehat{\mathbf{H}}$ because $\widehat{\mathbf{H}}^{-1}$ is always *dense*; unfortunately, such methods (based on the well-known Sherman-Morrison rank-update formula) are still popular in optimizing Takagi-Sugeno fuzzy systems and neural networks (e.g., by a version of natural-gradient learning).

B. A VP Algorithm implemented by Algorithm BA

One might immediately notice a close resemblance between Algorithms TR-GPK and BA; for instance, $\mathbf{V}$ in Eq.(18) is $\widetilde{\mathbf{B}}_{Q2}^T\widetilde{\mathbf{B}}_{Q2}$, the same in form as in the left-hand side of Eq.(12), although $\widetilde{\mathbf{B}}$ differs because VP methods evaluate $\mathbf{B}_k$ blocks ($k=1,...,F$) *after* the linear-least-squares (LLS) step is taken (i.e., the linear parameters $\boldsymbol{\theta}^A$ are updated) whereas Algorithm BA computes $\mathbf{B}_k$ *before* that. Also, the right-hand side of Eq.(12) is the negative gradient vector $-\mathbf{g}$ evaluated after the LLS-step is taken:

$$\begin{aligned}\widetilde{\mathbf{B}}_{Q2}^T\widetilde{\mathbf{t}}_{Q2} &= \sum_{k=1}^{F}\mathbf{B}_k^T\mathbf{Q}_2\mathbf{Q}_2^T\mathbf{t}_k = \sum_{k=1}^{F}\mathbf{B}_k^T\left(\mathbf{I}-\mathbf{A}\mathbf{A}^+\right)\mathbf{t}_k \\ &= \widetilde{\mathbf{B}}^T(\mathbf{t}-\widetilde{\mathbf{A}}\boldsymbol{\theta}_{\text{next}}^A) = -\widetilde{\mathbf{B}}^T\mathbf{r}_{\text{next}} = -\mathbf{g},\end{aligned} \tag{19}$$

where $\mathbf{A}^+$ is the pseudo-inverse of $\mathbf{A}$, and $\mathbf{r}_{\text{next}}=\widetilde{\mathbf{A}}\boldsymbol{\theta}_{\text{next}}^A-\mathbf{t}$, the *new* residual vector following the LLS-step, as explained for Eq.(8)(e). In spite of these distinctions and some others, we can recast VP, Algorithm TR-GPK, into the mold of Algorithm BA without using the full QR decomposition in Eq.(10); hence, a more economical implementation in both speed and memory, as shown next:

Algorithm BA-VP: *VP algorithm driven by* Algorithm BA.

(0) Initialize $\boldsymbol{\theta}_{\text{next}}^B=\boldsymbol{\theta}_{\text{now}}^B\leftarrow\boldsymbol{\theta}_{\text{init}}^B$ and $\Delta\boldsymbol{\theta}^B\leftarrow\mathbf{0}$.
(1) Evaluate $\mathbf{A}^T\mathbf{A}$ directly by stagewise backpropagation.
(2) Apply *Cholesky* to $\mathbf{A}^T\mathbf{A}$ [Step (1) of Algorithm BA].
(3) Get linear parameters by the LLS-step $\boldsymbol{\theta}_{\text{next}}^A$ (or $\boldsymbol{\Theta}_{\text{next}}^A$).
(4) Evaluate residuals: $\mathbf{r}_{\text{next}}=\widetilde{\mathbf{A}}\boldsymbol{\theta}_{\text{next}}^A-\mathbf{t}$.
(5) Follow Step (5) of Algorithm TR-GPK.
(6) Evaluate the n_B-length gradient vector $\mathbf{g}=\widetilde{\mathbf{B}}^T\mathbf{r}_{\text{next}}$, and the n_B-by-n_B (Gauss-Newton) Hessian $\mathbf{H}=\widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}}$.
(7) Compute the *Cauchy step* $\Delta\boldsymbol{\theta}_C^B\overset{\text{def}}{=}-\left(\frac{\mathbf{g}^T\mathbf{g}}{\mathbf{g}^T\mathbf{H}\mathbf{g}}\right)\mathbf{g}$.
(8) Determine a trust-region step $\Delta\boldsymbol{\theta}^B$ of length n_B by using a dogleg trust-region strategy of Eq.(13), for which, when $R_{tr}>\|\Delta\boldsymbol{\theta}_C^B\|$ holds, Steps (2) to (4) of Algorithm BA are performed to obtain $\Delta\boldsymbol{\theta}_{\text{GN}}^B$ by solving

$$\mathbf{X}\mathbf{X}^T\Delta\boldsymbol{\theta}_{\text{GN}}^B=-\mathbf{g}, \tag{20}$$

which is essentially Eq. (12), as detailed with Eq. (19).
(9) Set $\boldsymbol{\theta}_{\text{next}}^B=\boldsymbol{\theta}_{\text{now}}^B+\Delta\boldsymbol{\theta}^B$ [hence, $\mathbf{A}(\boldsymbol{\theta}_{\text{next}}^B)$ changes].

