Slim-DP: A Multi-Agent System for Communication-Efficient Distributed Deep Learning

Shizhao Sun*
College of Computer and Control Engineering, Nankai University
Tianjin, P.R.China
sunshizhao@mail.nankai.edu.cn

Wei Chen
Microsoft Research
Beijing, P.R.China
wche@microsoft.com

Jiang Bian
Microsoft Research
Beijing, P.R.China
Jiang.Bian@microsoft.com

Xiaoguang Liu
College of Computer and Control Engineering, Nankai University
Tianjin, P.R.China
liuxg@nbjl.nankai.edu.cn

Tie-Yan Liu
Microsoft Research
Beijing, P.R.China
Tie-Yan.Liu@microsoft.com

ABSTRACT

To afford the huge computational cost, large-scale deep neural networks (DNN) are usually trained on the distributed system, especially the widely-used parameter server architecture, consisting of a parameter server as well as multiple local workers with powerful GPU cards. During the training, local workers frequently pull the global model and push their computed gradients from/to the parameter server. Due to the limited bandwidth, such frequent communication will cause severe bottleneck for the training acceleration. As recent attempts to address this problem, quantization methods have been proposed to compress the gradients for efficient communication. However, such methods overlook the effects of compression on the model performance such that they either suffer from a low compression ratio or an accuracy drop. In this paper, to better address this problem, we investigate the distributed deep learning as a multi-agent system (MAS) problem. Specifically, 1) local workers and the parameter server are separate agents in the system; 2) the objective of these agents is to maximize the efficacy of the learned model through their cooperative interactions; 3) the strategy of the agents describes how they take actions, i.e. communicate their computed gradients or the global model; 4) rational agents always select the best-response strategy with the optimal utility. Inspired by this, we design a MAS approach for distributed training of DNN. In our method, the agents first estimate the utility (i.e., the benefit to help improve the model) of each action (i.e., transferring a subset of the gradients or the global model), and then take the best-response strategy based on their estimated utilities mixed with ϵ-random exploration. We call our new method *Slim-DP* as it, being different from the standard data-parallelism, only communicates a subset of the gradient or the global model. Our experimental results demonstrate that our proposed Slim-DP can reduce more communication cost and achieve better speedup without loss of accuracy than the standard data parallelism and its quantization version.

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

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

KEYWORDS

Application of multi-agent system; best response strategy; distributed training; deep learning

ACM Reference Format:
Shizhao Sun, Wei Chen, Jiang Bian, Xiaoguang Liu, and Tie-Yan Liu. 2018. Slim-DP: A Multi-Agent System for Communication-Efficient Distributed Deep Learning. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Rapid development of deep neural networks (DNN) has demonstrated that its great success is mainly due to the power of big models learned based on big data [8, 22]. However, the extremely time-consuming training has become a critical debt to obtain a large-scale DNN model. To accelerate the training of DNN, data parallelism powered by the architecture of parameter server [2–4, 16, 30] has emerged as a widely-used technique in recent years. In such a typical framework, the entire training data will be allocated into multiple local workers with powerful GPU cards, and the training procedure continues iterations of the following three steps: 1) First, the parameter server broadcasts the current global model to the local workers; 2) then, each local worker computes the gradients of the current model based on its own local data; 3) finally, the learned gradients are pushed to the parameter server, and are aggregated to produce a new glboal model at the parameter server side [13, 14]. As the parameter server and local workers communicate the gradients and values of all the parameters in each iteration, we refer this standard data parallelism approach as *Plump Data Parallelism*, abbreviated as *Plump-DP*.

Although Plump-DP is well-motivated, it suffers from the heavy communication cost by transferring huge amount of parameters and updates [1, 4, 19]. For example, as shown in [1], for the parallel training of AlexNet [11], GoogLeNet [24], ResNet152 [8] and VGG-19 [20], the communication takes 10% to 50% of the overall training time when there are 8 local workers, which are non-negligible compared to the training time; and the percentage of communication time continues to grow when the number of local workers is increased. Furthermore, such bottleneck will become more severe

when the bandwidth is small, e.g., the distributed training on mobile devices such as federated learning [10, 15].

Recent attempts proposed quantization techniques [1, 19, 27] to reduce heavy communication cost in distributed deep learning. In particular, the 32-bit gradient is compressed into a small number of bits during the communication. We call the standard data parallelism with quantization *Quantized Data Parallelism*, abbreviated as *Quant-DP*. However, Quant-DP overlooks the effects of compression on the efficacy of the learned model. More concretely, while Quant-DP uniformly sacrifices the precision of all the gradients, the influence on the model performance of sacrificing the precision of the gradient with large scale is larger than that of the gradient with the scale close to zero. As a result, Quant-DP either suffers from a low compression ratio or harms the accuracy.

To reduce communication cost and preserve the accuracy at the same time, we propose a novel multi-agent system (MAS) approach towards communication-efficient data parallelism[1]. Particularly, in a typical distributed deep learning system, the local workers and the parameter server naturally play as separate agents. Since the role of the local workers are different from that of the parameter server, we call local worker *local agents* and parameter server *center agent*. The eventual objective of all these agents is to obtain a model with maximized efficacy through their cooperative interactions during training.

