On derivation of stagewise second-order backpropagation by invariant imbedding for multi-stage neural-network learning

Eiji Mizutani and Stuart Dreyfus

***Abstract*— We present a simple, intuitive argument based on "invariant imbedding" in the spirit of dynamic programming to derive a stagewise second-order backpropagation (BP) algorithm. The method evaluates the Hessian matrix of a general objective function efficiently by exploiting the multi-stage structure embedded in a given neural-network model such as a multilayer perceptron (MLP). In consequence, for instance, our stagewise BP can compute the full Hessian matrix "faster" than the standard method that evaluates the Gauss-Newton Hessian matrix alone by rank updates in nonlinear least squares learning. Through our derivation, we also show how the procedure serves to develop advanced learning algorithms; in particular, we explain how the introduction of "stage costs" leads to alternative systematic implementations of multi-task learning and weight decay.**

I. Introduction

Learning with multilayer-perceptron (MLP) neural networks is a multiple N-stage decision-making (or discrete-time optimal control) problem [1]. The optimal-control theory available in control engineering dictates to us a variety of elaborate learning schemes in the context of N-layered MLP-learning with P_s nodes at layer s ($s = 1, ..., N$). For concreteness, we consider the following objective function J, known as the **problem of Bolza** in classical optimal control:

$$J(\boldsymbol{\theta}) = \sum_{s=1}^{N-1} L^s(\mathbf{y}^s, \boldsymbol{\theta}^s) + E(\mathbf{y}^N), \quad \text{(with } \mathbf{y}^1 \text{ given)} \tag{1}$$

where $E(.)$ is the terminal cost at stage N that depends only on $\mathbf{y}^N$, and $L^s(.,.)$ is the cost at stage s, which generally depends on the P_s-dimensional **state** vector $\mathbf{y}^s$ of "after-node" outputs and the n_s-dimensional **control** vector $\boldsymbol{\theta}^s$ of weight parameters, which includes threshold parameters; hence, the length $n_s \equiv (1 + P_s)P_{s+1}$. Introduction of such $L^s(.,.)$ at intermediate stages (i.e., at *hidden layers*; $1 < s < N$) leads to more advanced learning schemes (to be described in detail later). Both $E(.)$ and $L^s(.,.)$ could be chosen as the sum of squared residuals over D data between node outputs $\mathbf{y}^s$ and target outputs $\mathbf{t}^s$ of length P_s on each (training) datum d (for $d=1, ..., D$. That is, $E(.)$ is given by

$$\begin{aligned} E(\mathbf{y}^N) &= \frac{1}{2}\sum_{d=1}^{D}\sum_{k=1}^{P_N}(y^N_{k,d} - t^N_{k,d})^2 \\ &= \frac{1}{2}\sum_{d=1}^{D}\|\mathbf{y}^N_d - \mathbf{t}^N_d\|_2^2 = \frac{1}{2}\sum_{d=1}^{D}\mathbf{r}^{N\mathsf{T}}_d\,\mathbf{r}^N_d, \end{aligned} \tag{2}$$

Eiji Mizutani is with the Department of Computer Science, Tsing Hua University, Hsinchu, 300, TAIWAN (email: eiji@cs.nthu.edu.tw).

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

and the stage cost $L^s(.,.)$ ($1 < s < N$) can be defined similarly as

$$\begin{aligned} L^s(\mathbf{y}^s, \boldsymbol{\theta}^s) &= \frac{1}{2}\sum_{d=1}^{D}\sum_{k=1}^{P_s}(y^s_{k,d} - t^s_{k,d})^2 \\ &= \frac{1}{2}\sum_{d=1}^{D}\|\mathbf{y}^s_d - \mathbf{t}^s_d\|_2^2 = \frac{1}{2}\sum_{d=1}^{D}\mathbf{r}^{s\mathsf{T}}_d\,\mathbf{r}^s_d, \end{aligned} \tag{3}$$

where $\mathbf{r}^s_d$ is the P_s-length vector of residuals on datum d evaluated at intermediate layer s ($1 < s < N$). Although the above form of L^s is a natural extension of terminal cost $E(.)$ defined in Equation (2), this is a very special case in the theory of optimal control because, in MLP-learning, the posed form of L^s becomes independent of $\boldsymbol{\theta}^s$; namely $L^s(\mathbf{y}^s, -)$, which is what we call *hidden-node teaching* [2] to be discussed in Section III-A.

For minimizing J in Equation (1) (often with early stopping), the widely-employed first-order *stagewise* backpropagation (BP) algorithm can be summarized as follows: Given fixed weights and thresholds, the *forward pass* per datum (with subscript d omitted) evaluates the "after-node" outputs [see Figure 1(a)] in three steps below

$$\begin{cases} \text{(1) Forward pass from stage } s \text{ to stage } s+1 \\ \text{ to evaluate the "before-node" net inputs:} \\ \underbrace{\mathbf{x}^{s+1}}_{P_{s+1}\times 1} = \underbrace{\boldsymbol{\Theta}^s}_{P_{s+1}\times(1+P_s)}\ \underbrace{\mathbf{y}^s_+}_{(1+P_s)\times 1} \\ \text{(2) After-node output evaluation at stage } s\text{:} \\ \underbrace{\mathbf{y}^s}_{P_s\times 1} = \mathbf{f}^s(\mathbf{x}^s) \Longleftrightarrow \begin{cases} y^s_j = f^s_j(x^s_j)\,, \\ \text{for } j = 1, \ldots, P_s \end{cases} \\ \text{(3) Residual (stage cost) evaluation at stage } s\text{:} \\ \underbrace{\mathbf{r}^s}_{P_s\times 1} = \mathbf{y}^s - \mathbf{t}^s. \quad \text{[when Eq.(3) is used]} \end{cases} \tag{4}$$

Initially at stage 1 (namely, $s = 1$ at the first input layer), $\mathbf{y}^1 = \mathbf{x}^1$ (given). Subsequently, at stage $s\,(>1)$, a vector $\mathbf{y}^s$ of P_s after-node outputs are generated [see step (2)] by $f^s_j(.)$, a certain nonlinear node output function (e.g., tanh). Notice that subscript "+" on $\mathbf{y}^s_+$ in step (1) implies inclusion of a *constant* output of a bias node (node 0): This implies that $\mathbf{y}^s_+$, the $(1 + P_s)$-dimensional vector of "after-node" outputs at stage s, consists of the constant output ($y^s_0 = 1.0$) of bias-node 0 as the first element, and thus $\boldsymbol{\Theta}^s$, a P_{s+1}-by-$(1+P_s)$ matrix of parameters between adjacent layers s and $s+1$, includes the P_{s+1}-length vector $\boldsymbol{\theta}^s_{0,\cdot}$ of threshold parameters

