Hybrid Learning of Mapping and Its Jacobian in Multilayer Neural Networks

Jeong-Woo Lee
Machine Control Laboratory, Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology, Taejeon, Republic of Korea

Jun-Ho Oh
Department of Mechanical Engineering, KAIST, Taejeon, Republic of Korea

There are some learning problems for which a priori information, such as the Jacobian of mapping, is available in addition to input-output examples. This kind of information can be beneficial in neural network learning if it can be embedded into the network. This article is concerned with the method for learning the mapping and available Jacobian simultaneously. The basic idea is to minimize the cost function, which is composed of the mapping error and the Jacobian error. Prior to developing the Jacobian learning rule, we develop an explicit and general method for computing the Jacobian of the neural network using the parameters computed in error backpropagation. Then the Jacobian learning rule is derived. The method is simpler and more general than tangent propagation (Simard, Victorri, Le Cun, & Denker, 1992). Through hybridization of the error backpropagation and the Jacobian learning, the hybrid learning algorithm is presented. The method shows good performance in accelerating the learning speed and improving generalization. And through computer experiments, it is shown that using the Jacobian synthesized from noise-corrupted data can accelerate learning speed.

1 Introduction

Multilayer feedforward neural networks can be trained to learn a continuous function with desired accuracy (Cybenko, 1989; Hornik, Stinchcombe, & White, 1989). Since the publication of Rumelhart, Hinton, & Williams (1986), many improved learning algorithms have been reported in the literatures, and various learning methods have been reviewed (Ergezinger & Thomsen, 1995; Jin, Nikiforuk, & Gupta, 1995; Baba, Mogami, Shiraishi, & Yamashita, 1995). The multilayer neural network, with various squashing functions, can approximate even first-order derivatives (Gallant & White, 1992; Hornik et al., 1990; Cardaliaguet & Evrard, 1992). The method of learning first-order derivatives has not been clearly explored, however, and among the various types of learning methods, the error backpropagation...
learning rule (delta learning rule) and its variants are widely accepted for their simplicity and relatively good performance in training multilayer neural networks. Consider the following mapping:

$$\Psi : X \mapsto Y \quad X \in \mathbb{R}^K \text{ and } Y \in \mathbb{R}^J.$$ (1.1)

Assume that we are to train the neural network with a sufficient number of mapping examples taken from $\Psi$. Suppose further that we have additional a priori information on the Jacobian of $\Psi$ even for a small number of the mapping examples:

$$\Phi : X \mapsto \frac{\partial Y}{\partial X} \quad X \in \mathbb{R}^K \text{ and } \frac{\partial Y}{\partial X} \in \mathbb{R}^{K \times J}.$$ (1.2)

In other words, suppose that we have some Jacobian at some data points, which is very important information on $\Psi$.

To see the effect of Jacobian learning on neural network properties, consider single input–single output case. Suppose that sufficient learning data $(x_i, \Psi(x_i))$ and validation data $(x_i + \delta x, \Psi(x_i + \delta x))$ are provided $(i = 1, 2, \ldots, n)$. Then the learning error($J^L$) and the validation error($J^V$) can be expressed as:

$$J^L = \frac{1}{n} \sum_{i=1}^{n} (\Psi(x_i) - \hat{\Psi}(x_i))^2$$ (1.3)
$$J^V = \frac{1}{n} \sum_{i=1}^{n} (\Psi(x_i + \delta x) - \hat{\Psi}(x_i + \delta x))^2$$ (1.4)

where $\hat{\Psi}(x)$ denotes neural network realization of $\Psi(x)$. In this case, if the neural network learned the plant with best generalization, then it is natural that we expect $J^L \simeq J^V$. Therefore, although various measures are suggested and investigated (Niyogi & Girosi, 1996; Freeman & Saad, 1995), as candidates for generalization measures, we suggest using $J^G = \frac{1}{n}(J^L - J^V)^2$, which can be expanded as:

$$J^G = \frac{1}{n} \sum_{i=1}^{n} ((\Psi(x_i) - \hat{\Psi}(x_i))^2 - (\Psi(x_i + \delta x) - \hat{\Psi}(x_i + \delta x))^2)$$

$$\simeq \frac{1}{n} \sum_{i=1}^{n} \left( \frac{\partial(\Psi(x) - \hat{\Psi}(x))}{\partial x} \bigg|_{x=x_i} \right)^2 \delta x^2$$

$$\simeq \frac{4}{n} \sum_{i=1}^{n} (\Psi(x_i) - \hat{\Psi}(x_i))^2 \left( \frac{\partial \Psi(x)}{\partial x} \bigg|_{x=x_i} - \frac{\partial \hat{\Psi}(x)}{\partial x} \bigg|_{x=x_i} \right)^2 \delta x^2. \quad (1.5)$$
In this equation, term-A is related to the mapping error and term-B to the Jacobian error. If term-A is small but term-B remains large after learning, we cannot expect a good generalization because \( f^G \) remains large. Although this analysis is rough, we can expect that better generalization can be obtained through additional Jacobian learning, which minimizes term-B in equation 1.5. Furthermore, because Jacobian represents the approximate inclination of mapping near a given point, we can expect that Jacobian learning can have the effect of learning several examples near the point, and thus we can expect fast learning speed through Jacobian learning.

