2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

A Pairwise Algorithm for Training Multilayer Perceptrons with the Normalized Risk-Averting Error Criterion

Yichuan Gui James Ting-Ho Lo Yun Peng

***Abstract*— Proper use of the normalized risk-averting error (NRAE) criterion has been shown to avoid nonglobal local minima effectively in the mean squared error (MSE) criterion. For training on large datasets, a pairwise algorithm for the NRAE criterion similar to the widely-used least mean square algorithm for the MSE criterion is proposed. The gradual deconvexification method employing this pairwise algorithm is tested on examples with built-in nonglobal local minima that are difficult to avoid and on recognition of handwritten numerals with the MNIST dataset. Numerical experiments show that the pairwise algorithm for the NRAE criterion is computationally more economical than the corresponding batch algorithm and delivers multilayer perceptrons with better performances than training methods based on the MSE criterion.**

I. Introduction

The mean squared error (MSE) criteria for training multilayer perceptrons (MLPs) are nonconvex with many nonglobal local minima, causing the so-called local-minimum problem. This problem has plagued the neural network approach. Solving the problem has been a focal point of much research work [1–9]. In recent years, methods of convexifying and deconvexifying the MSE criterion have been developed [10–15] and have been shown very effective in numerous numerical experiments. The success rate of both the NRAE-MSE [13, 14] and the gradual deconvexification (GDC) [15] methods in all the performed training sessions with randomly selected initial guesses of the weights is 100%. In those experiments, the training sessions were conducted in the batch mode.

Generally, in training an MLP with backpropogation (BP) algorithm, a training method can be performed in either a batch or pairwise (also called "on-line" or "sequential") mode. Batch training accumulates weight changes over presentation of all training pairs before applying an update to weights, while pairwise training updates weights immediately after presentation of each training pair. Researchers often indicate that batch training is at least theoretically superior to pairwise training, because batch training uses the true gradient, or follow the true gradient more closely, to update weights in training MLPs [5, 16]. Some researchers also claim that batch training is as fast as or faster than pairwise training [3, 5, 17].

Yichuan Gui and Yun Peng are with the Department of Computer Science and Electrical Engineering of the University of Maryland, Baltimore County (email: {yichgui1, ypeng}@umbc.edu).

James Ting-Ho Lo is with the Department of Mathematics and Statistics of the University of Maryland, Baltimore County (email: jameslo@umbc.edu).

This material is based upon work supported in part by the National Science Foundation under Grant ECCS1028048, but does not necessarily reflect the position or policy of the Government.

However, advantages of pairwise training have been noticed in [6, 9, 18]. Several comprehensive summaries and comparisons between batch and pairwise training in [19, 20] point out that batch training is almost always slower by orders of magnitude than pairwise training especially in training with large datasets. Pairwise training is able to safely use a large learning rate to achieve a reasonably good result with a significantly fast convergence speed, but batch training can only follow the true gradient very well along the error surface when the learning rate is small enough to yield an optimal outcome. As the size of the training dataset gets larger, the magnitude of accumulated weights changed in batch training becomes larger too. In this case, batch training must use a small learning rate to prevent weight oscillations across the weight space and avoid the risk of the neuron saturation in MLPs. Accordingly, this learning manner lowers the convergence speed of batch training. On the other hand, pairwise training applies weight changes as soon as they are calculated, thus it can handle different sizes of training datasets without requiring a small learning rate. Therefore, pairwise training is expected to improve the convergence speed of NRAE-based training methods, thus leading these methods to handle large datasets more efficiently.

This paper first describes a pairwise training method for the NRAE criterion in Section II. Then, results of numerical experiments in Section III-A are reported on approximation of functions with fine features or under-sampled segments to demonstrate effects of the pairwise training method with GDC. Moreover, a handwritten numeral recognition task using the well-known MNIST dataset is tested in Section III-B, demonstrating the capability of the pairwise training method with GDC in solving a real-world task on a large dataset comparing to several reported benchmark results. At last, conclusion and future work are discussed in Section IV.

II. Pairwise NRAE Training Method

A standard problem of training a multilayer perceptron (MLP) is the following: Given a set of input/output pairs (x_k, y_k), $k = 1, ..., K$, that are related by

$$y_k = f(x_k) + \xi_k$$

where f is an unknown or known function and ξ_k are random noises or zero, find an MLP $y = \hat{f}(x, w)$ such that the standard MSE criterion

$$Q(w) := \frac{1}{K} \sum_{k=1}^{K} \left\| y_k - \hat{f}(x_k, w) \right\|^2 \tag{1}$$

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

is minimized. The GDC method is facilitated by the use of the normalized risk-averting error (NRAE) criterion [12–15]

$$C_\lambda(w) := \frac{1}{\lambda}\ln\left[\frac{1}{K}J_\lambda(w)\right] \tag{2}$$

where

$$J_\lambda(w) := \sum_{k=1}^{K}\exp\left(\lambda\left\|y_k - \hat{f}(x_k, w)\right\|^2\right) \tag{3}$$

is called the risk-averting error (RAE) criterion [10, 11].

