On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application

Eiji Mizutani [1,2], *Stuart E. Dreyfus* [1], *and Kenichi Nishio* [3]

eiji@biosys2.me.berkeley.edu, dreyfus@ieor.berkeley.edu, nishio@cv.sony.co.jp

1) Dept. of Industrial Engineering and Operations Research, Univ. of California at Berkeley, CA 94720, USA

2) Sony US Research Laboratories, 3300 Zanker Road, San Jose CA 95134, USA

3) Sony Corp. Personal IT Network Company, Tower C, 2-15-3 Kohnan Minato-ku, Tokyo 108-6201, JAPAN

Abstract

The well-known **backpropagation** (**BP**) derivative computation process for **multilayer perceptrons** (**MLP**) learning can be viewed as a simplified version of the Kelley-Bryson gradient formula in the classical discrete-time optimal control theory [1]. We detail the derivation in the spirit of **dynamic programming**, showing how they can serve to implement more elaborate learning whereby teacher signals can be presented to any nodes at any hidden layers, as well as at the terminal output layer.

We illustrate such an elaborate training scheme using a small-scale industrial problem as a concrete example, in which some hidden nodes are taught to produce specified target values. In this context, part of the hidden layer is no longer "hidden."

1 Introduction

Backpropagation has been a core procedure for computing derivatives in MLP learning, since Rumelhart et al. formulated it specially geared to MLPs in 1980s [2]. Similar formulations were derived by other individuals; often, Werbos's Ph.D thesis [3] is cited alone, but it is well-known that the theoretical framework of BP can be explained clearly in light of the classical optimal control theory [4, 1, 5].

The partial-derivative form of **sensitivity** is computed backward one stage after another from the terminal stage. In MLP terminology, this is the so-called **generalized delta rule**, which is equivalent to, what we call, the Kelley-Bryson formula, notably due to Kelley's *adjoint equations* [6] and Bryson's *multiplier rule* [7]. Also, in 1962, Dreyfus derived a *dynamic-programming-like recursive gradient formulation* [8, 9], which is almost identical to the *generalized delta rule*, for solving the multi-stage Mayer-type optimal control problem (see Eq. (5.4) in ref. [8]).

In conformity with the optimal control theory, we consider the following objective cost function:

$$E(\boldsymbol{\theta}) = \sum_{s=1}^{N-1} g^s(\mathbf{a}^s, \boldsymbol{\theta}^s) + h(\mathbf{a}^N), \tag{1}$$

where $\mathbf{a}^s$ and $\boldsymbol{\theta}^s$ signify the *state vector* and *decision vector* at stage s, and $g^s(.,.)$ and $h(.)$ denote the *immediate cost per stage s* and the *terminal cost*, respectively. In the MLP-learning, $g^s(.,.)$ usually drops off, and the $h(.)$ is often expressed as the sum of squared errors, resulting in the **neural networks nonlinear least-squares problem** [10, 11]. In this paper, we consider a case where both g and h have the squared-error cost measure between desired outputs and the MLP's outputs with a given set of m training data pairs; in this setup, $g^s(\mathbf{a}^s, \boldsymbol{\theta}^s)$ becomes independent of $\boldsymbol{\theta}^s$ [see Equation (3)].

The terminal cost function $h(.)$ at the final layer N, which consists of n_N neurons, can be expressed in a usual manner:

$$\begin{aligned} h(\mathbf{a}^N) &= \tfrac{1}{2}\sum_{p=1}^{m}\sum_{k=1}^{n_N}(t_{k,p}^N - a_{k,p}^N)^2 \\ &= \tfrac{1}{2}\sum_{p=1}^{m}\|\mathbf{t}_p^N - \mathbf{a}_p^N\|_2^2 \\ &= \tfrac{1}{2}\sum_{p=1}^{m}\|\mathbf{t}_p^N - f^N(\mathbf{a}_p^{N-1}, \boldsymbol{\theta}^{N-1})\|_2^2 \\ &= \tfrac{1}{2}\mathbf{r}_N^T(\boldsymbol{\theta})\mathbf{r}_N(\boldsymbol{\theta}), \end{aligned} \tag{2}$$

where $t_{k,p}^N$ is the kth desired output for the pth training data pattern at the terminal layer N; $a_{k,p}^N$ is the kth activation (i.e., output of the kth neuron function f) for the pth data; $\mathbf{t}_p^N$ is the desired output vector; $\mathbf{a}_p^N$ is the activation vector; and $\mathbf{r}_N(\boldsymbol{\theta})$ denotes the residual vector composed of $r_i(\boldsymbol{\theta})$, $i = 1, \ldots, mn_N$.