This article focuses on the following subjects:

- A method of computing the Jacobian in a multilayer neural network. Computing the Jacobian in an efficient and explicit way is important because it is used to derive the Jacobian learning rule.

- A method of learning the Jacobian in multilayer neural networks to use the available Jacobian for neural network learning.

- An investigation of the effect of learning the Jacobian through computer experiments.

In section 2, we derive the method of computing the Jacobian in terms of neural network parameters, which becomes very efficient if we use the error backpropagation learning rule. The derived explicit equation is used to derive the Jacobian learning rule. In section 3, we derive the Jacobian learning rule that minimizes the Jacobian error between the Jacobian of the neural network and the Jacobian provided. In section 4, we develop a hybrid learning method that minimizes the cost of the mapping error and the Jacobian error. In section 5, we present some computer experiments to demonstrate the effect of Jacobian learning. And finally, in section 6, we draw some conclusions and discuss some related topics.

2 Computing the Jacobian of a Multilayer Neural Network

In this section, we derive the method for computing the Jacobian of a multilayer neural network while learning the mapping by error backpropagation. There has been some evidence that the Jacobian of a multilayer neural network can be calculated using the delta and other related quantities (Jordan & Rumelhart, 1992; White & Jordan, 1992; Iwata & Kitamura, 1994). We will summarize and arrange the method in a formal and explicit way. The equations for computing the Jacobian will be used for deriving the Jacobian learning rule. We begin by explaining the notations we use throughout the article. See Figure 1 for a simplified structure of the neural network. Let \( X, \hat{Y}, \) and \( Y \) denote the neural network input vector, the neural network output vector, and the output vector of the underlying mapping from \( X, \)
respective. We will use the following notations:

\[ X \equiv [x_1, x_2, \ldots, x_K]^t \]  \hspace{1cm} (2.1)

\[ Y \equiv [y_1, y_2, \ldots, y_J]^t \]  \hspace{1cm} (2.2)

\[ \hat{Y} \equiv [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_J]^t \]  \hspace{1cm} (2.3)

\[ \hat{y}_i = \phi_L^i (f_L^i + \theta_L^i) \equiv o_L^i \]  \hspace{1cm} (2.4)

\[ O^p \equiv [o_1^p, o_2^p, \ldots, o_{hp}^p]^t \]  \hspace{1cm} (2.5)

\[ o_j^p = \phi_p^j (f_p^j + \theta_p^j) \]  \hspace{1cm} (2.6)

\[ f_p^j = \sum_k o_{k-1}^p w_{jk}^p \]  \hspace{1cm} (2.7)

where \( K \) and \( J \) denote the dimension of \( X \) and \( Y \) (or \( \hat{Y} \)), respectively. \((L - 1), h_p, \phi_j^i, f_j^i, \theta_j^i, \) and \( o_j^i \) denote the number of hidden layers, number of nodes in the \( p \)th layer, the sigmoidal function of node \( j \) in layer \( i \), input to the sigmoidal function, and bias and output of node \( j \) in layer \( i \), respectively. Superscript \( t \) means the transpose of a vector or a matrix.

Let \( \Theta^k \) and \( W^k \) denote the bias input vector at the \( k \)th layer and the weight matrix in the \( k \)th layer, respectively. (The 0th layer denotes the input layer and the \( L \)th layer denotes the output layer in the \( L \) layered neural network.)

\[ \Theta^k \equiv [\theta_1^k, \theta_2^k, \ldots, \theta_{h_k}^k]^t \]  \hspace{1cm} (2.8)

\[ W^k \equiv \begin{bmatrix} w_{11}^k & w_{12}^k & w_{13}^k & \cdots & w_{1h_{k-1}}^k \\ w_{21}^k & w_{22}^k & w_{23}^k & \cdots & w_{2h_{k-1}}^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{h_1}^k & \cdots & \cdots & \cdots & w_{h_{h-1}}^k \end{bmatrix} \]  \hspace{1cm} (2.9)
Mapping and Its Jacobian

where \( w_{ji}^k \) denotes the weight from neuron \( i \) in the \((k-1)\)th layer to neuron \( j \) in the \( k \)th layer. The weight in the first layer implicitly means the weights from the input to the first hidden layer. Define the input vector to the \( h \)th layer node as

\[
F_h \equiv [f_{h1}, f_{h2}, \ldots, f_{hh}] = W_h O^{h-1},
\]

and define the normalized error as

\[
\bar{e}_q \equiv \text{col}_q(I_{J \times J})
\]

where \( \text{col}_q(A) \) denotes the \( q \)th column vector taken from the matrix \( A \), and \( I_{J \times J} \) denotes the \( J \times J \) identity matrix. Define the normalized delta as

\[
\bar{\delta}_{ij}^q = \begin{cases} 
\phi_i' j \bar{e}_q & \text{when layer } i \text{ is output} \\
\sum_m \bar{\delta}_{im} w_{mj}^{i+1} & \text{when layer } i \text{ is hidden}
\end{cases}
\]

\[
\bar{\Delta}_q^i = [\bar{\delta}_{q1}^i \bar{\delta}_{q2}^i \ldots \bar{\delta}_{qh}^i]^t
\]

where, in \( \bar{\delta}_{ij}^q \), \( i \) and \( j \) denote the layer number and the node number, respectively. The subscript \( q \) in \( \bar{\delta}_{ij}^q \) denotes \( \delta_{ij}^q \) (defined in equation 4.5) calculated by the delta learning rule when the error is set to \( \bar{e}_q \). The terms \( \bar{e}_q \) and \( \bar{\delta}_{ij}^q \) are introduced to take the multiple input, multiple output (MIMO) system into account. Now we can state the first theorem, which provides an easy way of computing Jacobian from neural network parameters.

