Ensemble-Compression: A New Method for Parallel Training of Deep Neural Networks

Shizhao Sun[1*], Wei Chen[2], Jiang Bian[2], Xiaoguang Liu[1], and Tie-Yan Liu[2]

[1] College of Computer and Control Engineering, Nankai University, Tianjin, P.R.China
[2] Microsoft Research, Beijing, P.R.China
sunshizhao@mail.nankai.edu.cn, wche@microsoft.com, jiabia@microsoft.com, liuxg@nbjl.nankai.edu.cn, tyliu@microsoft.com.

Abstract. Parallelization framework has become a necessity to speed up the training of deep neural networks (DNN) recently. Such framework typically employs the *Model Average* approach, denoted as MA-DNN, in which parallel workers conduct respective training based on their own local data while the parameters of local models are periodically communicated and averaged to obtain a global model which serves as the new start of local models. 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 parallel training framework called *Ensemble-Compression*, denoted as EC-DNN. In this framework, we propose to aggregate the local models by 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 ensemble-based global model is guaranteed to be at least as good as the average performance of local models. However, a big challenge lies in the explosion of model size since each round of ensemble can give rise to multiple times size increment. Thus, we carry out model compression after each ensemble, specialized by a distillation based method in this paper, to reduce the size of the global model to be the same as the local ones. Our experimental results demonstrate the prominent advantage of EC-DNN over MA-DNN in terms of both accuracy and speedup.

Keywords: Parallel machine learning, Distributed machine learning, Deep learning, Ensemble method.

1 Introduction

Recent rapid development of deep neural networks (DNN) has demonstrated that its great success mainly comes from big data and big models [25, 13]. However, it is extremely time-consuming to train a large-scale DNN model over big data. To

* This work was done when the author was visiting Microsoft Research Asia.

accelerate the training of DNN, parallelization frameworks like MapReduce [7] and Parameter Server[18, 6] 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 identical parameter of each local models [27, 20]. 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-DNN might even perform worse than any local model, especially when the local models fall into different local-convex domains. As the global model will be used as a new starting point of the successive iterations of 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. Specifically, the ensemble method aggregates local models by averaging their outputs rather than their parameters. Equivalently, the global model produced by ensemble is a larger network with one additional layer which takes the outputs of local models as inputs with weights as $1/K$, where K is the number of local models. Since most of widely-used loss functions for DNN (e.g., cross entropy loss, square loss, and hinge loss) are convex with respect to the output vector of the model, the loss of the global model produced by ensemble can be upper bounded by the average loss of the local models. Empirical evidence in [25, 5] even show that the ensemble model of DNN, i.e., the global model, is usually better than any base model, i.e., the local model. According to previous theoretical and empirical studies [17, 24], 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 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 model with one additional layer will be K times wider than any of the local models. 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 [1, 22, 14], 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, w.r.t. different numbers of local workers and communication frequencies. The experimental results illustrate a couple of important observations: 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 test accuracy than MA-DNN in all the settings. 3) 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

In the following of this paper, we denote a DNN model as $f(\mathbf{w})$ where $\mathbf{w}$ represents the parameters of this DNN model. 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 c-th class. DNN is a highly non-convex model due to the non-linear activations and poolings after many layer.

In the parallel training of DNN, suppose that there are K workers and each of them 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 weights of the DNN model at the iteration t on the worker k as $\mathbf{w}_k^t$. The communication between the workers is invoked after every τ iterations of updates for the weights, and we call τ the communication frequency. A typical parallel training procedure for DNN implements the following three steps in an iterative manner until the training curve converges.

1. Local training: At iteration t, worker k updates its local model by using SGD. Such local model is updated for τ iterations before the cross-machine synchronization.

