# Training the Multilayer Perceptron Classifier with a Two-Stage Method

*Lipo Wang*
School of Electrical and Electronic Engineering
Nanyang Technological University
Block S2, Nanyang Avenue
Singapore 639798
URL: http://www.ntu.edu.sg/home/elpwang/
Email: elpwang@ntu.edu.sg

## Abstract

*We propose a two-stage training for the multilayer perceptron (MLP). The first stage is bottom-up, where we use a class separability measure to conduct hidden layer training and the least squared error criterion to train the output layer. The second stage is top-down, we use a criterion derived from classification error rate to further train the network weights. We demonstrate the effectiveness of the proposed training algorithms with computer simulations.*

## 1. Introduction

The multilayer perceptron (MLP) has been widely studied and applied in pattern recognition (e.g., [1]-[11]). There are many ways to improve the effectiveness of training and classification accuracy. For example, Bischel and Seitz [6] proposed a bottom-up training procedure by using a minimum class entropy criterion, in contrast with the traditional back-propagation (BP) algorithm, which is a top-down procedure. Juang and Katagiri [7] used a new cost function, called minimum classification error (MCE), to train the MLP.

In the present paper, we propose a new approach to the training of the MLP classifier. The network has a single hidden layer with a wavelet-like nonlinearity as the activation function. The output layer consists of $c$ nodes, where $c$ is the number of classes for the problem under consideration, with sigmoid nonlinearity. Training starts from bottom-up: the weights connecting the input layer to the hidden layer is trained first by using a interclass distance criterion as the cost function, followed by a least mean error (LME) procedure to train the weights connecting the hidden layer to the output layer. This initial training stage will be described in Section 2. Based on the initial training result, we then use a top-down procedure similar to the BP algorithm but with a cost function derived directly from the definition of classification error rate. This procedure is described in Section 3. In Section 4, we present computer simulations on a three-class problem to demonstrate the proposed training algorithm. Finally, conclusions and discussions are presented in Section 5.

## 2. Stage 1: Bottom-Up Training

We consider a feedforward network consisting of a single hidden layer with $H$ nodes and an output layer with $c$ nodes, where $H$ will be determined by training and $c$ is the number of classes specified by the problem at hand. We train the network to perform a classification task in such a way that an input pattern $\mathbf{x} = (x_1 \cdots x_n)^t$ is classified to class $i$ if the output from output neuron $i$ is the largest among all output neurons. That is, we use the following decision rule:

$$D(\mathbf{x}) = w_i \quad , \qquad g_i = \max_{j=1,\cdots,c} \{g_j\} \tag{2.1}$$

where

$$g_i = g(\sum_{j=1}^{H} w_{ij}' y_j + q_i') \tag{2.2}$$

is the output of output neuron $i$, g(.) denotes sigmoid function and $y_j$ is the output of hidden neuron $j$.

We observe that the hidden layer in the network acts as a feature extractor. The input to hidden neuron $j$ is

$$s_j = \sum_{k=1}^{n} w_{jk} x_k + \theta_j \tag{2.3}$$

and the output of hidden neuron $j$ is

$$y_j = f(s_j) \tag{2.4}$$

where f (.) is the activation function, which will be discussed shortly.

To train a feature extractor, we need a proper class separability measure. There exist two types of separability measure in the pattern recognition literature [8]. One is the probability measure, in which the entropy criterion is included. The other is the interclass distance measure. From the viewpoint of statistics theory, the former is considered to be more reliable. From a practical point of view, however, the latter is more convenient. Among the measures belonging to the interclass distance category cited in [8], we choose the following measure

$$J_d = \det(S_{t,y}) / \det(S_{w,y}) \tag{2.5}$$

as the objective function, where

$$S_{t,y} = E\{(\mathbf{y} - m)(\mathbf{y} - m)^t\} \tag{2.6}$$

is referred to as the total scatter matrix in the *feature* space, and

$$S_{w,y} = \sum P_i E_i\{(\mathbf{y} - m_i)(\mathbf{y} - m_i)^t\} \tag{2.7}$$

is the within-class scatter matrix.

We propose a new activation function as follows, which very similar to the derivative of a spline function or a wavelet [9]

$$f(t) = tgh^2(t+1) - tgh^2(t-1) \quad (2.8)$$

In the initial training stage, we first find the maximum of objective function (2.5) by a gradient method. We can compute the gradient vector by using the sample mean within a batch to replace the ensemble expectation and then modify the connection weights by using either gradient descent method or conjugate gradient method. This is repeated until convergence.

At this training stage, only the weights connecting the input nodes to the hidden neurons are involved. Thus the initialization problem is much less complex compared to that of training the entire network. We simply assign all weights to be random number within (-0.5, 0.5) and the offsets within (-1.5,1.5) as their initial values.