**Theorem 1** Computing the Jacobian of a Multilayer Neural Network. Let \( X \), defined in equation 2.1, and \( \hat{Y} \), defined in equation 2.3, denote the input vector and the output vector of a multilayer neural network. Then the following relation holds,

\[
\frac{\partial \hat{Y}}{\partial X} = \begin{bmatrix}
(\bar{\Delta}_1^q)^t W_1 \\
(\bar{\Delta}_2^q)^t W_1 \\
\vdots \\
(\bar{\Delta}_h^q)^t W_1
\end{bmatrix}
\]

where the notations are given above. (See the appendix of this article for the proof.)

This theorem is known and used by some authors (Jordan & Rumelhart, 1992; White & Jordan, 1992; Iwata & Kitamura, 1994). Although they address the single-output case, their method can easily be extended to the MIMO case.
case. We summarize it to express the formula in an explicit way, because an explicit formula is indispensable for deriving the Jacobian learning rule.

It is clear from equations 2.12 and 2.14 that the Jacobian matrix should be computed row by row by setting the corresponding normalized error, which involves setting corresponding output to 1 at a time. If we compute the Jacobian under the online computation of the error backpropagation learning rule, \( \phi_i \) and \( \phi'_i \) are computed to adjust the weights. Therefore, it is not so burdensome to compute the Jacobian under the online learning by error backpropagation. If the mapping to be learned is the multiple-input, single-output (MISO), then \( \delta \) is identical with \( \delta/y - \hat{y} \), which can be computed at the final stage of error learning. In this case, the theorem can be stated in a simpler way:

\[
\frac{\partial \hat{y}}{\partial X} = (\Delta^1)^t W^1. \tag{2.15}
\]

(In this case, since the output is one, the subscript in \( \Delta \) is omitted.) The computation of the Jacobian can be simplified to one multiplication of the weight matrix \( W^1 \) and the vector \( \Delta^1 \).

3 Derivation of the Jacobian Learning Rule

Simard et al. (1992) showed that for MISO cases, gradient learning combined with error learning can improve learning speed. Their method requires computing tangent propagation. The method cannot handle missing data in Jacobian, because it requires gradient forward propagation and more computation time. In this section, we derive an efficient Jacobian learning rule that is applicable to MIMO cases and does not require gradient forward propagation in MISO cases. The learning rule can be derived by minimizing the following cost function:

\[
J_g \equiv \frac{1}{2} \sum_{kl} \left( g_{kl} - \hat{g}_{kl} \right)^2 \quad \text{where} \quad g_{kl} = \frac{\partial y_l}{\partial x_k} \quad \hat{g}_{kl} = \frac{\partial \hat{y}_l}{\partial x_k}. \tag{3.1}
\]

To express the Jacobian learning rule in a compact recursive form, define the following quantities:

\[
\mu_{kl}^{pq} = \begin{cases} 
1 & \text{when } i = 1 \\
\omega_{lp}^1 & \text{when } i = 2 \\
\sum_r \mu_{lr}^{(i-1)p} \omega_{lr}^{(i-1)} \psi_{r}^{(i-2)} & \text{when } i \geq 3
\end{cases} \tag{3.2}
\]

\[
d_{kq}^l = \sum_m \delta_{q}^{(i+1)m} \omega_{kl}^{i+1} \omega_{mk} \tag{3.3}
\]

In equations 3.2 and 3.3, the subscripts \( p \) and \( q \) denote that we are working to learn \( \partial y_q / \partial x_p \).
To derive the Jacobian learning rule, we use the steepest descent method, which is also used for backpropagation:

\[
\Delta w^m_{kl} = -\frac{1}{2} \eta \frac{\partial (g_{pq} - \hat{g}_{pq})^2}{\partial w^m_{kl}} \\
= \eta \frac{\partial \hat{g}_{pq}}{\partial w^m_{kl}} (g_{pq} - \hat{g}_{pq}). \tag{3.4}
\]

We need only to compute \( \frac{\partial \hat{g}_{pq}}{\partial w^m_{kl}} \) to derive \( \Delta w^m_{kl} \). For the case of \( m = 1 \) (first layer),

\[
\frac{\partial \hat{g}_{pq}}{\partial w^1_{kp}} = \frac{\partial \left( \sum \delta^1_q w^1_{ip} \right)}{\partial w^1_{kp}} \\
= \delta^1_k + \frac{\partial \hat{\delta}^1_k}{\partial w^1_{kp}} w^1_{kp} \\
= \delta^1_k + \eta \phi^1_k \delta^1_k. \tag{3.5}
\]