For notational simplicity, let

$$\hat{y}_k(w) := \hat{f}(x_k, w)$$
$$\varepsilon_k(w) := y_k - \hat{y}_k(w)\ .$$

For a vector w, let $S(w) = \arg\max_{k\in\{1,\ldots,K\}}\|\varepsilon_k(w)\|^2$ which set may contain more than one elements if a tie exists, and $M = \min_k\{k|k \in S(w)\}$ which is the smallest index among all values in the set $S(w)$. It follows that

$$\|\varepsilon_k(w)\|^2 \le \|\varepsilon_M(w)\|^2\ .$$

Let

$$\eta_k(w) := e^{\lambda\left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2\right)}$$

then based on Eq. (3) and Eq. (2), we have

$$\begin{aligned} C_\lambda(w) &= \frac{1}{\lambda}\ln\left[\frac{1}{K}J_\lambda(w)\right] \\ &= \frac{1}{\lambda}\ln\left[\frac{1}{K}\sum_{k=1}^{K}e^{\left(\lambda\left\|y_k-\hat{f}(x_k,w)\right\|^2\right)}\right] \\ &= \frac{1}{\lambda}\ln\left[\frac{1}{K}e^{\lambda\|\varepsilon_M(w)\|^2}\sum_{k=1}^{K}\eta_k(w)\right] \\ &= \frac{1}{\lambda}\ln\frac{1}{K} + \|\varepsilon_M(w)\|^2 + \frac{1}{\lambda}\ln\left[\sum_{k=1}^{K}\eta_k(w)\right]\ . \end{aligned} \tag{4}$$

The first-order derivative of $C_\lambda(w)$ can be computed as

$$\begin{aligned} \frac{\partial C_\lambda(w)}{\partial w_j} &= \frac{1}{\lambda J_\lambda(w)}\frac{\partial J_\lambda(w)}{\partial w_j} \\ &= \frac{1}{\lambda J_\lambda(w)}\left[-2\lambda\sum_{k=1}^{K}e^{\lambda\|\varepsilon_k(w)\|^2}\varepsilon_k^T(w)\frac{\partial\hat{y}_k(w)}{\partial w_j}\right] \\ &= \frac{-2\lambda e^{\lambda\|\varepsilon_M(w)\|^2}\sum_{k=1}^{K}\eta_k(w)\varepsilon_k^T(w)\frac{\partial\hat{y}_k(w)}{\partial w_j}}{\lambda e^{\lambda\|\varepsilon_M(w)\|^2}\sum_{k=1}^{K}\eta_k(w)} \\ &= \frac{-2\sum_{k=1}^{K}\eta_k(w)\varepsilon_k^T(w)\frac{\partial\hat{y}_k(w)}{\partial w_j}}{\sum_{k=1}^{K}\eta_k(w)} \end{aligned} \tag{5}$$

In training an MLP on a pairwise mode with K input/output pairs, a training epoch often contains K training iterations where each iteration takes one training pair to evaluate the gradient of the objective function. It is noticed that Eq. (5) only works on a batch mode for calculating $\frac{\partial C_\lambda(w)}{\partial w_j}$. Because the evaluations of $\sum_{k=1}^{K}\eta_k(w)$ and $\sum_{k=1}^{K}\eta_k(w)\varepsilon_k^T(w)\frac{\partial\hat{y}_k(w)}{\partial w_j}$ require to compute summations with K training pairs, and Eq. (5) is unable to perform the pairwise training through taking only one training pair to compute $\frac{\partial C_\lambda(w)}{\partial w_j}$ in a training iteration. For a training dataset where K is very large, the evaluation of $\frac{\partial C_\lambda(w)}{\partial w_j}$ in Eq. (5) is not efficient on a batch mode.

Algorithm 1 Pairwise NRAE Training Method

Require: Initialize the weight vector w^n randomly, choose a desired training error ϵ, select $\lambda >> 1$, $n \leftarrow 1$;
while $C_\lambda(w^n) > \epsilon$ **do**
  for $k = 1$ to K **do**
    Evaluate $\|\varepsilon_k(w^n)\|^2$ in respect to a weight vector w^n;
  end for
  Determine $\varepsilon_M(w^n)$;
  $D \leftarrow 0$;
  for $k = 1$ to K **do**
    $\eta_k(w^n) \leftarrow e^{\lambda\left(\|\varepsilon_k(w^n)\|^2 - \|\varepsilon_M(w^n)\|^2\right)}$;
    $D \leftarrow D + \eta_k(w^n)$;
  end for
  Set $w^n(1) \leftarrow w^n$;
  for $k = 1$ to K **do**
    $\frac{\partial C_\lambda(w^n(k))}{\partial w_j} \leftarrow \frac{-2\eta_k(w^n)\varepsilon_k^T(w^n(k))\frac{\partial\hat{y}_k(w^n(k))}{\partial w_j}}{D}$;
    update $w^n(k)$ to $w^n(k+1)$ by using $\frac{\partial C_\lambda(w^n(k))}{\partial w_j}$;
  end for
  $w^n \leftarrow w^n(K+1)$ and $n \leftarrow n+1$;
end while
return the optimal weight vector w^*;