To achieve the goal of obtaining a model with maximized efficacy, each agent needs to determine its own strategy that describes their actions in terms of transferring computed gradients or broadcasting the global model within the certain bandwidth constraint. In such a multi-agent system, due to the limited communication cost, rational agents tend to select and transfer a subset of the important gradients or parameters instead of the whole set like Plump-DP. Therefore, the action of local workers corresponds to select a subset of their computed gradients to communicate, while the action of the parameter server corresponds to select a subset of the parameters in the global model to communicate. In each training iteration, 1) after the global model is updated by aggregating gradients transferred from local agents, the center agent takes the action in terms of broadcasting a subset of the current global model, 2) after computing the gradients of their local models, which merges the broadcasted parameters from the center agent, the local agents take the action in terms of transferring a subset of their computed gradients to the center agent. At the end of training process, all the agents will receive the utility in terms of the performance of the new global model.

Although the center agent and the local agents are cooperating with each other, the information available for them to make the decisions is still quite limited. Specifically, local agents only have the local model with parts of the parameter replaced by the current global model, and the center agent only has the current global model. Therefore, communication-efficient distributed deep learning system is essentially a multi-agent system with very limited partial information. In such situation, it is quite natural and reasonable to estimate the utility based on their own information, and then follow the best-response strategy to take the action that maximizes the estimated utility.

Consequently, a critical component of this MAS approach is the estimation of the utilities (i.e., the improvement to the global model) for each action (i.e., transferring a subset of the gradients or the global model). It is well known in optimization that, if the model can only be updated in some coordinates (corresponds to a subset of gradients), it will achieve the most objective improvement by updating the model in the coordinates with largest scale. Thus, it is reasonable for the local agents to estimate the utilities of the subset of the gradients by the total scales of the gradients. Another common observation in deep learning is that, the DNN models has high redundancy and we can preserve the performance if we remove parameters with small scales [7]. Therefore, the center agent can estimate the utilities of subsets of the parameters by the total scales of the parameters.

Moreover, in order for better exploration, we mix best-response strategy with ϵ-random strategy, i.e., taking best-response strategy that maximizes the expected utility with probability $1 - \epsilon$ and taking other actions with probability ϵ. Particularly, in the MAS of distributed deep learning, we propose to allocate ϵ proportion of the subset to be random selected in each interaction, instead of random selecting the whole subset with probability ϵ. In this way, the update of the global model can be more stable, which is very helpful for the convergence of DNN models. Compared with the standard Plum-DP, our approach can significantly reduce the communication cost by only transferring the subset of gradients (or parameters) instead of the whole set, and we call our method *Slim Data Parallelism*, abbreviated as Slim-DP.

To verify the effectiveness of Slim-DP, we conducted experiments on ImageNet [18], and evaluate our method on two state-of-the-art models, i.e., GoogLeNet and VGG-16. We have following observations from the experimental results: 1) compared to Plump-DP, Slim-DP can save about 55% and 70% of the communication time for GoogLeNet and VGG-16 respectively; 2) by saving communication time, Slim-DP can process 10%+ and 30%+ more images per second than Plump-DP for GoogLeNet and VGG-16 respectively; 3) Slim-DP reduces the communication cost and improves the speedup without loss of accuracy; 4) Slim-DP also outperforms other communication-efficient version of Plump-DP, i.e., Quant-DP, for both GoogLeNet and VGG-16, in terms of saved communication time, speed and accuracy.

In the remaining part of this paper, we first introduce the standard data parallelism of DNN in Section 2. Then, in Section 3, we give formal description of communication-efficient data parallelism as a multi-agent system, and propose Slim-DP based on such multi-agent system. We show experimental results in Section 4. At last, we discuss the related works and future works in Section 5 and Section 6 respectively.

2 PRELIMINARIES

In this section, we introduce the standard data parallelism (i.e., Plump-DP) with the popular parameter server architecture [13, 14]. We denote a DNN model as $f(\mathbf{w})$, where $\mathbf{w} = \{w_1, \ldots, w_N\}$ is the vector of the parameters and N is the number of the parameters. We assume that there are K local workers employed in the parallel

[1]In this paper, we take the popular parameter server architecture [13] as an example. Other architectures like MapReduce [5] can also be covered by our method with some simplifications.

architecture, and each local worker, say the k-th 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. We denote the parameters and gradients of the local model at the iteration t on the worker k as $\mathbf{w}_k^t$ and δ_k^t. And, we denote the global model at the iteration t on the parameter server as $\mathbf{w}^t$. The communication of gradients will be invoked after the local worker conducts every p iterations of local training on its local data. We call p the communication frequency. A typical data parallelism for DNN [2, 4] iteratively implements the following three steps until the training procedure converges[2].

1. Broadcast/Pull: The parameter server broadcasts the global model to the local workers.

2. Local training: Each local worker independently trains the local model based on its local data by stochastic gradient decent (SGD) or other stochastic algorithm. There will be no synchronization with the parameter server until every p local training iterations.