For the case of \( m = 2 \), similarly,

\[
\frac{\partial \hat{g}_{pq}}{\partial w^2_{kl}} = \frac{\partial \left( \sum \delta^1_q w^1_{ip} \right)}{\partial w^2_{kl}} \\
= \frac{\partial \left( \sum \psi^1_i \phi^1_i \sum \delta^2_q w^2_q \right)}{\partial w^2_{kl}} \\
= \psi^1_i \phi^1_i \delta^2_k + \sum \psi^1_i \phi^1_i \delta^2_k \frac{\partial \hat{\delta}^2_k}{\partial w^2_{kl}} w^2_{kl} \\
= \psi^1_i \phi^1_i \delta^2_k + \eta \phi^2_k \delta^2_k \frac{\partial \hat{\delta}^2_k}{\partial w^2_{kl}} w^2_{kl} \\
= \psi^1_i \phi^1_i \delta^2_k + \eta \phi^2_k \delta^2_k \frac{\partial \hat{\delta}^2_k}{\partial w^2_{kl}} w^2_{kl} \tag{3.6}
\]

Similarly, for the case of \( m = 3 \),

\[
\frac{\partial \hat{g}_{pq}}{\partial w^3_{km}} = \delta^3_k \phi^2_k \delta^2_j \sum \psi^1_i \phi^1_i \delta^2_k \frac{\partial \hat{\delta}^2_k}{\partial w^2_{kl}} w^2_{kl} \tag{3.7}
\]

We can obtain the weight adaptation rule for the Jacobian learning by substituting equations 3.5 through 3.7 into equation 3.4. For the biases, a similar adaptation rule for Jacobian learning can be obtained. In fact, the Jacobian...
leaning rule, including \( m = 1, 2, \) and 3, can be generalized as follows:

\[
\Delta_{g_{kl}}^{m} = \eta_{g} \left( \bar{\delta}_{mk}^{m} q_{\mu}^{l} \phi_{m}^{l} \right) \left( \psi_{q}^{m-1} \phi_{q}^{m-1} \phi_{m}^{l} \phi_{m}^{l} \right) (g_{pq} - \hat{g}_{pq}).
\] (3.8)

\[
\Delta_{s_{kl}}^{m} = \eta_{g} \left( \bar{\delta}_{mk}^{m} q_{\mu}^{l} \phi_{m}^{l} \right) \left( \psi_{q}^{m-1} \phi_{q}^{m-1} \phi_{m}^{l} \phi_{m}^{l} \right) (g_{pq} - \hat{g}_{pq}).
\] (3.9)

In equation 3.8, \( o_{l}^{0} \) means the inputs of neural network, \( x_{l} \). The derivation of the Jacobian learning rule for the general case in equations 3.8 and 3.9 can be found in the appendix. In equation 3.8, \textit{term1} represents the effect of the weight variation to the Jacobian of the neural network in forward direction. And \textit{term2} represents the effect of variation of \( \bar{\delta} \) in the \( m \)th layer neuron. Because \( \bar{\delta} \) affects the Jacobian of the neural network, \textit{term2} cannot be neglected. As can be seen in the above Jacobian learning rule, the \( i \)th layer of \( \mu \) in equation 3.2 is computed using their value at the \((i-1)\)th layer, so the direction of Jacobian learning is forward. The terms needed for computing the Jacobian, that is, \( \bar{\delta} \) in equations 4.5 and 2.13, are computed in a backward direction. This is the reverse direction to that of error learning.

4 Hybrid Learning of Mapping and Its Jacobian

If we set the cost function as follows, we can derive the weight adaptation rule, which minimizes the mapping error and the Jacobian error:

\[
J = \lambda_{e} J_{e} + \lambda_{g} J_{g}
\] (4.1)

where \[
J_{e} = \frac{1}{2} \sum_{k} (y_{k} - \hat{y}_{k})^{2}.
\] (4.2)

In equation 4.1, \( \lambda_{e} \) and \( \lambda_{g} \) denote the weighting coefficients for the mapping error and the Jacobian error, which may vary with iteration of learning. Weight correction, \( \Delta w_{kl}^{m} \), to the direction that minimizes \( J \) is assumed to be decomposed as

\[
\Delta w_{kl}^{m} = \lambda_{e} \Delta_{e} w_{kl}^{m} + \lambda_{g} \Delta_{g} w_{kl}^{m},
\] (4.3)

where \( \Delta_{e} w_{kl}^{m} \) denotes the weight correction to the direction that minimizes \( J_{e} \), and \( \Delta_{g} w_{kl}^{m} \) denotes the weight correction to the direction that minimizes \( J_{g} \). \( \Delta_{e} w_{kl}^{m} \) is calculated according to error backpropagation (Rumelhart et al., 1986),

\[
\Delta_{e} w_{kl}^{m} = -\eta_{e} \frac{\partial J_{e}}{\partial w_{kl}^{m}} = \eta_{e} \bar{\delta}_{l}^{m-1} o_{l}^{m-1}.
\] (4.4)
where $\delta$ is defined as

$$
\delta^m_k = \begin{cases} 
\epsilon^k w^m_k & \text{when neuron } k \text{ is output, } y_k = \hat{y}_k \\
\sum_l \delta^m_{l+1} w^m_{lk} + 1 & \text{when neuron } k \text{ is hidden.} \end{cases} \tag{4.5}
$$

The following steps are required for the hybrid learning of the mapping and its Jacobian:

1. Compute neural network outputs($\hat{y}_j$) and mapping errors in the forward direction.
2. Compute $\bar{\delta}$ and $w$ in the backward direction.
3. Compute Jacobian errors ($E_g$) at the end of step 2.
4. Correct weights according to equation 4.3 while computing $\bar{\delta}$ and $w$ in the forward direction.