- Go back to Step (1). □ (*End*) □

Remarks: At Step (4), the new residual vector $\mathbf{r}_{\text{next}}$ is a reshaped vector form of $\mathbf{R}_{\text{next}}=\mathbf{A}\boldsymbol{\Theta}_{\text{next}}^A-\mathbf{T}$ at Step (4) of Algorithm TR-GPK for the VP functional $E_B(.)$ in Eq.(9). Compared with TR-GPK, a primary advantage of the posed BA-VP resides in speed because $\mathbf{A}^T\mathbf{A}$, $\widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}}$, and $\mathbf{g}$ can be evaluated efficiently by the so-called *second-order stagewise backpropagation* [13], [14] without evaluating $\mathbf{A}$ and $\widetilde{\mathbf{B}}$ explicitly; further extensions will be described in Sec.V-C.

V. Numerical results

We compare the performance of preceding algorithms in three numerical examples, in which we check f_A, the fraction of *linear* parameters $\boldsymbol{\theta}^A$, of a given model:

$$f_A\equiv\left(\frac{\#\text{ of linear parameters}}{\#\text{ of all parameters}}\right)\times 100=100\left(\frac{n_A}{n}\right)\ (\%). \tag{21}$$

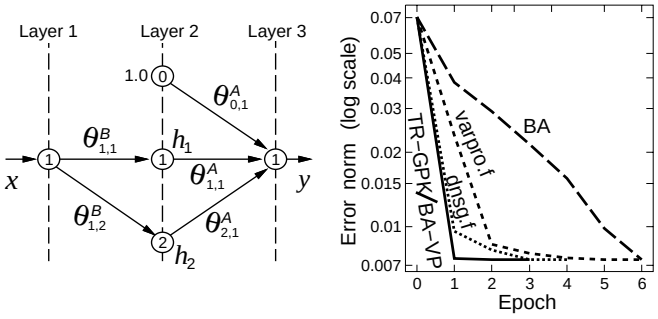

Fig. 2. A three-layered MLP representation (left figure) of the exponential-mixture benchmark problem (called OSBORNE1); and the learning curves (right) in terms of $\sqrt{\mathbf{r}^T\mathbf{r}}$ obtained by five methods: Algorithms TR-GPK, BA-VP, BA, varpro.f and dnsg.f. In the left figure, $\theta_{i,j}^B$ is a nonlinear (hidden) parameter between nodes i and j ($i=1;j=1,2$), and $\theta_{j,k}^A$ a linear (terminal) parameter between nodes j and k ($j=0,1,2$; $k=1$).

A. Three numerical examples

We first examine the *exponential-mixture* benchmark problem called "OSBORNE1" (e.g., see [9], [6], [7]); this is equivalent to optimizing a 1-2-1 multilayer perceptron (MLP) neural network model, as illustrated in Fig.2(left), with a single *hidden layer* (i.e., layer 2) that consists of two exponential-function nodes, $h_i(x)=\mathrm{e}^{-x}$ for $i=1,2$, plus one constant-output node [see "node 0" for $h_0\equiv 1$ in Fig.2(left)]. The final *linear* output y is generated at (terminal) layer 3 (in response to input x) as below

$$\begin{aligned}y &= h_0\theta_{0,1}^A+h_1\theta_{1,1}^A+h_2\theta_{2,1}^A \\ &= \theta_{0,1}^A+\exp\left(-x\theta_{1,1}^B\right)\theta_{1,1}^A+\exp\left(-x\theta_{1,2}^B\right)\theta_{2,1}^A.\end{aligned} \tag{22}$$