2. Model aggregation: The parameters of local models are communicated across machines. 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 weights 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 $t = \tau, 2\tau, \cdots$. A widely-used aggregation method is *model average (MA)*, which averages each parameter over all the local models, i.e.,

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

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

With the growing efforts in parallel training for DNN, many previous studies [27, 20, 6, 26, 2, 3] have paid attention to MA-DNN. NG-SGD [20] proposes an approximate and efficient implementation of Natural Gradient for SGD (NG-SGD) to improve the performance of MA-DNN. EASGD [26] improves MA-DNN by adding an elastic force which links the weights of the local models with the weights of the global model. BMUF [3] leverages data parallelism and blockwise model-update filtering to improve the speedup of MA-DNN. All these methods aim at solving different problems with us, and our method can be used with those methods simultaneously.

3 Model Aggregation: MA vs. Ensemble

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.

MA was originally proposed for convex optimization. If the model is convex w.r.t. the parameters and the loss is convex w.r.t. the model outputs, the performance of the global model produced by MA can be guaranteed to be no worse than the average performance of local models. This is because, when $f(\cdot)$ is a convex model, we have,

$$\mathcal{L}\left(f\left(\bar{\mathbf{w}}^t; x\right), y\right) = \mathcal{L}\left(f\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{w}_k^t; x\right), y\right) \leq \mathcal{L}\left(\frac{1}{K}\sum_{k=1}^{K} f\left(\mathbf{w}_k^t; x\right), y\right). \tag{2}$$

Moreover, when the loss is also convex w.r.t. the model outputs $f(\cdot; x)$, we have,

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

By combining inequalities (2) and (3), we can see that it is quite effective to apply MA in the context of convex optimization, since the loss of the global model by MA is no greater than the average loss of local models in such context.

However, DNN is indeed a highly non-convex model due to the existence of activation functions and pooling functions (for convolutional layers). Therefore,

the above properties of MA for convex optimization does not hold for DNN such that the MA method cannot produce any global model with guaranteed better performance than local ones. Especially, when the local models are in the neighborhoods of different local optima, the global model based on MA could be even worse than any of the local models. 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 5.3 and previous studies [20, 3] also revealed such problem.

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.(3) 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 *ensemble* the local models by averaging their outputs as follows,

$$G_E\left(\mathbf{w}_1^t, \ldots, \mathbf{w}_K^t\right) = \frac{1}{K}\sum_{k=1}^{K} f\left(\mathbf{w}_k^t; x\right), t = \tau, 2\tau, \cdots. \tag{4}$$

4 EC-DNN

In this section, we first introduce the EC-DNN framework, which employs ensemble for model aggregation. Then, we introduce a specific implementation of EC-DNN that adopts distillation for the compression. At last, we take some further discussions for the time complexity of EC-DNN and the comparison with traditional ensemble methods.

4.1 Framework

The details of EC-DNN framework is shown in Alg. 1. Note that, in this paper, we focus on the synchronous case[3] within the MapReduce framework, but EC-DNN can be generalized into the asynchronous case and parameter server framework as well. Similar to other popular parallel training methods for DNN, EC-DNN 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, 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

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

D_k. One local model will be updated for τ iterations before the cross-machine synchronization.

2. Model aggregation: The goal of model aggregation is to communicate the parameters of local models, i.e., $\mathbf{w}_1^t \dots \mathbf{w}_K^t$, across machines. To this end, a global model is produced by ensemble according to $G_E(\mathbf{w}_1^t, \dots, \mathbf{w}_K^t)$ in Eq.(4), i.e., averaging the outputs of the local models. Equivalently, the global model produced by ensemble is a larger network with one additional layer, whose outputs consist of C nodes representing C classes, and whose inputs are those outputs from local models with weights as $1/K$, where K is the number of local models. Therefore, the weights of global model $\bar{\mathbf{w}}^t$ can be denoted as $\bar{\mathbf{w}}^t = \{\mathbf{w}_1^t, \dots, \mathbf{w}_K^t, \frac{1}{K}\}$, $t = \tau, 2\tau, \dots$

Note that such ensemble-produced global model (i.e., $f(\bar{\mathbf{w}}_t)$) 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 Alg. 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 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 $t = \tau, 2\tau, \cdots$.