For a weight vector w^n obtained in the n-th training epoch, a formula which applies the pairwise training to compute $\frac{\partial C_\lambda(w^n(k))}{\partial w_j}$ is described as

$$\frac{\partial C_\lambda(w^n(k))}{\partial w_j} = \frac{-2\eta_k(w^n(k))\varepsilon_k^T(w^n(k))\frac{\partial\hat{y}_k(w^n(k))}{\partial w_j}}{\sum_{i=1}^{K}\eta_i(w^n(k))} \tag{6}$$

where

$$\eta_k(w^n(k)) = e^{\lambda\left(\|\varepsilon_k(w^n(k))\|^2 - \left\|\varepsilon_{M(k)}(w^n)\right\|^2\right)}$$
$$\sum_{i=1}^{K}\eta_i(w^n(k)) = \sum_{i=1}^{K}e^{\lambda\left(\|\varepsilon_i(w^n(k))\|^2 - \left\|\varepsilon_{M(k)}(w^n)\right\|^2\right)}\ .$$

Although Eq.(6) is able to perform the pairwise training, it costs too much computational time to evaluate $\eta_k(w^n(k))$ and $\sum_{i=1}^{K}\eta_i(w^n(k))$ especially when K is very large. Because Eq.(6) has to decide $\varepsilon_{M(k)}(w^n)$ for K times in each training iteration, while it will cost K^2 evaluations on $\eta_k(w^n(k))$ and K^2 evaluations on $\sum_{i=1}^{K}\eta_i(w^n(k))$ per training epoch.

In order to reduce the expensive cost in Eq.(6), we propose a method to estimate the true value of $\frac{\partial C_\lambda(w^n(k))}{\partial w_j}$ as

$$\frac{\partial C_\lambda(w^n(k))}{\partial w_j} = \frac{-2\eta_k(w^n)\varepsilon_k^T(w^n(k))\frac{\partial\hat{y}_k(w^n(k))}{\partial w_k}}{\sum_{k=1}^{K}\eta_k(w^n)} \tag{7}$$

where

$$\eta_k(w^n) = e^{\lambda\left(\|\varepsilon_k(w^n)\|^2 - \|\varepsilon_M(w^n)\|^2\right)}.$$

It can be observed that Eq. (7) uses $\varepsilon_M(w^n)$ to compute $\eta_k(w^n)$, thus it does not need K evaluations to decide $\varepsilon_{M(k)}(w^n)$ in each training iteration like Eq. (6). Moreover, Eq. (7) solely computes $\eta_k(w^n)$ K times and $\sum_{k=1}^{K}\eta_k(w^n)$ once at the beginning of each training epoch, then it estimates $\frac{\partial C_\lambda(w^n(k))}{\partial w_j}$ for K times without updating $\eta_k(w^n)$ and $\sum_{k=1}^{K}\eta_k(w^n)$ anymore in each training epoch. Therefore, only K evaluations on $\eta_k(w^n)$ and one evaluation on $\sum_{k=1}^{K}\eta_k(w^n)$ are needed per training epoch in Eq. (7). A pairwise NRAE training method using Eq. (7) is described in Algorithm 1.

III. Numerical Experiments

Four function approximation examples designed to have nonglobal local minima and one recognition of handwritten numerals task using the MNIST dataset are applied to demonstrate effects of the pairwise NRAE training method. The GDC method in [15] is chosen to perform the NRAE training for all numerical experiments. In all training sessions, the derivatives of the MLP outputs are computed by BP algorithm, and the MLP weights are updated by the standard gradient descent optimization method with a momentum term. Several general parameters in training MLPs are chosen with the aid of suggestions in [19]: each synaptic weight in a weight vector is randomly selected from a uniform distribution between $-2.4/F_i$ and $2.4/F_i$, where F_i is the number of input neurons of the connected unit; all input and output values defined in the training data are normalized into $[-1, 1]$; the activation function in each training neuron is chosen as the hyperbolic tangent function $\varphi(v) = a\tanh(bv)$, where $a = 1.7159$ and $b = 2/3$.