3. Push: Each local worker pushes the gradients δ_k^t of its local model to the parameter server. At the parameter server side, these gradients are aggregated to generate a new global model.

In spite of its straightforward motivation, Plump-DP suffers from the heavy communication cost by communicating huge amount of parameters and gradients. While some recent efforts attempted to leverage quantization techniques [1, 19, 27] to reduce the communication cost by sacrificing the precision of each gradient uniformly, they cannot preserve the accuracy and reduce the communication cost at the same time. To tackle these challenges, we view the communication-efficient data parallelism as a multi-agent system, and propose a new communication-efficient data parallelism framework called *Slim-DP*.

3 SLIM-DP

In this section, we propose a new communication-efficient data parallelism framework, i.e., Slim-DP, to address the challenge in terms of heavy communication cost of Plump-DP.

3.1 Communication-Efficient Data Parallelism as a Multi-Agent System

In the communication-efficient distributed deep learning, the parameter server and the local workers play as separate agents cooperating with each other to train a DNN model. The eventual objective of all these agents is to learn a model with maximized efficacy through their interactions during training. To achieve this goal, each agent needs to determine its own strategy. More concretely, the parameter server's strategy corresponds to the actions in terms of broadcasting the global model within the certain bandwidth constraint, while the local worker's strategy corresponds to the actions in terms of transferring computed gradients within the same bandwidth constraint. As the role of the local workers are different from that of the parameter server, we call local worker *local agents* and parameter server *center agent*.

Now, we describe the key components of the multi-agent system for communication-efficient data parallelism.

[2]In this paper, we focus on synchronous data parallelism, considering that synchronous data parallelism can achieve better convergence than asynchronous data parallelism [2]. Note that the algorithm and results can be generalized to the asynchronous data parallelism as well.

1. Action. When the communication bandwidth is limited, rational agents tend to select a subset of the gradients or parameters for communication instead of the whole set like Plump-DP. In other words, the center agent can specify its action a_0 as selecting a subset of the parameters to broadcast, and the k-th local agent can specify its action a_k as selecting a subset of the computed gradients to transfer to the center agent. Since there are same amounts of parameters and gradients (i.e., N), we denote $a_0, a_1, \ldots, a_k \in \mathcal{A}_C$, where the action space $\mathcal{A}_C = \{(c_1, \ldots, c_N) | \sum_{i=1}^N c_i = C, c_i \in \{0, 1\}, i = 1, \ldots, N\}$ contains all the subsets of $\{1, \ldots, N\}$, and C is the communication bandwidth.

2. Environments. There are center environment and local environments in this multi-agent system. Assume the current global model is $\mathbf{w}$, and the center agent takes an action a_0 to select a subset from it. Then, the selected parameter set to broadcast can be denoted as $a_0 \odot \mathbf{w}$, where $\odot$ is the element-wise product. We call the current global model $\mathbf{w}$ *center environment*. After receiving $a_0 \odot \mathbf{w}$, the k-th local agent first updates its local model $\mathbf{w}_k$ by $\mathbf{w}_k \oplus (a_0 \odot \mathbf{w})$, where $\oplus$ stands for the overwrite operation that overwrites the corresponding parameters in the local model by the received part of the global model. Then, the local agent computes the new gradients of the updated local model over its local mini-batch data s_k. We call the local models $\{\mathbf{w}_1, \ldots, \mathbf{w}_K\}$ and the local mini-batch data $\{s_1, \ldots, s_K\}$ *local environments*.

3. Utility. Given the actions and environments, the total utility can be defined as the performance of the new global model. Mathematically,

$$\begin{aligned} & u(a_0, a_1, \ldots, a_K; \mathbf{w}, \mathbf{w}_1, \ldots, \mathbf{w}_K, s_1, \ldots, s_K) \\ = \quad & -L\left(\mathbf{w} + \sum_{k=1}^{K} a_k \odot (g(\mathbf{w}_k \oplus (a_0 \odot \mathbf{w}), s_k))\right), \end{aligned} \tag{1}$$

where $L(\cdot)$ is the loss of a model and $g(\cdot)$ is the gradient of a model.

Whilst both center and local agents are cooperating with each other, the information available for them to make the decisions is limited. On the one hand, the center agent is only aware of the center environment; on the other hand, local agents know nothing but their own local environments. Therefore, communication-efficient distributed deep learning is a multi-agent system with very limited partial information.

