Training Multi-layer Perceptrons Using MiniMin Approach

Liefeng Bo, Ling Wang, and Licheng Jiao

Institute of Intelligent Information Processing,
Xidian University, Xi'an 710071, China
{blf0218, wliiip}@163.com

Abstract. Multi-layer perceptrons (MLPs) have been widely used in classification and regression task. How to improve the training speed of MLPs has been an interesting field of research. Instead of the classical method, we try to train MLPs by a MiniMin model which can ensure that the weights of the last layer are optimal at each step. Significant improvement on training speed has been made using our method for several big benchmark data sets.

1 Introduction

In many important application areas such as pattern recognition, signal processing and control, it is needed to approximate an unknown nonlinear mapping through learning from examples. Multi-layer perceptrons (MLPs) [1] have been widely used to tackle this task, since they are shown to be universal function approximators [2].

Classical backpropagation (BP) algorithm [3] is usually quite slow due to nonlinear of MLPs and global property of sigmoid neuron. Many efforts have been made to improve training time. In 1990, Battiti and Masulli [4] used quasi-Newton algorithm (BFGS) to speedup MLPs. In 1994, Hagan and Menhaj [5] improved training time of MLPs by Levenberg -Marquardt algorithm where the Jacobian matrix is computed through a standard backpropagation technique that is much less complex than computing the Hessian matrix. In 1993, Moller [5] proposed a quite effective algorithm named scaled conjugate gradient (SCG) for MLPs. SCG is fully automated including no user dependent parameters and avoiding a time consuming line-search. Demuth's test report [7] also shows that SCG performs well over a wide variety of problems, particularly for networks with a large number of weights. For a good introduction of these algorithms, reader can refer to [8-9].

Our focus will be on two-Layer perceptrons with sigmoidal hidden units and a linear output unit. Our fundamental idea is that the weights of MLPs can be computed by a MiniMin model named MM-MLPs. This novel model allows the weights of the last Layer to be analytically calculated by linear equation systems. In other words, MM-MLPs can ensure that the weights of the last Layer are optimal at each step. An empirical study on four big data sets shows that MM-MLP yields a significant speedup relative to MLPs with the same training algorithm (in this paper, SCG is used). The speedup depends on the learning task. Experimental results seem to support that MM-MLPs usually obtain a bigger speedup for regression than for classification task.

Y. Hao et al. (Eds.): CIS 2005, Part I, LNAI 3801, pp. 909 – 914, 2005.

2 MiniMin Multi-layer Perceptrons

We consider the training error to be the sum over output units of the squared difference between the desired output and actual output. Without loss of generality, we ignore the bias terms of network for convenience of formulation. In the classical MLPs, the weights is given by minimizing the following objective function

$$\min_{\boldsymbol{\alpha},\mathbf{W}}\left(E^{'}=\left(\sum_{s=1}^{c}\left(\mathbf{Y}^{s}-\mathbf{H}\boldsymbol{\alpha}^{s}\right)^{T}\left(\mathbf{Y}^{s}-\mathbf{H}\boldsymbol{\alpha}^{s}\right)\right)\right). \tag{1}$$

where $\mathbf{Y}^{s}$ and $\boldsymbol{\alpha}^{s}$ denote the s-th column of matrix $\mathbf{Y}$ and $\boldsymbol{\alpha}$ respectively, and $\mathbf{H}_{ij}=\phi\left(\sum_{t=1}^{n}w_{tj}\mathbf{X}_{it}\right)$. Due to nonlinear and compact structure, many algorithms such as BP result in poor performance in this model. To ease this problem, we try to optimize the weights of MLPs by MiniMin model

$$\min_{\mathbf{W}}\left(E=\min_{\boldsymbol{\alpha}}\left(f=\sum_{s=1}^{c}\left(\mathbf{Y}^{s}-\mathbf{H}\boldsymbol{\alpha}^{s}\right)^{T}\left(\mathbf{Y}^{s}-\mathbf{H}\boldsymbol{\alpha}^{s}\right)\right)\right). \tag{2}$$

By some mathematical tricks, we can get the analytical solution of the inner objective function. The inner objective function can be written as

$$E=\min_{\boldsymbol{\alpha}}\left(\sum_{s=1}^{c}\left(\left(\boldsymbol{\alpha}^{s}\right)^{T}\mathbf{H}^{T}\mathbf{H}\boldsymbol{\alpha}^{s}-2\left(\boldsymbol{\alpha}^{s}\right)^{T}\mathbf{H}^{T}\mathbf{Y}^{s}+\left(\mathbf{Y}^{s}\right)^{T}\mathbf{Y}^{s}\right)\right). \tag{3}$$