Here, $\boldsymbol{\theta}$, the five-parameter vector, of the 1-2-1 MLP separates into $\boldsymbol{\theta}^A=\left[\theta_{0,1}^A,\theta_{1,1}^A,\theta_{2,1}^A\right]^T$ (three "terminal" *linear* parameters between layers 2 and 3) and $\boldsymbol{\theta}^B=\left[\theta_{1,1}^B,\theta_{1,2}^B\right]^T$ (two "hidden" *nonlinear* parameters between layers 1 and 2); hence, $n=5$; $n_A=3$; $n_B=2$; and $f_A=60\%$. With this single-output ($F=1$) model, $n_A=FC_A=C_A(=3)$: In $\widetilde{\mathbf{A}}$ [see Eq.(5)], there is only one block $\mathbf{A}$, whose elements in the first column are all identical to $h_0=1$. We compare the performance between five algorithms: Algorithm BA as a full-functional approach of Group (1); and four VP (variable projection) algorithms of Group (4): Algorithms TR-GPK and BA-VP as well as two "standard" VP algorithms (in Fortran codes): varpro.f (available at www.netlib.org) and dnsg.f (available at www-out.bell-labs.com/project/PORT/). The VP-code varpro.f uses a classical Levenberg-Marquardt (LM) algorithm with a special *two-stage* QR decomposition (see pp.18–21 in [9]), and dnsg.f is a trust-region version that employs NL2SOL [6]. The stopping criteria was specified in squared residuals as $\mathbf{r}^T\mathbf{r}\le 0.5465\times 10^{-4}$ on given 33 data ($D=33$) (see p. 33 in [9]); the resulting learning behaviors are shown in Fig.2(right), where TR-GPK & BA-VP took three epochs; dnsg.f took four epochs; and varpro.f & BA took six epochs. Overall, VP algorithms worked faster than the full-functional approach BA, for which the linear parameters $\boldsymbol{\theta}^A$ are initialized by the LLS-step [e.g., see Step (3) of TR-GPK] common to VP methods; in this way,

this automatic initialization of $\boldsymbol{\theta}^A_{\text{init}}$ was used for all the five algorithms to set the exactly same starting error at Epoch 0 [see Fig.2(right)]. (Yet, the posed LLS-initialization may not be advantageous for standard MLP-learning; see later Table II.) We also confirmed that BA-VP has the behavior identical to TR-GPK, although these two are refreshingly different implementations of the same VP concept as detailed in previous sections: Notice therein that both TR-GPK and BA-VP are designed to take the Cauchy step initially at Epoch 1, which yielded a much larger error reduction than varpro.f that first took a classical LM step with the initial LM parameter $\mu = 1.0$ [see μ in Eq.(2)]. To attain a better behavior (similar to dnsg.f), the initial μ in varpro.f should be reduced (e.g., $\mu = 0.0244494$; see p.49 in [9]), but such a classical LM method (e.g., see trainlm.m [18]) has its limitations due to its own "μ-only" control; hence, we recommend the modern *trust-region* approach that controls the trust-region radius R_{tr} for robust *regularization* effects.

Next we test BA and BA-VP on the so-called 7-bit parity (7-dimensional XOR) two-class pattern classification problem ($D=2^7$ data) attacked by a 7-4-1 MLP (multi-layer perceptron) with only four tanh hidden nodes and a single *linear* output ($F=1$) at terminal layer 3, leading to $m=FD=128; n_A=C_A=5; n_B=32; n=n_A+n_B=37$. On each datum, in response to input x, the final output of the 7-4-1 MLP with 37 weight parameters is given below with $x_0=h_0=1$ like node 0 in Fig.2(left):

$$y=\sum_{j=0}^{4} h_j\theta^A_{j,1} \;=\; \theta^A_{0,1}+\sum_{j=1}^{4}\tanh\left(\sum_{i=0}^{7} x_i\theta^B_{i,j}\right)\theta^A_{j,1}. \qquad (23)$$