A. Approximation of Functions designed with Nonglobal Local Minima

In our experiments, we test and compare four different training methods: Pairwise training method with GDC (pairwise GDC training), batch training method with GDC (batch GDC training), pairwise training method with least mean square (LMS) error (LMS training), and batch training method with MSE (MSE training). For each function approximation example, ten different initial weight vectors of an MLP with a certain architecture are randomly chosen. Starting with such an initial weight vector, one pairwise GDC training, one batch GDC training, one LMS training, and one MSE training are performed. These four training sessions for the same initial weight vector are considered as a training group. The corresponding value of MSE of the MLP is recorded as the training error for each session after the training is converged. To compare different training methods in approximating functions, we collect the mean and standard deviation of training errors among ten training groups for each training method. In addition, we consider the time in seconds as the training cost of each session, and we compute the mean of training costs among ten training groups as the average training cost for each training method. The training time of each session is recorded after the corresponding training is converged.

For all training sessions performed on function approximation examples, we set the learning rate and the momentum term of the gradient descent optimization method equal to 0.01 and 0.5, respectively. For the GDC method, we set $\lambda = 10^4$ as the initial value, $E = 1000$ as the maximum training epochs to check the deviation of cost function values, $T = 10^{-8}$ as the threshold to decide when the deconvexification is needed, and $R = 0.9$ as the deconvexification rate of λ.

Target functions applied in function approximation examples are presented in Fig. 1. Definitions of these target functions with training data and MLP architectures for the experiments are described as following:

1) Three-notch: A function with three notches is defined by

$$y = f(x) = \begin{cases} 0 & \text{if } x \in [0, 1.0] \cup [2.2, 2.3] \cup [3.5, 4.5] \\ 0.25 & \text{if } x \in [2.8, 3.0] \\ 0.5 & \text{if } x \in [1.5, 1.7] \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

where $x \in X = [0, 4.5]$. For the training data, input values x_k are selected by randomly sampling 2000 different numbers from a uniform distribution on X, and corresponding output values y_k are computed by Eq. (8). Then, a training dataset of 2000 (x_k, y_k) pairs is obtained. MLPs with the 1:16:1 architecture are used in all training sessions to approximate the three-notch function.

2) Fine Features: A smooth function with two fine features as spikes is defined by

$$y = f(x) = g\left(x, \frac{1}{6}, \frac{1}{2}, \frac{1}{6}\right) + g\left(x, \frac{1}{64}, \frac{1}{4}, \frac{1}{128}\right) + g\left(x, \frac{1}{64}, \frac{11}{20}, \frac{1}{128}\right) \tag{9}$$

where $x \in X = [0, 1]$, and g is defined as

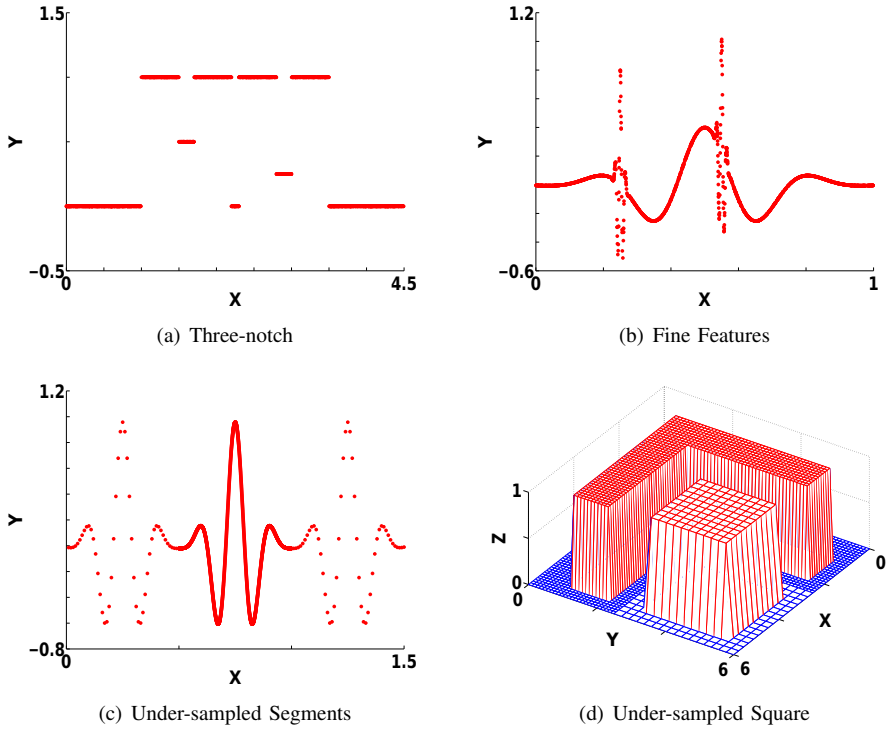
$$g(x, \alpha, \mu, \sigma) = \frac{\alpha}{\sqrt{2\pi}\sigma}\cos\left(\frac{(x-\mu)\pi}{\sigma}\right)\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{10}$$

For the training data, input values x_k are selected by randomly sampling 2000 different numbers from a uniform distribution on X, and corresponding output values y_k are computed by Eq. (9). Then, a training dataset of 2000 (x_k, y_k) pairs is obtained. MLPs with the 1:15:1 architecture are used in all training sessions to approximate the fine features function.