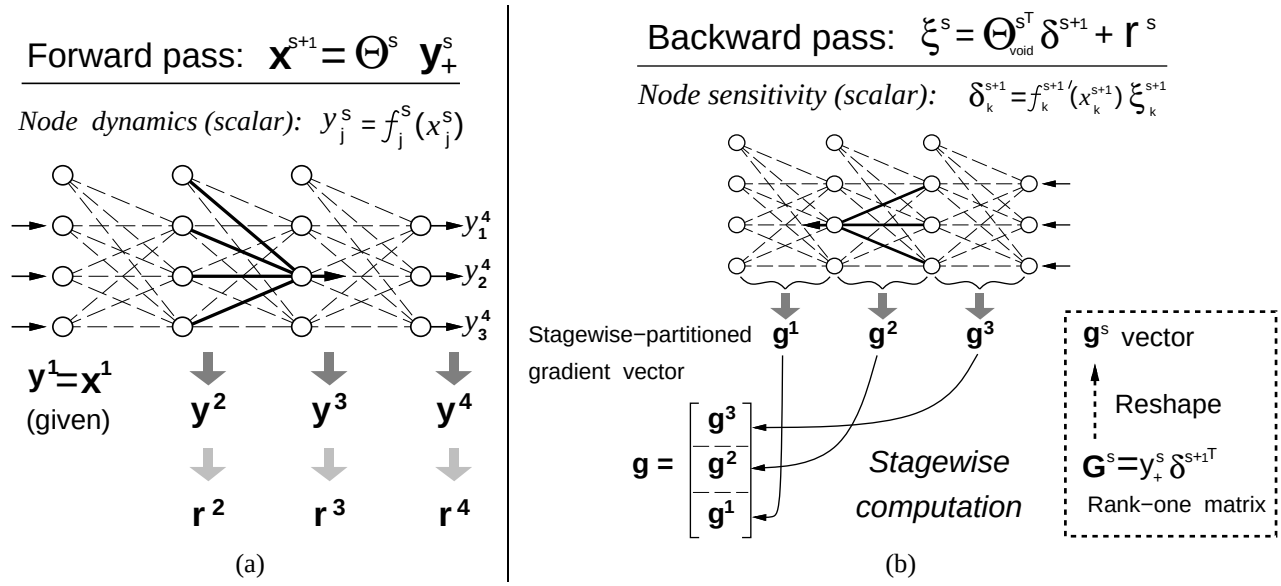


Fig. 1. First-order stagewise backpropagation in two-hidden-layer MLP-learning for the objective function J defined in Equation (1): (a) By forward pass, the "after-node" outputs $\mathbf{y}^s$ are propagated to the next stage as the "before-node" net-inputs $\mathbf{x}^{s+1}$; and (b) By backward pass, the "before-node" sensitivities $\boldsymbol{\delta}^{s+1}$ are propagated back to the previous stage as the "after-node" sensitivities $\boldsymbol{\xi}^s$. During the propagation process, the "hidden residual" vector $\mathbf{r}^s$ in (a) and the gradient vector $\mathbf{g}^s$ in (b) are also evaluated at each stage s in a stagewise fashion. Note in (a) that $\mathbf{r}^s = \mathbf{y}^s - \mathbf{t}^s$ (with $\mathbf{t}^s$, the vector of desired "hidden" outputs, for **hidden-node teaching** in Sec.III-A).

(between bias node 0 at layer s and P_{s+1} nodes at layer s+1) in the first column, as shown next:

$$\underbrace{\mathbf{y}^s_+}_{(1+P_s)\times 1} = \begin{bmatrix}1.0\\ \hline \mathbf{y}^s\end{bmatrix}; \quad \underbrace{\boldsymbol{\Theta}^s}_{P_{s+1}\times(1+P_s)} = \left[\begin{array}{c|c}\underbrace{\boldsymbol{\theta}^s_{0,.}}_{P_{s+1}\times 1} & \underbrace{\boldsymbol{\Theta}^s_{\text{void}}}_{P_{s+1}\times P_s}\end{array}\right], \quad (5)$$

where $\boldsymbol{\Theta}^s_{\text{void}}$ is a P_{s+1}-by-P_s matrix of parameters (with no thresholds) (to be used in backward pass). It should be clear that the kth element of $\mathbf{x}^{s+1}$, the P_{s+1} vector of *before-node net-inputs* at layer $s+1$, is given (with $y^s_0 = 1.0$) by

$$x^{s+1}_k = \sum_{j=0}^{P_s} y^s_j \theta^s_{j,k} \quad \text{for } k = 1, \ldots, P_{s+1}, \quad (6)$$

where $\theta^s_{j,k}$ is a control parameter (or weight) between node j at layer s and node k at the next layer.

Likewise, the backward pass [see Figure 1(b)] evaluates the *node sensitivities* (defined below) and obtains the gradient vector of J [defined in Eq.(1)] at each stage:

$$\begin{cases} \text{(1) Before-node sensitivity evaluation at stage } s+1\text{:} \\ \delta^{s+1}_k = f^{s+1\prime}_k\big(x^{s+1}_k\big)\,\xi^{s+1}_k, \text{ for } k = 1, \ldots, P_{s+1}. \\ \text{(2) Gradient evaluation for } \boldsymbol{\Theta}^s\text{:} \\ \underbrace{\mathbf{G}^s}_{(1+P_s)\times P_{s+1}} = \mathbf{y}^s_+ \boldsymbol{\delta}^{s+1\mathsf{T}} \Longleftrightarrow \begin{cases} g^s_{j,k} = y^s_j \delta^{s+1}_k, \\ \text{for } j = 0, \ldots, P_s; \\ \text{and } k = 1, \ldots, P_{s+1} \end{cases} \\ \text{(3) Backward pass from stage } s+1 \text{ down to } s\text{:} \\ \underbrace{\boldsymbol{\xi}^s}_{P_s\times 1} = \underbrace{\boldsymbol{\Theta}^{s\mathsf{T}}_{\text{void}}}_{P_s\times P_{s+1}} \underbrace{\boldsymbol{\delta}^{s+1}}_{P_{s+1}\times 1} + \underbrace{\mathbf{r}^s}_{P_s\times 1}, \end{cases} \quad (7)$$

TABLE I

A COMPARISON BETWEEN NEURAL-NETWORK AND OPTIMAL-CONTROL CONVENTIONS.

Notation	Optimal Control	Neural Networks
$\boldsymbol{\theta}$	decision / control	weight parameters
$\mathbf{y}=\mathbf{f}(\mathbf{x})$	state	"after-node" output
$\mathbf{x}$	N/A	"before-node" net input
s	stage	layer
$\boldsymbol{\xi}^s = \dfrac{\partial J}{\partial \mathbf{y}^s}$	costate adjoint variable influence function Lagrange multiplier	N/A
$\boldsymbol{\delta}^s = \dfrac{\partial J}{\partial \mathbf{x}^s}$	N/A	delta

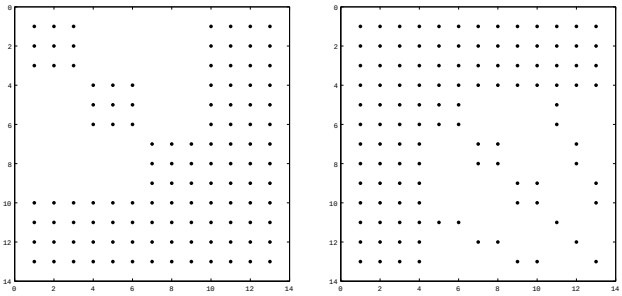
where the two forms of **node sensitivities** are defined as:

$$\begin{cases} \bullet\ \textbf{After-node sensitivity} \text{ at stage } s\text{:} \quad \boldsymbol{\xi}^s \overset{\text{def}}{=} \dfrac{\partial J}{\partial \mathbf{y}^s}, \\ \bullet\ \textbf{Before-node sensitivity} \text{ at stage } s\text{:} \quad \boldsymbol{\delta}^s \overset{\text{def}}{=} \dfrac{\partial J}{\partial \mathbf{x}^s}. \end{cases}$$

The procedure begins at terminal stage N, where $\boldsymbol{\xi}^N = \mathbf{r}^N$, the terminal residual vector [see Eq.(2)] when $s+1 = N$. Steps (1) and (3) in Equation (7) can be combined as

$$\boldsymbol{\delta}^s = \left[\frac{\partial \mathbf{y}^s}{\partial \mathbf{x}^s}\right]^{\mathsf{T}} \boldsymbol{\xi}^s = \left[\frac{\partial \mathbf{y}^s}{\partial \mathbf{x}^s}\right]^{\mathsf{T}} \left\{\boldsymbol{\Theta}^{s\mathsf{T}}_{\text{void}} \boldsymbol{\delta}^{s+1} + \mathbf{r}^s\right\}. \quad (8)$$

Table I compares the terminologies used in optimal-control and neural-network literatures. In the best-known BP formulation due to Rumelhart et al. [3], $\mathbf{x}^s$, the vector of "before-node" *net inputs* [see Equation (6)], is treated as the *state* vector, whereas in optimal control, $\mathbf{y}^s$, the vector of "after-node" *outputs*, is chosen as the *state* vector.



(a) (b)

Fig. 2. Two sparse patterns of the Hessian matrix in learning with a three-layered ($N=3$) 1-2-3 MLP with a single input ($P_1=1$), two hidden nodes ($P_2=2$), and three terminal outputs ($F\equiv P_3=3$); hence, 13 parameters in total including threshold parameters: (a) The desired **block-arrow** Hessian matrix, whose arrowhead should point downward to the right (see [4], [5]), with $F(=3)$ diagonal blocks; (b) A Hessian matrix with a complex sparse pattern, which is hard to exploit, obtained by the NETLAB (MATLAB-based software) (see mlphess.m at http://www.ncrg.aston.ac.uk/netlab/). For large-scale optimization, it is not recommendable to approximate the inverse of the Hessian because it always becomes *dense*. The posed sparsity can be exploited with Krylov subspace methods for optimization [4].