Once hidden layer training is completed, the output layer training reduces to a single layer perceptron problem. If linear neurons were used in the output layer the least mean-squared error solution can be obtained [8]. If sigmoid neurons are used, a gradient descent procedure could be performed; however, since the two types of output neuron result in almost the same partition boundaries and the training results will be refined further in our top-down procedure, we treat the output neurons as if they were linear.

## 3. Stage 2: Top-Down Training

From the discussions in the previous section, we realize that the resulted classifier is essentially a single layer perceptron, but it operates in the *feature* space in which the classes are best linearly separated. Thus it can be regarded as a promising start point to seek a better solution when we retrain the networks by the back-propagation (BP) algorithm.

However the conventional BP algorithm is conducted by the LME criterion, which is not directly related to the classification error. In an earlier paper an improvement was carried out by Do-Tu *et al* for the *two-class* problem [10]. In this paper, we attempt to generalize the method to the case of *multi-class* problem and nonlinear discriminant functions.

The classification error rate (CER) for a given partition is

$$E = \sum_i P_i \sum_{j \neq i \Omega_j} \int p(\mathbf{x} \mid w_i) d\mathbf{x} \quad , \quad (3.1)$$

where $\Omega_j$ is the decision region for $\omega_j$ by rule (2.1), we introduce an indicator function:

$$Z_i = \prod_{j \neq i} u(g_i(\mathbf{x}) - g_j(x)) \quad (3.2)$$

where u(.) is the unit step function. Obviously we have

$$Z_i(\mathbf{x}) = \begin{cases} 1 & x \in \Omega_i \\ 0 & otherwise \end{cases} \quad (3.3)$$

so that we can rewrite (3.1) as

$$E = \sum P_i \sum_{j \neq i} \int p(\mathbf{x} \mid w_i) Z_j(\mathbf{x}) d\mathbf{x} = \sum_{j \neq i} P_i E_i\{\sum Z_j(\mathbf{x})\} \quad (3.4)$$

This cost function is directly related to the definition of CER, contrasted with the MCE criterion defined in [7]. When we attempt to minimize it by gradient descent method, we need to calculate the derivative of the unit step function, which is a δ-function. As shown in [10], this difficulty can be overcome by a window technique, namely by replacing the unit step function in (3.2) with

$$u_a(t) = \int h_a(\tau) d\tau \quad (3.5)$$

where $h_a(\cdot)$ can be any function satisfying

$$\lim_{a \to 0} h_a(t) = \delta(t) \quad (3.6)$$

so that

$$\lim_{a \to 0} u_a(t) = u(t) \quad (3.7)$$

We can derive the following learning rule for the weight change by using gradient descent: if a training sample belonging to class s is present to the network, we have

$$\delta_i' = \begin{cases} h_a(g_i - g_s) \prod_{k \neq i,s} u_a(g_i - g_k) g_i(1-g_i) & i \neq s \\ -\sum_{q \neq s} h_a(g_s - g_q) \prod_{k \neq i,s} u_a(g_i - g_k) g_s(1-g_s) & i = s \end{cases} \quad (3.8)$$

and modify $w_{jj}'$

$$w_{jj}' = w_{jj}' - \eta' \delta_i' y_j \quad (3.9)$$

where η' is the learning rate.

The modification of the hidden layer weights $w_{jk}$ can be done in a way similar to the BP algorithm, namely

$$\frac{\partial E}{\partial w_{jk}} = \delta_j x_k \quad (3.10)$$

where

$$\delta_j = (\sum_i w_{ij}' \delta_i') \frac{\partial f}{\partial t}\Big|_{t=s_j} \quad (3.11)$$

and

$$w_{jk} = w_{jj} - \eta \delta_j x_k \quad (3.12)$$

It has been proven [10] that if the learning rate and window parameter satisfy certain condition such as

$$\eta = \eta_0 / r \; ; \; \eta' = \eta_0' / r \; ; \; \alpha = \alpha_0 / r^{1/2} \quad (3.13)$$

where r denotes the training count, this stochastic approximation procedure will converge.