Let the derivative of f with respect to $\boldsymbol{\alpha}^{s}$ be zeros, we can compute $\left(\boldsymbol{\alpha}^{opt}\right)^{s}$ by

$$\left(\boldsymbol{\alpha}^{opt}\right)^{s}=\left(\mathbf{H}^{T}\mathbf{H}\right)^{-1}\mathbf{H}^{T}\mathbf{Y}^{s} \tag{4}$$

where $\left(\boldsymbol{\alpha}^{opt}\right)^{s}$ is the s-th column of matrix $\boldsymbol{\alpha}^{opt}$.

Substituting Eq. (**4**) into Eq. (**3**), we have

$$E=\sum_{s=1}^{c}\left(-\left(\mathbf{Y}^{s}\right)^{T}\mathbf{H}\left(\mathbf{H}^{T}\mathbf{H}\right)^{-1}\mathbf{H}^{T}\mathbf{Y}^{s}+\left(\mathbf{Y}^{s}\right)^{T}\mathbf{Y}^{s}\right). \tag{5}$$

Thus Eq. (**2**) is simplified into

$$\min_{\mathbf{W}}\left(E=\sum_{s=1}^{c}\left(-\left(\mathbf{Y}^{s}\right)^{T}\mathbf{H}\left(\mathbf{H}^{T}\mathbf{H}\right)^{-1}\mathbf{H}^{T}\mathbf{Y}^{s}+\left(\mathbf{Y}^{s}\right)^{T}\mathbf{Y}^{s}\right)\right). \tag{6}$$

The derivative of E with respect to the weights $\mathbf{W}_{ij}$ can be computed by theorem 1.

Theorem 1

$$\frac{\partial E}{\partial \mathbf{W}_{ij}}=\sum_{t=1}^{c}2\left(\mathbf{Y}^{s}-\mathbf{H}\left(\boldsymbol{\alpha}^{opt}\right)^{s}\right)^{T}\left(\left(\boldsymbol{\alpha}^{opt}\right)_{js}\Delta\mathbf{H}^{j}\odot\mathbf{X}^{i}\right) \tag{7}$$

where $\odot$ denotes Hadamard product, $\Delta\mathbf{H}_{ij}=\Delta\phi\left(\sum_{t=1}^{n}w_{tj}\mathbf{X}_{it}\right)$ and $\Delta\mathbf{H}^{j}$ is the j-th column of matrix $\Delta\mathbf{H}$。

It is interesting to compare the derivative of E and the derivative of $E^{'}$.

The derivative of $E^{'}$ with respect to $\mathbf{W}_{ij}$ is

$$\frac{\partial E^{'}}{\partial \mathbf{W}_{ij}} = \sum_{s=1}^{c}\left(2\left(\mathbf{Y}^s - \mathbf{H}\boldsymbol{\alpha}^s\right)^T \left(-\frac{\partial \mathbf{H}}{\partial \mathbf{W}_{ij}}\boldsymbol{\alpha}^s \right)\right). \tag{8}$$

Because $\mathbf{W}_{ij}$ only appears in the *j*-th column of $\mathbf{H}$, Eq. (**8**) is further simplified as

$$\frac{\partial E^{'}}{\partial \mathbf{W}_{ij}} = \sum_{s=1}^{c}\left(2\left(\mathbf{Y}^s - \mathbf{H}\boldsymbol{\alpha}^s\right)^T \left(-\boldsymbol{\alpha}_{js}\frac{\partial \mathbf{H}^j}{\partial \mathbf{W}_{ij}} \right)\right). \tag{9}$$

In terms of $\frac{\partial \mathbf{H}^j}{\partial \mathbf{W}_{ij}} = \Delta\mathbf{H}^j \odot \mathbf{X}^i$, Eq. (**9**) can be transformed into

$$\frac{\partial E^{'}}{\partial \mathbf{W}_{ij}} = \sum_{s=1}^{c}\left(2\left(\mathbf{H}\boldsymbol{\alpha}^s - \mathbf{Y}^s\right)^T \left(\boldsymbol{\alpha}_{js}\Delta\mathbf{H}^j \odot \mathbf{X}^i \right)\right). \tag{10}$$

From Eq. (**7**) and Eq. (**10**), we can see that if replacing $\boldsymbol{\alpha}^s$ of Eq. (**10**) with $\boldsymbol{\alpha}^{opt}$, we can get Eq. (**7**). In MM-MLPs, $\boldsymbol{\alpha}^{opt}$ is optimal, which possible explains why our model usually has higher convergence rate.