The weight corrections $\lambda_e \Delta_e W$ and $\lambda_g \Delta_g W$ can conflict with each other. As shown in Figure 2, if the angle between $\lambda_e \Delta_e W$ and $\lambda_g \Delta_g W$ is relatively large and their magnitudes are approximately same, then the resultant weight correction, $\Delta W$, can be the direction that can minimize neither $J_e$ nor $J_g$. In this case, tuning the weighting coefficients $\lambda_e$ and $\lambda_g$ is recommended. If one is to weight the error, then it is better to increase $\lambda_e$ relative to $\lambda_g$. But in many cases, the weighting coefficients can be set to constant because, after some iteration of learning, the two directions of the weight correction can approximately coincide with each other. If $\lambda_e$ is very small compared with $\lambda_g$, then almost all learning is done for Jacobian learning. In this case, the learned neural network will have a function shape very similar to the underlying mapping, but the mapping error may be large. Clearly the tuning of the two weighting coefficients is important.
5 Computer Experiments

5.1 Experiment 1: Comparison of Learning Speed. In this experiment, we choose static mapping with one relatively sharp corner as an underlying mapping. The mapping to be learned by the neural network is generated by

\[ y = -13(x - 0.5)^2 + 8 \exp\left(\frac{(x - 0.05)^2}{0.01}\right). \]  \hspace{1cm} (5.1)

The shape of the mapping to be learned is shown in Figure 3. The neural networks are constructed as follows:

- One hidden layer with 60 neurons; bias inputs to hidden and output nodes.
- Squashing function: Hyperbolic tangent for hidden layers, linear output node.
- Error learning rate, \( \eta_e = 0.01 \); Jacobian learning rate, \( \eta_g = 0.001 \).
- Number of iterations for learning: 2000 epochs. For each epoch, 80 samples are presented.
- Initial weights: Random number between \(-1\) and 1.
- Learning examples: Chosen randomly within the range of \(-0.3 \leq x \leq 1.3\).
- Weighting coefficients \( \lambda_e \) and \( \lambda_g \): Set to unity.

No additional techniques are used to accelerate the learning speed. To compare the differences in the learning properties easily, we set the topology of the neural network and the learning rate identical for error backpropagation and hybrid learning. The only difference is whether Jacobian learning is added. The neural network mapping error \( J_e \) is shown in Figure 4. The top figure is the result of conventional error backpropagation, and the bottom figure is that of hybrid learning. The middle figure is the sum of the squared error \( J_e \) by assuming that the Jacobian is available on 50 percent of learning data.

Careful investigation of the results leads to the following conclusions:

- Even we take the computation time of hybrid learning (almost twice that of error backpropagation in this experiment) into consideration, learning is far more accelerated with the additional Jacobian learning.
- The result of the hybrid learning improves as more Jacobian information is available.

5.2 Experiment 2: Use of the Jacobian Synthesized from Noise-Corrupted Data. In this experiment, we investigate the possibility of using the
Jacobian synthesized from the noise-corrupted data, a similar experiment with a real-world problem. In real data sets, the Jacobian may not be provided, and even the data set may be corrupted by measurement noise. Using the data set in experiment 1, 80 noise-corrupted samples are generated by the following equation:

$$y_i = -13(x_i - 0.5)^2 + 8 \exp \left( \frac{- (x_i - 0.05)^2}{0.01} \right) + 0.5 \times \text{randn} \quad i = 1, 2, \ldots, 80$$

$$x_i = -0.3 + 0.2i \quad i = 1, 2, \ldots, 80$$

where \(\text{randn}\) denotes normal gaussian noise. The underlying mapping and noise-corrupted data are shown in Figure 5. To extract the rough Jacobian, we use the following simple formula:

$$g_i = \frac{1}{2} \left( \frac{y_i - y_{i-1}}{x_i - x_{i-1}} + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \right).$$

(5.2)

Because the Jacobian obtained in this way is very noisy, we use the following simple averaging filter:

$$g_i \text{ filtered} = \frac{g_{i-1} + 2g_i + g_{i+1}}{4}.$$  

(5.3)

Figure 6 shows the added noise (top), the synthesized Jacobian with equation 5.2 (middle panel,) and the filtered Jacobian (bottom panel). Because the
Figure 4: Neural network mapping error ($J_e$) during 2000 epochs of learning in experiment 1. Top: $\mu_e = 0.01$, $\mu_g = 0$. No Jacobian learning. Middle: $\mu_e = 0.01$, $\mu_g = 0.001$. Jacobian learning for 50 percent of the examples. Bottom: $\mu_e = 0.01$, $\mu_g = 0.001$. Jacobian learning for all examples.
Figure 5: Plot of the underlying mapping (smooth curve) and the noise-corrupted data set in experiment 2.