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

Algorithm 1: EC-DNN(D_k)

```
Randomly initialize w_k^0 and set t = 0;
while stop criteria are not satisfied do
    w_k^{t+1} ← w_k^t − η∆(L(f(w_k^t; x_k), y_k));
    t ← t + 1;
    if τ divides t then
        w̄^t ← {w_1^t, ..., w_K^t, 1/K};
        w̃_k^t ← Compression(w_k^t, w̄^t, D_k);
        w_k^t ← w̃_k^t.
return w_k^t
```

4.2 Implementations

Algorithm 1 contains two sub-problems 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.

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 [17, 24]. Therefore, in the local training phase, i.e., the third line

in Alg. 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 k-th worker, it is defined as follows,

$$\mathcal{L}_{\mathrm{LS}}^{k}(f(\mathbf{w}_k; x_{k,i}), y_{k,i}) = \sum_{i=1}^{m_k} \left(\mathcal{L}\left(f(\mathbf{w}_k; x_{k,i}), y_{k,i}\right) + \alpha \mathcal{L}_{\mathrm{sim}}\left(f\left(\mathbf{w}_k; x_{k,i}\right), \bar{z}_{k,i}\right)\right), \quad (5)$$

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.

Distillation Based 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[4] [1, 22, 14], 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 k-th worker, it is defined as follows,

$$\mathcal{L}_{\mathrm{LC}}^{k}(f(\mathbf{w}_k; x_{k,i}), y_{k,i}) = \sum_{i=1}^{m_k} \left(\mathcal{L}\left(f\left(\mathbf{w}_k; x_{k,i}\right), y_{k,i}\right) + \beta \mathcal{L}_{\mathrm{comp}}\left(f\left(\mathbf{w}_k; x_{k,i}\right), \bar{y}_{k,i}\right)\right), \quad (6)$$

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 $f\left(\mathbf{w}_k; x_{k,i}\right)$ 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 $\mathrm{Compression}_{\mathrm{distill}}$ ($\mathbf{w}_k^t$, $\bar{\mathbf{w}}^t$, D_k), and show its details in Alg. 2. First, on the k-th local worker, we construct a new training data $\hat{D}_k$ by relabeling the original dataset D_k with the pseudo labels produced by the global model, i.e., $\{\bar{y}_{k,i}, i \in [m_k]\}$. Specifically, when producing pseudo labels, we first produce the predictions of each local models respectively, and then average the predictions of all the local models. By this way, we keep using the same amount of GPU memory as MA-DNN throughout the training, because the big global model, which is K times larger than the local model, has never been established in GPU memory. Then, we optimize the accelerated compression loss $\mathcal{L}_{\mathrm{LC}}^{k}$ in Eq.(6) by SGD for p iterations. We initialize the parameters of the compressed model $\tilde{\mathbf{w}}_k^t$ by the parameters of the latest local model $\mathbf{w}_k^t$ instead of random numbers. Finally, the obtained compressed model $\tilde{\mathbf{w}}_k^{t+p}$ is returned, which will be set as the new starting point of next round of the local training.

[4] Other algorithms for the compression [4, 10, 8, 9, 21, 11] 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.

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

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

We can find that minimizing the diversity regularized loss $\mathcal{L}_{\text{LS}}^k$ for local training (Eq.(5)) and minimizing the accelerated compression loss $\mathcal{L}_{\text{LC}}^k$ for compression (Eq.(6)) are two opposite but complementary tasks. They need to leverage information generated by each other into their own optimization. Specifically, the local training phase leverages $\bar{z}_{k,i}$ based on compressed model while the compression process uses $\bar{y}_{k,i}$ provided by local models. Due to such structural duality, we take advantage of a new optimization framework, i.e. dual learning [12], to improve the performance of both tasks simultaneously.

4.3 Time Complexity

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

1. Communication time: Parallel DNN training process is usually sensitive to the communication frequency τ. Different parallelization frameworks yield various optimal τ. In particular, EC-DNN prefers larger τ compared to MA-DNN. Essentially, less frequent communication across 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 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. The poor performance of the global model will significantly slow down the convergence and harm the performance of the final model. Therefore, EC-DNN yields less communication time than MA-DNN.

