Multigradient for Neural Networks for Equalizers[1]

Chulhee Lee, Jinwook Go and Heeyoung Kim

Department of Electrical and Electronic Engineering
Yonsei University
134 Shinchon-Dong, Seodaemun-Ku, Seoul 120-749, Korea

ABSTRACT

Recently, a new training algorithm, multigradient, has been published for neural networks and it is reported that the multigradient outperforms the backpropagation when neural networks are used as a classifier. When neural networks are used as an equalizer in communications, they can be viewed as a classifier. In this paper, we apply the multigradient algorithm to train the neural networks that are used as equalizers. Experiments show that the neural networks trained using the multigradient noticeably outperforms the neural networks trained by the backpropagation.

Keywords: Equalizer, multigradient, neural networks, training algorithm, pattern classification.

1. Introduction

Neural networks have been successfully applied in pattern recognition, signal processing, and communications. In particular, there has been a great interest in using neural networks to implement equalizers which can be viewed as classification problems whose distribution functions are unknown [1, 2]. Many researchers reported that neural networks could be a promising solution to equalization problems and proposed various implementations. When neural networks are used as an equalizer, one of the most frequently used training algorithms is the backpropagation algorithm. Recently, a new training algorithm, which is called *multigradient*, has been proposed [3]. The multigradient is a specialized training algorithm when neural networks are used as a classifier. It has been reported that the multigradient outperforms the backpropagation algorithm in pattern classification [3]. Since neural networks are used as a classifier when they are used as equalizers, the multigradient algorithm can be used for such neural networks. In this paper, we apply the multigradient algorithm to neural networks that are used as equalizers and evaluate the performance.

2. Channel Equalization Problem

If input signal $x(n)$ is transmitted through a linear dispersive channel of finite impulse response with the coefficients a_k , the received signal $y(n)$ can be modeled by

$$y(n) = \sum_{k=-L}^{L} a_k x(n-k) + e(n)$$

where $e(n)$ is the additive white Gaussian noise specified by the following statistics:

$$E\left[e(n)\right] = 0, \quad E\left[e(n)e(m)\right] = \sigma_e^2 \delta(n-m)$$

where σ_e^2 is noise variance. The input signal $x(n)$ is chosen independently from {-1, 1} with equal probability and equalization is to estimate the original input signal $x(n)$ from the received signal $y(n)$ in the presence of noise and interference.

Equalizers have been important in digital communication systems to guarantee a reliable data transmission and numerous equalization algorithms have been proposed. Among various equalization methods, linear equalization has been widely used due to their speed and simplicity. The linear equalizer is frequently implemented using the least mean square (LMS) algorithm as follows:

$$W_{n+1} = W_n + c\lambda Y_n$$

[1] The Korea Science and Engineering Foundation partly supported the publication of this paper through BERC at Yonsei University.

where $Y_n = [y(n-2), y(n-1), y(n), y(n+1), y(n+2)]^T$, λ is the learning rate, c is 1 if signal 1 is transmitted and –1 if signal –1 is transmitted. The linear equalizer can perfectly reconstruct the original input signal if the received signal is linearly separable. However, the decision boundary for equalization is highly nonlinear in many cases and neural networks which can form an arbitrary nonlinear decision boundary can be better adopted for equalization.

3. Multigradient [3]

A typical neural network has the input layer, a number of hidden layers, and the output layer. Fig. 1 shows an example of 3-layer feedforward neural networks for a 2 pattern-class problem. The decision rule is to choose the class corresponding to the output neuron with the largest output [4]. In Fig. 1, $X_{in} = (x_1, x_2, ..., x_M)^T$ represents the input vector, $Y = (y_1, y_2)^T$ the output vector, and $B = (b_1, b_2)^T$ the bias vector. We may include the bias term in the input layer as follows:

$$X = (x_1, x_2, ..., x_M, 1)^T = (x_1, x_2, ..., x_M, x_{M+1})^T$$
$$\text{where } x_{M+1} = b_1 = 1.$$