II. Second-Order Stagewise Backpropagation

To exploit the stagewise structure embedded in a given multi-stage MLP model, both the Hessian matrix and the gradient vector are stagewise-partitioned, being evaluated block by block. In two-hidden-layer ($N=4$) MLP-learning, for instance, we consider the following partitioned form:

Hessian matrix **gradient vector**

$$\underbrace{\mathbf{H}}_{n\times n}=\left[\begin{array}{c|c|c} \underbrace{\mathbf{H}^{3,3}}_{n_3\times n_3} & \mathbf{H}^{2,3^T} & \mathbf{H}^{1,3^T} \\ \hline \underbrace{\mathbf{H}^{2,3}}_{n_2\times n_3} & \mathbf{H}^{2,2} & \mathbf{H}^{1,2^T} \\ \hline \underbrace{\mathbf{H}^{1,3}}_{n_1\times n_3} & \mathbf{H}^{1,2} & \mathbf{H}^{1,1} \end{array}\right];\quad \underbrace{\mathbf{g}}_{n\times 1}=\left[\begin{array}{c} \underbrace{\mathbf{g}^{3}}_{n_3\times 1} \\ \hline \underbrace{\mathbf{g}^{2}}_{n_2\times 1} \\ \hline \underbrace{\mathbf{g}^{1}}_{n_1\times 1} \end{array}\right].$$

Here, we want to evaluate (the lower triangular half of) three diagonal Hessian blocks $\mathbf{H}^{s,s}$ (for $s=1,2,3$) and three off-diagonal Hessian blocks $\mathbf{H}^{s,t}$ (for $s<t; s=1,2; t=2,3$) because $\mathbf{H}$ is *symmetric*. Note in multiple-output ($P_N>1$) problems, the Hessian matrix $\mathbf{H}$ has a so-called **block-arrow** form, as shown in Figure 2(a). Our *stagewise second-order BP* efficiently organizes the computed Hessian elements into such a block-arrow form with its arrowhead pointing downward to the right so as to exploit the posed *sparsity* [4].

We shall derive *stagewise* second-order backpropagation (BP) algorithm [5] that evaluates the above stagewise-partitioned Hessian matrix $\mathbf{H}$ of J [in Eq.(1)] using so-called *invariant imbedding* recurrence relations (i.e., dynamic-programming like recurrence relations with no optimization aspect). To this end, we define two "(non-optimal) cost-to-go value functions" and their "recurrence relations," as well as the "boundary condition" in the following way:

Value functions:

$$T^s(\mathbf{x}^s,\boldsymbol{\theta}^s) \stackrel{\text{def}}{=} \text{(non-optimal) cost-to-go, starting at state } \mathbf{x}^s \text{ at stage } s, \text{ using a guessed policy } \boldsymbol{\theta}^s. \quad (9)$$

$$V^s(\mathbf{x}^s,\boldsymbol{\theta}^s,\boldsymbol{\theta}^t) \stackrel{\text{def}}{=} \text{(non-optimal) cost-to-go, starting at state } \mathbf{x}^s \text{ at stage } s, \text{ using guessed policies } \boldsymbol{\theta}^s \text{ and } \boldsymbol{\theta}^t \text{ for stages } s \text{ and } t\ (s<t). \quad (10)$$

By definition in Equation (9), the cost-to-go function T^s depends on all future controls (i.e., decision parameters) starting at stage s; however, only its dependence on the control at the current stage s matters, and therefore, all the other subsequent controls $\boldsymbol{\theta}^t$ $(s<t)$ are suppressed in the argument of T^s, which can be used to evaluate *diagonal* Hessian blocks $\mathbf{H}^{s,s}$ as well as the gradient vector $\mathbf{g}^s$. In order to get *off-diagonal* Hessian blocks $\mathbf{H}^{s,t}$, we need another value function V^s, for which we are interested in controls at two different stages s and t $(s<t)$; that is, we define V^s in Equation (10) as a function of two controls at those two different stages that matter, suppressing the dependence on controls at all other future stages.

Recurrence relations:

• For *adjacent* stages s and t: $\quad t=s+1,$

$$T^s(\mathbf{x}^s,\boldsymbol{\theta}^s)=L^s(\mathbf{x}^s,\boldsymbol{\theta}^s)+T^{s+1}(\mathbf{x}^{s+1},\boldsymbol{\theta}^{s+1}), \quad (11)$$

and

$$V^s(\mathbf{x}^s,\boldsymbol{\theta}^s,\boldsymbol{\theta}^{s+1})=L^s(\mathbf{x}^s,\boldsymbol{\theta}^s)+T^{s+1}(\mathbf{x}^{s+1},\boldsymbol{\theta}^{s+1}). \quad (12)$$

• For *non-adjacent* stages s and t: $\quad t>s+1,$

$$V^s(\mathbf{x}^s,\boldsymbol{\theta}^s,\boldsymbol{\theta}^t)=L^s(\mathbf{x}^s,\boldsymbol{\theta}^s)+V^{s+1}(\mathbf{x}^{s+1},\boldsymbol{\theta}^{s+1},\boldsymbol{\theta}^t). \quad (13)$$

Notice here that $\boldsymbol{\theta}^t$ appears on both sides.

Boundary condition [see Equation (2)]:

$$T^N(\mathbf{x}^N,-)=E(\mathbf{f}^N(\mathbf{x}^N))=\frac{1}{2}\sum_{k=1}^{P_N}\|\mathbf{y}_k^N-\mathbf{t}_k^N\|_2^2. \quad (14)$$

In MLP-learning, an initial (nominal) guessed policy is often chosen as a randomly-initialized weight-parameter set. For deriving first-order BP (see [2]), we only need recurrence relation (11) to obtain the backward-pass recurrence relation between the two forms of node-sensitivity vectors $\boldsymbol{\xi}^s$ and $\boldsymbol{\delta}^{s+1}$ in Equation (7−3). The procedure [in Fig.1(b)] evaluates the stagwise-partitioned gradient vector sequence $\mathbf{g}^s=\frac{\partial T^s}{\partial \boldsymbol{\theta}^s}$, which represents the *first-order effect* on J in terms of control changes at each stage s along the nominal state trajectory $\mathbf{x}^s$. By extension, the second-order BP procedure allows us to investigate the *second-order effects* on how $\mathbf{g}^s=\frac{\partial T^s}{\partial \boldsymbol{\theta}^s}$ varies when

(1) the control changes as $\boldsymbol{\theta}^s \leftarrow \boldsymbol{\theta}^s+\delta\boldsymbol{\theta}^s$; and
(2) the state changes as $\mathbf{x}^s \leftarrow \mathbf{x}^s+\delta\mathbf{x}^s$.

For effect (1), we seek $\mathbf{H}^{s,s}\equiv\left[\frac{\partial^2 T^s}{\partial\boldsymbol{\theta}^s\partial\boldsymbol{\theta}^s}\right]$, *diagonal Hessian* blocks. For effect (2), we pursue $\mathbf{F}^{s,s}\equiv\left[\frac{\partial^2 T^s}{\partial\mathbf{x}^s\partial\boldsymbol{\theta}^s}\right]$, which requires to evaluate $\mathbf{Z}^s\equiv\left[\frac{\partial^2 T^s}{\partial\mathbf{x}^s\partial\mathbf{x}^s}\right]$, and then $\mathbf{F}^{s,u}\equiv\left[\frac{\partial^2 T^u}{\partial\mathbf{x}^s\partial\boldsymbol{\theta}^u}\right]$ (for $s<u$), yielding the evaluation of *off-diagonal Hessian*

blocks $\mathbf{H}^{s,u}$. In other words, how $\mathbf{g}^s$ varies when controls at two different stages s and u change is translated by the propagation of effect (2) through the *perturbations* of the *states* between those stages, leading to the evaluation of off-diagonal Hessian blocks $\mathbf{H}^{s,u}$.

