Neurocomputing 287 (2018) 118–127

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

EC-DNN: A new method for parallel training of deep neural networks

Shizhao Sun[*], Xiaoguang Liu

College of Computer and Control Engineering, Nankai University, Tianjin, PR China

A R T I C L E   I N F O

Article history:
Received 17 December 2017
Revised 16 January 2018
Accepted 24 January 2018
Available online 2 February 2018

Communicated by Dr. Tie-Yan Liu

Keywords:
Parallel machine learning
Distributed machine learning
Deep learning

A B S T R A C T

Parallelization framework has become a necessity to speed up the training of deep neural networks (DNN) recently. In the typical parallelization framework, called MA-DNN, the parameters of local models are periodically averaged to get a global model. However, since DNN is a highly non-convex model, averaging parameters cannot ensure that such global model can perform better than those local models. To tackle this problem, we introduce a new parallelization framework, called EC-DNN. In this framework, we propose to aggregate the local models by the simple ensemble, i.e., averaging the outputs of local models instead of the parameters. As most of prevalent loss functions are convex to the output of DNN, the performance of the global model produced by the simple ensemble is guaranteed to be at least as good as the average performance of local models. To get more performance improvement, we extend the simple ensemble to the generalized ensemble, which produces the global model by the weighted sum instead of the average of the outputs of the local models. However, the model size will explode since each round of ensemble can give rise to multiple times size increment. Thus, we carry out model compression after each ensemble to reduce the size of the global model to be the same as the local ones. Our experimental results show that EC-DNN can achieve better speedup than MA-DNN without loss of accuracy, and there is even accuracy improvement sometimes.

1. Introduction

Recent rapid development of deep neural networks (DNN) has demonstrated that its great success mainly comes from big data and big models [1,2]. However, it is extremely time-consuming to train a large-scale DNN model over big data. To accelerate the training of DNN, parallelization frameworks like MapReduce [3] and Parameter Server [4,5] have been widely used. A typical parallel training procedure for DNN consists of continuous iterations of the following three steps. First, each worker trains the local model based on its own local data by stochastic gradient decent (SGD) or any of its variants. Second, the parameters of the local DNN models are communicated and aggregated to obtain a global model, e.g., by averaging the parameters of each local models [6,7]. Finally, the obtained global model is used as a new starting point of the next round of local training. We refer the method that performs the aggregation by averaging model parameters as *MA*, and the corresponding parallel implementation of DNN as *MA-DNN*.

However, since DNN is a highly non-convex model, the loss of the global model produced by MA cannot be guaranteed to be upper bounded by the average loss of the local models. In other words, the global model obtained by MA might even perform worse than any local model. As the global model will be used as a new starting point of the successive local training, the poor performance of the global model will drastically slow down the convergence of the training process and further hurt the performance of the final model.

To tackle this problem, we propose a novel framework for parallel DNN training, called *Ensemble-Compression (EC-DNN)*, the key idea of which is to produce the global model by ensemble instead of MA. We first consider the simple ensemble. In the simple ensemble, the local models are aggregated by averaging their outputs instead of their parameters. Since most of widely-used loss functions for DNN are convex with respect to the output vector of the model, the loss of the global model produced by the simple ensemble can be upper bounded by the average loss of the local models. Then, to get more performance improvement, we extend the simple ensemble to the generalized ensemble, in which the coefficients of the local models are evaluated instead of directly set as $1/K$ (where K is the number of the local workers). For ease of reference, we collectively call simple ensemble and generalized ensemble *ensemble*. According to previous theoretical and empirical studies [8,9], ensemble model tend to yield better results when there exists a significant diversity among local models. Therefore, we train the local models for a longer period for EC-DNN to increase the diversity among them. In other words, EC-DNN yields

[*] Corresponding author.
E-mail address: sunshizhao@mail.nankai.edu.cn (S. Sun).

https://doi.org/10.1016/j.neucom.2018.01.072
0925-2312/© 2018 Elsevier B.V. All rights reserved.

less communication frequency than MA-DNN, which further emphasizes the advantages of EC-DNN by reducing communication cost as well as increasing robustness to limited network bandwidth.

There is, however, no free lunch. In particular, the ensemble method will critically increase the model complexity: the resultant global equals to a larger neural network that is K times wider and one layer deeper than the local model. Several ensemble iterations may result in explosion of the size of the global model. To address this problem, we further propose an additional compression step after the ensemble. This approach cannot only restrict the size of the resultant global model to be the same size as local ones, but also preserves the advantage of ensemble over MA. Given that both ensemble and compression steps are dispensable in our new parallelization framework, we name this framework as EC-DNN. As a specialization of the EC-DNN framework, we adopt the distillation based compression [10–12], which produces model compression by distilling the predictions of big models. Nevertheless, such distillation method requires extra time for training the compressed model. To tackle this problem, we seek to integrate the model compression into the process of local training by designing a new combination loss, a weighted interpolation between the loss based on the pseudo labels produced by global model and that based on the true labels. By optimizing such combination loss, we can achieve model compression in the meantime of local training.

We conducted comprehensive experiments on CIFAR-10, CIFAR-100, and ImageNet datasets. We have following important observations from the experiments: (1) Ensemble is a better model aggregation method than MA consistently. MA suffers from that the performance of the global model could vary drastically and even be much worse than the local models; meanwhile, the global model obtained by the ensemble method can consistently outperform the local models. (2) In terms of the end-to-end results, EC-DNN stably achieved better speedup than MA-DNN in all the settings. (3) The speedup of EC-DNN over MA-DNN is achieved without loss of accuracy, and there is even accuracy improvement sometimes. (4) EC-DNN can achieve better performance than MA-DNN even when EC-DNN communicates less frequently than MA-DNN, which emphasizes the advantage of EC-DNN in training a large-scale DNN model as it can significantly reduce the communication cost.

2. Preliminary: parallel training of DNN

We denote a DNN model as $f(\mathbf{w})$ where $\mathbf{w}$ represents the parameters. In addition, we denote the outputs of the model $f(\mathbf{w})$ on the input x as $f(\mathbf{w}; x) = \{f(\mathbf{w}; x, 1), \ldots, f(\mathbf{w}; x, C)\}$, where C is the number of classes and $f(\mathbf{w}; x, c)$ denotes the output (i.e., the score) for the cth class.

In the parallel training of DNN, suppose that there are K local workers. Each local worker holds a local dataset $D_k = \{(x_{k,1}, y_{k,1}), \ldots, (x_{k,m_k}, y_{k,m_k})\}$ with size m_k, $k \in \{1, \ldots, K\}$. Denote the parameters of the DNN model at the iteration t on the local worker k as $\mathbf{w}_k^t$. The communication between the local workers is invoked after every τ iterations of updates for the parameters, and we call τ the communication frequency.

A typical parallelization framework of DNN iteratively implements the following three steps:

1. *Local training:* At iteration t, local worker k updates its local model by SGD. Such local model is updated for τ iterations before the communication between different local workers.

2. *Model aggregation:* The parameters of local models are communicated across the local workers. Then, a global model is produced by aggregating local models according to certain aggregation method.

3. *Local model reset:* The global model is sent back to the local workers, and set as the starting point for the next round of local training.

We denote the aggregation method in the second step as $G(\mathbf{w}_1^t, \ldots, \mathbf{w}_K^t)$ and the parameters of the global model as $\bar{\mathbf{w}}^t$. That is, $f(\bar{\mathbf{w}}^t) = G(\mathbf{w}_1^t, \ldots, \mathbf{w}_K^t)$, where $t = \tau, 2\tau, \ldots$. A widely-used aggregation method is *model average (MA)* [5–7,13–15], which averages each parameter over all the local models, i.e.,

$$G_{MA}\left(\mathbf{w}_1^t, \ldots, \mathbf{w}_K^t\right) = f\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{w}_k^t\right), t = \tau, 2\tau, \ldots. \quad (1)$$

We denote the parallel training method of DNN that using MA as MA-DNN for ease of reference.