Here, Eq.(21) merely gives $f_A=13.5\%$. To increase f_A, we apply a so-called *weight-sharing* to $\boldsymbol{\theta}^B$ such that the jth set of seven parameters $\theta^B_{i,j}$ (for $i=1,...,7$) linking to hidden node j are merged into one parameter (for $j=1,...,4$). As a result, the 7-4-1 MLP can be transformed (see p.201 [14]) to a 1-4-1 MLP, for which the *single* input is the sum of all the seven binary inputs ($+1$ or -1) per datum, leading to $n_A=C_A=5; n_B=8; n=n_A+n_B=13; \frac{n_A}{n}\approx 0.385$; hence, f_A increased up to 38.5%. In any event, unlike the previous exponential mixture model, any algorithm could fail due to "hidden-node" *saturations* of sigmoidal-shaped tanh-functions, which work as an *implicit constraint*. With this and some other reasons, the parity is notoriously an *ill-conditioned* problem. We attempted to mitigate such hidden-node saturations simply by imposing an upper-bound $R^{\max}_{tr}$ on trust-region radius. Table I shows the results of ten experiments (averaged over 500 trials): three choices of $R^{\max}_{tr}$ for our two algorithms (BA-VP and BA) without weight-sharing and two choices for them with *weight-sharing* (denoted by "wts-share" in the first column). For simulation, 500 initial sets of parameters $\boldsymbol{\theta}_{\text{init}}$ ($\boldsymbol{\theta}^A_{\text{init}}$ and $\boldsymbol{\theta}^B_{\text{init}}$) were randomly generated uniformly in a small range $[-0.2, +0.2]$, which are common to all the ten cases; hence, the same starting points. In the last three columns, "Required epochs," unsuccessful trials were ruled out; here, each trial is deemed "successful" when a solution is discovered by our preset limit epoch (using epoch 1,000 for the four "weight-sharing" cases and epoch 5,000 for the other six cases). We say that a solution is found when the next inequality of y [see Eq.(23)] holds on all data with desired outputs t, ON($+1$) or Off(-1):

$$\text{minimum of } y \text{ on } t=\text{ON} > \text{maximum of } y \text{ on } t=\text{Off},$$

which implies that a *separating hyperplane* is found. Remarkably, VP (using $R^{\max}_{tr}=0.02$) achieved 100% success rate on 500 trials with weight-sharing.

TABLE I

TEN RESULTS IN THE 7-BIT PARITY PROBLEM USING AN MLP WITH FOUR HIDDEN NODES; EACH RESULT IS ON AVERAGE OVER 500 TRIALS.

Method		Solved by	Solved by	Required epochs		
Name	$R^{\max}_{tr}$	epoch 100	limit epoch	avg.	min.	max.
VP	1.0	12.8 %	73.4 %	396.5	44	4,157
	0.1	0.0 %	97.4 %	250.2	108	4,494
	0.02	0.0 %	99.6 %	319.8	193	2,494
wts-share	2.0	74.0 %	85.6 %	49.2	1	377
wts-share	0.02	90.8 %	100.0 %	60.4	12	972
BA	3.0	0.004 %	76.4 %	572.2	93	4,882
	1.0	0.008 %	83.0 %	316.0	98	4,831
	0.5	0.0 %	66.6 %	594.7	101	4,876
wts-share	2.0	69.2 %	94.6 %	99.4	20	732
wts-share	1.0	41.0 %	92.2 %	139.2	31	919

TABLE II

RESULTS IN A COLOR REPRODUCTION PROBLEM: FOR BOTH TRAINING AND TEST DATA, RMSE (ROOT MEAN SQUARED ERROR) WAS EVALUATED AT EPOCH 100 AND AVERAGED OVER 20 TRIALS.

Algorithm	Training RMSE	Test RMSE
TR-GPK (no delay)	0.0119	1.2548
TR-GPK (10-epoch delay)	0.0078	0.7011
BA	0.0044	0.1336

Finally, we compare Algorithms TR-GPK and BA in a nonlinear regression (color reproduction) problem, mappings from one color space to another under four different illuminant conditions, in which $F=3; D=100; m=300$ (plus 4400 test data). We employed a (five-input and three-output) 5-10-3 MLP with 10 tanh hidden nodes; hence, $n_A=33; C_A=11; n_B=40;\ n=n_A+n_B=73;\ f_A\approx 45.2\ \%$. All the 73 parameters ($\boldsymbol{\theta}^A_{\text{init}}$ and $\boldsymbol{\theta}^B_{\text{init}}$) were randomly initialized in a small range $[-0.3, +0.3]$; consequently, unlike the exponential fitting problem in Fig.2, the automatic initialization of $\boldsymbol{\theta}^A$ adversely affected the performance, leading to slow learning; see training RMSE (root mean squared errors) of Algorithm TR-GPK (no Delay) in Table II, where RMSE was obtained at epoch 100 and averaged over 20 trials. We then introduced an ad-hoc rule: *Delay taking the full LLS step for 10 epochs;* i.e., until epoch 10, only nonlinear parameters $\boldsymbol{\theta}^B$ (with randomly-initialized $\boldsymbol{\theta}^A_{\text{init}}$ held fixed) were optimized, which relatively improved learning; see TR-GPK (10-epoch delay) in Table II.

