J. KSIAM Vol.22, No.2, 115–123, 2018 http://dx.doi.org/10.12941/jksiam.2018.22.115

A MODIFIED EXTENDED KALMAN FILTER METHOD FOR MULTI-LAYERED NEURAL NETWORK TRAINING

KYUNGSUP KIM † AND YOOJAE WON

DEPARTMENT OF COMPUTER ENGINEERING, CHUNGNAM NATIONAL UNIVERSITY, YOOSEONG-GU, DAEJEON, KOREA

E-mail address: sclkim@cnu.ac.kr, yjwon@cnu.ac.kr

ABSTRACT. This paper discusses extended Kalman filter method for solving learning problems of multilayered neural networks. A lot of learning algorithms for deep layered network are sincerely suffered from complex computation and slow convergence because of a very large number of free parameters. We consider an efficient learning algorithm for deep neural network. Extended Kalman filter method is applied to parameter estimation of neural network to improve convergence and computation complexity. We discuss how an efficient algorithm should be developed for neural network learning by using Extended Kalman filter.

1. INTRODUCTION

In the last decade, a lot of efforts were made for introducing deep neural networks into practical applications on both theoretical and hardware levels. Neural networks have been studied to complex functions in various fields, including pattern recognition, identification, classification and control systems [4, 7]. The neural network is a system that approximates the process of the human brain. A multi-layered neural network (MNN) is a nonlinear system having a layered structure, and its learning algorithm is regarded as parameter estimation for such a nonlinear system [4]. Because of a very large number of free parameters, the sequential (on-line) training algorithms can be used for this kind of models. Back-propagation algorithms based first order gradient descent method are used to train the multi-layered neural network but the algorithm converges slowly [10].

In order to improve the convergence, various modified learning algoritms have been proposed by using the second order derivative for updating the weights of neural network. Popular second-order methods have included weight updates based on quasi-Newton, LevenburgMarquardt(LM), and conjugate gradient techniques [5, 6, 13]. LM can be thought of as a combination of steepest descent and the Gauss-Newton method. The extended Kalman filter (EKF) is well-known as a state estimation method for a nonlinear system [3]. The extended Kalman filter approach for neural network learning is one of second order methods under simplifying assumptions [1].

Kalman Filter based training can be used for different models of neural networks such as multilayer perceptrons, radial basis function networks, and neural networks for classification

Received by the editors May 16 2018; Accepted June 11 2018; Published online June 15 2018.

[2, 5]. Various learning algorithms for a multilayered neural network derived from the EKF have been proposed to improve the convergence performance in comparison with the backwards error propagation algorithm [5, 6, 10]. Since this EKF-based learning algorithm approximately gives the minimum variance estimate of the connection-weights, it is expected that it converges as fast as second order [6]. But, the algorithm based extended Kalman filter has a serious drawback in its computational complexity [6]. The main bottleneck for scalable implementation of the Kalman filter is the computation and representation of the state covariance matrix. Our object is to present a unified method combining the existing back-propagation algorithm based on gradient-descent method and extended Kalman filter. We discuss a various issues related to computation problems, including derivative calculations, computationally efficient formulations and methods for avoiding matrix inversions and computational stability.

This paper is organized as follows: in Section 2, we review multi-layered feed-forward neural network, back-propagation learning in the matrix form, and second order method as Levenverg-Marquardt method. In Section 3, we derive back-propagation algorithm using the extended Kalman Filter. In Section 4, we discuss a variety of issues related to fast computation implementation of EKF for MNN.

2. Neural Network

2.1. **Multilayered Neural Network.** We review multi-layered neural network(MNN) [4, 8]. An MNN consists of several layers of nodes which express artificial neural units. Each node which is connected by the links with all nodes in the adjacent layer computes a weighted sum of inputs, and then add an offset to the sum. An MLP(multilayerd perceptron) can be thought of as a function that maps from input to output vectors. Since the behaviour of the function is parameterized by the connection weights, a single MLP is capable of instantiating many different functions. One of the more popular activation functions for backpropagation networks is the sigmoid, a real function $f : R \to (0, 1)$ defined by the expression