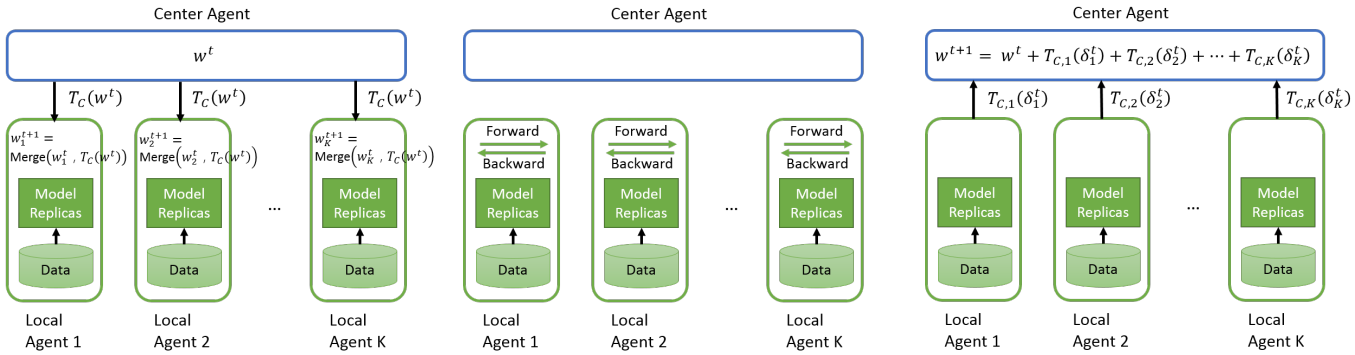
3.2 Best-Response Strategy

In a typical multi-agent system with very limited partial information, rational agents usually follow the best-response strategy to take the action that can maximize the utility estimated based on their own information.

In the communication-efficient distributed deep learning, we assume that the center agent estimates its utilities as,

$$\hat{u}(a_0; \mathbf{w}) = \|a_0 \odot \mathbf{w}\|_1. \tag{2}$$

In other words, the center agent estimates the utilities of the actions by the total scales of the parameters in corresponding subset. This is inspired by the common observation [7] that deep neural networks usually have many redundant parameters, especially those with small scales, removing which even causes no significant performance drop of the model.



(a) Pull **(b) Local Training** **(c) Push**

Figure 1: Slim-DP.

Meanwhile, we assume that the k-th local agent estimates its utilities as

$$\hat{u}_k(a_k; a_0 \odot \mathbf{w}, \mathbf{w}_k, s_k) = \|a_k \odot g(\mathbf{w}_k \oplus (a_0 \odot \mathbf{w}), s_k)\|_1. \quad (3)$$

In other words, the local agents estimate the utilities of the actions by the total scales of the gradients in corresponding subset. This is inspired by that when we can only update the model in some coordinates (corresponding to a subset of gradients), updating the model in the coordinates with the scale close to zero will not bring much improvement to the loss function [12].

As all the agents follow the best response strategy based on estimated utilities to take actions, it is necessary to mix best-response strategy with ϵ-random strategy, which traditionally takes best-response strategy that maximizes the expected utility with probability $1-\epsilon$ and taking other actions with probability ϵ. Particularly, in the MAS of communication-efficient distributed deep learning, we propose to allocate ϵ proportion of the subset of the gradients (or parameters) to be random selected in each interaction, instead of random selecting the whole subset with probability ϵ. In this way, the update of the global model can be more stable, which is very helpful for the convergence of the deep learning model. Specifically, agents will finally take the following action:

$$a_0^* = \left(\arg \max_{a_0 \in \mathcal{A}_{(1-\epsilon)C}} \hat{u}(a_0; \mathbf{w})\right) \wedge a^\epsilon; \quad (4)$$

$$a_k^* = \left(\arg \max_{a_k \in \mathcal{A}_{(1-\epsilon)C}} \hat{u}(a_k; a_0 \odot \mathbf{w}, \mathbf{w}_k, s_k)\right) \wedge a^\epsilon,\ k = 1, \ldots, K$$

where a^ϵ is randomly sampled from $\mathcal{A}_{\epsilon C}$.

3.3 Algorithm Description

Figure 1 illustrates the whole Slim-DP. Similar to Plump-DP, it also consists of three iterative steps, i.e., pull, local training and push. In this figure and the remaining parts of this paper, for ease of reference, we call the subset of gradients transferred during the communication the *communication set*. In addition, we call the portion of the communication set that is randomly selected the *explorer*, denoted as $T_{R,k}(\delta_k^t)$, and the portion of the communication set that is selected to maximize the estimated utility (i.e., the scale of the gradients) the *core*, denoted as $T_{S,k}(\delta_k^t)$. It is clear that $T_{C,k}(\delta_k^t) = T_{R,k}(\delta_k^t) \cup T_{S,k}(\delta_k^t)$, where k indicates the k-th local agent. Similarly, for the center agent, we use the notation $T_C(\mathbf{w}^t)$, $T_R(\mathbf{w}^t)$ and $T_S(\mathbf{w}^t)$.

Firstly, we introduce the inputs for Slim-DP as follows.