Assuming that there are K neurons in the hidden layer, the weight matrices W_1 and W_2 for the 2 pattern class neural network can be represented by

$$W_1 = \begin{bmatrix} w_{1,1}^{hi} & w_{1,2}^{hi} & \cdots & w_{1,M}^{hi} & w_{1,M+1}^{hi} \\ w_{2,1}^{hi} & w_{2,2}^{hi} & \cdots & w_{2,M}^{hi} & w_{2,M+1}^{hi} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ w_{K,1}^{hi} & w_{K,2}^{hi} & \cdots & w_{K,M}^{hi} & w_{K,M+1}^{hi} \end{bmatrix}$$

$$W_2 = \begin{bmatrix} w_{1,1}^{oh} & w_{1,2}^{oh} & \cdots & w_{1,K}^{oh} & w_{1,K+1}^{oh} \\ w_{2,1}^{oh} & w_{2,2}^{oh} & \cdots & w_{2,K}^{oh} & w_{2,K+1}^{oh} \end{bmatrix}$$

where $w_{j,i}^{hi}$ is the weight between input neuron i and hidden neuron j and $w_{k,j}^{oh}$ is the weight between hidden neuron j and output neuron k. In order to train the neural network, we need to find matrices W_1 and W_2 that produce a desirable sequence of output vectors for a given sequence of input vectors.

Let W be the vector containing all weights. In other words,

$$W = (w_{1,1}^{hi}, w_{1,2}^{hi}, ..., w_{K,M+1}^{hi}, w_{1,1}^{oh}, w_{1,2}^{oh}, ..., w_{2,K+1}^{oh})^T$$
$$= (w_1, w_2, w_3, ..., w_L)^T$$

where L=((M+1)K+2(K+1)) and K is the number of hidden neurons. Then, we may view W as a point in the L dimensional space. In the above example, there are ((M+1)K+2(K+1)) weights to adjust. Let W be the vector containing all the elements of W_1 and W_2:

$$W = (w_{1,1}^{hi}, w_{1,2}^{hi}, ..., w_{K,M+1}^{hi}, w_{1,1}^{oh}, w_{1,2}^{oh}, ..., w_{2,K+1}^{oh})^T$$
$$= (w_1, w_2, w_3, ..., w_L)^T$$

where L=((M+1)K+2(K+1)). Then, W can be viewed as a point in the L dimensional space. In this paradigm, the learning process can be viewed as finding a solution point in the L dimensional space.

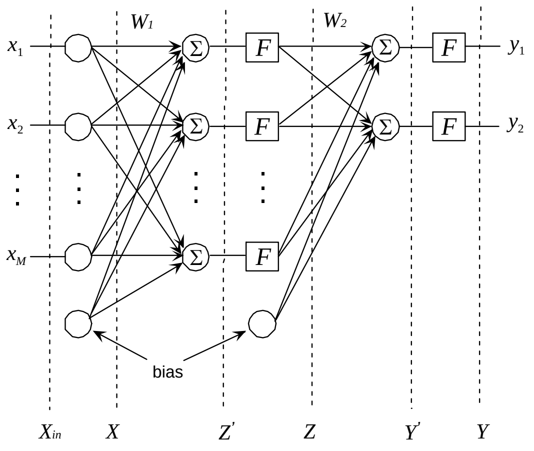


Fig. 1. An example of 3-layer feedforward neural networks
(2 pattern classes).

In multilayer feedforward neural networks, the output vector Y can be represented as a function of X and W:

$$Y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} F_1(X, W) \\ F_2(X, W) \end{bmatrix}$$

assuming a 2 pattern-class classification problem. During learning phase, if X belongs to class ω_1, we move W in such a direction that y_1 increases and y_2 decreases. We can find the direction by taking the gradients of y_1 and y_2 with respect to W:

$$\nabla y_i = \frac{\partial y_i}{\partial w_1}\overline{w}_1 + \frac{\partial y_i}{\partial w_2}\overline{w}_2 + \ldots + \frac{\partial y_i}{\partial w_L}\overline{w}_L$$

where $\{\overline{w}_i\}$ is a basis of the L-dimensional space. Thus, if we update W in the direction of $\alpha\nabla y_1 - \beta\nabla y_2$, where $\alpha, \beta > 0$, y_1 will increase and y_2 will decrease. In general, we update the weight vector W as follows:

$$W^{updated} = W + \gamma(c_1\nabla y_1 + c_2\nabla y_2) \qquad (1)$$

where γ is the learning rate. This procedure is illustrated in Fig. 2. If there are N output neurons, then the weight vector W is updated as follows:

$$W^{updated} = W + \gamma(c_1\nabla y_1 + c_2\nabla y_2 + \ldots + c_N\nabla y_N)$$

where $c_i \geq 0$ if X belongs to class ω_i and $c_i \leq 0$ otherwise.