The immediate cost g at stage s is defined in a similar fashion:

$$\begin{aligned} g^s(\mathbf{a}^s, \boldsymbol{\theta}^s) &= \tfrac{1}{2}\sum_{p=1}^{m}\sum_{k=1}^{n_s}(t_{k,p}^s - a_{k,p}^s)^2 \\ &= \tfrac{1}{2}\sum_{p=1}^{m}\|\mathbf{t}_p^s - \mathbf{a}_p^s\|_2^2 \\ &= \tfrac{1}{2}\sum_{p=1}^{m}\|\mathbf{t}_p^s - f^s(\mathbf{a}_p^{s-1}, \boldsymbol{\theta}^{s-1})\|_2^2. \end{aligned} \tag{3}$$

In words, $g^s(\mathbf{a}^s, \boldsymbol{\theta}^s)$ is the immediate cost at stage s, or at the sth (hidden) layer, between the (hidden) activations $\mathbf{a}^s$ and the desired values $\mathbf{t}^s$. The n_s neurons at the sth hidden layer are supervised and expected to produce specific target values $\mathbf{t}^s$. We call this **hidden-node teaching**. Notice that the hidden nodes at layer s to be supervised can be a subset of all n_s nodes at layer s; we shall demonstrate such a "*partial* hidden-node teaching" later in Section 3.

In the rest of this paper, the subscript p will be omitted for simplicity, and the *activation* (e.g., at node j at layer s) is expressed in several forms, as shown next:

$$\begin{aligned} a_j^s &= f^s(\mathbf{a}^{s-1}, \boldsymbol{\theta}^{s-1}) \\ &= f^s\left(\sum_{i=1}^{n_{s-1}} a_i^{s-1}\theta_{ij}^{s-1}\right) \\ &= f^s(net_j^s), \end{aligned}$$

where net_j^s denotes the **net input** to the jth neuron at the sth (hidden) layer (see Figure 1). In the next section, we show how the decision-making scheme for optimal control plays an important role in MLP learning. We then advance to its practical application.

2 Backpropagation

We first summerize the terminologies in the next table:

Notation	Optimal Control	Neural Network
θ	decision variables	weight parameters
a	state variables	(neuron) activation
s	stage	layer
δ	adjoint variables Lagrange multipliers costates	delta

It should be noted that the best-known BP formulation due to Rumelhart et al. was made by choosing the *net input* but not the *activation* as the *state* variable; hence, it looks slightly different.

Using the aforementioned notations, we describe the derivation of BP for the objective cost function in Equation (1) in the spirit of *dynamic programming*; we define "value function," "recurrence relation," and "boundary condition" subsequently:

Value function:

$$T^s(\mathbf{a}^s, \boldsymbol{\theta}^s) \overset{\text{def}}{=} \text{cost-to-go, starting at state } \mathbf{a}^s \text{ at stage s, using a guessed policy } \boldsymbol{\theta}^s. \tag{4}$$

The initial guessed policy is often given by a randomly-initialized weight-parameter set.

Recurrence relation:

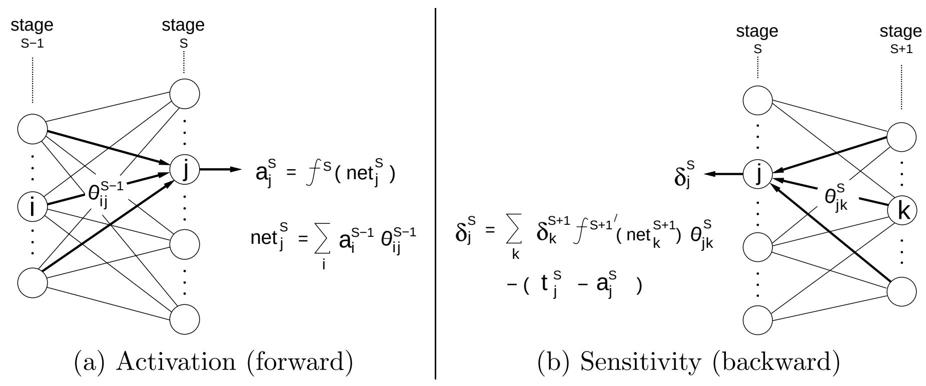
$$\begin{aligned} T^s(\mathbf{a}^s, \boldsymbol{\theta}^s) &= T^{s+1}(\mathbf{a}^{s+1}, \boldsymbol{\theta}^{s+1}) + g^s(\mathbf{a}^s, \boldsymbol{\theta}^s) \\ &= T^{s+1}(f^{s+1}(\mathbf{a}^s, \boldsymbol{\theta}^s), \boldsymbol{\theta}^{s+1}) + g^s(\mathbf{a}^s, \boldsymbol{\theta}^s), \end{aligned} \tag{5}$$