synthesized Jacobian is not the exact Jacobian of the underlying mapping, learning of filtered Jacobian at the fine-tuning stage is undesirable and may lead to an inferior model, so we tune the learning rate as follows:

$$\eta_g = \begin{cases} 0.001 & 0 < \text{epoch} \leq 500 \\ \text{decreased linearly from 0.001 to 0.} & 500 < \text{epoch} \leq 1000 \\ 0 & 1000 < \text{epoch} \leq 2000 \end{cases}$$ (5.4)

$$\eta_e = \begin{cases} 0.01 & 0 < \text{epoch} \leq 1000 \\ \text{decreased linearly from 0.001 to 0.} & 1000 < \text{epoch} \leq 2000 \end{cases}$$ (5.5)

All the other conditions except the learning rate are the same as in experiment 1. The neural network mapping error ($L_e$) is shown in Figure 7. Although we believe that hybrid learning requires two passes of weight updates, the result shows that the learning is much faster, by three to four times, compared with that of error backpropagation. This experiment indicates that the use of the filtered Jacobian synthesized from the raw data can be effective in accelerating learning; however, the structure of the filter should vary according to the characteristics of the given data set.

5.3 Experiment 3: Suppression of Overfitting. In this experiment, we investigate the ability to prevent overfitting by additional Jacobian learning.
For this purpose, the experiment is designed to fit a straight line with a small number of examples. Experimental conditions were as follows:

- For the neural network, there was one hidden layer with 100 neurons of hyperbolic tangent with bias and one linear output node with bias. The error learning rate was 0.0005, and the Jacobian learning rate was 0.0005. There were 20,000 iterations for learning.

- Seven sample data and seven validation data, which were noise corrupted, were generated by the following equations:

\[
\begin{align*}
    x_i &= 0.4i + \text{rand} \quad i = 1, 2, \ldots, 7 \\
    y_i &= 0.5x_i + \text{rand} \quad i = 1, 2, \ldots, 7
\end{align*}
\]

where \( \text{rand} \) denotes the random number between \(-1\) and \(1\).

- Learning examples were shown in a random sequence.
• For each learning of the sample points in Jacobian learning, the noise-corrupted Jacobian was provided by the following equation:

$$g = 0.5 + \text{randn} \times 0.3$$  \hspace{1cm} (5.6)

where \text{randn} denotes normal gaussian noise.

• The weighting coefficients were set to unity.
No other modifications were attempted so that we could see the effects of the Jacobian learning more clearly. As in the previous experiment, the same topology of the neural network was used, and the only difference was whether the Jacobian learning was added. The learning error ($J_L$) and validation error ($J_V$) versus learning epoch are shown in Figures 8 and 9. The result of neural network learning and the average of the squared distance between the neural network and underlying mapping ($y = 0.5x$) is also shown. The learning error ($J_L$) and validation error ($J_V$) are computed by summing the square of error at the points. The averages of $J_L$ and $J_V$ in the last 100 iterations for error learning are 0.51 and 4.41, respectively. Those for hybrid learning are 1.88 and 3.38, respectively. The difference between $J_L$ and $J_V$ in hybrid learning (1.50) is smaller than that in error learning (3.90). A comparison of the two sets of figures shows that the mean distance between the neural network and underlying mapping in error backpropagation is larger than that in hybrid learning. Although the learning error for learning samples in hybrid learning is larger than that in error backpropagation, there is better generalization. These observations support the hypothesis set out in section 1 regarding the effect of Jacobian learning on generalization.

6 Conclusions

We have proposed an algorithm that learns the mapping and its Jacobian. For this purpose, we derived the explicit form of computing the Jacobian using the neural network parameters, and we derive the hybrid learning algorithm that minimizes the cost function consisting of mapping error and Jacobian error. The proposed Jacobian learning method is similar to that of the error backpropagation learning rule but in the opposite direction of calculation.

Computer experiments proved that the proposed method is faster than conventional error backpropagation, because the proposed hybrid learning algorithm learns to minimize both the mapping error and the Jacobian error simultaneously. We show as well the possibility that using the rough Jacobian synthesized from noise-corrupted raw data set can accelerate learning speed and that Jacobian learning improves generalization through computer experiment.

Appendix

A.1 Proof of Theorem 1.

Proof. From the definition of the feedforward neural network (at output layer),

$$\hat{y}_j = \phi_j^L(f_j^L + \theta_j^L)$$
Figure 8: Experiment 3: Error backpropagation learning. Upper left: Learned result of neural network. Dotted line: underlying mapping, o: validation data, *: sample learning data. Upper right: Mean squared distance between neural network and underlying mapping ($-0.5 < x < 6.5$). Lower left: Sum of squared error for learning data. Lower right: Sum of squared error for validation data.

\[
\hat{y}_j = \sum_k w^L_{jk} o^{L-1}_k + \theta^L_j \\
= \psi^L_j \left( \sum_k w^L_{jk} \psi^{L-1}_k \left( f^{L-1}_k + \theta^{L-1}_k \right) + \theta^L_j \right) \\
= \psi^L_j \left( \sum_k w^L_{jk} \psi^{L-1}_k \left( \sum_i w^{L-1}_{ki} o^{L-2}_i + \theta^{L-1}_k \right) + \theta^L_j \right).
\]

We differentiate the outputs directly by $o^{l-2}_i$ to get

\[
\frac{\partial \hat{y}_j}{\partial o^{l-2}_i} = \psi^L_j \sum_k (w^L_{jk} \psi^{L-1}_k \frac{d^{L-1}_k}{d o^{l-2}_i}) \equiv \hat{S}^{l-2}_j.
\]
Figure 9: Experiment 3: Hybrid learning. Upper left: Learned result of neural network. Dotted line: underlying mapping; o: validation data; *: sample learning data. Upper right: Mean squared distance between neural network and underlying mapping (−0.5 < x < 6.5). Lower left: Sum of squared error for learning data. Lower right: Sum of squared error for validation data.