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.1}$$

Consider a layered feed-forward neural network as Fig. 2.1. The network consists of M layers, in which the first layer denotes the input, the last M layer is the output, and the other layers are intermediate (or hidden) layers. Here layer 0 is the layer of input sites. It is assumed that the $(k - l)$th layer has N_{k-1} units. The model of the network is based on the following equations

$$p_j^k = \sum_{i=1}^{N_{k-1}} w_{j,i}^k o_i^{k-1} + w_{i,0}^k \tag{2.2}$$

$$o_j^k = f(p_j^k) \tag{2.3}$$

Assume that a connection-weight matrix between layer $n - 1$ and n is denoted by an $N_n \times (N_{n-1} + 1)$ matrix $W^n = (w_{ij}^n)$ and bias terms $w_{i,0}^n$ are put on the last column of W. The

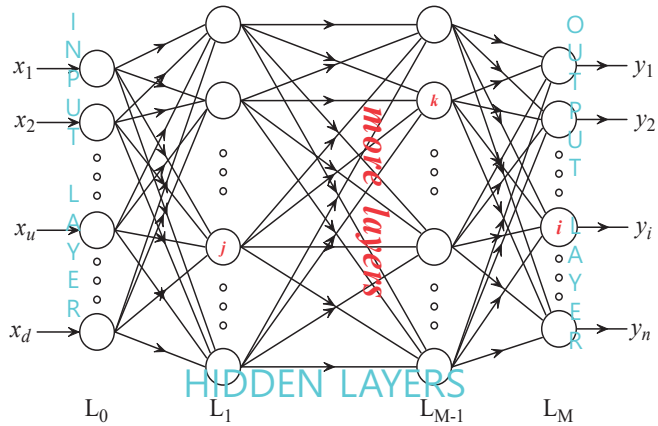

FIGURE 1. A multi-layered feed-forward neural network

equation (2.2) is rewritten in the matrix term such that

$$p^i = W o^{i-1} \tag{2.4}$$

$$o^k = f(p^j) \tag{2.5}$$

where p^i is a column vector with entries p^i_k for i layer and o^i is a column vector with entries o^i_k. The output vector of the nodes in the n-th layer

$$o^n(t) = \begin{bmatrix} o^n_1(t) & o^n_2(t) & \cdots & o^n_{N_n}(t) \end{bmatrix}^T$$

The desired output vector of the MNN for M layer are defined by

$$y(t) = \begin{bmatrix} y_1(t) & y_2(t) & \cdots & y_{N_M} \end{bmatrix}^T$$

The purpose of the learning procedure is to find a set of weights such that, when the network is presented with each input vector, the output vector produced by the network is the same as the desired output vector. The total error in the performance of the network with a particular set of weights can be computed by comparing the actual and desired output vectors for every case. The total error E at t is defined by

$$E(t) = \frac{1}{2} \sum_{j=1}^{N_M} (o^M_j(t) - y_j(t))^2. \tag{2.6}$$

If the neural network solves the classification problem, then it is possible to determine a cross-entropy error such as

$$E(t) = \sum_{j=1}^{N_M} y_j(t) \ln o^M_j(t) - (1 - y_j(t)) \ln(1 - o^M_j(t)). \tag{2.7}$$

The objective is to determine an adaptive algorithm or rule which adjusts the parameters of the network based on a given set of input-output pairs. If the weights of the networks are considered as elements of a parameter vector θ, the learning process involves the determination of the vector θ^* which optimizes a performance function E based on the output error.

2.2. **Gradient descent method and back propagation in matrix form.** The procedure determining gradient descent in Neural network is called a back-propagation algorithm [9]. Since the EKF is a method of estimating the state vector, we put the unknown connection-weights as the state vector

$$w = \begin{bmatrix}(w^1)^T & (w^2) & \cdots & (w^M)^T\end{bmatrix}^T$$

where the vectors w^n and w_i^n are defined by

$$w^n = \begin{bmatrix}(w_1^n)^T & (w_2^n)^T & \cdots & (w_{N_n}^n)^T\end{bmatrix}^T$$
$$w_i^n = \begin{bmatrix}w_{i,0}^n & w_{i,1}^n & \cdots a_{i,N_{n-1}}^n\end{bmatrix}^T.$$