B. Behaviors of VP *observed in three examples*

VP algorithms would prove very useful especially when nonlinear parameters $\boldsymbol{\theta}^B$ need to be merely fine-tuned provided that their good initial guesses are available, and/or

when f_A in Eq.(21) is relatively high. The first exponential-fit benchmark is really such a case since $f_A = 60\%$, and the initial values $\boldsymbol{\theta}^B_{\text{init}} = [0.01,\ 0.02]^T$ were fairly close to the desired ones $\boldsymbol{\theta}^B_* = [0.0129,\ 0.0221]^T$.

In MLP-learning, however, the initial parameters are usually randomly generated uniformly in a small range. Given a poor starting point $\boldsymbol{\theta}^B_{\text{init}}$ in the nonlinear parameter space, VP's automatic initialization of linear parameters ($\boldsymbol{\theta}^A$), given by the linear-least-squares (LLS) step [see Step (3) in Algorithms TR-GPK and BA-VP (with $\boldsymbol{\theta}^A_{\text{init}}$ ignored)], tends to yield much greater $\boldsymbol{\theta}^A_{\text{next}}$ in magnitude than the full functional approaches such as Algorithm BA. Therefore, even if the initial starting points ($\boldsymbol{\theta}^A_{\text{init}}$ and $\boldsymbol{\theta}^B_{\text{init}}$) are the exactly same for VP and BA, they typically find a "different" *solution* point, and VP may not always work faster than BA. In the last color application, Table II clearly shows such slow learning (i.e., slow in reducing training errors). In the 7-bit parity problem, a careful observation also reveals slow learning even in a weight-sharing case; there were 106 trials (out of 500), in which VP (with $R^{\max}_{tr} = 0.02$ in the reduced parameter space) worked slower than BA (with $R^{\max}_{tr} = 2.0$).

We employed weight-sharing to increase f_A in Eq.(21), which most likely improves the performance of VP; overall, Table I confirms that the aforementioned model transformation via weight-sharing renders the problem "VP-favorite." We may further increase f_A by omitting constant-output nodes (or node 0) from a weight-shared 1-4-1 MLP; then, f_A goes up to 50% (with four linear parameters and four nonlinear ones). If more knowledge on parity is applied, VP's first automatic LLS-step could always find the solution; that is, observe only eight distinct data (essentially four by symmetry) among 128 data, and thus use those four alone for VP. Consequently, the first LLS-step reduces to a four-by-four linear "square" system solving, certainly yielding a solution as long as the four nonlinear parameters are initialized differently (in magnitude owing to randomness) as usual. This quick solution may not be fulfilled by any full-functional approach; in this sense, parity is very special.

Furthermore, Table II shows that VP led to poor generalization in terms of *test* errors; probably, this is also due to the aforementioned LLS-initialization of linear parameters $\boldsymbol{\theta}^A$. Here is a possible explanation: Suppose that two MLPs are optimized that attained the same error level in training data: one obtained by VP, and another by BA (full-functional approach). As mentioned above, the resulting $\boldsymbol{\theta}^A$ in the former MLP obtained by VP tends to be larger in modulus than that in the latter MLP by BA. In other words, the variance of linear parameters $\boldsymbol{\theta}^A$ is higher in the former MLP. For any previously-unseen datum from the test data set, the behavior of the former MLP is more unexpected due to higher variance of $\boldsymbol{\theta}^A$. This is consistent with the effect of weight-decay terms that may be introduced to penalize large weights in some machine learning applications.

C. Discussions for improvements and two examples

In the nonlinear least squares literature, there are two conflicting arguments: "The VP procedure not only reduces the dimension of the parameter space but also results in a *better-conditioned* problem (see [10])," versus "VP functional $E_B(.)$ in Eq.(9) may increase *nonlinearity* compared with the original full functional $E(.)$ in Eq.(7) (see page 299 in [5])." In [10], a better-conditioned argument is somehow overemphasized, stating "The same optimization method applied to the original and reduced problems will *always converge faster* for the latter." This is not always true in MLP-learning, as discussed in Sec.V-B for two applications. Furthermore, when the results of classical (small) benchmark problems reported in [6], [7] were examined carefully, there already existed several separable problems, for which NL2SOL [6], a "standard" *full functional approach* (FFA), worked better than VP.