2. Computational time: According to the analysis in Sec 4.2, EC-DNN does not consume extra computation time for model compression since the compression process has been integrated into the local training phase, as shown in Eq.(6). 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 maximal time of the feed-forward propagation over the local dataset. We call such extra time "*relabeling time*" for ease of reference. To limit the relabeling time on large datasets, we choose to relabel a portion of the local data, denoted as μ. Our experimental results in Section 5.3 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, compared to MA-DNN, EC-DNN is essentially more time-efficient as it can reduce the communication cost without significantly increasing computational time.

4.4 Comparison with Traditional Ensemble Methods

Traditional ensemble methods for DNN [25, 5] usually first train several DNN models independently without communication and make ensemble of them in the end. 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 Experiments

5.1 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 [16] and ImageNet (ILSVRC 2015 Classification Challenge) [23]. 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 [19], a 9-layer convolutional network. On ImageNet, we use GoogLeNet [25], 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 [15].

Parallel Setting. On experiments on CIFAR-10 and CIFAR-100, we explore the number of 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 $\tau \in \{1, 1000, 10000\}$. The communication across local workers is implemented using MPI.

Hyperparameters 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.(5), 2) the coefficient of the model compression loss, i.e., β in Eq.(6), 3) the length of the compression process, i.e., p in Alg. 2, and 4) the portion of the data needed to be relabeled in the compression process μ as mentioned in Sec 4.3. We tune these hyperparameters by exploring a certain range of values and then choose the one resulting in best performance. In particular, we explored α among $\{0.2, 0.4, \ldots, 1\}$, and finally choose $\alpha = 0.6$ on all the datasets. To decide the value of β, we explored two strategies, one of which uses consistent β at each compression process while the other employs increasing β after a certain percentage of compression process. In the first strategy, we explored β among $\{0.2, 0.4, \ldots, 1\}$, and in the second one, we explored β among $\{0.2, 0.4, \ldots, 1\}$, the incremental step of β among $\{0.1, 0.2\}$, and the percentage of compression process from which β begins to increase among $\{10\%, 20\%, 30\%\}$. On CIFAR datasets, we choose to use $\beta = 0.4$ for the first 20% of compression processes and $\beta = 0.6$ for all the other compression processes. And, on ImageNet, we choose to use consistent $\beta = 1$ throughout the compression. Moreover, we explored p's value among $\{5\%, 10\%, \ldots, 20\%\}$ of the number of the mini-batches that the whole training lasts, and finally pick $p = 10\%$ on all the datasets. Furthermore, we explored μ's value among $\{30\%, 50\%, 70\%\}$. And, we finally select $\mu = 70\%$ on CIFAR datasets, and $\mu = 30\%$ on ImageNet.

5.2 Compared Methods

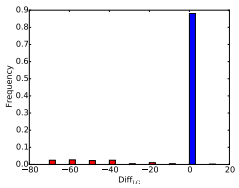
We conduct performance comparisons on four methods:

- S-DNN denotes the sequential training on one GPU until convergence [19, 25].
- E-DNN denotes the method that trains local models independently and makes ensemble of the local models merely at the end of the training [25, 5].
- MA-DNN refers the parallel DNN training framework with the aggregation by averaging model parameters [27, 20, 6, 26, 2, 3].
- EC-DNN refers the parallel DNN training framework with the aggregation by averaging model outputs. EC-DNN applies $\text{Compression}_{\text{distill}}$ for the compression for all the experiments in this paper.

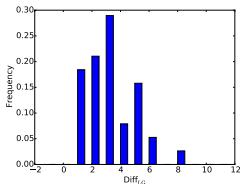
Furthermore, we use EC-DNN$_L$, MA-DNN$_L$ and E-DNN$_L$ to denote the corresponding methods that take the local model with the smallest training loss as the final model, and use EC-DNN$_G$, MA-DNN$_G$ and E-DNN$_G$ to represent the respective methods that take the global model (i.e., the ensemble of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