3. Model aggregation

In this section, we first reveal why the MA method cannot guarantee to produce a global model with better performance than local models. Then, we propose to use ensemble method to perform the model aggregation, which in contrast can ensure to perform better over local models.

3.1. MA

For any model aggregation method, we hope that the performance of the global model is at least not worse than the average performance of the local models, i.e.,

$$\mathcal{L}(f(\bar{\mathbf{w}}^t; x), y) \le \frac{1}{K}\sum_{k=1}^{K}\mathcal{L}(f(\bar{\mathbf{w}}_k^t; x), y), \quad (2)$$

where the performance is evaluated by the loss. Otherwise, the aggregation would be meaningless.

For MA, two conditions must be satisfied if we want to guarantee that the performance of the global model is not worse than the average performance of the local models. First, the model should be convex with respect to the model parameters $\mathbf{w}_k^t$, i.e.,

$$\mathcal{L}\left(f\left(\bar{\mathbf{w}}^t; x\right), y\right) = \mathcal{L}\left(f\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{w}_k^t; x\right), y\right) \le \mathcal{L}\left(\frac{1}{K}\sum_{k=1}^{K}f\left(\mathbf{w}_k^t; x\right), y\right). \quad (3)$$

Second, the loss should be convex with respect to the model outputs $f(\cdot x)$, i.e.,

$$\mathcal{L}\left(\frac{1}{K}\sum_{k=1}^{K}f\left(\mathbf{w}_k^t; x\right), y\right) \le \frac{1}{K}\sum_{k=1}^{K}\mathcal{L}\left(f\left(\mathbf{w}_k^t; x\right), y\right). \quad (4)$$

However, DNN is a highly non-convex model due to the existence of activation functions and pooling functions (for convolutional layers). Therefore, MA method cannot always produce the global model with guaranteed better performance than local ones. Furthermore, given that the global model is usually used as the starting point of the next round of local training, the performance of the final model could hardly be good if a global model in any round fails to achieve good performance. Beyond the theoretical analysis above, the experimental results reported in Section 6.3 and previous studies [7,15] also revealed such problem.

3.2. Simple ensemble and generalized ensemble

While the DNN model itself is non-convex, we notice that most of widely-used loss functions for DNN are convex w.r.t. the model outputs (e.g., cross entropy loss, square loss, and hinge loss). Therefore, Eq. (4) holds, which indicates that averaging the output of the local models instead of their parameters guarantees to yield better performance than local models. To this end, we propose to use *simple ensemble* to aggregate different local models, which averages outputs of the local models as follows:

$$G_{SE}\left(\mathbf{w}_1^t,\dots,\mathbf{w}_K^t\right)=\frac{1}{K}\sum_{k=1}^{K}f\left(\mathbf{w}_k^t;x\right),t=\tau,2\tau,\dots. \tag{5}$$

Moreover, if the coefficients of the local models can be evaluated or tuned instead of directly set as $1/K$, we have the chance to further improve the performance of simple ensemble. To this end, we extend the simple ensemble to the *generalized ensemble*, which produces the global model by the weighted sum instead of the direct average of the outputs of the local models, i.e.,

$$G_{GE}\left(\mathbf{w}_1^t,\dots,\mathbf{w}_K^t\right)=\sum_{k=1}^{K}\alpha_k f\left(\mathbf{w}_k^t;x\right),t=\tau,2\tau,\dots. \tag{6}$$

For the coefficients $\{\alpha_1,\dots,\alpha_K\}$ in the generalized ensemble, we propose two methods to evaluate them, i.e., regression based method and margin based method.

1. *Regression based method:* The generalized ensemble can be viewed as a linear regression problem with the target to produce a global model as accurate as possible. Specifically, the coefficients can be viewed as the parameters and the predictions of the local models can be viewed as inputs.

Therefore, a direct way is to learn the coefficients in the generalized ensemble by minimizing the loss based on the true labels on the training data [16,17], i.e.,

$$\{\alpha_1,\dots,\alpha_K\}=\arg\min_{\{\alpha_1',\dots,\alpha_K'\}}\sum_{(x,y)\in D}\left(y-\sum_{k=1}^{K}\alpha_k' f\left(\mathbf{w}_k^t;x\right)\right)^2, \tag{7}$$

where l_2 loss is used for simplicity, and D denote the local training data. We denote the corresponding aggregation method as G_{GE--R}.

Compared to the huge training time cost by training the deep neural networks, the extra learning time introduced by G_{GE--R} is negligible. The reason is two-fold. First, the learning of coefficients is only a linear regression problem, which is much easier than the learning of the DNN model. Second, the dimension of the coefficients in the generalized ensemble equals to the number of the local workers, which is quite small compared to the huge number of parameters in the DNN model.

2. *Margin based method:* Besides assigning coefficient to each local model, a fine-grained way is to also assign coefficient to each data point (x, y). By this way, the coefficients are specified for each data point, and thus the performance can be further improved.

To this end, for each data point, we consider putting more confidence on the local model that is more confident for its classification result on this data point. Specifically, we use the margin [18] to measure such confidence. The margin measures the gap between the prediction for the true class and the maximum prediction for all the other classes. By this way, the margin not only indicates the correctness of the classification but also shows the confidence of the classification result.

Therefore, the coefficient can be set to be proportional to the margin of the local model on the data point, i.e.,

$$\alpha_{k,x,y}\propto\max\left(f\left(\mathbf{w}_k^t;x,y\right)-\max_{i\neq y}f\left(\mathbf{w}_k^t;x,i\right),0\right),$$
$$\times(x,y)\in D,k\in\{1,\dots,K\}. \tag{8}$$

where $f\left(\mathbf{w}_k^t;x,y\right)$ is the prediction for the true class and $\max_{i\neq y}f\left(\mathbf{w}_k^t;x,i\right)$ is the maximum prediction for all the other classes. We denote the corresponding aggregation method as G_{GE--M}.

During the test, since we cannot access the true labels, we just directly set $\alpha_1=\dots=\alpha_K=\frac{1}{K}$. That is, G_{GE--M} can only improve the training performance of the simple ensemble, but cannot improve the test performance. However, G_{GE--M} is used iteratively during the training process of EC-DNN (see more details in Section 4) and thus can improve the training performance of EC-DNN. Therefore, although G_{GE--M} cannot directly improve the test performance of the simple ensemble, it can improve the test performance of the whole EC-DNN by improving the training performance of EC-DNN.

In the remaining part of this paper, we collectively call the simple ensemble, i.e., G_{SE}, and the generalized ensemble, i.e., G_{GE--R} and G_{GE--M}, *ensemble*, for ease of reference.

4. EC-DNN

4.1. Framework

The details of EC-DNN framework[1] is shown in Algorithm 1. Note that, in this paper, we focus on the synchronous case[2] within the MapReduce framework, but EC-DNN can be generalized into the asynchronous case and parameter server framework as well. Similar to MA-DNN, EC-DNN also iteratively conducts local training, model aggregation, and local model reset.

1. *Local training:* The local training process of EC-DNN is the same as that of MA-DNN, in which the local model is updated by SGD. Specifically, at iteration t, local worker k updates its local model from $\mathbf{w}_k^t$ to $\mathbf{w}_k^{t+1}$ by minimizing the training loss using SGD, i.e., $\mathbf{w}_k^{t+1}=\mathbf{w}_k^t-\eta\Delta(\mathcal{L}(f(\mathbf{w}_k^t;x_k),y_k))$, where η is the learning rate, and $\Delta(\mathcal{L}(f(\mathbf{w}_k^t;x_k),y_k))$ is the gradients of the empirical loss $\mathcal{L}(f(\mathbf{w}_k^t;x_k),y_k)$ of the local model $f(\mathbf{w}_k^t)$ on one mini batch of the local dataset D_k. One local model will be updated for τ iterations before the communication between local workers.