Assuming the sigmoid function is used as the activation function, it can be shown that differentiating y_1, y_2 with respect to the weights between the hidden layer and the output layer can be obtained as follows:

$$\frac{\partial y_{k'}}{\partial w^{oh}_{k,j}} = \begin{cases} y_{k'}(1-y_{k'})z_j & (k' = k) \\ 0 & (k' \neq k) \end{cases}$$

where $w^{oh}_{k,j}$ is the weight between hidden neuron j and output neuron k and z_j is the output of hidden neuron j. Similarly, differentiating y_1, y_2 with respect to weights between the input layer and hidden layer yields

$$\frac{\partial y_k}{\partial w^{hi}_{j,i}} = y_k(1-y_k)w^{oh}_{k,j}\, z_j(1-z_j)x_i$$

where $w^{hi}_{j,i}$ is the weight between input neuron i and hidden neuron j and $w^{oh}_{k,j}$ is the weight between hidden neuron j and output neuron k. There are a number of possibilities to set c_i in (1). If we set c_i to be the difference between the target value and the output value, the multi-gradient algorithm is equivalent to the backpropagation algorithm. In [3], assuming that the target value is either 0.1 or 0.9, c_i was set as follows:

$$c_i = \begin{cases} t_i - y_i & \text{if target value } t_i = 0.9 \text{ and } y_i < 0.9 \\ t_i - y_i & \text{if target value } t_i = 0.1 \text{ and } y_i > 0.1 \\ 0 & \text{otherwise} \end{cases}$$

In other words, we ignore the output neurons that exceed the target values and concentrate on the output neurons that do not meet the target values, updating weights accordingly. Since the classification accuracy is the most important criterion when neural networks are used as a classifier, this weight update strategy can be effective, providing better classification accuracies.

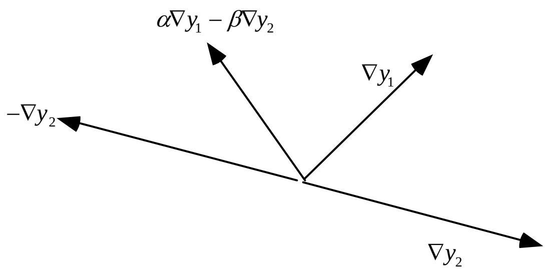


Fig. 2. Adjusting weights by adding the gradients.

4. Experiments and Results

Experiments were conducted for a symmetric channel and a non-symmetric channel. In the first experiment, we generated 10,000 samples for the following symmetric channel:

$$y(n) = \sum_{k=-L}^{L} a_k x(n-k) + e(n)$$

where L=2, $a_1 = a_5 = 0.5$, $a_2 = a_4 = 0.7$, $a_3 = 1$, and $\sigma_e = 0.1$. Among the 10,000 samples, the first 1000 samples are used for training and the rest are used for testing. Fig. 3 shows the performance comparison of the multigradient and the backpropagation algorithms. As can be seen, the multigradient noticeably outperforms the backpropagation. When the backpropagation was used, the classification accuracies for the training and test data are 87.8% and 86.5%, respectively. When the networks are trained by the multigradient, the classification accuracies for the training and test data are 90.1% and 88.3%, respectively.

In the second experiment, we generated 10,000 samples for the following channel:

$$y(n) = \sum_{k=-L}^{L} a_k x(n-k) + e(n)$$

where L=2, $a_1 = 0.2$, $a_2 = 0.8$, $a_3 = 1$, $a_4 = 0.7$, $a_5 = 0.3$ and $\sigma_e = 0.2$. It is noted that the channel is non-symmetric. As previously, the first 1000 samples are used for training and the rest are used for testing. Fig. 4 shows the performance comparison. With the backpropagation, the classification accuracies for the training and test data are 93.9% and 93.9%, respectively. When the networks are trained by the multigradient, the classification accuracies for the training and test data are 95.6% and 94.9%, respectively. As in the symmetric channel, the multigradient outperforms the backpropagation. The multigradient also converges faster the backpropagation.

5. Conclusions

In this paper, we applied the recently published multigradient training algorithm to neural networks that are used as an equalizer. It was reported that the multigradient algorithm outperforms the backpropagation when neural networks are to be used as a classifier. Experiments with symmetric and non-symmetric channels showed that the multigradient algorithm provided noticeable improvements over the conventional backpropagation.