Our derivation of the backward-pass recursive formula of stagewise second-order BP begins at *terminal stage* N (i.e., $s=t=N$), by evaluating a matrix of second partial derivatives of the scalar terminal cost $E(.)$ with respect to $\mathbf{x}^N$, the P_N-vector of terminal before-node net inputs:

$$\underbrace{\mathbf{Z}^N}_{P_N\times P_N} \stackrel{\text{def}}{=} \frac{\partial^2 T^N}{\partial \mathbf{x}^N \partial \mathbf{x}^N} = \left[\frac{\partial^2 E}{\partial \mathbf{x}^N \partial \mathbf{x}^N}\right] = \left[\frac{\partial \boldsymbol{\delta}^N}{\partial \mathbf{x}^N}\right], \tag{15}$$

$$= \underbrace{\left[\frac{\partial \mathbf{y}^N}{\partial \mathbf{x}^N}\right]^T}_{P_N\times P_N} \underbrace{\left[\frac{\partial^2 E}{\partial \mathbf{y}^N \partial \mathbf{y}^N}\right]}_{P_N\times P_N} \underbrace{\left[\frac{\partial \mathbf{y}^N}{\partial \mathbf{x}^N}\right]}_{P_N\times P_N} + \underbrace{\left\langle \left[\frac{\partial^2 \mathbf{y}^N}{\partial \mathbf{x}^N \partial \mathbf{x}^N}\right], \boldsymbol{\xi}^N \right\rangle}_{P_N\times P_N}.$$

Here, as defined just after Equation (7), $\boldsymbol{\delta}^N = \frac{\partial T^N}{\partial \mathbf{x}^N} = \frac{\partial E}{\partial \mathbf{x}^N}$ is the terminal before-node sensitivity vector, obtainable from Equation (7-1) with $\boldsymbol{\xi}^N = \mathbf{r}^N$ when Equation (2) is used, and the (i,j)-element of the last P_N-by-P_N *symmetric* matrix is obtainable from the following particular $\langle .,. \rangle$-operation (set $s=N$ below):

$$\left\langle \left[\frac{\partial^2 \mathbf{y}^s}{\partial \mathbf{x}^s \partial \mathbf{x}^s}\right], \boldsymbol{\xi}^s \right\rangle_{ij} \stackrel{\text{def}}{=} \sum_{k=1}^{P_s}\sum_{j=1}^{P_s}\sum_{i=1}^{P_s} \xi_k^s \left[\frac{\partial^2 y_k^s}{\partial x_i^s \partial x_j^s}\right], \tag{16}$$

which is just a diagonal matrix in standard MLP-learning.

At *non-terminal* stage s (for $s=N-1, N-2, \ldots,$ 2, 1), when $s=t\,(<N)$, we evaluate the following quantities:

$$\begin{cases} (a)\ \underbrace{\boldsymbol{\delta}^s}_{P_s\times 1} \stackrel{\text{def}}{=} \frac{\partial T^s}{\partial \mathbf{x}^s}; \quad (b)\ \underbrace{\mathbf{g}^s}_{P_s\times 1} \stackrel{\text{def}}{=} \frac{\partial T^s}{\partial \boldsymbol{\theta}^s} \\ (c)\ \underbrace{\mathbf{Z}^s}_{P_s\times P_s} \stackrel{\text{def}}{=} \frac{\partial^2 T^s}{\partial \mathbf{x}^s \partial \mathbf{x}^s} \quad \text{—— see Eq.(27)} \\ (d)\ \underbrace{\mathbf{F}^{s,s}}_{P_s\times n_s} \stackrel{\text{def}}{=} \frac{\partial^2 T^s}{\partial \mathbf{x}^s \partial \boldsymbol{\theta}^s} \quad \text{—— see Eq.(29)} \\ (e)\ \underbrace{\mathbf{H}^{s,s}}_{n_s\times n_s} \stackrel{\text{def}}{=} \frac{\partial^2 T^s}{\partial \boldsymbol{\theta}^s \partial \boldsymbol{\theta}^s} \quad \text{—— see Eq.(31)} \end{cases} \tag{17}$$

Next, we consider a case where $s=N-2$ and $t=N-1$ for **adjacent** stages s and $s+1\,(=t)$; the recurrence relation in Equation (12) is given by

$$\begin{aligned} &V^{N-2}(\mathbf{x}^{N-2}, \boldsymbol{\theta}^{N-2}, \boldsymbol{\theta}^{N-1}) \\ &= L^{N-2}(\mathbf{x}^{N-2}, \boldsymbol{\theta}^{N-2}) + T^{N-1}(\mathbf{x}^{N-1}, \boldsymbol{\theta}^{N-1}). \end{aligned} \tag{18}$$

By differentiating Equation (18) once and twice, we obtain a gradient vector and an off-diagonal Hessian block, as shown next:

$$\begin{cases} (a)\ \dfrac{\partial V^{N-2}}{\partial \boldsymbol{\theta}^{N-1}} = \dfrac{\partial \left(L^{N-2} + T^{N-1}\right)}{\partial \boldsymbol{\theta}^{N-1}} \\ \qquad\qquad = \dfrac{\partial T^{N-1}}{\partial \boldsymbol{\theta}^{N-1}} = \mathbf{g}^{N-1} \\ (b)\ \dfrac{\partial^2 V^{N-2}}{\partial \boldsymbol{\theta}^{N-2} \partial \boldsymbol{\theta}^{N-1}} = \dfrac{\partial^2 T^{N-1}}{\partial \boldsymbol{\theta}^{N-2} \partial \boldsymbol{\theta}^{N-1}} \\ \qquad\qquad = \left[\dfrac{\partial \mathbf{x}^{N-1}}{\partial \boldsymbol{\theta}^{N-2}}\right]^{\mathsf{T}} \left[\dfrac{\partial^2 T^{N-1}}{\partial \mathbf{x}^{N-1} \partial \boldsymbol{\theta}^{N-1}}\right] \\ \qquad\qquad = \left[\dfrac{\partial \mathbf{x}^{N-1}}{\partial \boldsymbol{\theta}^{N-2}}\right]^{\mathsf{T}} \underbrace{\mathbf{F}^{N-1,N-1}}_{P_{N-1}\times n_{N-1}} \\ \qquad\qquad = \underbrace{\mathbf{H}^{N-2,N-1}}_{n_{N-2}\times n_{N-1}}. \end{cases} \tag{19}$$

For **non-adjacent** stages s and u, such as when $s=N-3$ (and $u=N-1$), the recurrence relation in Equation (13) is given by

$$\begin{aligned} &V^{N-3}(\mathbf{x}^{N-3}, \boldsymbol{\theta}^{N-3}, \boldsymbol{\theta}^{N-1}) \\ &= L^{N-3}(\mathbf{x}^{N-3}, \boldsymbol{\theta}^{N-3}) + V^{N-2}(\mathbf{x}^{N-2}, \boldsymbol{\theta}^{N-2}, \boldsymbol{\theta}^{N-1}). \end{aligned} \tag{20}$$

Differentiating Equation (20) once [with Eq.(18)] and twice yields the following:

$$\begin{cases} (a)\ \dfrac{\partial V^{N-3}}{\partial \boldsymbol{\theta}^{N-1}} = \dfrac{\partial V^{N-2}}{\partial \boldsymbol{\theta}^{N-1}} = \dfrac{\partial T^{N-1}}{\partial \boldsymbol{\theta}^{N-1}} = \mathbf{g}^{N-1} \\ (b)\ \dfrac{\partial^2 V^{N-3}}{\partial \boldsymbol{\theta}^{N-3} \partial \boldsymbol{\theta}^{N-1}} = \dfrac{\partial^2 V^{N-2}}{\partial \boldsymbol{\theta}^{N-3} \partial \boldsymbol{\theta}^{N-1}} \\ \qquad\qquad = \left[\dfrac{\partial \mathbf{x}^{N-2}}{\partial \boldsymbol{\theta}^{N-3}}\right]^{\mathsf{T}} \left[\dfrac{\partial^2 V^{N-2}}{\partial \mathbf{x}^{N-2} \partial \boldsymbol{\theta}^{N-1}}\right] \\ \qquad\qquad = \left[\dfrac{\partial \mathbf{x}^{N-2}}{\partial \boldsymbol{\theta}^{N-3}}\right]^{\mathsf{T}} \underbrace{\mathbf{F}^{N-2,N-1}}_{P_{N-2}\times n_{N-1}} \\ \qquad\qquad = \underbrace{\mathbf{H}^{N-3,N-1}}_{n_{N-3}\times n_{N-1}}. \end{cases} \tag{21}$$