2. *Model aggregation:* During the model aggregation, the parameters of local models, i.e., $\mathbf{w}_1^t\dots\mathbf{w}_K^t$, are communicated across local workers. To this end, a global model is produced by ensemble according to G_{SE}, G_{GE--R} or G_{GE--M} (see Eqs. (5), (7) and (8)). Equivalently, the global model produced by ensemble is a larger network with one additional layer with the parameters as $1/K$ for G_{SE} and $\{\alpha_1,\dots,\alpha_K\}$ for G_{GE--R} and G_{GE--M}, whose outputs consist of C nodes representing C classes, and whose inputs are those outputs from local models, where K is the number of local models.

Note that such global model (i.e., $f(\bar{\mathbf{w}}_t)$) produced by the ensemble is one layer deeper and K times wider than the local model (i.e., $f(\mathbf{w}_t^k)$). Therefore, continuous rounds of ensemble process will easily give rise to a global model with exploding size. To avoid this problem, we propose introducing a compression process (i.e., Compression($\mathbf{w}_k^t$, $\bar{\mathbf{w}}^t$, D_k) in Algorithm 1) after ensemble process, to compress the resultant global model to be the same size as those local models while preserving the advantage of the ensemble over MA. We denote the compressed model for the global model $\bar{\mathbf{w}}_t$ on the local worker k as $\tilde{\mathbf{w}}_k^t$.

3. *Local model reset:* The compressed model will be set as the new starting point of the next round of local training, i.e., $\mathbf{w}_k^t=\tilde{\mathbf{w}}^t$ where $t=\tau,2\tau,\dots$.

At the end of the training process, EC-DNN will output K local models and we choose the model with the smallest training loss as the final one. Note that, we can also take the global model (i.e., the

[1] In this paper, we limit the discussion of EC-DNN in the area of feed-forward neural network (including fully connected layers and convolutional layers). For the recurrent neural network, we leave it for future work because it needs special methods for model compression.

[2] As shown in [14], MA-DNN in synchronous case converges faster and achieves better test accuracy than that in asynchronous case.

ensemble of K local models) as the final model if there are enough computation and storage resources for the test.

4.2. Comparison with traditional ensemble methods

Traditional ensemble methods for DNN [1,19] usually first train several DNN models independently without communication and make ensemble of them only at the end of the training. We denote such method as E-DNN. E-DNN was proposed to improve the accuracy of DNN models by reducing variance and it has no necessity to train base models with parallelization framework. In contrast, EC-DNN is a parallel algorithm aiming at training DNN models faster without the loss of the accuracy by leveraging a cluster of machines.

Although E-DNN can be viewed as a special case of EC-DNN with only one final communication and no compression process, the intermediate communications in EC-DNN will make it outperform E-DNN. The reasons are as follows: (1) local workers has different local data, the communications during the training will help local models to be consensus towards the whole training data; (2) the local models of EC-DNN can be continuously optimized by compressing the ensemble model after each ensemble process. Then, another round of ensemble will result in more advantage for EC-DNN over E-DNN since the local models of EC-DNN has been much improved.

5. Implementations

In this section, we introduce one concrete implementation of the EC-DNN framework. Algorithm 1 contains two sub-problems

Algorithm 1: EC-DNN(D_k).

```
Randomly initialize $\mathbf{w}_k^0$ and set $t = 0$;
while stop criteria are not satisfied do
    $\mathbf{w}_k^{t+1} \leftarrow \mathbf{w}_k^t - \eta\Delta(\mathcal{L}(f(\mathbf{w}_k^t; x_k), y_k))$;
    $t \leftarrow t + 1$;
    if $\tau$ divides $t$ then
        $\bar{\mathbf{w}}^t \leftarrow$Ensemble($\{\mathbf{w}_1^t, \ldots, \mathbf{w}_K^t\}$);
        $\tilde{\mathbf{w}}_k^t \leftarrow$ Compression($\mathbf{w}_k^t, \bar{\mathbf{w}}^t, D_k$);
        $\mathbf{w}_k^t \leftarrow \tilde{\mathbf{w}}_k^t$.
return $\mathbf{w}_k^t$
```

that need to be addressed more concretely: (1) how to train local models that can benefit more to the ensemble model; (2) how to compress the global model without costing too much extra time.

5.1. Diversity driven local training

In order to improve the performance of ensemble, it is necessary to generate diverse local models other than merely accurate ones [8,9]. Therefore, in the local training phase, i.e., the third line in Algorithm 1, we minimize both the loss on training data and the similarity between the local models, which we call *diversity regularized local training loss*. For the kth local worker, it is defined as follows,

$$\mathcal{L}_{\mathrm{LS}}^k(f(\mathbf{w}_k; x_{k,i}), y_{k,i}) = \sum_{i=1}^{m_k}\Big(\mathcal{L}\big(f\big(\mathbf{w}_k; x_{k,i}\big), y_{k,i}\big) + \alpha\mathcal{L}_{\mathrm{sim}}\big(f\big(\mathbf{w}_k; x_{k,i}\big), \bar{z}_{k,i}\big)\Big), \quad (9)$$

where $\bar{z}_{y,i}$ is the average of the outputs of the latest compressed models for input $x_{k,\,i}$. In our experiments, the local training loss $\mathcal{L}$ takes the form of cross entropy, and the similarity loss $\mathcal{L}_{\mathrm{sim}}$ takes the form of $-l_2$ distance. The smaller $\mathcal{L}_{\mathrm{sim}}$ is, the farther the outputs of a local model is from the average of outputs of the latest compressed models, and hence the farther (or the more diverse) the local models are from (or with) each other.

5.2. Accelerated compression

In order to compress the global model to the one with the same size as the local model, we use distillation base compression method[3] [10–12], which obtains a compressed model by letting it mimic the predictions of the global model. In order to save the time for compression, in compression process, we minimize the weighted combination of the local training loss and the pure compression loss, which we call *accelerated compression loss*. For the kth local worker, it is defined as follows:

$$\mathcal{L}_{\mathrm{LC}}^k(f(\mathbf{w}_k; x_{k,i}), y_{k,i}) = \sum_{i=1}^{m_k}\Big(\mathcal{L}\big(f\big(\mathbf{w}_k; x_{k,i}\big), y_{k,i}\big) + \beta\mathcal{L}_{\mathrm{comp}}\big(f\big(\mathbf{w}_k; x_{k,i}\big), \bar{y}_{k,i}\big)\Big), \quad (10)$$

where $\bar{y}_{k,i}$ is the output of the latest ensemble model for the input $x_{k,\,i}$. In our experiments, the local training loss $\mathcal{L}$ and the pure compression loss $\mathcal{L}_{\mathrm{comp}}$ both take the form of cross entropy loss. By reducing the loss between $f(\mathbf{w}_k;\, x_{k,\,i})$ and the pseudo labels $\{\bar{y}_{k,i}; i \in [m_k]\}$, the compressed model can play the similar function as the ensemble model. We denote the distillation based compression process as Compression$_{\mathrm{distill}}$ ($\mathbf{w}_k^t$, $\bar{\mathbf{w}}^t$, D_k), and show its details in Algorithm 2 .[4] Note that before minimizing the accelerated compression loss, we first produce the pseudo labels by the ensemble model and construct a new training dataset $\hat{D}_k$ by the pseudo labels. And, the parameters of the compressed model $\tilde{\mathbf{w}}_k^t$ are initialized by the parameters of the latest local model $\mathbf{w}_k^t$ instead of random numbers.