5.3 Experimental Results

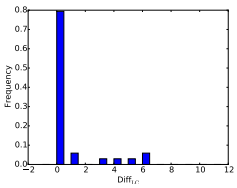
Model Aggregation. We first compare the performance of aggregation methods, i.e. MA and Ensemble. We employ Diff_{LG} as the evaluation metric, which



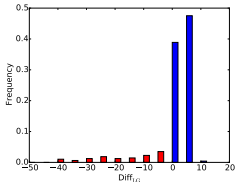
(a) CIFAR-10



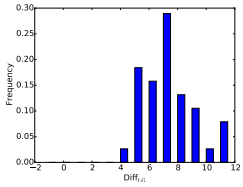
(a) CIFAR-10



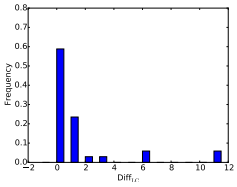
(a) CIFAR-10



(b) CIFAR-100



(b) CIFAR-100



(b) CIFAR-100

Fig. 1: MA.

Fig. 2: Ensemble.

Fig. 3: Compression.

measures the improvement of the test error of the global model compared to that of the local models, i.e.,

$$\text{Diff}_{\text{LG}} = \frac{1}{K}\sum_{k=1}^{K}\text{error}_k - \text{error}_{\text{global}}, \tag{7}$$

where error_k denotes the test error of the local model on worker k, and $\text{error}_{\text{global}}$ 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_{LG} means performance improvement (or drop) of global models over local models. On each dataset, we produce a distribution for Diff_{LG} over all the communications and all the parallel settings (including numbers of workers and communication frequencies). We show the distribution for Diff_{LG} of MA and ensemble on CIFAR datasets in Fig. 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.

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%. On the other hand, for ensemble, we can observe from Fig. 2 that 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.

Model Compression. In order to avoid model size explosion, the ensemble model is compressed before the next round of local training. However, such com-

pression may result in a risk of performance loss. To examine if the performance improvement of the compressed global models over those local ones in EC-DNN can still outperform such kind of improvement in MA-DNN, we compare Diff_{LG} in MA-DNN (see Eq.(7)) with Diff_{LC} in EC-DNN,

$$\text{Diff}_{\text{LC}} = \frac{1}{K}\sum_{k=1}^{K}\left(\text{error}_k - \text{error}_{\text{compress},k}\right), \tag{8}$$

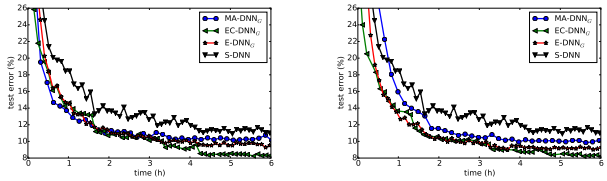
where error_k denotes the test error of the local model on worker k, and $\text{error}_{\text{compress},k}$ denotes the test error of the corresponding compressed model after compressing the ensemble model of those local models on worker k. The positive (or negative) Diff_{LC} means performance improvement (or drop) of the compressed model over local ones. Figure 3 illustrates the distribution of Diff_{LC} over all the communications and various settings of communication frequency and the number of workers on two CIFAR datasets.

From Fig. 3, we can observe that the average performance of the compressed models is consistently better than that of the local models on both datasets in EC-DNN, while Figure 1 indicates that there are over 10% global models do not reaching better performance than the local ones in MA-DNN. In addition, the average improvement of compressed models over local ones in EC-DNN is greater than that in MA-DNN. Specifically, the average of such improvements in EC-DNN are 1.03% and 1.95% on CIFAR-10 and CIFAR-100, respectively, while the average performance difference in MA-DNN are -3.53% and 1.72% on CIFAR-10 and CIFAR-100 respectively. All these results can indicate that EC-DNN is a superior method than MA-DNN.