According to Eq. (**7**), the computational complexity for the derivative of each weight is $O(cl)$, where l is the size of training samples. To obtain the gradient vector, we need to do it $(Ne \times n)$ times, which incurs a computational cost of $O(Ne \times ncl)$, where Ne is the size of hidden units. This cost seems too large. A better solution is based on the following theorem.

Theorem 2

$$\frac{\partial E}{\partial \mathbf{W}} = 2\mathbf{X}^T\left(\Delta\mathbf{H} \odot \left(\left(\mathbf{H}\boldsymbol{\alpha} - \mathbf{Y}\right)\left(\boldsymbol{\alpha}^{opt}\right)^T \right)\right) \tag{11}$$

A straightforward corollary of theorem 2 is that the gradient vector can be computed at $O(Ne \times nl + cnl)$ cost. Similar conclusion holds for MLPs model. Another time-consuming operation is computing $\boldsymbol{\alpha}^{opt}$ whose cost is $O\left(Ne^3 + Ne^2 l\right)$. Thus the computational complexity of MM-MLPs at each step is $O\left(Ne \times nl + cnl + Ne^3 + Ne^2 l\right)$. The computational complexity of MLPs at each step is $O(Ne \times nl + cnl)$.

As a result, we can derive that if $Ne < m$, then MM-MLPs and MLPs have the same computational complexity at each step. Since many practical problems satisfy this condition ($Ne < m$), our model should find wide applications.

3 Empirical Study

In order to know how well MM-MLPs work, we compare it with MLPs on two big data sets, each of which contains several thousand samples. Classification problems

are from Statlog [10] and regression problems are from Delve [11]. These data sets have been extensively used in testing the performance of diversified kinds of learning algorithms. Here, SCG algorithm is used to train networks. All the experiments are run on a personal computer with 2.4 GHz P4 processors, 2 GB memory and Windows XP operation system.

To avoid that the features with large magnitude dominate the output, all the training data are scaled in [-1, 1], then the test data are adjusted using the same linear transformation. The input weights are randomly initialized in the range $[-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}]$ and hidden-to-output weights in the range $[-\frac{1}{\sqrt{Ne}}, \frac{1}{\sqrt{Ne}}]$. The number of hidden units depends on the task at hand; hence there is no foolproof method for setting the number of hidden units before training. In our experiments, the number of hidden units is determined by the 10-fold cross validation method.

The aim with this test is to compare the performance of MM-MLPs and MLPs on the classification problems. This task is to recognize the splice-junction gene sequences. This data set consists of 2000 training samples and 1186 test samples, 180 attributes of each sample. Three-layer network with five hidden units is used for this task. The convergence criterion is set to 0.030, 0.020, 0.010, 0.008, 0.006, 0.004 and 0.002. MM-MLPs and MLPs are tested on 10 random initial weights for each criterion.

From Table 1, we can see that MM-MLPs obtain the speedup range from 2 to 5 under the seven different convergence criterions. From Fig. 1, we can see that MM-MLPs reach the best test error with significantly fewer epochs.

Table 1. Epochs and training time of MM-MLPs and MLPs under the different criterion on Dna data set

	MM-MLPs		**MLPs**	
Criterion	Epoch	Time	Epoch	Time
0.030	11.200	1.482	22.600	2.833
0.020	17.200	2.217	30.700	3.803
0.010	28.900	3.659	53.900	6.609
0.008	32.500	4.249	60.200	7.486
0.006	38.000	4.861	102.000	12.655
0.004	55.000	6.847	229.400	29.788
0.002	130.200	15.541	472.400	60.971

The aim with this test is to compare the performance of MM-MLPs and MLPs on the regression problem. This task is to predict portion of time that CPUs run in user mode. This data set consists of 8192 samples, 21 attributes of each. Three-layer network with fifteen hidden units is used for this task. The convergence criterion is set to 0.000200, 0.00100, 0.00090, 0.00080, 0.00070, 0.00065, 0.00060 and 0.00055. MM-MLPs and MLPs are tested on 10 random initial weights for each criterion.

From Table 2, we can see that MM-MLPs obtain the speedup range from 6 to 12 under the seven different convergence criterions. From Fig. 2, we can see that MM-MLPs reach the best 10-fold cross validation error with significantly fewer epochs.