where $g(.,.)$ denotes the immediate cost, as shown in Equation (3).

Boundary condition:

$$T^N(\mathbf{a}^N, -) = h(\mathbf{a}^N) = \frac{1}{2}\sum_{k=1}^{n_N}\|\mathbf{t}_k^N - \mathbf{a}_k^N\|_2^2, \tag{6}$$

which corresponds to Equation (2).



(a) Activation (forward) (b) Sensitivity (backward)

Figure 1: *Forward and backward propagations in the MLP. In our notation, θ_{jk}^s denotes the weight of neuron j at layer s to neuron k at layer (s+1).*

In this setting, the **sensitivity** δ at the terminal stage is given by

$$\delta_k^N \overset{\text{def}}{=} \frac{\partial T^N}{\partial a_k^N} = -(t_k^N - a_k^N).$$

Departing from this *terminal* value at layer N, the sensitivity δ is backpropagated and computed backward one after another by the the following recurrence relation, our version of the **delta-rule**:

$$\begin{aligned}
\delta_j^s &\overset{\text{def}}{=} \frac{\partial T^s}{\partial a_j^s} \\
&= \frac{\partial T^{s+1}(\mathbf{a}^{s+1}, \boldsymbol{\theta}^{s+1})}{\partial a_j^s} + \frac{\partial g^s}{\partial a_j^s} \quad \text{(due to the recurrence relation)} \\
&= \sum_{k=1}^{n_{s+1}} \frac{\partial T^{s+1}}{\partial a_k^{s+1}} \frac{\partial a_k^{s+1}}{\partial a_j^s} + \frac{\partial g^s}{\partial a_j^s} \\
&= \sum_{k=1}^{n_{s+1}} \frac{\partial T^{s+1}}{\partial a_k^{s+1}} \frac{\partial f^{s+1}(\sum_{i=1}^{n_s} a_i^s \theta_{ik}^s)}{\partial a_j^s} - (t_j^s - a_j^s) \\
&= \sum_{k=1}^{n_{s+1}} \frac{\partial T^{s+1}}{\partial a_k^{s+1}} \; f^{s+1'}(net_k^{s+1})\;\theta_{jk}^s - (t_j^s - a_j^s) \\
&= \sum_{k=1}^{n_{s+1}} \delta_k^{s+1} \; f^{s+1'}(net_k^{s+1})\;\theta_{jk}^s - (t_j^s - a_j^s).
\end{aligned}$$

Of course, the last term (for hidden-node teaching) would be omitted for any node not being supervised.

The parameter-updating formula can be written with a learning rate η as:

$$\begin{aligned}
\Delta\theta_{jk}^s &= -\eta \frac{\partial T^s}{\partial \theta_{jk}^s} \\
&= -\eta \frac{\partial T^{s+1}(\mathbf{a}^{s+1}, \boldsymbol{\theta}^{s+1})}{\partial \theta_{jk}^s} \\
&\quad \text{(due to the recurrence relation and the fact that } g^s \text{ is independent of } \boldsymbol{\theta}^s) \\
&= -\eta \frac{\partial T^{s+1}(f^{s+1}(\sum_{i=1}^{n_s} a_i^s \theta_{ik}^s), \boldsymbol{\theta}^{s+1})}{\partial \theta_{jk}^s} \\
&= -\eta \frac{\partial T^{s+1}}{\partial a_k^{s+1}} \frac{\partial a_k^{s+1}}{\partial \theta_{jk}^s}
\end{aligned}$$