Algorithm 2: Compression$_{\mathrm{distill}}$($\mathbf{w}_k^t, \bar{\mathbf{w}}_k^t, D_k$).

```
for $j \in [m_k]$ do
    for $c \in [C]$ do
        $\bar{y}_{k,j,c} \leftarrow \sum_{r=1}^K \alpha_k f(\mathbf{w}_r^t; x_{k,j}, c)$;
    $\bar{y}_{k,j} = (\bar{y}_{k,j,1}, \ldots, \bar{y}_{k,j,C})$;
$\hat{D}_k \leftarrow \{(x_{k,1}, y_{k,1}, \bar{y}_{k,1}), \ldots, (x_{k,m_k}, y_{k,m_k}, \bar{y}_{k,m_k})\}$;
Set $\tilde{\mathbf{w}}_k^t = \mathbf{w}_k^t$ and $i = 0$;
while $i \le p$ do
    $\tilde{\mathbf{w}}_k^{t+i+1} \leftarrow \tilde{\mathbf{w}}_k^{t+i} - \eta\Delta(\mathcal{L}_{\mathrm{LC}}^k(f(\tilde{\mathbf{w}}_k^{t+i}; x_k), y_k))$;
    $i \leftarrow i + 1$;
return $\tilde{\mathbf{w}}_k^{t+p}$.
```

5.3. Time complexity

We compare the time complexity of MA-DNN and EC-DNN from two aspects:

1. *Communication time:* EC-DNN prefers larger τ compared to MA-DNN. Essentially, less frequent communication across the local workers can give rise to more diverse local models, which will lead to better ensemble performance for EC-DNN. On the other hand, much diverse local models may indicate greater probability that

[3] Other algorithms for the compression [20–25] can also be used for the same purpose, but different techniques may be required in order to plug these compression algorithms into the EC-DNN framework.

[4] In the algorithm, we take G_{GE--R} as an example and it can be directly generalized to G_{SE} and G_{GE--M}.

local models are in the neighboring of different local optima such that the global model in MA-DNN is more likely to perform worse than local ones. Therefore, EC-DNN yields less communication time than MA-DNN.

2. *Computational time:* EC-DNN does not consume extra computation time for compression since the compression process has been integrated into the local training phase (see discussions in Section 5.2). Therefore, compared with MA-DNN, EC-DNN only requires additional time to relabel the local data using the global model, which approximately equals to the time of the feed-forward propagation over the local dataset. To limit the relabeling time, we choose to relabel a portion of the local data, denoted as μ. Our experimental results in Section 6.3 will demonstrate that the relabeling time can be controlled within a very small amount compared to the training time of DNN. Therefore, EC-DNN can cost only a slightly more or roughly equal computational time over MA-DNN.

Overall, EC-DNN is essentially more time-efficient than MA-DNN as it can reduce the communication cost without significantly increasing computational time.

6. Experiments

6.1. Experimental setup

Platform: Our experiments are conducted on a GPU cluster interconnected with an InfiniBand network, each machine of which is equipped with two Nvdia's K20 GPU processors. One GPU processor corresponds to one local worker.

Data: We conducted experiments on public datasets CIFAR-10, CIFAR-100 [26] and ImageNet (ILSVRC 2015 Classification Challenge) [27]. CIFAR-10 and CIFAR-100 contain 50,000 training images and 10,000 test images for 10-class and 100-class classification respectively. ImageNet contains 1,281,167 training images and 50,000 test images for 1000-class classification. For all the datasets, each image is normalized by subtracting the per-pixel mean computed over the whole training set. The training images are horizontally flipped but not cropped, and the test data are neither flipped nor cropped.

Model: On CIFAR-10 and CIFAR-100, we employ NiN [28], a 9-layer convolutional network. On ImageNet, we use GoogLeNet [1], a 22-layer convolutional network. We used the same tricks, including random initialization, l_2-regularization, Dropout, and momentum, as the original paper. All the experiments are implemented using Caffe [29].

Parallel setting: On experiments on CIFAR-10 and CIFAR-100, we explore the number of the local workers $K \in \{4, 8\}$ and the communication frequency $\tau \in \{1, 16, 2000, 4000\}$ for both MA-DNN and EC-DNN. On experiments on ImageNet, we explore $K \in \{4, 8\}$ and $\tau \in \{1, 1000, 10, 000\}$. The communication across the local workers is implemented using MPI.

Hyperparameter setting of EC-DNN: There are four hyperparameters in EC-DNN, including (1) the coefficient of the regularization in terms of similarity between local models, i.e., α in Eq. (9); (2) the coefficient of the model compression loss, i.e., β in Eq. (10); (3) the length of the compression process, i.e., p in Algorithm 2, measured by the percentage of the number of the mini-batches that the whole training lasts; and (4) the portion of the data needed to be relabeled in the compression process μ as mentioned in Section 5.3. We tune these hyperparameters by exploring a certain range of values and then choose the one resulting in best performance. The explored values and the chosen value on each datasets are shown in Table 1.

6.2. Compared methods

We conduct performance comparisons on four methods:

Table 1
Hyperparameter setting of EC-DNN.

Hyperparameter	Explored values	Chosen value	
		CIFAR 10, CIFAR 100	ImageNet
α	{0.2, 0.4, 0.6, 0.8, 1}	0.6	0.6
β	{0.2, 0.4, 0.6, 0.8, 1}	0.4	1
p	{5%, 10%, 15%, 20%}	10%	10%
μ	{30%, 50%, 70%}	70%	30%

- S-DNN denotes the sequential training on one GPU until convergence [1,28].
- E-DNN denotes the method that trains local models independently and makes ensemble of the local models merely at the end of the training [1,19].
- MA-DNN refers the parallel DNN training framework with the aggregation by averaging model parameters [5–7,13–15].
- EC-DNN refers the parallel DNN training framework with the aggregation by ensemble. EC-DNN applies Compression$_{\text{distill}}$ for the compression for all the experiments in this paper.

Furthermore, for different versions of EC-DNN method, we use EC-DNN-S to denote the one with simple ensemble G_{SE}, and use EC-DNN-R and EC-DNN-M to denote the method with generalized ensemble G_{GE--R} and G_{GE--M}, respectively. And, we collectively denote EC-DNN-S, EC-DNN-R and EC-DNN-M as EC-DNN.

In addition, we use EC-DNN-S$_L$ (or EC-DNN-R$_L$ and EC-DNN-M$_L$), MA-DNN$_L$ and E-DNN$_L$ to denote the corresponding methods that take the local model with the smallest training loss as the final model, and use EC-DNN-S$_G$ (or EC-DNN-R$_G$ and EC-DNN-M$_G$), MA-DNN$_G$ and E-DNN$_G$ to represent the respective methods that take the global model (i.e., the ensemble of local models for EC-DNN-S (or EC-DNN-R and EC-DNN-M) and E-DNN, and the average parameters of local models for MA-DNN) as the final model.

6.3. Experimental results

6.3.1. Model aggregation

We first compare the performance of aggregation methods, i.e. MA and Ensemble. To this end, we employ Diff$_{\text{LG}}$ as the evaluation metric, which measures the improvement of the test error of the global model compared to that of the local models, i.e.,

$$\text{Diff}_{\text{LG}} = \frac{1}{K}\sum_{k=1}^{K}\text{error}_k - \text{error}_{\text{global}}, \tag{11}$$

where error$_k$ denotes the test error of the local model on local worker k, and error$_{\text{global}}$ denotes the test error of the corresponding global model produced by MA (or ensemble) in MA-DNN (or EC-DNN). The positive (or negative) Diff$_{\text{LG}}$ means performance improvement (or drop) of global models over local models. On each dataset, we produce a distribution for Diff$_{\text{LG}}$ over all the communications and all the parallel settings (including numbers of the local workers and communication frequencies). We show the distribution for Diff$_{\text{LG}}$ of MA and ensemble on CIFAR datasets in Figs. 1 and 2, respectively, in which red bars (or blue bars) stand for that the performance of the global model is worse (or better) than the average performance of local models. In these figures, we take the simple ensemble G_{SE} as an example because it is the least powerful one over all the discussed ensemble methods. If it can outperform MA, all the other ensemble methods can also outperform MA.