1. The local data set D_k, where $k \in \{1, \ldots, K\}$;
2. The hyperparameter $\alpha \in (0, 1)$ that controls the size of communication set in Slim-DP comparing to Plump-DP, i.e., $\alpha = \frac{|T_{C,k}(\delta_k^t)|}{|\delta_k^t|} = \frac{|T_C(\mathbf{w}^t)|}{|\mathbf{w}^t|}$;
3. The hyperparameter $\epsilon \in [0, \alpha]$ that controls the size of the explorer, i.e., $\epsilon = \frac{|T_{R,k}(\delta_k^t)|}{|\delta_k^t|} = \frac{|T_R(\mathbf{w}^t)|}{|\mathbf{w}^t|}$;
4. The communication frequency p, i.e., the frequency that the local worker push $T_{C,k}(\delta_k^t)$ to or pull $T_C(\mathbf{w}^t)$ from the parameter server.

Then, we introduce the Slim-DP algorithm (described in Algorithm 1), which consists of the following six steps. All the steps are executed iteratively until the training converges.

1. LocalTrain ($\mathbf{w}_k^t, D_k, p$): Each local agent computes the gradients of the parameters $\mathbf{w}_k^t$ by minimizing the cross entropy loss using SGD on its local dataset D_k. Such computation lasts for p mini-batches before the communication with the parameter server. We accumulate the model updates over p mini-batches and denote the result as δ_k^t.

2. CoreSelection ($\delta_k^t, \alpha - \epsilon$): Each local agent chooses $\alpha - \epsilon$ of gradients δ_k^t to maximize the expected utility, i.e., the scale of the gradients. Similarly, this step can also be executed by the center agent, in which the parameters is chosen.

3. Exploration ($\delta_k^t \backslash T_{S,k}(\delta_k^t), \epsilon$): Each local agent randomly samples ϵ of gradients from the set of gradients outsides the core, i.e., $\delta_k^t \backslash T_{S,k}(\delta_k^t)$. Similarly, this step can also be executed by the center agent, in which the parameters is randomly sampled.

4. Push ($T_{C,k}(\delta_k^t)$): At the local agent side, we execute Push ($T_{C,k}(\delta_k^t)$), i.e., send the subset of local updates (i.e., $T_{C,k}(\delta_k^t)$) to the center agent. At the center agent side, we add the gradients received from all the local agents to the global model, i.e., $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \sum_{k=1}^K T_C(\delta_k^t)$.

Algorithm 1: Slim-DP (D_k, α, p, ϵ)

Local Agent k:
$\mathbf{w}_k^0 \leftarrow$Pull($\mathbf{w}^0$);
while $f(\tilde{\mathbf{w}}^t; x)$ *dose not converge* **do**
  $\delta_k^t \leftarrow$ LocalTrain($\mathbf{w}_k^t, D_k, p$);
  $T_{S,k} \leftarrow$ CoreSelection($\delta_k^t, \alpha - \epsilon$);
  $T_{R,k} \leftarrow$ Exploration($\delta_k^t \backslash T_{S,k}(\delta_k^t), \epsilon$);
  $T_{C,k} \leftarrow T_{S,k} \cup T_{R,k}$;
  Push($T_{C,k}(\delta_k^t)$);
  $\tilde{\mathbf{w}}^t \leftarrow$Pull($T_C(\mathbf{w}^t)$);
  $\mathbf{w}_k^t \leftarrow$ Merge($\mathbf{w}_k^t, \tilde{\mathbf{w}}^t$);
Center Agent:
while $f(\tilde{\mathbf{w}}^t; x)$ *does not converge* **do**
  $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta \sum_{k=1}^K T_{C,k}(\delta_k^t)$;
  $t \leftarrow t + 1$;
  $T_S \leftarrow$ CoreSelection($\mathbf{w}^t, \alpha - \epsilon$);
  $T_R \leftarrow$ Exploration($\mathbf{w}^t \backslash T_S(\mathbf{w}^t), \epsilon$);
  $T_C \leftarrow T_S \cup T_R$.

5. Pull ($T_C(\mathbf{w}^t)$): Each local agent pulls the subset of the parameters $T_C(\mathbf{w}^t)$ from the center agent.

6. Merge ($\mathbf{w}_k^t, \tilde{\mathbf{w}}^t$): The parameter $\tilde{\mathbf{w}}^t$, which is pulled from the center agent, will be merged with the current local parameters $\mathbf{w}_k^t$ to produce a new local model. This new model will be set as the starting point of next round of local training.

3.4 Communication Efficiency and Time Efficiency

In this subsection, we discuss the communication and time efficiency of Slim-DP, and introduce some implementation details to further improve the communication efficiency and time efficiency.

3.4.1 Communication Efficiency. In Slim-DP, since we transfer a subset instead of the whole set of the gradients like in Plump-DP, we should make the receiver know which gradients are transferred. Therefore, the information transfered between local agent and the master agent should be represented as ⟨key, value⟩ pairs, where the key is the index of the gradients (or parameters) and the value is the corresponding gradients (or parameters). Thus, the real amount of transferred information is the double the size of the communicated gradients (or parameters).