## 4. Computer Simulations

Let us consider a two-dimensional, three-class problem ($c$=3 and $n$=2). The classes are distributed with equal *a priori* probability and the following class probability density functions:

$$P(\mathbf{x} \mid w_1) = 0.5N(\Sigma_1^{(1)}, m_1) + 0.5N(\Sigma_1^{(2)}, m_1)$$
$$P(\mathbf{x} \mid w_2) = 0.5N(\Sigma_2^{(1)}, m_2) + 0.5N(\Sigma_2^{(2)}, m_2)$$
$$P(\mathbf{x} \mid w_3) = N(\Sigma_3, m_3)$$

where $N(\Sigma, m)$ denotes a normal distribution with covariance matrix Σ and mean vector $m$. The parameters are specified as follows:

$$\Sigma_1^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad \Sigma_1^{(2)} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$
$$\Sigma_2^{(1)} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \qquad \Sigma_2^{(2)} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\Sigma_3 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

and

$$m_1^{(1)} = (3,3)^t \qquad m_1^{(2)} = (-3,-3)^t$$

$$m_2^{(1)} = (3,-3)^t \qquad m_2^{(2)} = (-3,3)^t$$

$$m_3 = (0,0)^t$$

Without the third class in the middle, the problem reduces to a typical XOR problem. Hence this is a multi-class classification problem more difficult than an XOR problem. We generated 1000 samples in total as training set and another 1000 as test set.

Table 1 shows the experimental results for H = 3, 4, and 5 (the numbers of hidden neurons). For each H, 10 independent simulations were carried out.

Table 2 shows the results for 10 independent runs for using sigmoid neurons in the hidden layer, with the same network with the same procedure described above, are listed in. One can see that the results are not as good as the results showed in Table 1.

To show the importance of the initial bottom-up procedure, let us start the training described in Section 3 with initialization of all connection weights to be small random numbers with 5 hidden nodes (H=5). Table 3 lists the error counts of 10 runs. The results were also not as good.

If we adopt sigmoid neurons in the hidden layer and train the same network with the same procedure described above, the results for 10 independent runs are listed in. One can see that the results are not as good as the results showed in Table 1.

## 5. Conclusions and Discussions

In this paper, a two-stage training procedure is proposed to make the overall design of the MLP classifier more effective. The initial training can be considered as an initialization strategy for final training. During the initial training, an interclass distance measure is used for hidden layer training and an MLE criterion for output layer training. We derive a new CER cost function directly from the classification error rate to conduct our final training in a way similar to the BP algorithm. Computer simulations demonstrated the effectiveness of the above methods.

As a general design approach, the proposal needs to be examined by more real-world classification problems, which is subject to our future studies.

| | H =3 | H = 4 | H = 5 |
|---|---|---|---|
| 1 | 301* | 227 | 203 |
| 2 | 322 | 219 | 185 |
| 3 | 302 | 360 | 327 |
| 4 | 291 | 209 | 238 |
| 5 | 496 | 191 | 215 |
| 6 | 221 | 276 | 205* |
| 7 | 437 | 285 | 315 |
| 8 | 220 | 317 | 288 |
| 9 | 215 | 199 | 309 |
| 10 | 322 | 264 | 239 |
| Average | 310 | 254.7 | 252.4 |

Table 1. Error counts with the proposed algorithm.

| Simulation No. | Error count |
|---|---|
| 1 | 298 |
| 2 | 432 |
| 3 | 346 |
| 4 | 337 |
| 5 | 532 |
| 6 | 235 |
| 7 | 318 |
| 8 | 552 |
| 9 | 447 |
| 10 | 298 |
| Average | 379.5 |

Table 2. Error counts (no bottom-up stage)

## References


[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, U.K., 1995.

[2] R. P. Lippmann, "An Introduction to computing with neural nets", *IEEE ASSP Mag.* 4, 2, 4-22 (1987).

[3] Y. LeCun, et al, "Back propagation applied to handwritten zipcode recognition", *Neural Computation*, 1, 4 (1990)

[4] N. Morgan, H. A. Bourlard, "Neural network for statistical recognition of continuous speech", *Proc. IEEE*, 83, 5, 741-770 (1995).

[5] R. Chellappa, et al, "Guest Editorial: Application of artificial neural network to image processing," *IEEE Trans on Image Processing*, 7, 8 (1998).

[6] M. Nischel & P. Seitz, "Minimum class entropy: a maximum information approach to layered networks", *Neural Networks*, 2, 133-141 (1989)

[7] B. H. Juang & S. Katagiri, "Discriminative learning for minimum error classification ", *IEEE Trans. SP*, 40, 12 (1992).

[8] P. A. Devijer & J. Kittler, *Pattern Recognition: a Statistical Approach*, PHI, (1982).

[9] S. Mallat & S. Zhong, "Characterization of signals from multiscale edge", *IEEE Trans. PAMI*, 14, 7, 710-732 (1992).

[10] Hai Do-Tu & M. Installe, "Learning algorithms for nonparametric solution to the minimum error classification problem", *IEEE Trans on Comput.* 27,7,648-659 (1978).

[11] D. E. Rumelhart & J.L. McClelland (Eds), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol.1: Foundations.* MIT Press (1986).