In (a), Equation (20) is differentiated once and then Equation (18) is used, and in (b) a rectangular matrix $\mathbf{F}^{s,u}$ below is used for different stages s and u $(s<u)$

$$\underbrace{\mathbf{F}^{s,u}}_{P_s\times n_u} \stackrel{\text{def}}{=} \left[\frac{\partial^2 V^s}{\partial \mathbf{x}^s \partial \boldsymbol{\theta}^u}\right] = \left[\frac{\partial^2 T^u}{\partial \mathbf{x}^s \partial \boldsymbol{\theta}^u}\right]; \quad (\text{for } s<u). \tag{22}$$

In general, $\mathbf{F}^{s,u}$ (for $s<u$) is given with a suitable number of $\mathbf{N}^{s,s+1} \equiv \left[\frac{\partial \mathbf{x}^{s+1}}{\partial \mathbf{x}^s}\right]$, which is a so-called *before-node state*

transition matrix of size $P_{s+1} \times P_s$ (see [5]). That is,

$$\underbrace{\mathbf{F}^{s,u}}_{P_s \times n_u} = \underbrace{\mathbf{N}^{s,s+1}}_{P_s \times P_{s+1}}{}^{\mathsf{T}} \underbrace{\mathbf{N}^{s+1,s+2}}_{P_{s+1} \times P_{s+2}}{}^{\mathsf{T}}, \cdots, \underbrace{\mathbf{N}^{u-1,u}}_{P_{u-1} \times P_u}{}^{\mathsf{T}} \underbrace{\mathbf{F}^{u,u}}_{P_u \times n_u}. \quad (23)$$

In reality, this chain matrix multiplication is performed by using the second-order backward-pass recurrence below

$$\underbrace{\mathbf{F}^{s,u}_{\text{(new)}}}_{P_s \times n_u} = \underbrace{\mathbf{N}^{s,s+1}}_{P_s \times P_{s+1}}{}^{\mathsf{T}} \underbrace{\mathbf{F}^{s+1,u}_{\text{(old)}}}_{P_{s+1} \times n_u} = \left[\frac{\partial \mathbf{y}^s}{\partial \mathbf{x}^s}\right]^{\mathsf{T}} \mathbf{\Theta}^{s,s+1\mathsf{T}}_{\text{void}} \mathbf{F}^{s+1,u}_{\text{(old)}}, \quad (24)$$

where the last matrix is computed already at a previous stage; hence, the subscript *old* attached. Here, notice in standard MLP-learning that the first matrix $\left[\frac{\partial \mathbf{y}^s}{\partial \mathbf{x}^s}\right]$ on the right-hand side becomes *diagonal* with the jth element $f_j^{s'}(.)$ $(j=1,...,P_s)$ [see Eq.(8)]. Furthermore, there is no need to form all those $\mathbf{N}$ matrices explicitly [5] unlike another second-order BP that evaluates $\mathbf{H}$ differently in a *nodewise* manner described on pp.155–157 in [6].

When $s = N-4$ (and $t = N-1$), we proceed in the same manner as in the aforementioned case where $s=N-3$ for non-adjacent stages. We first write down the recurrence relation (13) as below:

$$\begin{aligned} &V^{N-4}(\mathbf{x}^{N-4}, \boldsymbol{\theta}^{N-4}, \boldsymbol{\theta}^{N-1}) \\ &= L^{N-4}(\mathbf{x}^{N-4}, \boldsymbol{\theta}^{N-4}) + V^{N-3}(\mathbf{x}^{N-3}, \boldsymbol{\theta}^{N-3}, \boldsymbol{\theta}^{N-1}). \end{aligned} \quad (25)$$

By differentiating recurrence relation (25) once and twice [see Eq.(21)], we obtain the following:

$$\left\{ \begin{aligned} &(a)\ \ \frac{\partial V^{N-4}}{\partial \boldsymbol{\theta}^{N-1}} = \frac{\partial V^{N-3}}{\partial \boldsymbol{\theta}^{N-1}} = \frac{\partial T^{N-1}}{\partial \boldsymbol{\theta}^{N-1}} = \mathbf{g}^{N-1} \\ &(b)\ \ \frac{\partial^2 V^{N-4}}{\partial \boldsymbol{\theta}^{N-4} \partial \boldsymbol{\theta}^{N-1}} = \frac{\partial^2 V^{N-3}}{\partial \boldsymbol{\theta}^{N-4} \partial \boldsymbol{\theta}^{N-1}} \\ &\qquad = \left[\frac{\partial \mathbf{x}^{N-3}}{\partial \boldsymbol{\theta}^{N-4}}\right]^{\mathsf{T}} \left[\frac{\partial^2 V^{N-3}}{\partial \mathbf{x}^{N-3} \partial \boldsymbol{\theta}^{N-1}}\right] \\ &\qquad = \left[\frac{\partial \mathbf{x}^{N-3}}{\partial \boldsymbol{\theta}^{N-4}}\right]^{\mathsf{T}} \underbrace{\mathbf{F}^{N-3,N-1}}_{P_{N-3} \times n_{N-1}} \\ &\qquad = \underbrace{\mathbf{H}^{N-4,N-1}}_{n_{N-4} \times n_{N-1}}. \end{aligned} \right. \quad (26)$$

For completeness, we show *recursive formulas* for $\mathbf{Z}^s$, $\mathbf{F}^{s,s}$, and $\mathbf{H}^{s,s}$ in Equation (17). To obtain $\mathbf{Z}^s$ in Equation (17)(c) for non-terminal stage s for $s=N-1, \cdots, 2$, we differentiate twice the recurrence relation (11) with respect to the state vector, yielding

$$\begin{aligned} \underbrace{\mathbf{Z}^s}_{P_s \times P_s} &\stackrel{\text{def}}{=} \left[\frac{\partial^2 T^s}{\partial \mathbf{x}^s \partial \mathbf{x}^s}\right] \quad (27) \\ &= \left[\frac{\partial^2 L^s}{\partial \mathbf{x}^s \partial \mathbf{x}^s}\right] + \underbrace{\mathbf{N}^{s,s+1}}_{P_s \times P_{s+1}}{}^{\mathsf{T}} \underbrace{\mathbf{Z}^{s+1}}_{P_{s+1} \times P_{s+1}} \underbrace{\mathbf{N}^{s,s+1}}_{P_{s+1} \times P_s} + \underbrace{\left\langle \left[\frac{\partial^2 \mathbf{y}^s}{\partial \mathbf{x}^s \partial \mathbf{x}^s}\right], \boldsymbol{\xi}^s \right\rangle}_{P_s \times P_s} \end{aligned}$$

where the (i,j)-element of the last matrix is obtainable from

$$\left\langle \left[\frac{\partial^2 \mathbf{y}^s}{\partial \mathbf{x}^s \partial \mathbf{x}^s}\right], \boldsymbol{\xi}^s \right\rangle_{ij} = \sum_{k=1}^{P_s} \xi_k^s \left[\frac{\partial^2 y^s{}_k}{\partial x_i^s \partial x_j^s}\right]. \quad (28)$$

We derive another key recursive formula for matrix $\mathbf{F}^{s,s}$ in Equation (17)(d) by differentiating twice the recurrence equation (11) with respect to state and control vectors, yielding (after a little algebra)

$$\begin{aligned} \underbrace{\mathbf{F}^{s,s}}_{P_s \times n_s} &\stackrel{\text{def}}{=} \left[\frac{\partial^2 T^s}{\partial \mathbf{x}^s \partial \boldsymbol{\theta}^s}\right] \quad (29) \\ &= \left[\frac{\partial^2 L^s}{\partial \mathbf{x}^s \partial \boldsymbol{\theta}^s}\right] + \underbrace{\left[\frac{\partial \mathbf{y}^s}{\partial \mathbf{x}^s}\right]^{\mathsf{T}}}_{P_s \times P_s} \left\{ \underbrace{\mathbf{\Theta}^{s,s+1\mathsf{T}}_{\text{void}}}_{P_s \times P_{s+1}} \underbrace{\mathbf{Z}^{s+1}}_{P_{s+1} \times P_{s+1}} \underbrace{\left[\frac{\partial \mathbf{x}^{s+1}}{\partial \boldsymbol{\theta}^{s,s+1}}\right]}_{P_{s+1} \times n_s} \right. \\ &\qquad \left. + \underbrace{\left\langle \left[\frac{\partial \boldsymbol{\theta}^{s,s+1}_{\text{void}}}{\partial \boldsymbol{\theta}^{s,s+1}}\right], \boldsymbol{\delta}^{s+1} \right\rangle}_{P_s \times n_s} \right\}, \end{aligned}$$

where the (i,j)-element of the last P_s-by-n_s rectangular matrix is given by

$$\left\langle \left[\frac{\partial \boldsymbol{\theta}^{s,s+1}_{\text{void}}}{\partial \boldsymbol{\theta}^{s,s+1}}\right], \boldsymbol{\delta}^{s+1} \right\rangle_{ij} \stackrel{\text{def}}{=} \sum_{k=1}^{P_{s+1}} \sum_{i=1}^{P_s} \sum_{l=0}^{P_s} \delta_k^{s+1} \left[\frac{\partial \theta^{s,s+1}_{i,k}}{\partial \theta^{s,s+1}_{l,k}}\right]. \quad (30)$$

Here, index j is subject to $j = (1+P_s)(k-1)+l+1$, and $\boldsymbol{\theta}^s_{\text{void}}$ and $\boldsymbol{\theta}^s$ are the vector forms of the parameter matrices $\mathbf{\Theta}^s_{\text{void}}$ and $\mathbf{\Theta}^s$ in Equation (5), respectively.