3) Under-sampled Segments: A smooth function with two under-sampled segments is defined by

$$y = f(x) = g\left(x, \frac{1}{5}, \frac{1}{4}, \frac{1}{12}\right) + g\left(x, \frac{1}{5}, \frac{3}{4}, \frac{1}{12}\right) + g\left(x, \frac{1}{64}, \frac{5}{4}, \frac{1}{12}\right) \tag{11}$$

where $x \in X = [0, 1.5]$ and g is defined in Eq. (10). For the training data, input values x_k are collected by using 50



(a) Three-notch (b) Fine Features

(c) Under-sampled Segments (d) Under-sampled Square

Fig. 1. Target functions for function approximation examples in Section III-A. Numbers on horizontal and vertical axes in each subfigure represent the input and output of the function, respectively. From Fig. 1(a) to Fig. 1(c), red dots denote to the target training data. In Fig. 1(d), different colors (blue and red) are used to distinguish different output values (0 and 1) of the target function on vertical axis.

grid points from a uniform grid on $[0, 0.5]$, 50 grid points from a uniform grid on $[1.0, 1.5]$, and 2000 grid points from a uniform grid on $(0.5, 1.0)$. Corresponding output values y_k are computed by Eq. (11). These form a training dataset of 2100 (x_k, y_k) input/output pairs. MLPs with the 1:12:1 architecture are used in all training sessions to approximate the under-sampled segments function.

4) Under-sampled Square: A three-dimensional function, which has a letter 'L' shape and an under-sampled square raised from a plane, is defined on $[0, 6] \times [0, 6]$ by

$$z = f(x, y) = \begin{cases} 1 & \text{if } x \in [1.0, 5.5] \text{ and } y \in [1.0, 2.0] \\ 1 & \text{if } x \in [1.0, 2.0] \text{ and } y \in [2.0, 5.5] \\ 1 & \text{if } x \in [3.0, 5.5] \text{ and } y \in [3.0, 5.5] \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

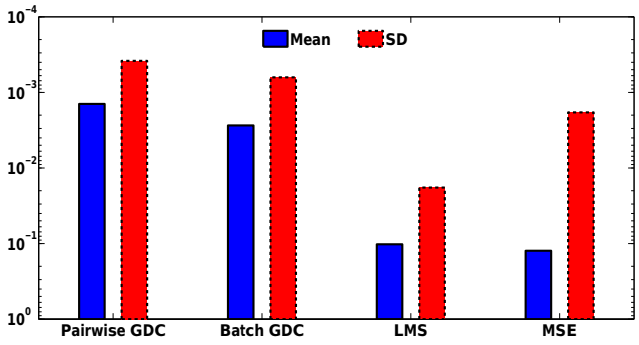
In the training data, input values x_k and y_k are the 289 grid points from the uniform grid on $(2.5, 6] \times (2.5, 6]$ and 2522 grid points from the uniform grid on $[0, 6] \times [0, 6] - (2.5, 6] \times (2.5, 6]$. Corresponding output values z_k are computed by Eq. (12). These form a training dataset of 2811 (x_k, y_k) pairs. MLPs with the 2:9:3:1 architecture are used in all training sessions to approximate the under-sampled square function.

Fig. 2 shows means and standard deviations of training errors achieved by four tested training methods in function approximation examples. The result demonstrates that both the pairwise and batch GDC training methods consistently achieve lower average training errors and standard deviations than the LMS and MSE training methods. Therefore, it confirms that the GDC method has the ability to avoid nonglobal local minima, thus producing better training results than the LMS and MSE method in training MLPs. Furthermore, the pairwise GDC training method reaches the same level of average training errors as the batch GDC training, implying that the pairwise GDC training method is capable to be applied to train MLPs in solving practical tasks such as the function approximation.