In practice, the communication efficiency can be further improved by executing *CoreSelection* step after each q rounds of communication (i.e., $q > 1$) instead of for each round of communication. The reason is that the model will not differ very significantly after recent updates and thus the scale of gradients will not change that fastly when it is computed based on very close model. Then, as the*CoreSelection* step is executed after each q rounds of communication, for the information about the core, we can use key catching filter [14] to transfer it because the core is not frequently renewed during the training, and thus the keys for the core can keep unchanged for a period. In the key catching filter, both the sender and receiver have cached the keys, and the sender then only needs to send the values with a signature of the keys. Therefore, the real amount of transferred information for the core is approximated to the same size of the core, i.e., $(\alpha-\epsilon)N$, where N is the number of the gradients (or parameters). For the information about the explorer, we still need to transfer it by the ⟨key, value⟩ pair, and thus the real amount of transferred information for the explorer is $2\epsilon N$.

Therefore, in Slim-DP, the real amount of total transferred information is $(\alpha + \epsilon)N$. In Plump-DP, the real communication amount equals to the number of the gradients, i.e., N, since we need to transfer the whole set of the gradients (or parameters).

3.4.2 Time Efficiency. Compared to Plump-DP, Slim-DP may bring in two kinds of potential extra time. The first kind of potential extra time is the time to generate the index for the explorer and the core. For the generation of the index for the explorer, since it is not related to the scale of the gradients, it can be executed off-line or overlapped with the gradient computation. For the generation of the index for the core, since it is less frequent than the communication in practice, the time for it can be ignored comparing to the communication time. Therefore, generating the index for explorer and core will not bring in extra time.

The second kind of potential extra time is the time to extract corresponding gradients (or parameters) from the generated index of the explorer and the core. In the worst case, such extraction is done by scanning the whole set of the gradients (or parameters), whose time cost is proportional to the number of the parameters N. In practical implementation, multi-thread scanning can be easily leveraged to ensure limited such time cost.

Overall, Slim-DP will bring a very small amount of extra time, which is less than $O(N)$.

3.5 Discussions

We make the following discussions for Slim-DP:

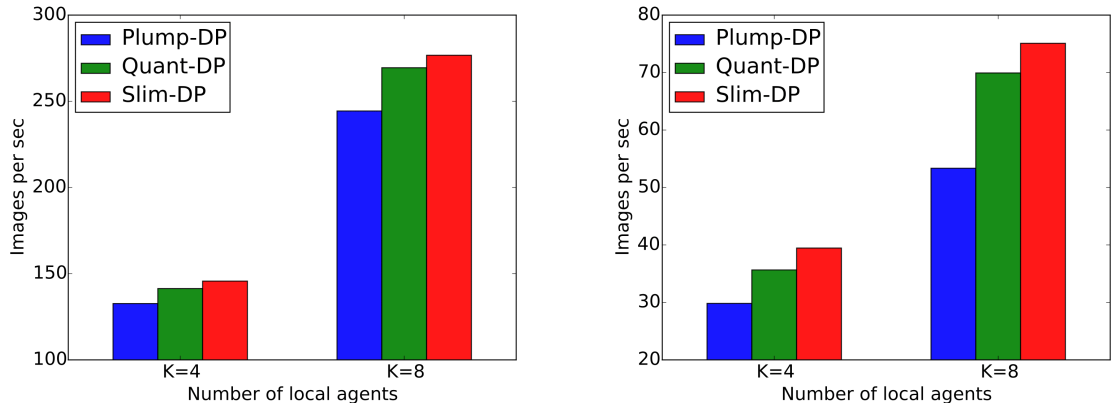
1. Trade-off between Accuracy and Speed. The size of the communication set α trades-off the accuracy and the speed. On the one hand, larger α indicates that Slim-DP can communicate more gradients, including both the core and the explorer, which will result in better accuracy. The reason is that a larger core ensures the coverage of sufficient number of important gradients (i.e., the gradients with large scale), and a larger set of random explored parameters results in sufficient exploration outside the core. On the other hand, larger α also implies more communication cost, which slows down the training.

2. Trade-off between Exploration and Exploitation. For the fixed size of the communication set α, the value of ϵ trades-off the exploration and the exploitation. On the one hand, when ϵ is very small or even $\epsilon = 0$, there is no sufficient exploration of gradients other than those in the core (i.e., the best response) during the training, which hurts the performance. On the other hand, when ϵ yields a greater value or even $\epsilon = \alpha$, the selected core cannot cover enough important gradients (i.e., the gradients with large scale), which will hurt the performance as well.

3. Relationship with Dropout/DropConnect. The motivation of the two methods are different. Slim-DP aims at designing a data parallelism approach to accelerate the training of big DNN models by reducing communication cost, while Dropout/ Dropconnect [21, 25] is a trick in the sequential training to avoid overfitting. Moreover,

Table 1: Top-5 Accuracy (%) and Saved Communication Time.

Model	Method	K=4		K=8	
		Top-5 Accuracy	Saved Communication Time	Top-5 Accuracy	Saved Communication Time
GoogLeNet	Plump-DP	88.06	-	88.03	-
	Quant-DP	88.02 (-0.04)	47.85%	88.08 (+0.05)	49.87%
	Slim-DP	**88.29 (+0.23)**	**55.12%**	**88.23 (+0.20)**	**56.14%**
VGG-16	Plump-DP	86.53	-	86.48	-
	Quant-DP	86.55 (+0.02)	63.05%	86.53 (+0.05)	63.97%
	Slim-DP	**87.03 (+0.50)**	**71.15%**	**86.91 (+0.43)**	**70.06%**



(a) GoogLeNet **(b) VGG-16**

Figure 2: Speed.