The diagonal Hessian block $\mathbf{H}^{s,s}$ is obtainable from twice differentiation of the recurrence equation (11) with respect to the control vector:

$$\begin{aligned} \underbrace{\mathbf{H}^{s,s}}_{n_s \times n_s} &\stackrel{\text{def}}{=} \left[\frac{\partial^2 T^s}{\partial \boldsymbol{\theta}^s \partial \boldsymbol{\theta}^s}\right] \quad (31) \\ &= \left[\frac{\partial^2 L^s}{\partial \boldsymbol{\theta}^s \partial \boldsymbol{\theta}^s}\right] + \underbrace{\left[\frac{\partial \mathbf{x}^{s+1}}{\partial \boldsymbol{\theta}^s}\right]^{\mathsf{T}}}_{n_s \times P_{s+1}} \underbrace{\mathbf{Z}^{s+1}}_{P_{s+1} \times P_{s+1}} \underbrace{\left[\frac{\partial \mathbf{x}^{s+1}}{\partial \boldsymbol{\theta}^s}\right]}_{P_{s+1} \times n_s}. \end{aligned}$$

The off-diagonal Hessian blocks can be obtained in conjunction with Equation (24), as shown next

$$\underbrace{\mathbf{H}^{r,s}}_{n_r \times n_s} = \underbrace{\left[\frac{\partial \mathbf{x}^{r+1}}{\partial \boldsymbol{\theta}^r}\right]^{\mathsf{T}}}_{n_r \times P_{r+1}} \underbrace{\mathbf{F}^{r+1,s}}_{P_{r+1} \times n_s} \quad (\text{for } r+1 \le s), \quad (32)$$

where $n_r \equiv (1+P_r)P_{r+1}$; $\mathbf{F}^{r+1,s}$ is obtainable from Equation (24); and the n_r-by-P_{r+1} *transposed* matrix has a "block-diagonal" form below in MLP-learning

$$\underbrace{\left[\frac{\partial \mathbf{x}^{r+1}}{\partial \boldsymbol{\theta}^r}\right]^{\mathsf{T}}}_{n_r \times P_{r+1}} = \begin{bmatrix} * & & \\ & * & \\ & & \ddots \\ & & & * \end{bmatrix}.$$

Here, there are P_{r+1} *identical* blocks, denoted by the same symbol "$*$" (hence, need to store one block "$*$" alone) because each block is just a $(1+P_r)$-length vector $\mathbf{y}^r_+$ defined

in Equation (5) with $s=r$, where the first component is a bias-node *constant* output y_0^r (e.g., $y_0^r = 1.0$). Our stagewise second-order BP can be summarized as follows:

Algorithm: *Stagewise second-order BP on each datum.*

(0) Do forward pass from stage 1 to N for evaluating J in Equation (1) on a given training datum.
(1) At terminal stage N, obtain $\mathbf{Z}^N$ by Equation (15).
- Set $s=N-1$, and repeat the following steps:

(2) Evaluate diagonal Hessian blocks $\mathbf{H}^{s,s}$ at each stage s by Equation (31).
- When $s=N-1$, go to Step (5); otherwise, continue:

(3) Obtain off-diagonal Hessian blocks $\mathbf{H}^{s,t}$ ($s < t \leq N$–1) by Equation (32).
- If $s=1$, then terminate; otherwise, continue:

(4) Compute rectangular matrices $\mathbf{F}^{s,t}$ ($s < t \leq N-1$) by Equation (24).
(5) Evaluate a new P_s-by-n_s rectangular matrix $\mathbf{F}^{s,s}$ by Equation (29) for the current stage s.
(6) Compute a P_s-by-P_s matrix $\mathbf{Z}^s$ by Equation (27).
- Set $s=s-1$, and go back to Step (2). □ (*End*) □

All the foregoing formulas are derived by choosing the before-node net-input vector $\mathbf{x}^s$ as the *state* vector at stage s to conform with neural-network (NN) convention; see Table I. In optimal control, the after-node output vector $\mathbf{y}^s$ is used as the *state* vector. Despite this distinction and some others, using a state vector as a basic ingredient allows us to adopt analogous formulas available in the second-order optimal control theory (see [1] and references therein). Therefore, in the table below, we show which equations in this paper "loosely" match up to the formulas appeared in three other papers [5], [7], [1] for further reference:

This paper	Paper [5]	Paper [7]	Paper [1]
$\mathbf{Z}^s$ in Eq.(27)	Eq.(13)	Eq.(17)	Eq.(9)
$\mathbf{F}^{s,s}$ in Eq.(29)	Eq.(11)	Eq.(16)	Eq.(10)
$\mathbf{F}^{s,u}$ in Eq.(24)	Eq.(10)	N/A	N/A
$\mathbf{H}^{s,s}$ in Eq.(31)	Eq.(8)	Eq.(22)	N/A
$\mathbf{H}^{r,s}$ in Eq.(32)	Eq.(9)	N/A	N/A

For details on algorithmic descriptions of the posed *stagewise second-order BP*, refer to [5], where the formulas do not include the terms associated with the stage costs L^s. In what follows, we describe the methods that use L^s.

III. Introduction of Stage Costs

We shall show how stage costs L^s serve to develop advanced learning schemes.

A. Hidden-node teaching

In certain learning situations, some *additional* (or *extra*) information related to a posed learning task might be available. *Hidden-node teaching* supplies such information to any nodes at any intermediate stage s ($s = 2, ..., N-1$) as the "desired" *hidden* outputs, leading to stage costs L^s; e.g., see Equation (3). Here are three major purposes for using hidden-node teaching:

(α) Provide useful *hints* related to a given target task [2];
(β) Alleviate *hidden-node saturations*;
(γ) Perform "desired" signal encoding or decoding.

For purpose (α), one might resort to the methods of *hints* [8] and *multi-task learning* [9]; however, those methods need to set up additional nodes at terminal layer to receive additional information. Accordingly, the number of parameters is increased, resulting in slower learning. Furthermore, for the second-order methods, due to increase in size of the Hessian matrix $\mathbf{H}$ of the objective function J, more *sparsity* would be inevitably involved in the "block-arrow" Hessian $\mathbf{H}$ in Figure 2 (hence, worth exploiting such sparsity). On the other hand, *hidden-node teaching* can supply such information without altering a given NN structure for the purpose of (α).

Aim (β) above is specially designed to circumvent the hidden-node saturation problems: When all the sigmoidal hidden-node functions (e.g., tanh) get driven to their saturation limits, the associated Hessian matrix $\mathbf{H}$ becomes *rank deficient* (consequently, trapped in a lower-dimensional subspace). To alleviate this concern, we could supply certain target signals at (a subset of) hidden nodes to avoid hidden-node saturations. For showing the value of this concept, a sample MATLAB code has been made available on the web at http://www.ieor.berkeley.edu/People/Faculty/dreyfus-pubs/hidteach.m: The code attacks the well-known *7-bit parity problem*, the 7-dimensional XOR-problem with $2^7 (= 128)$ data, by a 7-4-1 MLP with only four hidden (tanh) nodes (hence, 37 parameters in total), demonstrating that an *on-line* hidden-node teaching algorithm can render the 7-4-1 MLP able to solve the posed problem always perfectly as long as the initial parameters are randomly generated uniformly in a small range (e.g., [-0.2, +0.2]). Furthermore, the number of epoch required for solution does not differ greatly. That is, owing to the hidden-node teaching, the 7-4-1 MLP can successfully *develop insensitivity to the initial parameters*, and thus never fails to solve the 7-bit parity problem perfectly (see hidteach.m and references therein).