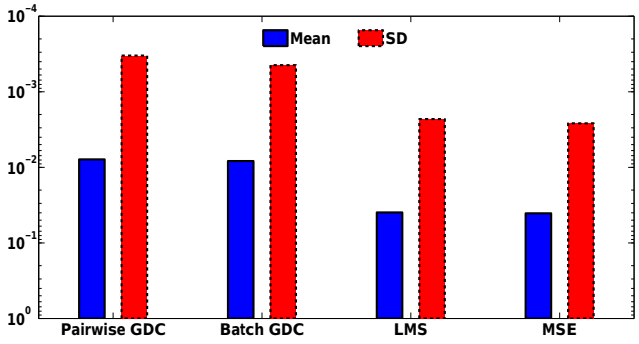
In Fig. 3, it presents average training costs of four tested training methods in function approximation examples. The pairwise GDC training reduces nearly 42%, 56%, 16%, and 11% average training costs comparing to the batch GDC training in approximating the three-notch, fine features, under-sampled segments, and under-sampled square functions, respectively. In addition, it can be observed in Fig. 3 that the LMS and MSE training take less computational time than the pairwise and batch GDC training, respectively. The reason is that the formulas of the objective functions and gradients of the LMS and MSE criteria are simpler than the NRAE criterion to be evaluated. However, comparing to the GDC training, even the LMS and MSE training are fast, training errors achieved by both of them never outperform the results obtained by the GDC method in all training sessions of our experiments.

B. Handwritten Numeral Recognition using the MNIST Dataset

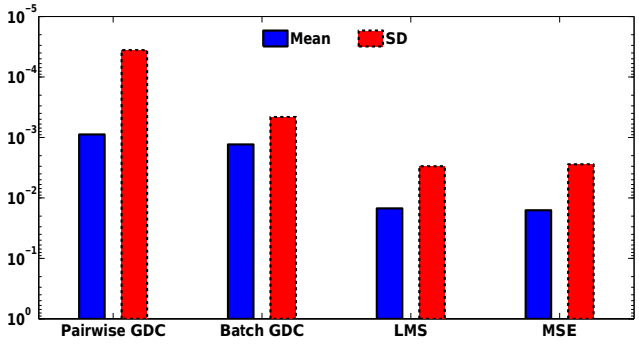
To test the capability of the pairwise training method with GDC in training a large MLP on a large real-world



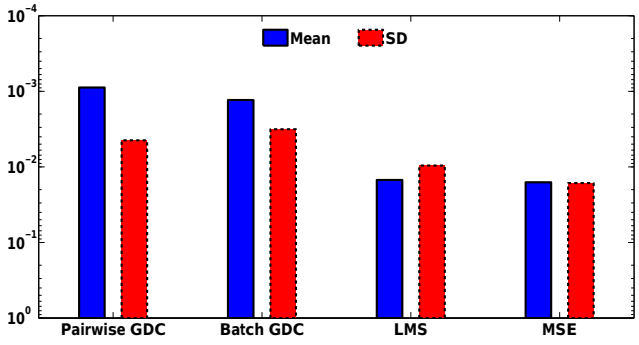
(a) Three-notch



(b) Fine Features



(c) Under-sampled Segments



(d) Under-sampled Square

Fig. 2. Means and standard deviations of training errors for function approximation examples in Section III-A. Blue bars surrounding by solid lines and red bars surrounding by dash lines in each subfigure denote to the means and standard deviations of training errors collected among ten different training sessions, respectively. The corresponding value of MSE of the MLP is collected as the training error for each session after the training is converged. Vertical axes in each subfigure apply the logarithmic scale with the base 10.