CoreSelection step (corresponding to the best response) is quite indispensable for Slim-DP, while Dropout/DropConnect simply applies pure random sampling.

4 EXPERIMENTS

4.1 Experimental Settings

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

Data. We conduct experiments on ImageNet (ILSVRC 2015 Classification Challenge) [18]. In our experiments, each image is normalized by subtracting the per-pixel mean computed over the whole training set, and cropped to the size of 224×224. In addition, no data augmentation is used during the training.

Model. We employ two models, i.e., VGG-16 [20] and GoogLeNet [24]. VGG-16 is a 16-layer convolutional neural network with about 140M parameters and GoogLeNet is a 22-layer convolutional neural network with about 13M parameters. All the hyperparameters of the models, e.g., initialization, learning rate, dropout ratio, weight decay and coefficient for momentum, are set the same as that in the Caffe [9] model zoo.

Parallel Setting. We explore the number of local workers $K \in \{4, 8\}$. Local workers communicate with the parameter server after the updates for every mini-batch, i.e., $p = 1$. We use DMTK framework[3] to implement the related operations of the parameter server, i.e., the center agent.

Hyperparameter Setting of Slim-DP. For each dataset, we explore the size of the communication set $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the size of the explorer $\epsilon \in \{0, 0.25\alpha, 0.5\alpha, 0.75\alpha, \alpha\}$, and then report the one that achieves best performance over all these hyperparameters. Finally, we choose to use $\alpha = 0.2$ and $\epsilon = 0.1$ for VGG-16, and $\alpha = 0.3$ and $\epsilon = 0.15$ for GoogLeNet.

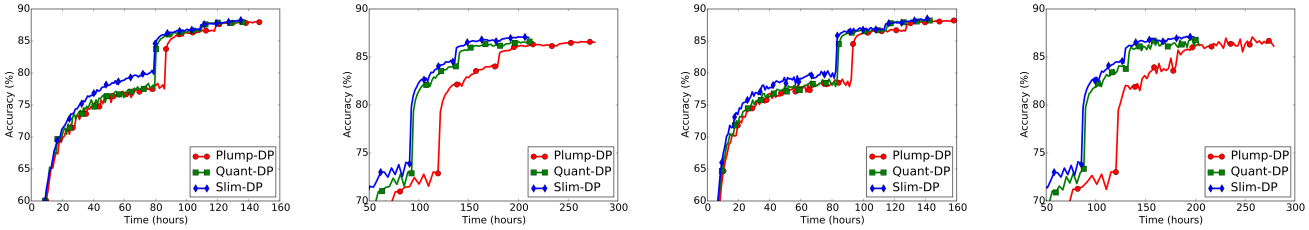
4.2 Compared Methods

We compare performance of the following three methods.

- **Plump-DP** denotes the standard data parallelism framework that transfers the updates and parameters of the whole global model [2, 4].
- **Quant-DP** denotes the method that reduces the communication cost by quantizing each gradient to a small number of bits (less than 32 bits) during the communication[4]. There are

[3] https://github.com/Microsoft/multiverso.

[4] Actually, Quant-DP and Slim-DP can be used simultaneously to reduce the communication cost because Quant-DP aims at reducing precision of each float while Slim-DP aims at reducing the number of such float. The comparison between Quant-DP and



(a) **GoogLeNet,** $K = 4$ (b) **VGG-16,** $K = 4$ (c) **GoogLeNet,** $K = 8$ (d) **VGG-16,** $K = 8$

Figure 3: Top-5 Test Accuracy w.r.t. Time

a few kinds of such quantization method, i.e. 1-bit SGD [19], TernGrad [27] and random quantization SGD [1]. In our experiments, we implement the last one since it yields better performance by introducing randomization. And, we use the same hyper-parameters as in [1], i.e., we employ the 8-bit version and set the bucket size as 512.

- **Slim-DP** refers to the communication-efficient data parallelism framework proposed in this paper, which reduces the communication cost by transferring the gradients (or parameters) with large scale (corresponding to the best response) together with a random explored set of other gradients (or parameters).

4.3 Experimental Results

4.3.1 Communication Cost. We first compare the communication cost of different methods. To this end, we count the communication time caused by that each local agent processes 10k mini-batches of data. For Quant-DP, the extra decoding and encoding time has been counted into the communication time. For Slim-DP, the extra time to extract parameters/updates according to exploration and core-selection (see Section 3.4 for the analysis of the time efficiency) has been counted into the communication time.

Table 1 shows the reduced communication time of Quant-DP and Slim-DP compared to Plump-DP on both GoogleNet and VGG-16. From this table, Slim-DP can reduce more communication time than Quant-DP. For example, when $K = 4$, Slim-DP saves about 55% and 71% of the communication time for GoogLeNet and VGG-16 respectively, while Quant-DP only saves about 47% and 63% of the communication time respectively. The observations of $K = 8$ are similar.