We have following observations from Figs. 1 and 2: (1) for MA, from Fig. 1, we can observe that, on both datasets, over 10% global models achieve worse performance than the average performance of local models, and the average performance of locals model can be worse than the global model by a large margin, e.g., 30%; (2) on the other hand, for ensemble, we can observe from Fig. 2 that

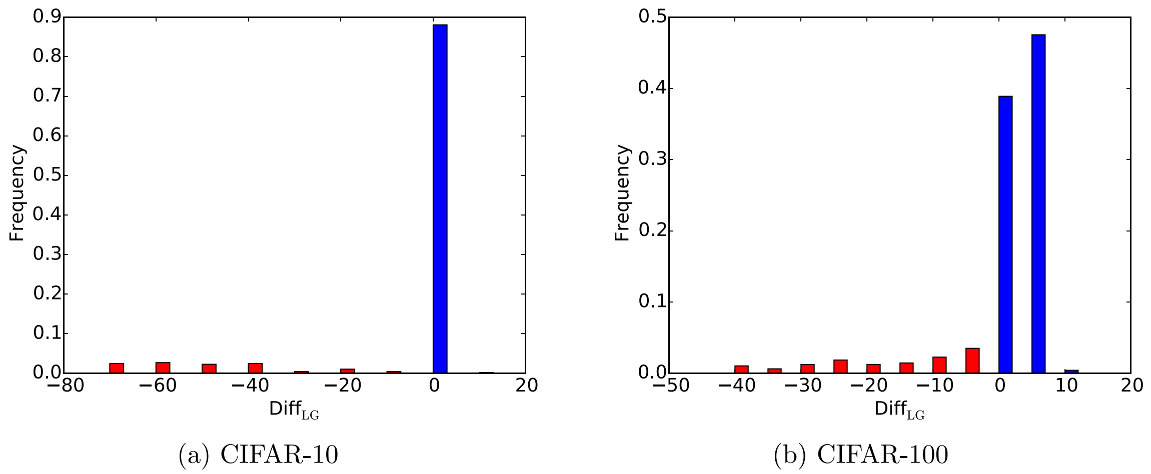


Fig. 1. MA. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

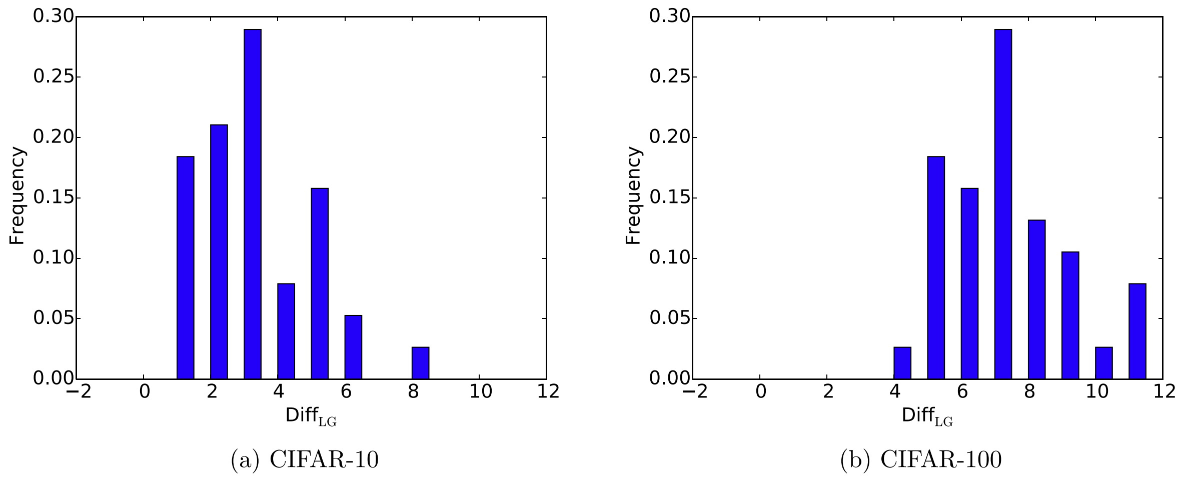


Fig. 2. Ensemble. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

the performance of the global model is consistently better than the average performance of the local models on both datasets. Specifically, the performances of over 20% global models are 5+% better than the average performance of local models on both datasets.

6.3.2. *EC-DNN vs. MA-DNN*

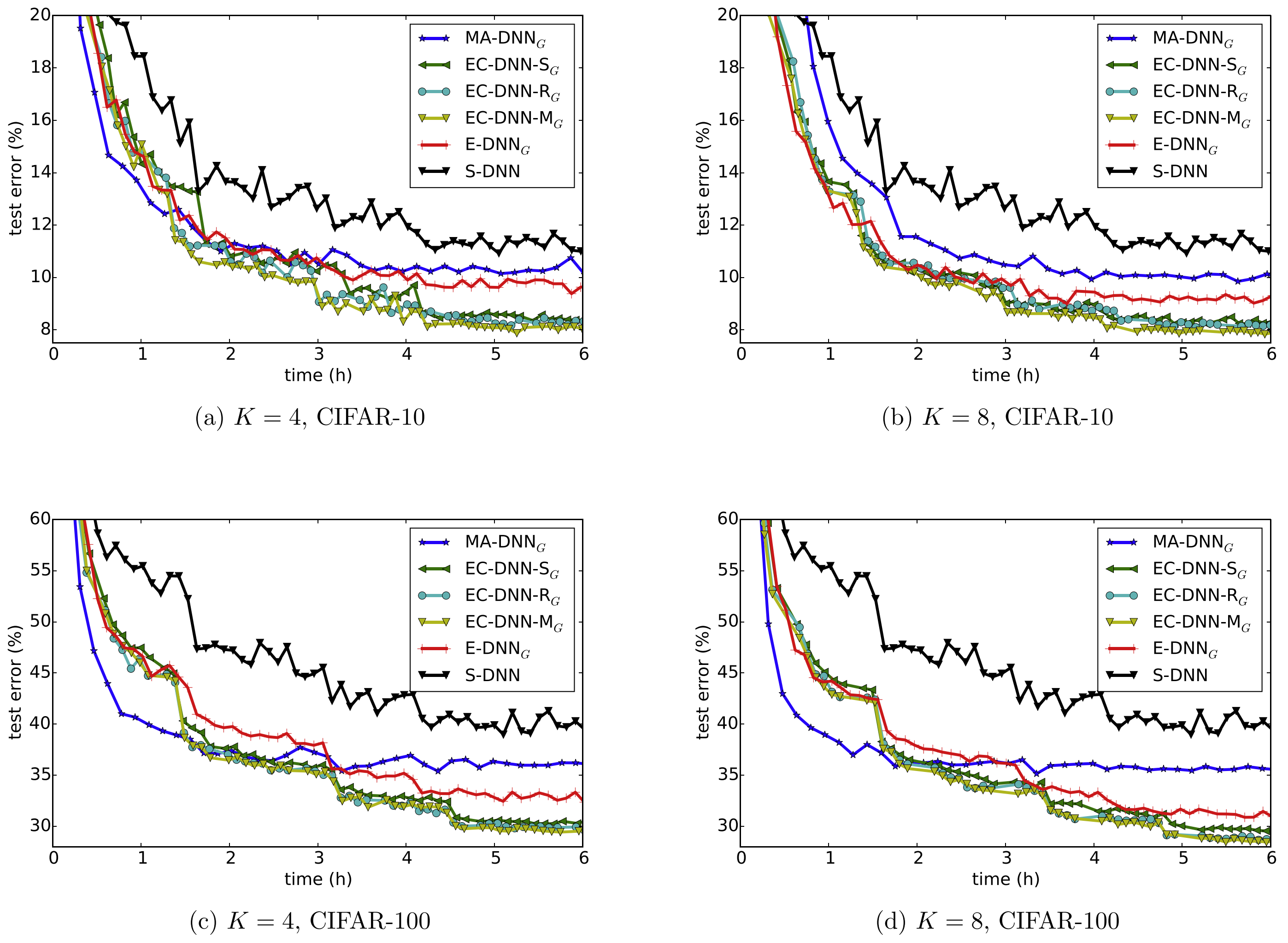
We compare the performance of EC-DNN and MA-DNN from three aspects, i.e., speed, accuracy and communication frequency.