Sec.V-B attributes VP's slow learning to the issue of large $\boldsymbol{\theta}^A$. Here, we argue in another perspective. In MLP-learning, it is frequently encountered that $\mathbf{J}$ is (nearly) rank-deficient; hence, $\mathbf{J}^T\mathbf{J}$ is (nearly) singular: This rank-deficient situation often induces plateaus in learning (e.g., see [16]). Note in double precision that if $\mathsf{cond}(\mathbf{J})$, the condition number of $\mathbf{J}$, is $O(10^8)$, the order of 10^8, or worse, then $\mathbf{J}^T\mathbf{J}$ is *numerically singular*; hence, QR on $\mathbf{J}$ is preferred to Cholesky on $\mathbf{J}^T\mathbf{J}$ when we solve the Jacobian system (in Sec.II). Yet, this criticism against normal-equation approaches may not always apply to the full Hessian $\mathbf{H}$ based algorithms; when $\mathbf{J}^T\mathbf{J}$ is singular, $\mathbf{H}$ likely becomes *indefinite* (with both positive and negative eigenvalues) because the residual Hessian $\mathbf{S} \equiv \sum_i^m r_i \nabla^2 r_i$ inevitably has an indefinite structure. To see this better, using Eq.(5), we put the Hessian matrix, $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S}$, in a partitioned form below

$$\left[\begin{array}{c|c} \widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}} & \widetilde{\mathbf{A}}^T\widetilde{\mathbf{B}} \\ \hline \widetilde{\mathbf{B}}^T\widetilde{\mathbf{A}} & \widetilde{\mathbf{B}}^T\widetilde{\mathbf{B}} \end{array}\right] + \left[\begin{array}{c|c} \mathbf{0} & \mathbf{S}^T_{1,2} \\ \hline \mathbf{S}_{1,2} & \mathbf{S}_{1,1} \end{array}\right] \stackrel{\text{def}}{=} \left[\begin{array}{c|c} \widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}} & \mathbf{H}^T_{1,2} \\ \hline \mathbf{H}_{1,2} & \mathbf{H}_{1,1} \end{array}\right]. \quad (24)$$

All the preceding algorithms omit $\mathbf{S}$ from $\mathbf{H}$ by conforming to the original VP, and then solve the subproblem in Eq.(2) with $\mathbf{H} \approx \mathbf{J}^T\mathbf{J}$. Now, a question arises: If $\mathbf{H}$ is (strongly) indefinite, and/or if $\mathbf{S}$ is dominant in $\mathbf{H}$, is it a good idea to omit $\mathbf{S}$? The answer is no since the trust-region theory has thrived on negative curvature. We should modify VP to include $\mathbf{S}$. A quick remedy is made possible in the framework of Algorithm BA, where $\mathbf{V}$ in Step (3) should be $\mathbf{W}$ below as the projected Hessian (in the reduced nonlinear parameter space); namely, the Schur complement $\mathbf{W}$ below of $\widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}}$ in $\mathbf{H}$ when block $\widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}}$ is positive definite (non-singular) even if $\mathbf{J}^T\mathbf{J}$ is *singular*,

$$\mathbf{W} = \mathbf{H}_{1,1} - \mathbf{H}_{1,2}\left[\widetilde{\mathbf{A}}^T\widetilde{\mathbf{A}}\right]^{-1}\mathbf{H}^T_{1,2}. \quad (25)$$

In consequence, if the projected Hessian $\mathbf{W}$ is indefinite, we may exploit negative curvature to update $\boldsymbol{\theta}^B$ alone in the reduced parameter space in the same spirit as the original VP.

Example (1): A four-weight 1-2-1 MLP with two exp(x)-hidden-node functions, for which "Node 0" in layer 2 is omitted from Fig.2(left), using five data, five pairs of input x and target t: $\mathsf{Data}(\mathbf{x};\mathbf{t}) = \{(-2;3),(-1;1),(0,2),(1,1),(2,3)\}$. Note in this example that no hidden-node saturations occur. Consider a point: $\boldsymbol{\theta}^T_{\text{now}} = [\boldsymbol{\theta}^{A^T}_{\text{now}} | \boldsymbol{\theta}^{B^T}_{\text{now}}] = [-1, 1 | 0.1, 0.11]$; VP first finds $\boldsymbol{\theta}^A_{\text{next}}$, the new two linear parameters, by the linear