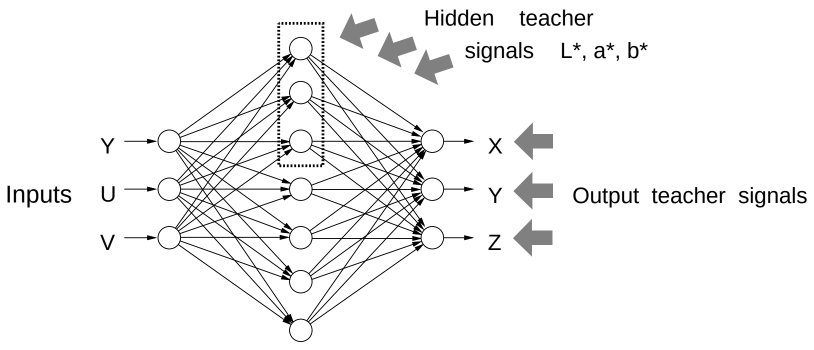


Figure 2: *An MLP architecture with partial hidden-node teaching. The three neurons, marked by the dotted rectangle, at the hidden layer are instructed to produce specified target values.*

$$
\begin{aligned}
&= -\eta\, \delta_k^{s+1}\, f^{s+1'}(net_k^{s+1})\, a_j^s \\
&\quad \text{If s+1 = N, then ...} \\
(\; &= -\eta\, \delta_k^{N}\, f^{N'}(net_k^{N})\, a_j^{N-1} \;) \\
(\; &= -\eta \left[-(t_k^N - a_k^N)\right]\, f^{N'}(net_k^{N})\, a_j^{N-1}. \;)
\end{aligned}
$$

3 Experiments

We used the **color device characterization** problem [12] to demonstrate the "partial hidden-node teaching" by the $3 \times 7 \times 3$ MLP, as illustrated in Figure 2. The basic MLP's task is to learn the mapping from YUV camera signals to XYZ color signals using Equation (2). Furthermore, the three designated hidden nodes (inside the dotted rectangle in Figure 2) are supervised in conjunction with Equation (3) by presenting the scaled L*, a*, and b* color signals [13], which are associated with the desired outputs X, Y, and Z.

The training data set consists of 25 color samples (24 color samples from Machbeth ColorChecker plus the "perfect black" color sample). To check generalization capacity, we also prepared 51 checking data (the "perfect black" and 50 other color samples outside the training data set). All 75 data were collected under the standard illuminant D65. The XYZ values were *scaled* in the sense that those values were just divided by 110 for MLP training; all RMSE (root mean squared errors) were computed after this pre-processing. The MLP models were trained by the steepest descent-type pattern-by-pattern learning method.

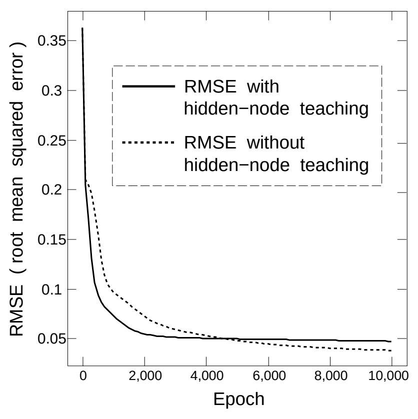
Figure 3 presents sample learning curves of RMSE and color difference, and Figure 4 shows color differences for each color sample obtained by the MLP with hidden-node teaching at the epoch of 10,000.

4 Discussion

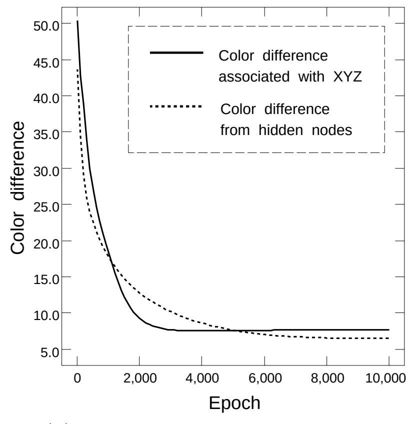
The hidden-node teaching might work as constraint, slowing down the entire MLP-learning process, but Figures 3(a) and (b) indicate that it is not always the case; in fact, MLP with hidden-node teaching learned the mapping from YUV to XYZ slightly faster at the early stage of learning.

The dotted curves in Figures 3(c) and (e) describes how the immediate cost in Equation (3); namely, the color difference obtained from the three designated hidden nodes, decreased thanks to the hidden-node teaching during the learning phase. In contrast, the extraordinarily large color difference (in dotted curves) in Figures 3(d) and (f) shows that the three designated hidden-node activations became totally irrelevant to L*, a*, and b*, when those target values had not been presented to them. In this way, we have verified that the hidden-node teaching was accomplished successfully.