For any parallel algorithm, speed is the most important measure. To this end, we measure the overall training time by MA-DNN and EC-DNN to achieve the same accuracy. Fig. 3 shows the test error curves w.r.t. overall training time. For all the EC-DNN methods, the relabeling time has been counted in the overall time when plotting the figures. We report EC-DNN and MA-DNN that achieve best test performance among all the communication frequencies. We calculate the speed of compared methods from the figures and show results in Tables 2 and 3. In these tables, we denote the speed of MA-DNN$_G$ as 1, and normalize the speed of other methods by dividing that of MA-DNN$_G$. If one method never achieves the same performance with MA-DNN$_G$, we denote its speed as 0. Therefore, larger value of speed indicates better speedup. From these tables, we can observe that all the EC-DNN methods can achieve better speedup than MA-DNN on all the datasets. On average, EC-DNN-S$_G$ and EC-DNN-S$_L$ runs about 2.24 and 1.33 times faster than MA-DNN$_G$, respectively (which is averaged over all the datasets and the number of the local workers). And, EC-DNN-R and EC-DNN-M can achieve more speedup.

In addition, the speedup of EC-DNN over MA-DNN is achieved without loss of accuracy, and there is even accuracy improvement sometimes. In Tables 2 and 3, each EC-DNN$_G$ outperforms MA-DNN$_L$ and MA-DNN$_G$. The average improvements of EC-DNN$_G$ (which is computed by averaging the performance of EC-DNN-S$_G$, EC-DNN-R$_G$ and EC-DNN-M$_G$) over MA-DNN$_L$ and MA-DNN$_G$ are around 1% and 5% for CIFAR-10 and CIFAR-100 respectively. Be-

Table 2
Test error (%), speed, and communication frequency τ on CIFAR-10.

	K=4			K=8		
	Error	Speed	τ	Error	Speed	τ
MA-DNN$_G$	10.3	1	16	9.99	1	2k
E-DNN$_G$	9.44	1.58	–	9.05	1.92	–
EC-DNN-S$_G$	8.43	1.92	4k	8.19	2.05	4k
EC-DNN-R$_G$	8.32	2.05	2k	8.16	2.11	2k
EC-DNN-M$_G$	8.13	2.05	4k	8.06	2.14	2k
MA-DNN$_L$	10.55	0	16	10.54	0	2k
E-DNN$_L$	11.04	0	–	10.95	0	–
EC-DNN-S$_L$	10.04	1.36	4k	9.88	1.26	4k
EC-DNN-R$_L$	9.92	1.42	2k	9.79	1.32	2k
EC-DNN-M$_L$	9.84	1.45	4k	9.70	1.38	2k
S-DNN			10.41			



(a) $K = 4$, CIFAR-10 (b) $K = 8$, CIFAR-10

(c) $K = 4$, CIFAR-100 (d) $K = 8$, CIFAR-100

Fig. 3. Test error curves on CIFAR datasets.

Table 3
Test error (%), speed, and communication frequency τ on CIFAR-100.

	K=4			K=8		
	Error	Speed	τ	Error	Speed	τ
MA-DNN$_G$	36.18	1	16	35.55	1	16
E-DNN$_G$	32.49	1.95	–	30.9	1.97	–
EC-DNN-S$_G$	30.26	2.52	4k	29.31	2.48	2k
EC-DNN-R$_G$	30.19	2.61	2k	29.10	**2.61**	4k
EC-DNN-M$_G$	**29.85**	**2.63**	4k	**28.93**	**2.61**	4k
MA-DNN$_L$	36.39	0	16	35.56	0	16
E-DNN$_L$	39.57	0	–	39.55	0	–
EC-DNN-S$_L$	34.80	1.42	4k	35.1	1.27	2k
EC-DNN-R$_L$	34.33	1.48	2k	34.93	1.36	4k
EC-DNN-M$_L$	**34.05**	**1.50**	4k	**34.75**	**1.38**	4k
S-DNN	35.68					

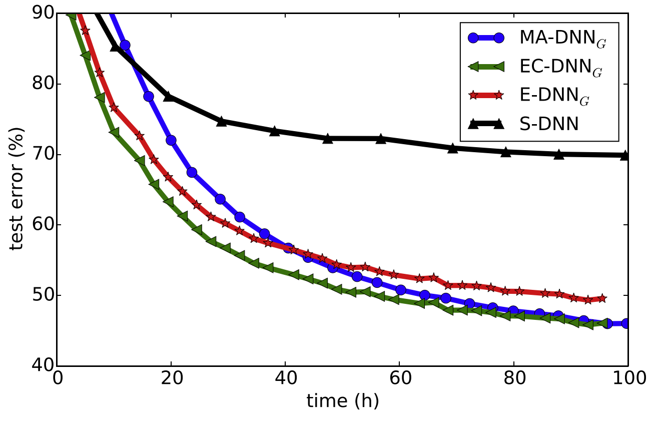
sides, we also report the final performance of EC-DNN$_L$ considering that it can save test time and still outperform both MA-DNN$_L$ and MA-DNN$_G$ when we do not have enough computational and storage resource. We can observe that all the EC-DNN$_L$ (including EC-DNN-S$_L$, EC-DNN-R$_L$ and EC-DNN-M$_L$) methods achieve comparable or better performance than both MA-DNN$_L$ and MA-DNN$_G$. Specifically, for example, EC-DNN-S$_L$ achieved test errors of 10.04% and 9.88% for $K = 4$ and $K = 8$ respectively on CIFAR-10, while it achieved test errors of 34.8% and 35.1% for $K = 4$ and $K = 8$ respectively on CIFAR-100. In addition, for all the datasets, all the EC-DNN methods achieve comparable or better accuracy than S-DNN.

We also compare the communication frequency τ that makes MA-DNN and EC-DNN achieve the best speed respectively. Tables 2 and 3 show the results. We can observe that all the EC-DNN methods tend to communicate less frequently than MA-DNN. Specifically, MA-DNN usually achieves the best performance with a small τ (i.e., 16), while EC-DNN cannot reach its best performance before τ is not as large as 2000. These observations are consistent with our analysis in Section 5.3, that EC-DNN requires less frequent communication than MA-DNN.

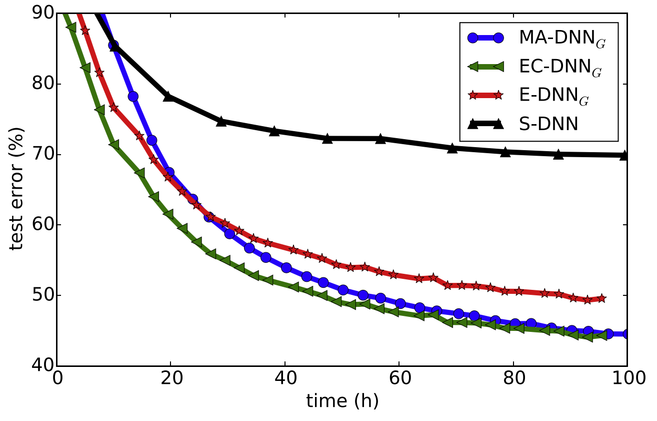
6.3.3. *EC-DNN vs. E-DNN*

We compare the performance of EC-DNN and E-DNN from two aspects, i.e., speed and accuracy.

For the speed, we show the test error curves w.r.t. overall training time of EC-DNN and E-DNN in Fig. 3. The speed of E-DNN is calculated by the same way as EC-DNN (see details in Section 6.3.2), and shown in Tables 2 and 3. We can observe that all the EC-DNN methods consistently results in better speedup than E-DNN on all the datasets. Specifically, on average, E-DNN$_G$ only runs about 1.85 times faster than MA-DNN$_G$ while EC-DNN-S$_G$ (the slowest one over all the EC-DNN methods) can reach about 2.24 times faster speed. In addition, we can also find that E-DNN$_L$ never achieves the same performance with MA-DNN$_G$ while all the EC-DNN$_L$ methods can contrarily run much faster than MA-DNN$_G$.