In Sec. IV, we shall demonstrate our hidden-node teaching for accomplishing two goals (α) and (γ) simultaneously.

B. Stage costs for weight decay

Our neural-network (NN) model is expected to perform well when previously-unseen "test" data are presented, which are outside the "training" data set. This is an important matter of *generalization*. Often, the optimization process is to be stopped partway before the *training*-error measure is minimized (i.e., early stopping). Furthermore, one may consider **regularization** by adding some *penalty* cost C (to be imposed on control parameters) to the objective function. One regularization approach is to *penalize large control parameters*, which is known as *weight decay*; an easily-implementable scheme is to employ the *sum of squared controls (i.e., decision parameters)* as C (e.g., see [10] and pp.338–343 in [6]), given with some scalar constant μ as $C \equiv \frac{1}{2}\mu^2 \boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{\theta}$. In the spirit of *optimal control*, the cost C for such a simple *weight decay* can be merged at each stage s into the stage cost L^s (*scalar*) with (or without) *hidden-node*

teaching in Equation (3), leading to J in Equation (1) with L^s (per datum) below:

$$L^s(\mathbf{y}^s, \boldsymbol{\theta}^s) = \underbrace{\frac{1}{2}\mu^2 {\boldsymbol{\theta}^s}^{\mathsf{T}} \boldsymbol{\theta}^s}_{\text{weight decay}} + \underbrace{\frac{1}{2}\left\|\mathbf{y}^s - \mathbf{t}^s\right\|_2^2}_{\text{hidden-node teaching}}. \tag{33}$$

The gradient vector $\mathbf{g}$ of the posed J can be derived by recurrence (11); for parameter $\theta^s_{j,k}$ [see Eq.(6) for notations], an element of $\mathbf{g}^s$ at stage s (per datum) is given as

$$\begin{aligned} g^s_{jk} &= \frac{\partial T^s(\mathbf{y}^s, \boldsymbol{\theta}^s)}{\partial \theta^s_{j,k}} = \frac{\partial L^s(\mathbf{y}^s, \boldsymbol{\theta}^s)}{\partial \theta^s_{j,k}} + \frac{\partial T^{s+1}(\mathbf{y}^{s+1}, \boldsymbol{\theta}^{s+1})}{\partial \theta^s_{j,k}} \\ &= \underbrace{\mu^2 \theta^s_{j,k}}_{\text{weight decay}} + \left(\frac{\partial x^{s+1}_k}{\partial \theta^s_{j,k}} \frac{\partial y^{s+1}_k}{\partial x^{s+1}_k}\right) \Bigg\{ \underbrace{\left(y^{s+1}_k - t^{s+1}_k\right)}_{\text{residual } r^{s+1}_k} + \underbrace{\sum_{l=1}^{P_{s+2}} \theta^{s+1}_{kl} \delta^{s+2}_l}_{\text{backward pass}} \Bigg\} \\ &= \mu^2 \theta^s_{j,k} + y^s_j \, \delta^{s+1}_k. \end{aligned} \tag{34}$$

Because $\left[\frac{\partial^2 L^s}{\partial \boldsymbol{\theta}^s \partial \boldsymbol{\theta}^s}\right] = \mu^2 \left[\frac{\partial^2 C}{\partial \boldsymbol{\theta}^s \partial \boldsymbol{\theta}^s}\right] = \mu^2 \mathbf{I}$, the n-by-n symmetric Hessian matrix $\mathbf{H}$ of J is given with the identity matrix $\mathbf{I}$ as $\mathbf{H} + \mu^2 \mathbf{I}$. Consequently, weight-decay methods encourage the magnitude of decision parameters $\boldsymbol{\theta}^s$ to be kept small by penalizing large controls in magnitude, which can be implemented by introducing stage costs L^s.

This should not be confused with the widely-employed *Levenberg-Marquardt method* in the *nonlinear least squares* (with no weight decay) context, where the n-by-n full Hessian matrix $\mathbf{H}$ of J [with weight-decay terms excluded from L^s in Eq.(33)] has such a special form as $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S}$; here, $\mathbf{J}^{\mathsf{T}}\mathbf{J}$ is called the Gauss-Newton Hessian, $\mathbf{J}$ the m-by-n Jacobian matrix of the m-dimensional residual vector $\mathbf{r}$, and $\mathbf{S}$ a matrix of terms involving the second derivatives of residuals. In normal-equation approach, the method may use $\mathbf{J}^{\mathsf{T}}\mathbf{J} + \mu^2\mathbf{I}$ for solving approximately the subproblem below

$$\begin{aligned} &\min_{\Delta\boldsymbol{\theta}} \left\{ \|\mathbf{r} + \mathbf{J}\Delta\boldsymbol{\theta}\|_2^2 + \mu \|\Delta\boldsymbol{\theta}\|_2^2 \right\} \\ \Longleftrightarrow \; &\min_{\Delta\boldsymbol{\theta}} \left\{ \|\mathbf{r} + \mathbf{J}\Delta\boldsymbol{\theta}\|_2^2 \right\} \text{ subject to } \|\Delta\boldsymbol{\theta}\|_2^2 < R^2, \end{aligned} \tag{35}$$

where R is called trust-region radius. There is a vast literature on trust-region methods that solve the above trust-region subproblem at each epoch efficiently for parameter optimization; e.g., see [11]. The trust-region methods may employ the same form of the Hessian matrix, but uses the ordinary gradient vector $\mathbf{g} = \mathbf{J}^T\mathbf{r}$ of J excluding such a weight-decay term from Equation (34); therefore, the trust-region methods penalize the large step $\Delta\boldsymbol{\theta}$ (or *implicitly* avoids making control parameters "large") by solving the trust-region subproblem, whereas weight-decay methods impose a penalty *explicitly* on large control without telling us how to determine μ efficiently. Overall, we recommend *trust-region regularization* (e.g., see [4]) rather than weight-decay type. Notice that when hidden-node teaching is employed, $\mathbf{J}$ has more "sparse" rows due to hidden residuals at L^s in Equation (3), but our stagewise second-order BP can efficiently compute $\mathbf{H}$; Section IV shows our numerical evidence with and without hidden-node teaching under *trust-region regularization*.

IV. Numerical Results and Discussions

First, we show how efficiently our stagewise second-order BP evaluates the n-by-n full Hessian matrix of $E(.)$ defined in Equation (2): $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S}$ [recall discussion for Eq.(35)]. In simulation on a 2-GHz Pentium-4 PC, we employed a single-hidden-layer (i.e., three-stage: $N = 3$) 5-130-3 MLP with three linear outputs ($F = 3$) on 400 data ($D = 400$); i.e.,

$$\begin{cases} N = 3;\ P_1 = 5; P_2 = 130; P_3 (\equiv F) = 3;\ C_A = 1 + P_2 = 131; \\ n_B = (1 + P_1) P_2 = 780; n = FC_A + n_B = 1,173; m = FD = 1,200. \end{cases}$$

We compared performance in speed to obtain a "block-arrow" Hessian matrix with $F(= 3)$ diagonal blocks [recall *sparsity* in Fig.2(a)] between the following four algorithms: (A) Stagewise second-order BP that evaluates the *full* Hessian matrix $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S}$; (B) A simplified version that evaluates the Gauss-Newton Hessian $\mathbf{J}^T\mathbf{J}$ alone (see [12]); (C) A *standard* method that computes row vectors $\mathbf{u}^T$ of $\mathbf{J}$ per datum, and then forms $\mathbf{J}^T\mathbf{J}$ alone by rank updates $\mathbf{u}\mathbf{u}^T$ with sparsity-exploitation, and (D) the same *standard* method as (C) but with such sparsity totally ignored as in calcjejj.m (for trainlm.m) on the MATLAB Neural Network Toolbox. The average execution time measured in (C) was 3.8 second (averaged over 30 trials) and was 6.8 second in (D), whereas it was 2.9 second in (A) and (B); remarkably, no time difference between (A) and (B). This implies that our stagewise algorithm economically evaluates $\mathbf{S}$ during the backward process; see Equations (16), (28), and (30), which yield *sparse* matrices in MLP-learning (see [5]).