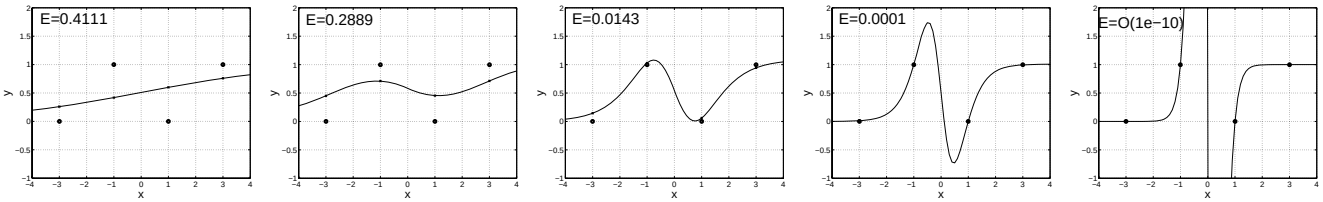

Fig. 3. For Example (2), E in Eq.(1) can be made arbitrary small along a descent direction of negative curvature obtainable from $\mathbf{W}$ in Eq.(25).

TABLE III
THE CONDITION NUMBERS AND EIGENVALUES OF SEVERAL MATRICES INCLUDING $\mathbf{J} \equiv [\mathbf{A}|\mathbf{B}_1]$ IN EXAMPLE (1) BEFORE AND AFTER THE LLS (LINEAR LEAST SQUARES) STEP.

	Before LLS-step	After LLS-step
$\mathsf{cond}(\mathbf{H})$	$O(10^3)$	$O(10^9)$
$\mathsf{cond}(\mathbf{J})$	$O(10^7)$	$O(10^8)$
$\mathsf{cond}(\mathbf{W})$	$O(10^3)$	$O(1)$
$\min\{\mathsf{eig}(\mathbf{H})\}$	-17.9	-7.22×10^{-7}
$\max\{\mathsf{eig}(\mathbf{H})\}$	40.1	4.53×10^3
$\min/\max\{\mathsf{eig}(\mathbf{W})\}$	-1.31×10^4 / 1.97	-6.05 / 4.17

least squares (LLS)-step as $\boldsymbol{\theta}^{A^T}_{\text{next}} = [\ 15.5073, -13.5055\ ]$. We then evaluate the condition numbers and eigenvalues of several matrices of our interest before and after the LLS-step. Table III summarizes the results: Here, $\mathbf{J} = [\mathbf{A}|\mathbf{B}_1]$ from Eq.(5), and $\mathbf{W}$ is given by Eq.(25). The LLS-step won't change $\boldsymbol{\theta}^B$; hence, $\mathbf{A}$ remains the same; so does $\mathsf{cond}(\mathbf{A}) = O(10^2)$. Intriguingly, $\mathsf{cond}(\mathbf{B}_1) = O(10^2)$ before and after the LLS-step although the entries in $\mathbf{B}_1$ change greatly. By contrast, $\mathbf{J}$ is more ill-conditioned in any event (and $\mathbf{J}^T\mathbf{J}$ is *numerically singular* after LLS). This is a better-conditioned claim behind VP using reduced column spaces. After the first LLS-step, the Hessian $\mathbf{H}$ becomes much more ill-conditioned and much less indefinite; in this way, the negative curvature information is lost in a sense. Yet, $\mathbf{W}$ becomes strongly indefinite; hence, worth exploiting it.

Example (2): A four-weight 1-2-1 MLP with two sigmoidal logistic hidden-node functions $h_i(x) = \frac{1}{1+e^{-x}}$ for $i = 1, 2$ on four data: $\mathsf{Data}(\mathbf{x}; \mathbf{t}) \equiv \{(-3; 0), (-1; 1), (1; 0), (3; 1)\}$. At a point $\boldsymbol{\theta}^T = [\boldsymbol{\theta}^{A^T} | \boldsymbol{\theta}^{B^T}] = [0.7645, 0.2539 | 0.3571, 0.3572]$, where $E \approx 0.4111$, we had the following: $\mathbf{J}^T\mathbf{J}$ was singular, and $\mathbf{H}$ was slightly indefinite with $\min\{\mathsf{eig}(\mathbf{H})\} = -1.8453 \times 10^{-9}$ and $\mathsf{cond}(\mathbf{H}) = O(10^9)$, whereas $\mathbf{W}$ in Eq.(25) was indefinite with $\min\{\mathsf{eig}(\mathbf{W})\} = -0.3189$ and $\mathsf{cond}(\mathbf{W}) = O(1)$. Then, E can be made small (*no plateau*) along a negative-curvature direction $\Delta\boldsymbol{\theta}^B$ (while $\boldsymbol{\theta}^A$ updated by LLS steps); see Fig.3.