The total number of the linkweights is defined by

$$L = \sum_{n=1}^{M}(N_{n-1}+1)N_n. \tag{2.8}$$

The back-propagation algorithm is applicable for networks with trainable hidden units. We can thus minimize E in Eq.(2.6) by using an iterative process of gradient descent, for which we need to calculate the gradient

$$\nabla E = \begin{bmatrix}\frac{\partial E}{\partial w_1} & \cdots & \frac{\partial E}{\partial w_L}\end{bmatrix}^T$$

A simple back propagation method is based on the gradient descent method that the change Δw_{ij}^k in weight be proportional to $\partial E/\partial w_{ij}^k$ [4]. The gradient of the performance function with respect to θ is computed as $\nabla_\theta E$ and θ is adjusted along the negative gradient as

$$\theta = \theta_{nom} - \epsilon\nabla_\theta E|_{\theta=\theta_{nom}}$$

where the step size ϵ is a suitably chosen constant and θ_{nom} denotes the nominal value of θ at which the gradient is computed. The procedure determining gradient descent in Neural network is called a back-propagation algorithm [9].

The back-propagation algorithm based on the gradient descent method can be summarized in three equations as

$$\begin{aligned} w_{ij}^k &= w_{ij}^k - \epsilon\delta_i^k o_j^{k-1}, \\ \delta_j^M &= (o_j^M - y_j)f'(p_i^M), \\ \delta_i^k &= \sum_{l=1}\delta_l^{k+1}w_{lj}^k f'(p_j^k). \end{aligned}$$

By using matrix concept, the back-propagation algorithm is more generally explained to multi-layered architectures [8]. A error vector e of the stored derivatives of the quadratic deviations is defined by

$$e = \begin{pmatrix} (o_1^M - t_1) & (o_2^M - t_2) & \cdots & (o_{N_M}^M - t_{N_M}) \end{pmatrix}$$

If f is sigmoid, then the derivative stored in the feed-forward step at the N_i units of i-th layer can be written as diagonal matrix

$$D^i = \begin{pmatrix} o_1^i(1-o_1^i) & 0 & \cdots & 0 \\ 0 & o_2^i(1-o_2^i) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & o_{N_i}^i(1-o_{N_i}^i) \end{pmatrix}$$

for each $1 \leq i \leq M$. The back-propagated error to the output layer is then obtained in the form $\delta^M = D^M e$. The back-propagation error to the i-th computing layer is defined as the following, recursively,

$$\delta^i = eD^M W^M D^{M-1} W^{M-1} \cdots W^{i+1} D^i \tag{2.9}$$

$$\delta^i = \delta^{i+1} W^{i+1} D^i. \tag{2.10}$$

We reformulate the back-propagation algorithm in the matrix form for multilayered neural network. Matrix form representation is helpful to understand the total flow of neural network learning algorithm. This give some motivation to improve the back-propagation algorithm.

2.3. **Second order method of Back propagation.** As the second-order derivatives of total error function, Hessian matrix H gives the proper evaluation on the change of gradient vector. The update rule for Newtons method is

$$w_{k+1} = w_k - H^{-1} g_k. \tag{2.11}$$

where $g_k = J_k e$ is the gradient vector of a weght function and J is a Jacobian matrix. The relationship between Hessian matrix H and Jacobian matrix J can be rewritten as $H = J^T J$. The update rule is called the GaussNewton algorithm

Levenberg-Marquardt algorithm introduces another approximation to Hessian matrix H:

$$H = J^T J + \mu I. \tag{2.12}$$

where J is a Jacobian matrix, μ is always positive, and I is the identity matrix. The elements on the main diagonal of the approximated Hessian matrix will be larger than zero. It can be sure that matrix H is always invertible. The update rule of LevenbergMarquardt algorithm can be presented as

$$w_{k+1} = w_k - (J_k^T J_k + \mu I) J_k e_k. \tag{2.13}$$

As the combination of the steepest descent algorithm and the GaussNewton algorithm, the Levenberg Marquardt algorithm switches between the two algorithms during the training process [13]. When the combination coefficient μ is very small, GaussNewton algorithm is used. When combination coefficient μ is very large, the steepest descent method is used.