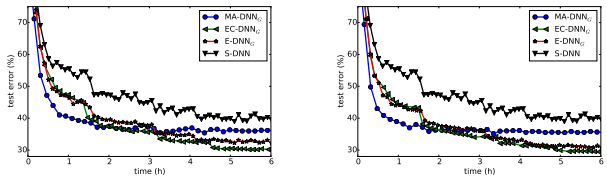
Accuracy. In the following, we examine the accuracy of compared methods. Figure 4 shows the test error of the global model during the training process w.r.t. the overall time, and Table 1 reports the final performance after the training process converges. For EC-DNN, the relabeling time has been counted in the overall time when plotting the figure and the table. We report EC-DNN and MA-DNN that achieve best test performance among all the communication frequencies.

From Fig. 4, we can observe that EC-DNN$_G$ outperforms MA-DNN$_G$ and S-DNN on both datasets for all the number of workers, which demonstrates that EC-DNN is superior to MA-DNN. Specifically, at the early stage of training, EC-DNN$_G$ may not outperform MA-DNN$_G$. We hypothesize the reason as the very limited number of communications among local works at the early stage of EC-DNN training. Along with increasing rounds of communications, EC-DNN will catch up with and then keep outperforming MA-DNN after the certain time slot. Besides, EC-DNN$_G$ outperforms E-DNN$_G$ consistently for different datasets and number of workers, indicating that technologies in EC-DNN are not trivial improvements of E-DNN but is the key factor of the success of EC-DNN.

In Table 1, each EC-DNN$_G$ outperforms MA-DNN$_L$ and MA-DNN$_G$. The average improvements of EC-DNN$_G$ over MA-DNN$_L$ and MA-DNN$_G$ are around 1% and 5% for CIFAR-10 and CIFAR-100 respectively. Besides, we also report



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



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

Fig. 4: Test error curves on CIFAR datasets.

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. Specifically, the best EC-DNN$_L$ 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, E-DNN$_L$ never outperforms MA-DNN$_L$ and MA-DNN$_G$.

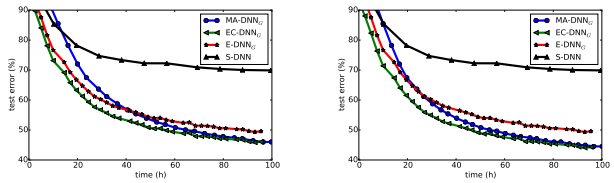
Speed. According to our analysis in Sec 4.3, EC-DNN is more time-efficient than MA-DNN because it communicates less frequently than MA-DNN and thus costs less time on communication. To verify this, we measure the overall time cost by each method to achieve the same accuracy. Table 1 shows the speed of compared methods. In this table, 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 Table 1, we can observe that EC-DNN can achieve better speedup than MA-DNN on all the datasets. On average, EC-DNN$_G$ and EC-DNN$_L$ runs about 2.24 and 1.33 times faster than MA-DNN$_G$, respectively. Furthermore, EC-DNN consistently results in better speedup than E-DNN on all the datasets. On average, E-DNN$_G$ only runs about 1.85 times faster than MA-DNN$_G$ while EC-DNN$_G$ can reach about 2.24 times faster speed. From this table, we can also find that E-DNN$_L$ never achieves the same performance with MA-DNN$_G$ while EC-DNN$_L$ can contrarily run much faster than MA-DNN$_G$.

Furthermore, Table 1 demonstrates the communication frequency τ that makes compared methods achieve the corresponding speed. We can observe that

Table 1: Test error (%), speed and τ on CIFAR datasets.

	CIFAR-10						CIFAR-100					
	K=4			K=8			K=4			K=8		
	Error	Speed	τ	Error	Speed	τ	Error	Speed	τ	Error	Speed	τ
MA-DNN$_G$	10.3	1	16	9.99	1	2k	36.18	1	16	35.55	1	16
E-DNN$_G$	9.44	1.58	-	9.05	1.92	-	32.49	1.95	-	30.9	1.97	-
EC-DNN$_G$	**8.43**	**1.92**	4k	**8.19**	**2.05**	4k	**30.26**	**2.52**	4k	**29.31**	**2.48**	2k
MA-DNN$_L$	10.55	0	16	10.54	0	2k	36.39	0	16	35.56	0	16
E-DNN$_L$	11.04	0	-	10.95	0	-	39.57	0	-	39.55	0	-
EC-DNN$_L$	**10.04**	**1.36**	4k	**9.88**	**1.26**	4k	**34.8**	**1.42**	4k	**35.1**	**1.27**	2k
S-DNN	10.41						35.68					



(a) $K = 4$ (b) $K = 8$

Fig. 5: Test error curves on ImageNet.