VI. CONCLUSION

We have described in detail the trust-region variable projection (VP) method in the framework of block-arrow least squares algorithm (BA-VP). A major breakthrough is that BA-VP can work with the full Hessian $\mathbf{H} (= \mathbf{J}^T\mathbf{J} + \mathbf{S})$, where $\mathbf{S}$ is important to efficiency in *large-residual* problems [5], in conjunction with a so-called *second-order stagewise backpropagation* [13] that can evaluate $\mathbf{J}^T\mathbf{J}$ and $\mathbf{H}$ at the essentially same cost [14]. Furthermore, in MLP-learning, $\mathbf{J}^T\mathbf{J}$ tends to be numerically singular, rendering $\mathbf{H}$ *indefinite* [16]. The resulting trust-region BA-VP exploits negative curvature when it arises. In this way, VP would be improved significantly.

ACKNOWLEDGMENTS

The authors would like to thank John E. Dennis, Jr. (Rice University) for valuable discussions via e-mail in the past. A special thanks must go to Gene Golub and Victor Pereyra for a personal meeting at Stanford on August 4, 2006.

REFERENCES

[1] F. Biegler-Konig and F. Barmann. A learning algorithm for multilayer neural networks based on linear least squares problems. *Neural Networks*, 6:127–131, 1993.
[2] Vladimir Cherkassky and Filip Mulier. *Learning from data: concepts, theory, and methods*. John Wiley & Sons, Inc., New York, 1998.
[3] Andrew R. Conn, Nicholas IM Gould, and Philippe L. Toint. *Trust-Region Methods*. SIAM, 2000.
[4] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.
[5] John E. Dennis., Jr. Nonlinear least squares and equations. In D. Jacobs, editor, *State of the art in numerical analysis*, pages 269–312. Academic Press, London, 1977.
[6] John E. Dennis, David M. Gay, and Roy E. Welsch. An adaptive nonlinear least-squares algorithm. *ACM Transactions on Mathematical Software*, 7(3):348–368, September 1981.
[7] David Gay and Linda Kaufman. Tradeoffs in algorithms for separable nonlinear least squares. In *Proc. of the 13th World Congress on Computational and Applied Mathematics*, pp. 157–158, December 1990.
[8] P. E. Gill, W. Murray, and M. H. Wright. *Practical optimization*. Academic Press, New York, 1981.
[9] Gene H. Golub and Victor Pereyra. The differentiation of pseudoinverses and nonlinear least squares problems whose variables separate. Computer Science Technical Report STAN-CS-72-261, Stanford University, 1972.
[10] Gene H. Golub and Victor Pereyra. Separable nonlinear least squares: the variable projection method and its applications. *Inverse Problems*, 19:R1–R26, 2003.
[11] Gene H. Golub and Randall J. LeVeque. Extensions and uses of the variable projection algorithm for solving nonlinear least squares problems. Computer Science Technical Report SU-326, Stanford University, 1978.
[12] Tomas Hrycej. *Modular Learning in Neural Networks: A Modularized Approach to Neural Network Classification*. John Wiley & Sons, Inc., New York, 1992.
[13] Eiji Mizutani, Stuart E. Dreyfus, and James W. Demmel. Second-order backpropagation algorithms for a stagewise-partitioned separable Hessian matrix. In *Proc. of the International Joint Conference on Neural Networks (INNS-IEEE IJCNN'05)*, Montreal Quebec, CANADA, 2005.
[14] Eiji Mizutani and Stuart Dreyfus. Second-order stagewise backpropagation for Hessian-matrix analyses and investigation of negative curvature. *Neural Networks*, vol.21 (issues 2–3): pp.193-203, 2008. (See its Corrigendum in vol.21, issue 9, page 1418).
[15] Eiji Mizutani and James W. Demmel. On structure-exploiting trust-region regularized nonlinear least squares algorithms for neural-network learning. *Neural Networks*, Elsevier Science, vol.16: pp.745-753, 2003.
[16] Eiji Mizutani and Stuart Dreyfus. An analysis on negative curvature induced by singularity in multi-layer neural-network learning. In Advances in Neural Information Processing Systems, vol. 23, MIT Press, 2010.
[17] MATLAB Fuzzy Logic Toolbox, Mathworks, Inc.
[18] MATLAB Neural Network Toolbox, Mathworks, Inc.