3. Extended Kalman Filter approach

We have reviewed the back-propagation algorithms based on the first and second order. The first order algorithms converge slowly for large systems. A Kalman filter approach is based on the use of the information of the second order derivative for updating the weights of the neural network. We show how a time-varying learning rate can be computed by using the extended Kalman filter. The MNN is then expressed by the following nonlinear system equations

$$\begin{aligned} w(t+1) &= w(t) + u(t) \\ y(t) &= h_t(w(t)) + v(t) \\ &= o^M(t) + v(t) \end{aligned}$$

Note here that $o^M(t)$ is the output vector of the nodes in the output layer. The input combined with the structure of the MNN is expressed by a nonlinear time-variant function ϕ_t,. The observation vector is represented by the desired output vector $y(t)$, and $u(t)$ and $v(t)$ are assumed to be a white noise vectors with covariance matrix $R(t)$ and $Q(t)$, respectively.

We will be concerned with discrete-time systems such that

$$w(t+1) = f_t[w(t)] + u(t) \tag{3.1}$$

$$y(t) = h_t[w(t)] + v(t) \tag{3.2}$$

where $u(\cdot)$, $x(\cdot)$, and $y(\cdot)$ are discrete time sequences. We assume that h is sufficiently differentiable with respect to x. By using a Taylor series, we can make a linearization of the nonlinear function h_t about a nominal trajectory $\hat{w}$ and $\hat{y} = h_t(\hat{w}(t))$ such as

$$h_t(w(t)) = h_t(\hat{w}(t)) + \left.\frac{\partial h_t}{\partial w}\right|_{\hat{w}} \cdot \tilde{x} + \text{higher order terms}$$

$$\delta y = \left.\frac{\partial h_t}{\partial w}\right|_{\hat{w}} \cdot \tilde{x} + \text{higher order terms}$$

where $\tilde{y} = y - \hat{y}$ and $\tilde{x} = w - \hat{w}$. We can also obtain the linearization of f, similarly. The linearized equations about nominal values are reformulated as

$$\tilde{x}(t+1) = F_t\tilde{x}(t) + u(t), \tag{3.3}$$

$$\tilde{y}(t) = H_t\tilde{x}(t) + v(t) \tag{3.4}$$

where $F_t = \nabla f_t \in \mathbb{R}^{L\times L}$, $H_t = \nabla h_t \in \mathbb{R}^{N_M\times L}$. F_t and H_t are defined by Jacobian matrices of a function f_t and h_t with respect to a nominal trajectory $\hat{x}$. If the problem is such that the actual trajectory x is sufficiently close to the nominal trajectory $\hat{x}(t)$ so that the higher order terms in the expansion can be ignored, then this method transforms the problem into a linear problem [3]. The $N_M \times L$ Jacobian matrix H_t is expressed by

$$\begin{aligned} H_t = \left(\frac{\partial h_t}{\partial w}\right)_{w=\hat{w}} &= \begin{bmatrix} H^1 & H^2 & \cdots & H^M \end{bmatrix} \\ H^n &= \begin{bmatrix} H_1^n & H_2^n & \cdots & H_{N_{n+1}}^n \end{bmatrix} \end{aligned}$$

where H_k^n is defined by an $N_M \times (N_n + 1)$ matrix such as

$$H_k^n = \left(\frac{\partial o^M(t)}{\partial w_k^n} \right)_{w_k^n = \hat{w}_k^n}. \tag{3.5}$$

We have $o^M = F_{k+1}(p^{k+1}) = F_{k+1}(W^{k+1} f(p^k))$. The derivatives of o^M with respect to $w_{i,j}^k$ and $w_{i,j}^{k+1}$ is obtain as

$$\frac{\partial o^M}{\partial w_{ij}^k} = \frac{\partial F_{k+1}}{\partial p^{k+1}} \frac{\partial p^{k+1}}{\partial w_{ij}^k} = [\frac{\partial F_{k+1}}{\partial p^{k+1}} W^{k+1}]_i o_i^k (1 - o_i^k) o_j^{k-1}$$

$$\frac{\partial o^M}{\partial w_{i,j}^{k+1}} = \frac{\partial F_{k+1}}{\partial p^{k+1}} \frac{\partial p^{k+1}}{\partial w_{ij}^{k+1}} = \left[\frac{\partial F_{k+1}}{\partial p^{k+1}} \right]_i o_j^k$$

Define a row vector Δ_i^k with j entry $\Delta_{i,j}^k = \frac{\partial o^M}{\partial w_{ij}^k} / o_j^{k-1}$ similar to the back-propagation. Recursively, $\Delta_i^k = \Delta_i^{k+1} W^{k+1} D^k$. H_k^n is computed by $H_k^n = \Delta_k^n o_k^n$.