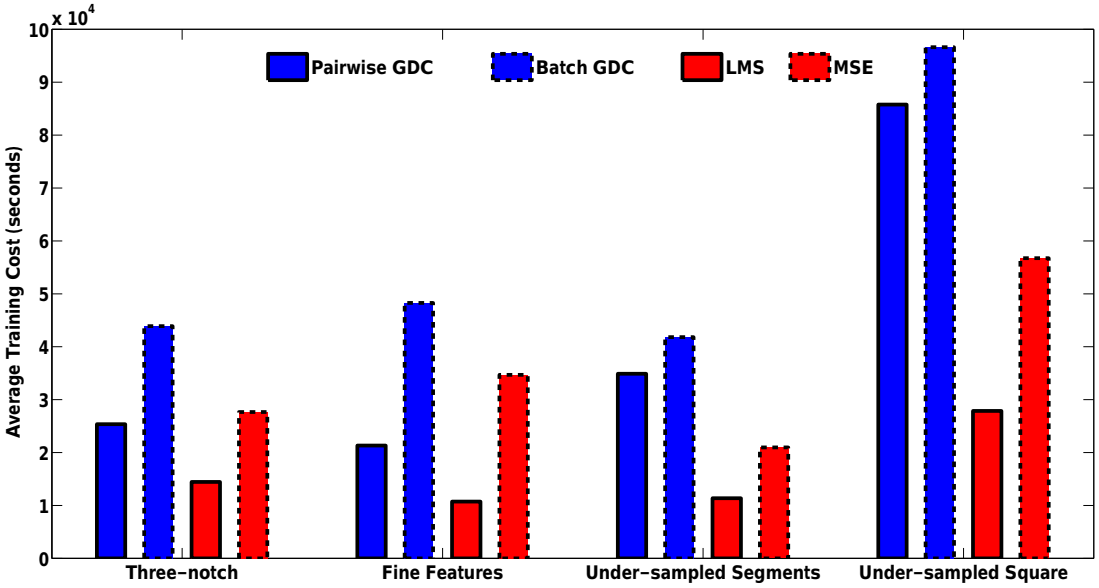


Fig. 3. Average training costs for function approximation examples in Section III-A. Blue bars and red bars denote to the pairwise/batch GDC training and the LMS/MSE training, respectively. Bars surrounding by solid lines represent pairwise training methods, and bars surrounding by dash lines represent batch training methods. Numbers on the vertical axis shows the average training costs in seconds for tested training methods.

task, it is used to train an MLP for recognizing handwritten numerals on the MNIST dataset [21]. The MNIST dataset is commonly used as a benchmark to compare performances of different classifiers including many neural networks. The MNIST dataset contains 60,000 training samples and 10,000 test samples of handwritten numerals. Each sample has 784 features which are obtained from a 28×28 black and white image. Each feature value is the anti-aliasing normalized gray level of the corresponding pixel in an image.

In our experiments, we test the MNIST dataset on a 2-layer MLP with the architecture of 784:300:10, where each of the ten output nodes is associated with one of the ten numerals, 1, 2, ..., 9, and 0. For the training of the MLP, the target value of an output node is 1 if the input is the numeral associated with the node, and is -1 otherwise. After the MLP is trained, the numeral associated with the output node of the MLP outputting the highest value among the ten output nodes is selected as the result of classifying the input image. The classification accuracy of the trained MLP is defined as the percentage of test images that are correctly classified in the test dataset, and the test error rate is defined as 100% minus the classification accuracy of the trained MLP. Five different initial weight vectors are used to perform five training sessions. In each training session, 60000 training samples are used to train the MLP. After the corresponding training is converged, 10000 testing samples are used to test the trained MLP, and the test error rate is computed for each testing session as well.

For all training sessions performed on the MNIST dataset, we set the learning rate and the momentum term of the gradient descent optimization method equal to 0.0001 and 0.5, respectively. For the GDC method, we set $\lambda = 10^4$ as the initial value, $E = 10$ as the maximum training epochs to check the deviation of cost function values, $T = 10^{-5}$ as the threshold to decide when the deconvexification is needed, and $R = 0.9$ as the deconvexification rate of λ.

Table I presents test error rates of the MNIST dataset obtained by different MLP classifiers. For the pairwise GDC training, the mean and standard deviation of the test error rates among five testing sessions are shown in the table. Four major advantages of the pairwise GDC method in training an MLP classifier are described in following:

1) Experimental results in Table I show that the MLP classifier trained by the pairwise GDC method has the lowest test error rate than many benchmark MLP classifiers. Based on our experiments, the MLP classifiers trained by the pairwise GDC method consistently achieve low test error rates on all five testing sessions with the same settings of MLPs. It indicates that the pairwise GDC training method has the ability to provide satisfactory generalization results with different initial weight vectors, thus requiring no multiple selections of initial weight vectors for the training.
2) For three benchmark MLP classifiers listed in Table I, one classifier uses 1000 hidden nodes on a 2-layer MLP, and other two classifiers are built by 3-layer MLPs where each classifier has more than 300 hidden nodes in its hidden layers. However, the MLP classifier trained by the pairwise GDC method solely uses 300 hidden nodes on a 2-layer MLP. It demonstrates that the pairwise GDC method has the ability to avoid nonglobal local minima and maintain

TABLE I
TEST ERROR RATES OF THE MNIST DATASET FOR MLP CLASSIFIERS[1]

CLASSIFIER	TEST ERROR RATE (%)
Pairwise Linear classifier (1-layer MLP) with image deskewing	7.6
2-layer MLP 784:300:10, MSE	4.7
2-layer MLP 784:300:10, Pairwise GDC	$\mathbf{2.7 \pm 0.03}$
2-layer MLP 784:1000:10, MSE	4.5
2-layer MLP 784:300:10, MSE with image distortions	3.6
2-layer MLP 784:1000:10, MSE with image distortions	3.8
3-layer MLP 784:300:100:10, MSE	3.05
3-layer MLP 784:500:150:10, MSE	2.95

a high level of generalization without applying a large MLP architecture comparing to the listed benchmark MLP classifiers.