(a) $K = 4$



(b) $K = 8$

Fig. 4. Test error curves on ImageNet.

For the accuracy, we can observer from Tables 2 and 3 that EC-DNN-S $_G$ (the worst one over all the EC-DNN methods) outperforms E-DNN$_G$ consistently for different datasets and number of the local workers. Specifically, on average, EC-DNN-S$_G$ can achieve 1+% and 1.8+% better performance than E-DNN$_G$ on CIFAR-10 and CIFAR-100, respectively. In addition, E-DNN$_L$ never outperforms MA-DNN$_L$ and MA-DNN$_G$, while all the EC-DNN$_L$ methods can always achieve comparable performance with or better performance than both MA-DNN$_L$ and MA-DNN$_G$.

6.3.4. EC-DNN with different ensemble methods

Fig. 3, Tables 2 and 3 also show the performances for different versions of EC-DNN methods, which use different ensemble methods. First, we can observe that all the EC-DNN methods consistently outperforms MA-DNN and E-DNN in terms of speedup and accuracy, indicating that technologies in EC-DNN are very powerful (see detailed discuss in Sections 6.3.2 and 6.3.3). In addition, with delicate design of ensemble methods in EC-DNN-R and EC-DNN-M, the performance of simple version of EC-DNN, i.e., EC-DNN-S, can be further improved. For example, for CIFAR-100 and $K = 8$, by generalized ensemble G_{GE--M}, the speedup of EC-DNN-S can be improved from 2.48 to 2.61 without loss of accuracy, and the test error can even be reduced from 29.31% to 28.93%. Furthermore, by the fine-grained way that assigns coefficient to each data point in EC-DNN-M, EC-DNN-M usually achieves better performance than EC-DNN-R.

6.3.5. Large-scale experiments

In the following, we will conduct experiments to compare the performance of MA-DNN with that of EC-DNN with the setting of much bigger model and more data, i.e., GoogleNet on ImageNet. Fig. 4 shows the test error of the global model w.r.t the overall time. In the figure, we take EC-DNN-S as an example since it is the least powerful one over all the versions of EC-DNN. The communication frequencies τ that makes MA-DNN and EC-DNN achieve best performance are 1 and 1000, respectively. We can observe that EC-DNN consistently achieves better speed than S-DNN, MA-DNN and E-DNN throughout the training. Besides, we can observe that EC-DNN outperforms MA-DNN even at the early stage of the training, while EC-DNN cannot achieve this on CIFAR datasets because it communicates less frequently than MA-DNN. The reason is that frequent communication will make the training much slower for very big model, i.e., use less mini-batches of data within the same time. When the improvements introduced by MA cannot compensate the decrease of the number of used data, MA-DNN no longer outperforms EC-DNN at the early stage of the training. In this case, the advantage of EC-DNN becomes even more outstanding.

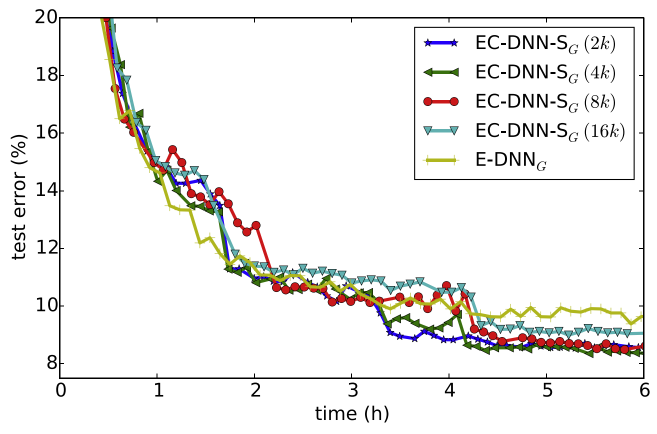
6.3.6. Discussions

The influence of the communication frequency: To study the influence of the communication frequency τ on our proposed EC-DNN, we explore the communication frequency $\tau \in \{2k, 4k, 8k, 16k\}$, and show the test error curves in Fig. 5. We take $K = 4$ and EC-DNN-S as an example and the results for $K = 8$ and other versions of EC-DNN are similar. From the figure, we can observe that, on both datasets, EC-DNN can achieve good performance in a wide range of communication frequency, e.g., from 2k to 8k on CIFAR-10 and from 2k to 4k on CIFAR-100. This indicates that EC-DNN is robust to the change of communication frequency. In addition, when τ is too large, e.g., 16k, the accuracy and the speed drop due to the insufficient communication between different local workers. In the extreme case that there is only one communication at the end of the training, i.e., E-DNN, the accuracy and the speed drop more severely.

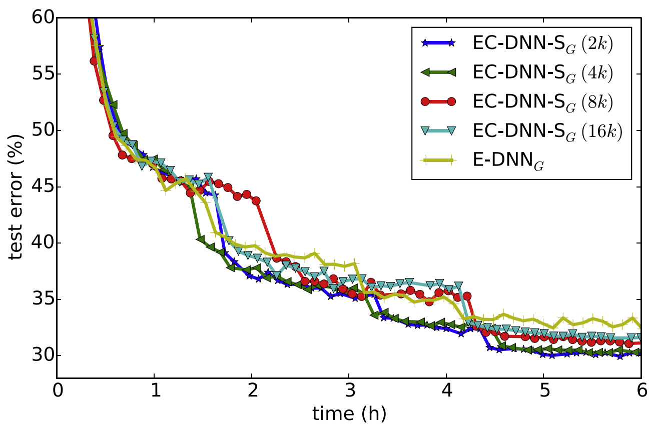
Local training as a indispensable part: Since we minimize both the local training loss and the pure compression loss in the accelerated compression step (see details in Section 5.2), a natural question is whether it is still necessary to have a separate local training step that merely minimizes the local training loss (see details in Section 5.1). To answer this question, we compare the following settings of EC-DNN: (1) EC-DNN without local training step; (2) EC-DNN with local training step but without the diversity based term; (3) EC-DNN with local training step and the diversity based term, which is the one we used in all the above experiments. Fig. 6 shows the experimental results. We take $K = 4$ and EC-DNN-S as an example and the results for $K = 8$ and other versions of EC-DNN are similar. From the figure, we can observe that even if there is no diversity based term in case 2, case 2 significantly outperforms case 1. This indicates that the local training is an indispensable part. We hypothesis the reason as that during the separate local training step, the different local models do not need to learn to mimic the common ensemble model, and thus becomes more diverse. Such diversity benefit the ensemble and thus the performance of the whole EC-DNN. In addition, we can observe that by introducing diversity in case 3, case 3 further improve the performance of case 2.