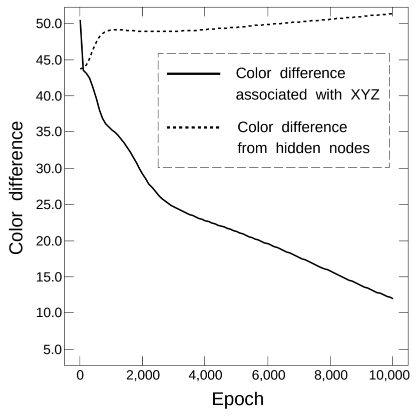
Intriguing enough, the partial hidden-node teaching contributed greatly to decreasing the color difference associated with the final outputs XYZ; for instance, compare the solid curves in Figures 3(c) and (d). At the epoch of 3,000, the *training* color difference (associated with the final outputs XYZ) obtained by the MLP with hidden-node teaching was 7.7008, which is much smaller than 24.5868, the color difference obtained by the MLP *without* hidden-node teaching. (Similar observation can be found in the *checking* result.) Without



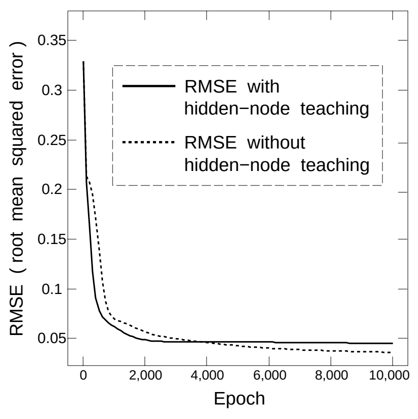
(a) Training RMSE of XYZ



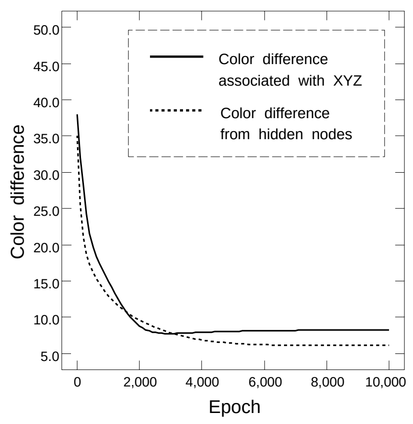
(c) Training color difference with hidden-node teaching



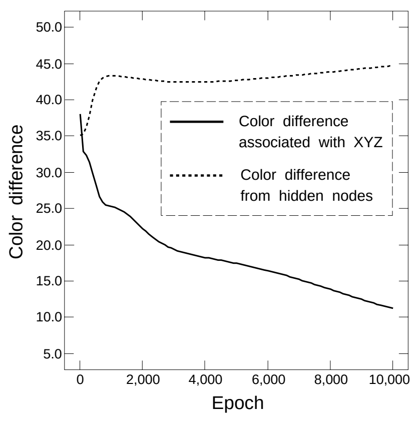
(d) Training color difference without hidden-node teaching