At the output layer, normalized delta is expressed as

$$\bar{\Delta}_m^{L} = \left[ 0, \ldots, 0, \varphi_m^{L'}, 0, \ldots, 0 \right]^T,$$

(A.2)

in which $\varphi_m^{L'}$ is the first-order derivative of the activation function of the $m$th element and the superscript $L$ denotes the output layers. From equation 2.12,

$$\delta_m^{(L-1)k} = \psi_k^{(L-1)'} \sum_j \bar{z}_m^{Lj} w_{jk}^{L}$$

$$= \psi_k^{(L-1)'} \varphi_m^{L'} w_{mk}^{L}$$

(A.3)
and

\[ \tilde{\Delta}^{(L-1)}_m = \left[ \tilde{\delta}^{(L-1)}_m, \tilde{\delta}^{(L-1)2}_m, \ldots, \tilde{\delta}^{(L-1)h_{L-1}}_m \right]^t \]

\[ \equiv \left[ \varphi^{(L-1)'}_m w^{(L-1)1}_m \varphi^{(L-1)'}_m w^{(L-1)2}_m \cdots \varphi^{(L-1)'}_m w^{(L-1)h_{L-1}}_m \right]^t. \quad (A.4) \]

where \( h_{L-1} \) is the number of nodes in the \((L - 1)\)th hidden layer. If we apply the theorem, assuming \((L - 1)\) to be the only hidden layer and \((L - 2)\) the input layer,

\[ ((W^{(L-1)}')_m \tilde{\Delta}^{(L-1)}_m)^t \]

\[ = (\tilde{\Delta}^{(L-1)}_m)^t \left[ \begin{array}{cccc}
  w^{(L-1)1}_{11} & \cdots & w^{(L-1)1}_{1h_{L-2}} \\
  w^{(L-1)2}_{21} & \cdots & \cdots \\
  \vdots & \ddots & \vdots \\
  w^{(L-1)h_{L-1,1}}_{h_{L-1,1}} & \cdots & w^{(L-1)h_{L-1,h_{L-2}}}_{h_{L-1,h_{L-2}}} 
\end{array} \right] 
\]

\[ \equiv \tilde{G}^{L-2}_m = \left[ \tilde{\delta}^{L-2}_1, \tilde{\delta}^{L-2}_2, \ldots, \tilde{\delta}^{L-2}_{h_{L-2}} \right]. \quad (A.5) \]

Equation A.5 denotes the gradient of \( \tilde{y}_m \) with respect to \( O^{L-2} \). If we assume that the only hidden layer is \((L - 1)\) and layer \((L - 2)\) is the input layer, equation A.5 is equivalent to the Jacobian of the output vector with respect to the input vector. If we rewrite \( \tilde{G}^{L-2}_m \) in equation A.5 using equation A.4,

\[ \tilde{\delta}^{L-2}_{\tilde{\eta}_j} = \varphi^{(L-1)'}_j \left( \sum_k w^{(L-1)1}_{ki} w^{(L-1)k}_j \varphi^{(L-1)k}_j \right). \quad (A.6) \]

Equations A.1 and A.6 are identical for all possible choice of \( j \). This proves the theorem when only one hidden layer exists in the neural network.

Next, we prove that if

\[ \frac{\partial \tilde{y}_j}{\partial O^l} = (\tilde{\Delta}^{l+1}_j)^t W^{l+1} \quad (A.7) \]

is true, then

\[ \frac{\partial \tilde{y}_j}{\partial O^{l-1}} = (\tilde{\Delta}^{l}_j)^t W^l \quad (A.8) \]

also holds. This means that if the theorem holds for \((L - l - 1)\) hidden layers, it holds for \((L - l)\) hidden layers. If we extract one component \( \frac{\partial \tilde{y}_j}{\partial O^{l-1}} \) from equation A.8,

\[ \frac{\partial \tilde{y}_j}{\partial O^{l-1}} = \sum_k \tilde{\delta}^{l+1}_j w^{l+1}_{ki} \]
\[ \sum_{k} \left( \psi_{j}^{l} w_{j}^{l} \sum_{m} \tilde{\delta}_{j}^{l(m)} w_{mk}^{l+1} \right). \tag{A.9} \]

And if we apply the chain rule and use equations 2.5 through 2.7,

\[ \frac{\partial \hat{y}_{i}}{\partial o_{l}^{-1}} = \frac{\partial \hat{y}_{i}}{\partial O^{l}} \frac{\partial O^{l}}{\partial o_{l}^{-1}} \quad \text{(using equation A.7)} \]

\[ = (\tilde{\Delta}_{j}^{l+1})^{T} W^{l+1} \frac{\partial O^{l}}{\partial o_{l}^{-1}}. \tag{A.10} \]