We regard the learning of network as an estimation (or identification) problem of constant parameters. The measurement noise $u(t)$ is typically characterized as zero mean, white noise with covariance given by $E[u(t)u(k)^T] = \delta(t-k)Q_t$. The process noise $v(t)$ is also characterized as zero-mean, white noise with covariance given by $E[v(t)v(k)^T] = \delta(t-k)R_t$.

The training problem using Kalman filter theory can now be described as finding the minimum mean-squared error estimate of the state w using all observed data. The EKF(extended Kalman filter) solution to the training problem is given by the following recursion

$$\tilde{x}(t+1) = \tilde{x}(t) + K(t)\xi(t) \tag{3.6}$$

$$S = R_t + H_t P(t) H_t^T = R_t + \sum H^k P^k (H^k)^T \tag{3.7}$$

$$K(t) = P(t|t-1) H_t^T [R_t + H_t P(t|t-1) H_t^T]^{-1} \tag{3.8}$$

$$P(t+1|t) = P(t|t-1) - K(t) H_t P(t|t-1) + Q_t. \tag{3.9}$$

where this estimate of $\tilde{x}(t)$ is a function of the Kalman gain matrix $K(t)$. $P(t+1|t)$ is the approximate error covariance matrix. The Kalman gain matrix $K(t)$ is a function of the approximate error covariance matrix of $\tilde{x}(t)$. The $N_M \times N_M$ covariance matrix S in (3.7) is the zero-mean innovation vector $\xi(t) = y(t) - \hat{y}(t)$ and P^k is the error covariance matrix of $\tilde{x}^k$ for k layer.

4. MODIFICATION OF EKF

In this section, we discuss a various issues related to computational problems, including derivative calculations, computationally efficient formulations and methods for avoiding matrix inversions and computational stability. We should compute various matrices such as the estimation error covariance matrix, the measurement covariance matrix, and the additional process noise matrix [5, 10]. The algorithm based extended Kalman filter has a serious drawback in its

computational complexity [6]. Its computational complexity is $O(N_M L^2)$ per pattern, which becomes intractable as the size of the MNN grows large.

The main objective is to consider several computational methods to avoid full matrix multiplication and matrix inversion in computing innovation error matrix S, Kalman gain K and error covariance matrix P.

First, we note that if the error weights defined by $\tilde{x}^n = w^n - \hat{w}^n$ for n layer are jointly independent for other layers, then the error covariance matrix $P(t)$ is block diagonal matrix [6, 11, 12]. Then the weight $\tilde{x}^n$ of independently of other layers' weights can be updated.

Second, we note that the term S^{-1} in the Kalman gain in (3.8) can not be separable for different layers and it is difficult to make the innovation error matrix S and the error covariance matrix P diagonal matrices since the matrix H^n induced in (3.4) and (3.5) is dense matrix. So we proposed a new model for the Kalman gain to be separable. We modify the desired output function $y(t)$ in (3.2) and (3.4) by adding fictitious measurement outputs. The back-propagation error δ^i of the i-th layer in (2.10) is regarded as the measurement error $\tilde{y}^i$ of i-the layer. Then a new desired output is defined by

$$\mathbf{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^M \end{bmatrix} = \begin{bmatrix} H^1 & O & \cdots & O \\ O & H^2 & \cdots & O \\ \vdots & \vdots & \ddots & \vdots \\ O & O & \cdots & H^M \end{bmatrix} \begin{bmatrix} \tilde{x}^1 \\ \tilde{x}^2 \\ \vdots \\ \tilde{x}^M \end{bmatrix} + \begin{bmatrix} v^1 \\ v^2 \\ \vdots \\ v^M \end{bmatrix} \tag{4.1}$$