EC-DNN 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.

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. Figure 5 shows the test error of the global model w.r.t the overall time. 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 test performance than S-DNN, MA-DNN and E-DNN throughout the training. Besiedes, 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 Conclusion and Future Work

In this paper, we propose EC-DNN, a new Ensemble-Compression based parallel training framework for DNN. As compared to the traditional approach, MA-DNN, that averages the parameters of different local models, our proposed method uses the ensemble method 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 will investigate the theoretical properties of the ensemble method, compression method, and the whole EC-DNN framework.

7 Acknowledgments

This work is partially supported by NSF of China (grant numbers: 61373018, 61602266, 11550110491) and NSF of Tianjin (grant number: 4117JCYBJC15300).

References

1. Bucilua, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: Proceedings of the 12th ACM Conference on Knowledge Discovery and Data Mining. pp. 535–541. ACM (2006)
2. Chen, J., Monga, R., Bengio, S., Jozefowicz, R.: Revisiting distributed synchronous sgd. arXiv preprint arXiv:1604.00981 (2016)
3. Chen, K., Huo, Q.: Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In: Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on. pp. 5880–5884. IEEE (2016)
4. Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y.: Compressing neural networks with the hashing trick. In: Proceedings of the 32st International Conference on Machine Learning (2015)
5. Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 3642–3649. IEEE (2012)
6. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al.: Large scale distributed deep networks. In: Advances in Neural Information Processing Systems. pp. 1223–1231 (2012)
7. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
8. Denil, M., Shakibi, B., Dinh, L., de Freitas, N., et al.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems. pp. 2148–2156 (2013)
9. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems. pp. 1269–1277 (2014)

10. Gong, Y., Liu, L., Yang, M., Bourdev, L.: Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115 (2014)
11. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems 28. pp. 1135–1143 (2015)
12. He, D., Xia, Y., Qin, T., Wang, L., Yu, N., Liu, T., Ma, W.Y.: Dual learning for machine translation. In: Advances in Neural Information Processing Systems 29, pp. 820–828 (2016)
13. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1026–1034 (2015)
14. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
15. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
16. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
17. Kuncheva, L., Whitaker, C.: Measures of diversity in classifier ensembles. In: Machine Learning, pp. 181–207 (2003)
18. Li, M., Andersen, D.G., Park, J.W., Smola, A.J., Ahmed, A., Josifovski, V., Long, J., Shekita, E.J., Su, B.Y.: Scaling distributed machine learning with the parameter server. In: 11th USENIX Symposium on Operating Systems Design and Implementation. pp. 583–598 (2014)
19. Min, L., Qiang, C., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2014)
20. Povey, D., Zhang, X., Khudanpur, S.: Parallel training of dnns with natural gradient and parameter averaging. arXiv preprint arXiv:1410.7455 (2014)
21. Rigamonti, R., Sironi, A., Lepetit, V., Fua, P.: Learning separable filters. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 2754–2761. IEEE (2013)
22. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: Hints for thin deep nets. arXiv preprint arXiv:1412.6550 (2014)
23. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115(3), 211–252 (2015)
24. Sollich, P., Krogh, A.: Learning with ensembles: How overfitting can be useful. In: Advances in Neural Information Processing Systems, vol. 8, pp. 190–196 (1996)
25. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. arXiv preprint arXiv:1409.4842 (2014)
26. Zhang, S., Choromanska, A.E., LeCun, Y.: Deep learning with elastic averaging sgd. In: Advances in Neural Information Processing Systems 28, pp. 685–693 (2015)
27. Zhang, X., Trmal, J., Povey, D., Khudanpur, S.: Improving deep neural network acoustic models using generalized maxout networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 215–219. IEEE (2014)