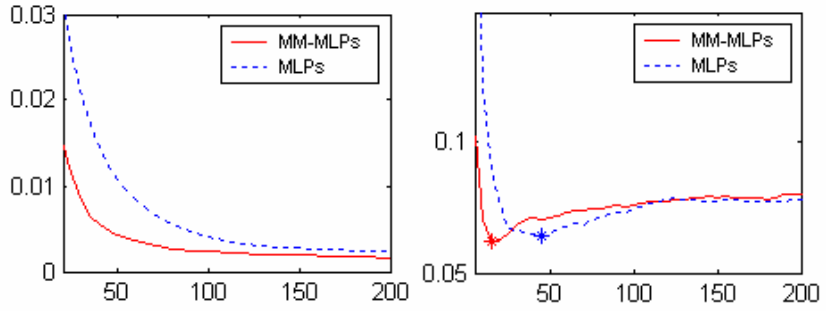

Fig. 1. Variation of the average training errors with epochs (left) and variation of average test errors with epochs (right) on Dna data set. "*" denotes the best test error.

Table 2. Epochs and training times of MM-MLPs and MLPs under the different criterion on Computer Activity data set

Criterion	MM-MLP		MLP	
	Epoch	Time	Epoch	Time
0.00200/19.6020	8.400	7.361	73.300	50.055
0.00100/9.8010	14.000	11.891	120.700	85.627
0.00090/8.8209	17.000	14.014	154.500	108.704
0.00080/7.8408	20.700	17.111	229.000	161.663
0.00070/6.8607	28.900	23.909	379.300	274.800
0.00065/6.3706	49.700	40.422	647.900	470.228
0.00060/5.8806	79.500	64.567	1156.800	835.763
0.00055/5.3906	306.200	246.749	2027.900	1483.817

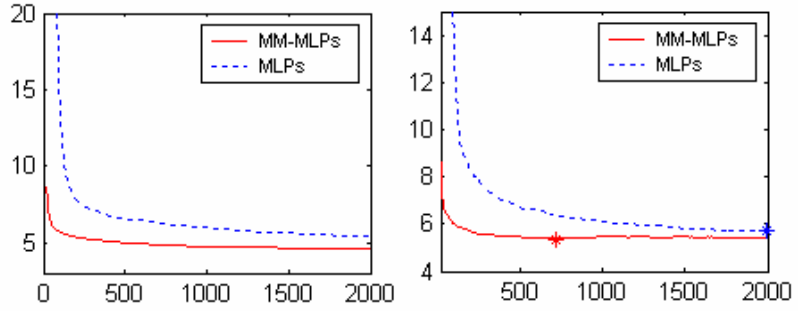

Fig. 2. Variation of average training errors with epochs (left) and variation of 10-fold cross validation errors with epochs(right) on Computer Activity data set. "*" denotes the best 10-fold cross validation error.

4 Conclusion

In this paper, MiniMin model is presented to train MLPs, which can ensure that the weights of the last layer are optimal at each step. The empirical comparisons on the four big benchmark data sets show that our method obtains a significant speedup relative to the classical formulation.

References

1. Rumelnart, D.E., Hinton, G.E. and Williams, R.J.: Learning reprensentation of backprogation errors. Nature 223 (1986) 533-536
2. Hornik, K., Stinchcombe, M. and White, H.: Multilayer feedforward networks are univesal approximators. Neural Networks 2 (1989) 359-366
3. Rumelhart, D.E., Hinton, G.E. and Williams, R.J.: Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing: Exploration in the Microstructure of Cognition (1986) 318–362
4. Battiti R. and Masulli, F.: BFGS Optimization for faster and automated supervised learing. International Neural Network conference (1990) 757-760
5. Hagan, M.T. and Menhaj, M.: Training feedforward networks with the Marquardt algrithm. IEEE Transactions on Neural Networks 5 (1994) 989-993
6. Moller, M.F.: A Scaled conjugate gradient algorithm for fast supervised learning. Neural Network 6 (1993) 525-533
7. Demuth, H. and Beale, M.: Neural network toolbox for use with MATLAB. The MatWorks Inc. Natick, MA (1998)
8. Hagan, M.T., Demuth, H.B. and Beale, M.H.: Neural Network Design. Boston. MA: PWS Publishing (1996)
9. Boyd, S. and Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004).
10. Michie, D., Spiegelhalter, D.J. and Taylor, C.C.: Machine Learning, Neural and Statistical Classification. Prentice Hall (1994)
11. Rasmussen, C.E., Neal, R.M., Hinton, C.E., Van Gamp, D., Revow, M., Ghahramani, Z., Kustra, R. and Tibshirani, R.: The Deleve Manual (1996)