Furthermore, the following two equations hold:

\[ (\tilde{\Delta}_{j}^{l+1})^{T} W^{l+1} = \begin{bmatrix} \sum_{m} w_{m1}^{l+1} \tilde{\delta}_{i}^{l+1} \\
\sum_{m} w_{m2}^{l+1} \tilde{\delta}_{i}^{l+1} \\
\vdots \\
\sum_{m} w_{ml}^{l+1} \tilde{\delta}_{i}^{l+1} \end{bmatrix}. \tag{A.11} \]

\[ \frac{\partial O^{l}}{\partial o_{l}^{-1}} = \begin{bmatrix} \phi_{1}^{l} w_{1}^{l} & \phi_{2}^{l} w_{2}^{l} & \ldots & \phi_{h_{l}}^{l} w_{h_{l}}^{l} \end{bmatrix}^{T}. \tag{A.12} \]

If we substitute equations A.11 and A.12 into equation A.10, then we can easily see the result is identical with equation (A.9). Thus, by induction, the theorem is proved for any number of hidden layers.

**A.2 Derivation of Jacobian Learning Rule for General Case.** In the following derivation, we use the notation that, in subscript \(i_{m}, \) \(m\) is a variable. For example, if \(m = 3,\) subscript \(i_{m-1}\) denotes subscript \(i_{2}\). For any \(m,\) we can rewrite \(\hat{g}_{pq}\) as

\[ \hat{g}_{pq} = \sum_{i_{p}} \tilde{\delta}_{p}^{q+1} w_{i_{p}p}^{q+1}. \]

\[ = \sum_{i_{1}} \psi_{1}^{q+1} w_{i_{1}p}^{q+1} \sum_{i_{2}} \tilde{\delta}_{q}^{i_{2}} w_{i_{2}i_{1}}^{i_{2}} \]

\[ \quad \ldots \ldots \]

\[ = \sum_{i_{1}} \psi_{1}^{q+1} w_{i_{1}p}^{q+1} \sum_{i_{2}} \psi_{2}^{q+1} w_{i_{2}i_{1}}^{i_{2}} \ldots \]

\[ \sum_{i_{m-1}} \psi_{m-1}^{q+1} w_{i_{m-1}i_{m-2}}^{q+1} \sum_{i_{m}} \tilde{\delta}_{q}^{i_{m}} w_{i_{m}i_{m-1}}^{i_{m}}. \tag{A.13} \]

And from equation 2.12,

\[ \frac{\partial \hat{g}_{q}^{\hat{y}_{j}}}{\partial w_{j}^{l+1}} = \phi_{l}^{j} \sum_{t} \tilde{\delta}_{q}^{l+1} w_{t}^{l+1}. \tag{A.14} \]
Then we can write:

\[
\frac{\partial \delta_j}{\partial \bar{w}_{kl}} = \delta_{mkl} \sum_{t_1} \psi_{i_1} \bar{w}_{t_1} \bar{q}_{i_2} \bar{w}_{i_2} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{q}_{l_{n-2}}}{\partial \bar{w}_{kl}}
\]

\[
+ \sum_{t_1} \psi_{i_1} \bar{w}_{t_1} \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{q}_{l_{n-2}}}{\partial \bar{w}_{kl}}
\]

\[
= \bar{q}_{kl} \sum_{t_1} \psi_{i_1} \bar{w}_{t_1} \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{q}_{l_{n-2}}}{\partial \bar{w}_{kl}}
\]

\[
+ \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{q}_{l_{n-2}}}{\partial \bar{w}_{kl}}
\]

\[
= \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{q}_{l_{n-2}}}{\partial \bar{w}_{kl}}
\]

If we substitute equation A.15 into equation 3.4, we can obtain equation 3.8. Similar derivations can be made for equation 3.9. From equation 2.12,

\[
\frac{\partial \delta_k}{\partial \theta_k} = \psi_k \sum_t \bar{q}_{t} \bar{w}_{t+1}.
\]  

(A.15)

(A.16)

Using equations A.13 and A.16, we can write:

\[
\frac{\partial \hat{\delta}_{ij}}{\partial \theta_k} = \frac{\partial \hat{\delta}_{ij}}{\partial \theta_k} = \sum_{l_1} \psi_{l_1} \bar{w}_{l_1} \bar{q}_{l_2} \bar{w}_{l_2} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{w}_{l_{n-2}}}{\partial \theta_k}
\]

\[
= \sum_{l_1} \psi_{l_1} \bar{w}_{l_1} \bar{q}_{l_2} \bar{w}_{l_2} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{w}_{l_{n-2}}}{\partial \theta_k}
\]

\[
= \psi_k \sum_{l_1} \psi_{l_1} \bar{w}_{l_1} \bar{q}_{l_2} \bar{w}_{l_2} \cdots \sum_{l_{n-2}} \psi_{l_{n-2}} \bar{w}_{l_{n-2}} \psi_{l_{n-2} \cdots l_{n-2}} \frac{\partial \bar{w}_{l_{n-2}}}{\partial \theta_k}
\]

If we substitute equation A.17 into equation 3.4, we can obtain equation 3.9.

References


Received February 6, 1996; accepted September 10, 1996.