where $E(v^i v^j) = R^i \delta_{i,j}$. We assume that $y(t) = \sum_n y^n$, $v(t) = \sum_k v^k(t)$ and $R = \sum_k R^k$. y^k for hidden layer denotes the fictitious output similar to the method in [12]. The covariance matrix of the zero-mean innovation vector of $\mathbf{y} - \tilde{\mathbf{y}}$ is a block diagonal matrix. The EKF(extended Kalman filter) solution to the training problem is obtained by the following recursion

$$\tilde{x}^n(t+1) = \tilde{x}^n(t) + K^n(t)\tilde{y}^n(t) \tag{4.2}$$

$$S^n = R^n + H^n P^n(t|t-1)(H^n)^T \tag{4.3}$$

$$K^n(t) = P^n(t|t-1)(H^n)^T[R^n + H^n P^n(t|t-1)(H^n)^T]^{-1} \tag{4.4}$$

$$P^n(t+1|t) = P^n(t|t-1) - K^n(t)H^n P^n(t|t-1) + Q^n. \tag{4.5}$$

By assuming that R^n is a diagonal matrix $\lambda_n I$, we can avoid a matrix inversion computation of innovation covariance matrix S in (4.3) and conserve the block diagonal property of P and S by giving G and R the assumption as diagonal matrices. There are many variables that affect EKF training algorithm performances. These variables are matrices that must be initialized properly. However the initial values of the EKF training algorithm can influence algorithm performance.

5. CONCLUSION

We reformulate the back-propagation algorithm in the matrix form for multilayered neural network. Matrix form representation is helpful to understand the total flow of neural network

learning algorithm. This give some motivation to improve the back-propagation algorithm. We discussed some first and second order back-propagation algorithms network gradient-based training algorithms in matrix-form. The EKF method for MNN has been presented. We discussed some issues of learning algorithms related to computation problems, including derivative calculations, computationally efficient formulations and methods for avoiding matrix inversions and computational stability. A modified learning algorithm for MNN was proposed by modifying EKF framework to reduce the computation complexity.

ACKNOWLEDGMENTS

The first author (Kyungsup Kim) was supported by research fund of Chungnam National University.

REFERENCES

[1] Ben Cherif Aissa and Chouireb Fatima. Neural Networks Trained with Levenberg-Marquardt-Iterated Extended Kalman Filter for Mobile Robot Trajectory Tracking. *Journal of Engineering Science and Technology Review*, 10(4):191–198, 2017.

[2] A. N. Chernodub. Training Neural Networks for classification using the Extended Kalman Filter: A comparative study. *Optical Memory and Neural Networks*, 23(2):96–103, 2014.

[3] Mohinder S. Grewal and Angus P. Adnrews. *Kalman filtering - theory and practice using matlab*. John Wiley & Sons, Inc., 2008.

[4] Simon Haykin. *Neural Networks and Learning Machines*. 1999.

[5] SS Haykin. *Kalman Filtering and Neural Networks*. 2001.

[6] Y. Iiguni, H. Sakai, and H. Tokumaru. A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter. *IEEE Transactions on Signal Processing*, 40(4):959–966, apr 1992.

[7] K.S. Narendra and K Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[8] Raul Rojas. *Neural Networks*. Springer, 1996.

[9] David E Rumelhart, Geoffrey E Hinton, and R J Williams. Learining Internal Representations by Error Propagation, 1986.

[10] Shared Singhal and Lance Wu. Training Multilayer Perceptrons with the Extended Kalman Algorithm. *Nips*, pages 133–140, 1988.

[11] Keigo Watanabe, Toshio Fukuda, and Spyros G. Tzafestas. Learning algorithms of layered neural networks via extended kalman filters. *International Journal of Systems Science*, 22(4):753–768, 1991.

[12] Keigo Watanabe and Spyros G. Tzafestas. Learning Algorithms for Neural Networks with the Kalman Filters. *Journal of lntelligent and Robotic Systems*, 3:305–319, 1990.

[13] Hao Yu and Bogdan Wilamowski. LevenbergMarquardt Training. 2011.