3) It is noticed that there are two benchmark MLP classifiers listed in Table I perform training sessions by applying image distortions to increase the sizes of the training samples. The technique of image distortions improves test error rates of the two benchmark MLP classifiers, but the results obtained by those MLP classifiers are still worse than the test error rate of the MLP classifier trained by the pairwise GDC method without image distortions. It indicates that the MLP classifier is able to be trained effectively by the pairwise GDC method in achieving low test error rate. In fact, through applying image distortions, the pairwise GDC method is expected to lead the MLP classifier to achieve a lower test error rate than the result of the MLP classifier trained without the aiding of image distortions.
4) A batch GDC training applying the same experimental settings as the pairwise GDC training is also tested on the MNIST dataset. Since the batch GDC training converges very slow comparing to the pairwise GDC training, there is no converged batch GDC training session has been obtained in our experiments, thus no tested result is included in Table I. The lowest test error rate achieved by the batch GDC training is nearly 10%, which costs almost 15,000 training epochs. However, in all pairwise GDC training sessions, the total number of training epochs for each session is less than 1,000. It presents a significant advantage of the pairwise GDC training comparing to the batch GDC training in saving computational costs on a large training dataset.

[1]The row highlighted by bold fonts is the result of the MLP classifier trained by the pairwise GDC method. The test error rate of this classifier is presented as the mean and standard deviation computed among five testing sessions. Other results without highlighted are benchmark test error rates trained by different kinds of MLP classifiers. More classifiers and benchmark results are available at the MNIST website: http://yann.lecun.com/exdb/mnist/index.html.

IV. CONCLUSION

A pairwise training method to execute gradual deconvexification (GDC) is proposed and tested on both function approximation and handwritten numeral recognition. Numerical experiments demonstrate that the pairwise GDC method achieves an accuracy level similar to that of the batch GDC training, but requires much less computation especially in training a large dataset. It is appropriate to note that the pairwise training method with GDC is able to train an MLP classifier on the MNIST dataset and achieve a better test error rate than those in many benchmark results. Future work will be done on improving the pairwise training method to speedup GDC and enhancing the generalization capability for challenging image recognition tasks, such as CIFAR-10 / CIFAR-100 and ImageNet datasets.

REFERENCES

[1] E. Aarts and J. Korst, *The Neuron*. Oxford University Press, 1989.
[2] J. M. Zurada, *Introduction to Artificial Neural Networks*. St. Paul, MN: West Publishing Company, 1992.
[3] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. Cambridge, Massachusetts: MIT Press, 1995.
[4] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs, third edition*. New York: Springer, 1999.
[5] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals through Simulations*. New York: John Wiley and Sons, Inc., 2000.
[6] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY: Springer, 2006.
[7] K.-L. Du and M. Swamy, *Neural Networks in a Softcomputing Framework*. New York, NY: Springer, 2006.
[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 3rd ed. New York, NY: Cambridge University Press, 2007.
[9] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 2008.
[10] J. T.-H. Lo and D. Bassu, "An adaptive method of training multilayer perceptrons," in *Proceedings of the 2001 International Joint Conference on Neural Networks*, vol. 3, Jul. 2001, pp. 2013–2018.
[11] J. T.-H. Lo, "Convexification for data fitting," *Journal of Global Optimization*, vol. 46, no. 2, pp. 307–315, Feb. 2010.
[12] J. T.-H. Lo, Y. Gui, and Y. Peng, "Overcoming the local-minimum problem in training multilayer perceptrons with the NRAE training method," in *Advances in Neural Networks - ISNN 2012*, vol. Part I, Jul. 2012, pp. 440–447.
[13] J. T.-H. Lo, Y. Gui, and Y. Peng, "Overcoming the local-minimum problem in training multilayer perceptrons with the NRAE-MSE training method," in *Advances in Neural Networks - ISNN 2013*, vol. Part I, Jul. 2013, pp. 83–90.

[14] J. T.-H. Lo, Y. Gui, and Y. Peng, "The normalized risk-averting error criterion for avoiding nonglobal local minima in training neural networks," *to appear in Neurocomputing*, 2014.
[15] J. T.-H. Lo, Y. Gui, and Y. Peng, "Overcoming the local-minimum problem in training multilayer perceptrons by gradual deconvexification," in *Proceedings of the 2013 International Joint Conference on Neural Networks*, Aug. 2013, pp. 635–640.
[16] L. Fausett, *Fundamentals of Neural Networks*. Englewood Cliffs, New Jersey: Prentice Hall, 1994.
[17] H. Demuth and M. Beale, *Neural Network Toolbox User's Guide*. Natick, MA: MathWorks, Inc., 1994.
[18] Y. Bengio, *Neural Networks for Speech and Sequence Recognition*. New York, NY: International Thomson Computer Press, 1996.
[19] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, "Efficient backprop," *Lecture Notes in Computer Science in Neural Networks: Tricks of the Trade*, vol. 1524, pp. 9–50, 1998.
[20] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, Dec 2003.
[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.