(b) Checking RMSE of XYZ

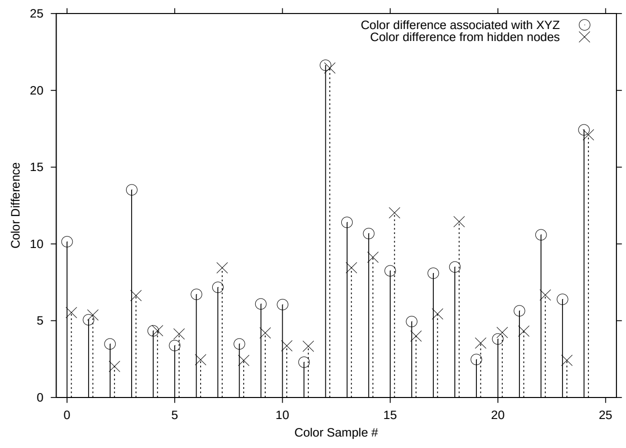


(e) Checking color difference with hidden-node teaching

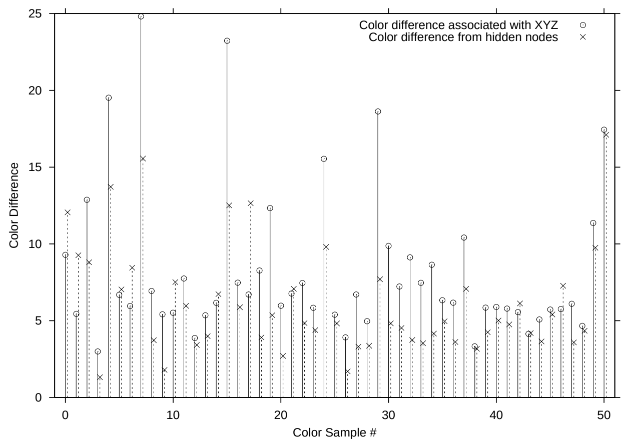


(f) Checking color difference without hidden-node teaching

Figure 3: *Comparison in learning behaviors between the MLP with hidden-node teaching and the MLP without hidden-node teaching.*



(a) Color difference for 25 training color samples



(b) Color difference for 51 checking color samples

Figure 4: *The resultant color difference per color sample obtained by the MLP with hidden-node teaching at 10,000 epoch. The solid bars show the color difference associated with XYZ, whereas the broken bars signify the color difference obtained from the three designated hidden-node activations.*

hidden-node teaching, the color difference associated with the final outputs XYZ was still larger than 10.0 even at the epoch of 10,000, as shown in solid curves in Figures 3(d) and (f), although their RMSEs in XYZ [the dotted curves in Figures 3(a) and (b)] were smaller than the XYZ-RMSEs obtained by the MLPs with hidden-node teaching [the solid curves in Figures 3(a) and (b)].

Figure 4 displays the color difference for each color sample, resulting from hidden-node teaching; the solid bars denote the color difference associated with the final outputs XYZ, while the broken bars signify the color difference obtained from the three designated hidden nodes. They are not completely the same, but somehow reflect the distinctions in color difference. In other words, those hidden nodes can approximate color difference associated with the final outputs XYZ. This finding suggests that such a hidden-node teaching can be modified for application to another industrial problem *color recipe prediction* [14], in which there is no explicit formula for computing color difference associated with a given recipe output vector.

5 Conclusion

We have confirmed that teacher signals can be presented to any selected nodes at any hidden layers in MLPs, and in some applications such a partial hidden-node teaching might produce meaningful and intriguing results. Indeed, it is observed in our experiments that presenting some relevant teacher signals to a subset of the hidden nodes made a positive impact on decreasing color difference.

In the future, it would be of our great interest to perform more elaborate learning based on the Kelley-Bryson formula in conjunction with more sophisticated nonlinear least squares techniques, such as described in ref. [11].

References

[1] Stuart E. Dreyfus. Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure. *Journal of Guidance, Control, and Dynamics*, 13(5):926–928, 1990.

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

[3] Paul Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.

[4] Yann le Cun. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, 1988.

[5] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Massachusetts, 1996.

[6] Henry J. Kelley. Gradient theory of optimal flight paths. *Am. Rocket Soc. Journal*, 30(10):941–954, 1960.

[7] Arthur E. Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proceedings of the Harvard University Symposium on Digital Computers and Their Applications*, April 1961.

[8] Stuart E. Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962.

[9] Stuart E. Dreyfus and Averill M. Law. *The Art and Theory of Dynamic Programming*, volume 130 of *Mathematics in Science and Engineering.* Academic Press Inc., 1977.

[10] E. Mizutani and J.-S. Roger Jang. Chapter 6: Derivative-based Optimization. In *Neuro-Fuzzy and Soft Computing: a computational approach to learning and machine intelligence*, pages 129–172. J.-S. Roger Jang, C.-T. Sun and E. Mizutani. Prentice Hall, 1997.

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

[12] E. Mizutani, K. Nishio, N. Katoh, and M. Blasgen. Color device characterization of electronic cameras by solving adaptive networks nonlinear least squares problems. In *Proceedings of the 8th IEEE International Conference on Fuzzy Systems, vol. 2*, pages 858–862, Seoul, Korea, August 1999.

[13] G. Wyszecki and W. S. Stiles. *Color science: concepts and methods, quantitative data and formulae.* John Wiley & Sons, New York, 2nd edition, 1982.

[14] E. Mizutani, J.-S. R. Jang, K. Nishio, H. Takagi, and D. M. Auslander. Coactive neuro-fuzzy modeling for color recipe prediction. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 2252–2257, November 1995.