References

[1] G. J. Gibson, S. Siu and C. F. N. Cowan, "Multilayer perceptron structures applied to adaptive equalizers for data communications," Proc. IEEE ICASSP’89, pp. 1183-1186, May 1989.

[2] S. Chen, B. Mulgrew, and P. M. Grant, "A clustering technique for digital communications channel equalization using radial basis function networks," IEEE Trans. Neural Networks, vol. 4, no. 4, pp. 570-579, July 1993.

[3] J. Go, G. Han, H. Kim and C. Lee, "Multigradient: a new neural network learning algorithm for pattern classification," IEEE Trans. Geoscience and Remote Sensing, vol. 39, no. 5, pp. 986-993, May 2001.

[4] R. P. Lippmann, "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, vol. 4, no. 2, pp. 4-22, 1987.

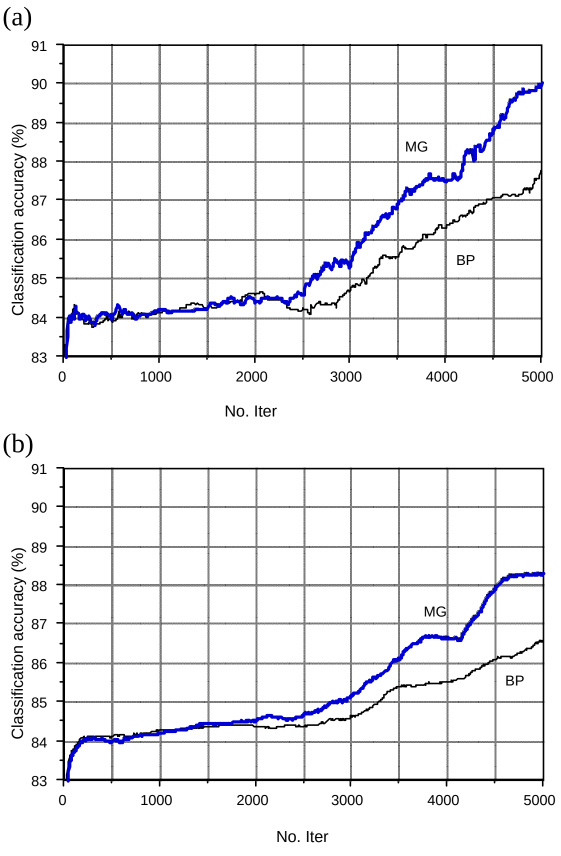
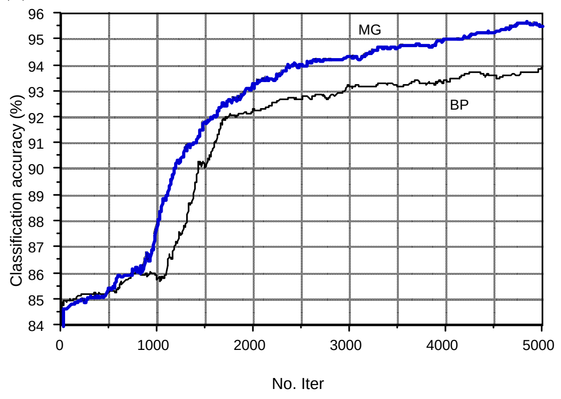


Fig. 3. Performance comparison for a symmetric channel.
(a) training data, (b) test data.

(a)



(b)

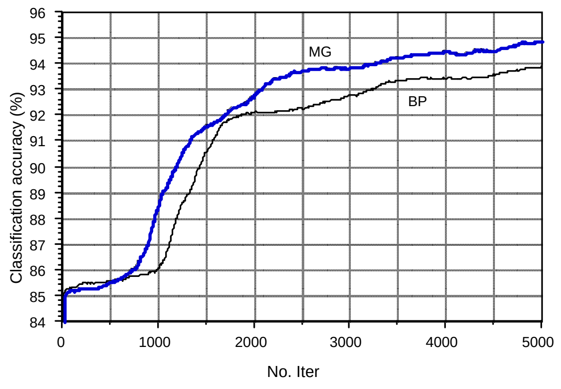


Fig. 4. Performance comparison for a non-symmetric channel.
(a) training data, (b) test data.