Next, we shall demonstrate *hidden-node teaching* for data compression using an encoder neural network with a bottleneck structure. In our experiments, we used a 8-3-8 MLP (with three tanh hidden nodes) illustrated in Figure 3 for solving a classical eight-bit *encoding problem* described on pages 336-337 in [3]. Their original concept is "automatic" encoding; that is, present the same eight-bit *binary* data (see the first column in table below) both at the first and the terminal layers for training purposes, and read off the three hidden-node outputs. Their resulting values are tabulated below in symbol as the PDP-patterns (see the last column), which contain an intermediate value labeled *middle* (nearly 0) between ON ($+1$) and off (-1):

Eight-bit patterns (Inputs/Targets)	HID-patterns	PDP-patterns
(ON,off,off,off,off,off,off,off)	(ON,off,off)	(*middle*,off,off)
(off,ON,off,off,off,off,off,off)	(off,ON,off)	(off,ON,off)
(off,off,ON,off,off,off,off,off)	(ON,ON,off)	(ON,ON,off)
(off,off,off,ON,off,off,off,off)	(off,off,ON)	(ON,ON,ON)
(off,off,off,off,ON,off,off,off)	(ON,off,ON)	(off,ON,ON)
(off,off,off,off,off,ON,off,off)	(off,ON,ON)	(*middle*,off,ON)
(off,off,off,off,off,off,ON,off)	(ON,ON,ON)	(ON,off,*middle*)
(off,off,off,off,off,off,off,ON)	(off,off,off)	(off,off,*middle*)

However, the posed *automatic* scheme is highly questionable because the three hidden-node outputs very often represented meaningless signals (including unexpectedly many *middle* values), and learning those eight-bit training patterns hardly succeeded by the 8-3-8 MLP. The next table present our simulation results obtained by two methods: (F) First-order on-line incremental gradient method (limit epoch 1,000), and

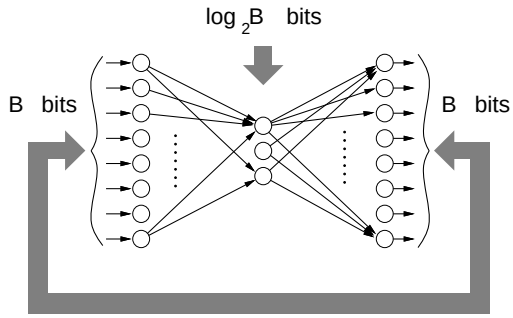


Fig. 3. A B-bit encoder MLP with *hidden-node teaching*. For attacking the eight-bit ($B=8$) encoding problem posed on pages 336-337 in [3] by this single-hidden-layer 8-3-8 MLP, one may use tanh hidden-node functions. The same eight-bit patterns are used as input and terminal desired output, while user-oriented three-bit patterns are supplied as hidden target signals.

(S) Second-order batch-mode trust-region method with our stagewise second-order BP (limit epoch 100):

	Automatic scheme			**Hidden-node teaching**			
	Success	Epoch	RMSE	Success	Epoch	RMSE	H-RMSE
(F)	12%	477.5	0.531	100%	105.0	0.543	0.328
(S)	16%	6.8	0.506	100%	4.7	0.509	0.159

Here, "Success" in columns 2 and 5 shows the success rate for learning the eight-bit patterns in the sense that the largest terminal output node *matches* the ON-bit on all the eight data, which can occur even when the terminal residual error $E(.)$ [in Eq.(2)] was greater than 0.5 in terms of root mean square error (RMSE); see columns 4 and 7. The above results were averaged over the successful cases among 50 trials (using 50 sets of initial parameters randomly generated uniformly in a small range [-0.2,+0.2]). For *hidden-node teaching*, the HID-patterns in the first table were used as the desired hidden outputs for L^2 [$s=2$ in Eq.(3)]. The following three observations confirm that hidden-node teaching with stagewise second-order BP can work efficiently: (1) The 8-3-8 MLP can reduce the hidden residuals L^2 (see "H-RMSE" in the last column) faster than the terminal residuals $E(.)$ (compare "RMSE" columns); (2) Second-order method (S) decreased both terminal and hidden residuals more efficiently than first-order method (F) (see the difference in "H-RMSE"); and (3) Hidden-node teaching helped increase the success rate for learning the eight-bit patterns also (compare columns 2 & 5). Interested readers are encouraged to attack the posed problem by a 8-3-8 MLP using the data in the first table.

Without hidden-node teaching, the 8-3-8 MLP most likely failed to learn the training patterns; under this poor learning situation, it is hopeless to expect meaningful signals *automatically* "encoded" at the three hidden nodes. By contrast, "hidden-node teaching" can control the hidden-node outputs by presenting certain desired hidden outputs (i.e., by introducing stage costs L^2), as discussed in Section III-A. Of course, by hidden-node teaching, it is no longer "automatic," but in a certain circumstance, where the test data (beside the training data) are available to check generalization capacity, hidden-node teaching would undoubtedly ensure better quality of "encoded" signals produced at (a subset of) hidden nodes on test data in the context of multi-task learning.

V. Conclusion

We have shown a simple derivation of stagewise second-order BP by invariant-imbedding recurrences. The procedure exploits "stagewise" structure embedded in an N-stage neural network and is based on the optimal-control theory, where N is usually very large and the objective function can be of any form (see [1]). This leads to a systematic evaluation of the Hessian even with hidden-node teaching and weight-decay regularization. In nonlinear least squares, where the Hessian matrix ($\mathbf{H}=\mathbf{J}^T\mathbf{J}+\mathbf{S}$) is of block-arrow form [see Fig.2(a)], the usual practice is to use only the Gauss-Newton Hessian $\mathbf{J}^T\mathbf{J}$ because $\mathbf{S}$ is expensive to compute. In Sec.IV, however, our numerical results verify that our stagewise second-order BP forms $\mathbf{H}$ faster than the standard method that obtains $\mathbf{J}^T\mathbf{J}$ alone by rank updates [12]. This is significant because $\mathbf{S}$ is important to efficiency in large-residual problems, and the *trust-region* method works excellently with the *indefinite* Hessian $\mathbf{H}$ as well as positive (semi-)definite $\mathbf{H}$. Furthermore, the results indicate that stagewise-BP can improve the efficiency of *trust-region* second-order learning with *hidden-node teaching* in a classical encoding problem. For a more realistic application, one might employ a five-stage ($N=5$) bottleneck MLP with three hidden layers, where (a subset of) hidden nodes in the middle layer (i.e., stage 3) may be used for encoding by hidden-node teaching. In this multi-task learning with such a larger N, the utility of our stagewise second-order BP would be magnified.

References

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

[2] E. Mizutani, S.E. Dreyfus, and K. Nishio. On derivation of MLP back-propagation from the Kelley-Bryson optimal-control gradient formula and its application. In *Proc. of the IEEE International Conference on Neural Networks (vol.2)*, pages 167–172, Como, Italy, July 2000.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, volume 1. MIT press, Cambridge, MA., 1986.

[4] Eiji Mizutani and James W. Demmel. On structure-exploiting trust-region regularized nonlinear least squares algorithms for neural-network learning. *Neural Networks*, 16:745–753, 2003.

[5] Eiji Mizutani, Stuart E. Dreyfus, and James W. Demmel. Second-order backpropagation algorithms for a stagewise-partitioned separable Hessian matrix. In *Proc. of 2005 Int'l Joint Conf. on Neural Networks* (see www.ieor.berkeley.edu/People/Faculty/dreyfus-pubs/ijcnnESJ05.pdf).

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

[7] 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.

[8] Yaser S. Abu-Mostafa. Hints. *Neural Computation*, 7:639–671, 1995.

[9] Rich Caruana. A dozen tricks with multitask learning. In G. B. Orr and K.-R. Muller, editors, *Neural networks : tricks of the trade*, pages 165–191. Springer, 1998. Lecture notes in computer science 1524.

[10] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, vol.4, pp.950–957. Morgan Kauffmann Publishers, 1995.

[11] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. SIAM MPS/SIAM Series on Optimization, 2000.

[12] Eiji Mizutani. On computing the Gauss-Newton Hessian matrix for neural-network learning. In *Proc. of the 12th Int'l Conf. on Neural Information Processing (ICONIP 2005)*, pp.43–48, Taipei, 2005.