7. Related works

With the growing efforts in parallel training for DNN, many previous studies have paid attention to MA-DNN. Some studies aim

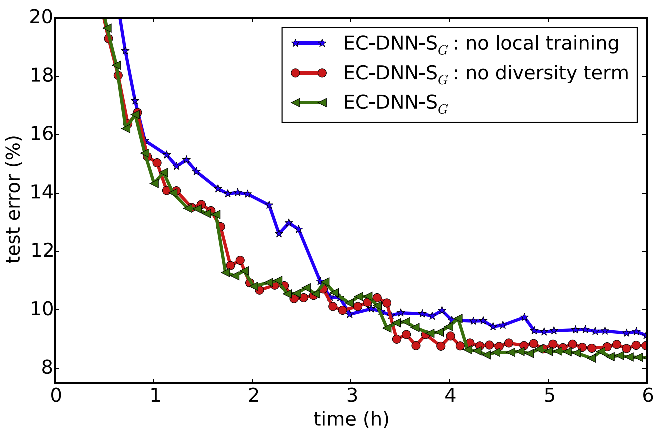


(a) CIFAR-10

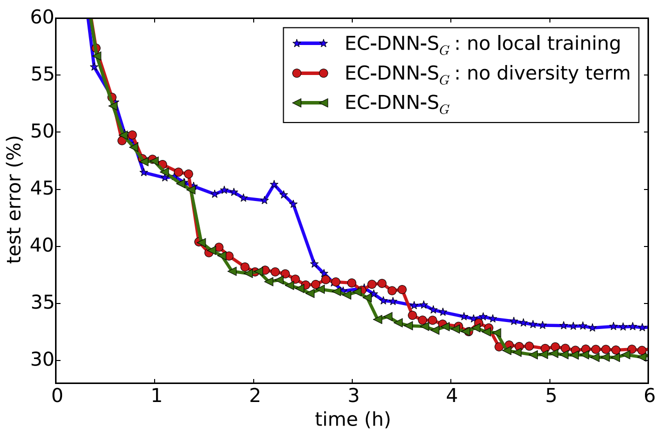


(b) CIFAR-100

Fig. 5. The influence of the communication frequency.



(a) CIFAR-10



(b) CIFAR-100 (place holder)

Fig. 6. Local training as an indispensable part.

at improving the speedup or convergence of MA-DNN. For example, NG-SGD [7] proposes an approximate and efficient implementation of Natural Gradient for SGD (NG-SGD) to improve the performance of MA-DNN; EASGD [13] improves MA-DNN by adding an elastic force which links the parameters of the local models with the parameters of the global model; BMUF [15] leverages data parallelism and blockwise model-update filtering to improve the speedup of MA-DNN; large mini-batch methods [30,31] increase the learning rate and the mini-batch size to accelerate the convergence; DC-ASGD improves the accuracy of the asynchronous data parallelism by compensating the delayed gradients using the second order term of the Taylor expansion of the gradients. Some studies focus on reducing the communication cost. For example, for NLP tasks, sampling method [32] reduces communication cost by only transferring the gradients of the parameters that corresponds to the most frequent words in the vocabulary in the RNN model; Quantization method [33–35] reduces communication cost by quantizing each gradient to a small number of bits (less than 32 bits) during the communication. All these methods aim at solving different problems with us, and our method can be used with those methods simultaneously.

8. Conclusion and future work

In this work, we propose a new parallelization training framework for DNN, called EC-DNN. As compared to the traditional approach, MA-DNN, which averages the parameters of different local models, our proposed method uses the ensemble method (including both the simple ensemble and the generalized ensemble) to aggregate local models. In this way, we can guarantee that the error of the global model in EC-DNN is upper bounded by the average error of the local models and can consistently achieve better performance than MA-DNN. In the future, we plan to consider other compression methods for EC-DNN. Besides, we plan to investigate how to generalize EC-DNN to more types of deep neural networks, such as recurrent neural networks.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Computer Vision and Pattern Recognition, 2015, pp. 1–9.
[2] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on ImageNet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.
[3] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[4] M. Li, D.G. Andersen, J.W. Park, A.J. Smola, A. Ahmed, V. Josifovski, J. Long, E.J. Shekita, B.-Y. Su, Scaling distributed machine learning with the parameter server, in: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, 2014, pp. 583–598.
[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q.V. Le, et al., Large scale distributed deep networks, in: Proceedings of Advances in Neural Information Processing Systems, 2012, pp. 1223–1231.

[6] X. Zhang, J. Trmal, D. Povey, S. Khudanpur, Improving deep neural network acoustic models using generalized maxout networks, in: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2014, pp. 215–219.
[7] D. Povey, X. Zhang, S. Khudanpur, Parallel training of DNNs with natural gradient and parameter averagin, in: International Conference on Learning Representations, 2015.
[8] L.I. Kuncheva, C.J. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Mach. Learn. 51 (2) (2003) 181–207.
[9] P. Sollich, A. Krogh, Learning with ensembles: how overfitting can be useful, in: Proceedings of Advances in Neural Information Processing Systems, 8, 1996, pp. 190–196.
[10] C. Bucilua, R. Caruana, A. Niculescu-Mizil, Model compression, in: Proceedings of the 12th ACM Conference on Knowledge Discovery and Data Mining, ACM, 2006, pp. 535–541.
[11] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: Hints for thin deep nets, in: International Conference on Learning Representation, 2015.
[12] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, in: NIPS Deep Learning and Representation Learning Workshop, 2015.
[13] S. Zhang, A.E. Choromanska, Y. LeCun, Deep learning with elastic averaging SGD, in: Proceedings of Advances in Neural Information Processing Systems 28, 2015, pp. 685–693.
[14] J. Chen, R. Monga, S. Bengio, R. Jozefowicz, Revisiting distributed synchronous SGD, in: International Conference on Learning Representations, 2016.
[15] K. Chen, Q. Huo, Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering, in: Proceedings of 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 5880–5884.
[16] D.H. Wolpert, Stacked generalization, Neural Netw. 5 (2) (1992) 241–259.
[17] L. Breiman, Stacked regressions, Mach. Learn. 24 (1) (1996) 49–64.
[18] X. Shen, L. Wang, et al., Generalization error for multi-class margin classification, Electron. J. Stat. 1 (2007) 307–330.
[19] D. Ciresan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2012, pp. 3642–3649.
[20] W. Chen, J.T. Wilson, S. Tyree, K.Q. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: Proceedings of the 32st International Conference on Machine Learning, 2015.
[21] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, in: International Conference on Learning Representations, 2014.
[22] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al., Predicting parameters in deep learning, in: Proceedings of Advances in Neural Information Processing Systems, 2013, pp. 2148–2156.
[23] E.L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Proceedings of Advances in Neural Information Processing Systems, 2014, pp. 1269–1277.
[24] R. Rigamonti, A. Sironi, V. Lepetit, P. Fua, Learning separable filters, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2013, pp. 2754–2761.
[25] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Proceedings of Advances in Neural Information Processing Systems 28, 2015, pp. 1135–1143.
[26] A. Krizhevsky, Learning Multiple Layers of Features From Tiny Images, Technical Report, University of Toronto, 2009.
[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.
[28] L. Min, C. Qiang, S. Yan, Network in network, in: International Conference on Learning Representations, 2014.
[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM international conference on Multimedia, ACM, 2014, pp. 675–678.
[30] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, large minibatch SGD: training ImageNet in 1 hour, arXiv:1706.02677 (2017).
[31] Y. You, I. Gitman, B. Ginsburg, Scaling SGD batch size to 32k for ImageNet training, arXiv:1708.03888 (2017).
[32] T. Xiao, J. Zhu, T. Liu, C. Zhang, Fast parallel training of neural language models, in: Proceedings of International Joint Conference on Artificial Intelligence, 2017.
[33] F. Seide, H. Fu, J. Droppo, G. Li, D. Yu, 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs, in: Proceedings of Interspeech, 2014, pp. 1058–1062.
[34] D. Alistarh, D. Grubic, J. Li, R. Tomioka, M. Vojnovic, QSGD: Communication–efficient SGD via gradient quantization and encoding, in: Advances in Neural Information Processing Systems 30, 2017, pp. 1707–1718.
[35] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, H. Li, TernGrad: ternary gradients to reduce communication in distributed deep learning, in: Proceedings of Advances in Neural Information Processing Systems 28, 2017.

Shizhao Sun is a Ph.D. student in Nankai University, majored in computer science. Before that, she received her bachelor degree in computer science from Nankai University in 2013. Her research interests center around parallel/distributed machine learning and statistical learning theory.

Xiaoguang Liu received the B.Sc. degree, M.Sc. degree and Ph.D. degree in computer science from Nankai University, Tianjin, China, in 1996, 1999 and 2002, respectively. He is currently a professor in computer science at Nankai University, Tianjin, China. His research interests include parallel computing, cloud storage and search engine system.