4.3.2 Speed. To compare the speed of different methods, we count the number of images can be processed by each method per second. Figure 2 shows the results. From the figure, we can observe that Plump-DP can process more images per second than both Plump-DP and Quant-DP. Specifically, the improvement of speed is related to the communication-to-computation ratio and the number of local agents. First, DNN models with larger communication-to-computation ratio (e.g., VGG-16) can benefit more from Slim-DP than those with smaller ratio (e.g., GoogLeNet). For example, when $K = 8$, Slim-DP can process 40%+ more images than Plump-DP on VGG-16, while it processed 10%+ more images than Plump-DP on GoogLeNet. Second, the situation with more local agents can benefit more from Slim-DP than that with less local agents. For example, on VGG-16, Slim-DP can process 30%+ more images than Plump-DP when $K = 4$ and 40%+ more images when $K = 8$.

4.3.3 Accuracy. For a distributed training method, the communication cost should be reduced (and the speed should be improved) without significant loss of accuracy. Therefore, we compare the accuracy of all the methods. Table 1 summarizes the accuracy when the method is trained to convergence. From the table, we can observe that Slim-DP reduces the communication cost (and improves the speed) without introducing loss of accuracy, and it even achieves better accuracy than Plump-DP while Quant-DP only achieves comparable performance with Plump-DP. Specifically, the accuracy improvement of Slim-DP over Plump-DP is about 0.2% and 0.5% for GoogLeNet and VGG-16 respectively. In addition, except for the final test accuracy when the model is trained to convergence, we also show the test accuracy during the training in Figure 3. From the figure, we can observe that Slim-DP consistently achieves better performance than both Plump-DP and Quant-DP during the training.

4.3.4 Trade-off between Exploration and Exploitation. To investigate the effects of exploration (i.e., randomly choosing the gradients or parameters) and exploitation (i.e., choosing the gradients or parameters that maximize the expected utility), we fix the size of the communication set (i.e., α) and vary the size of the explorer (i.e., ϵ). We set $\alpha = 0.3$, and compare the performance of Slim-DP when $\epsilon = 0.3$ (no exploitation), $\epsilon = 0.15$ (the one used in the former experiments), and $\epsilon = 0$ (no exploration). For ease of reference, we denote Slim-DP with constraint α on the communication set and the ratio ϵ that controls the size of the explorer as Slim-DP (α, ϵ).

Figure 4 shows the test accuracy curves w.r.t. the overall time. Note that we take GoogLeNet and $K = 4$ as an example and the observations on VGG-16 and $K = 8$ are similar. From this figure, we can observe that Slim-DP (0.3,0.15), which considers both exploration and exploitation, achieves best performance, indicating that both exploration and exploitation are indispensable for the success of Slim-DP. When there is no exploitation, i.e., Slim-DP (0.3,0.3), Slim-DP equals to DropConnect. We observe that Slim-DP in such case slightly improves the performance of Plump-DP in terms of accuracy as a regularization method. When there is no exploration,

Slim-DP in the experiments only aims at emphasizing the benefit of leveraging the knowledge about multi-agent system to solve the problems in the distributed deep learning but not to negate Quant-DP.

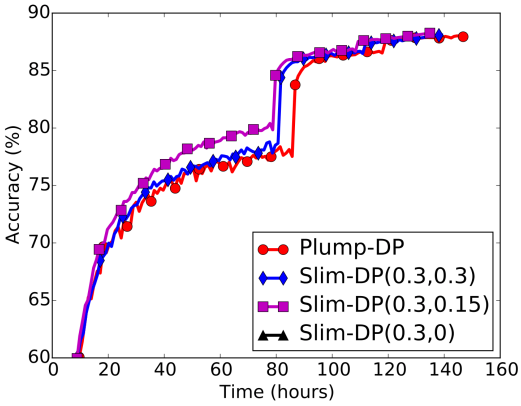


Figure 4: Exploration and Exploitation Trade-off.

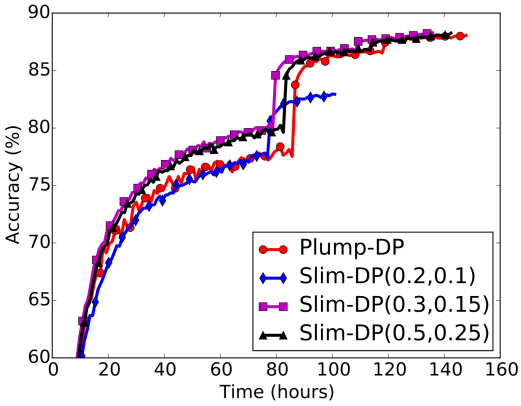


Figure 5: Accuracy and Speedup Trade-off.

i.e., Slim-DP (0.3,0), Slim-DP fails to converge and thus we do not show it in the figure.

4.3.5 Trade-off between Accuracy and Speedup. We plot the test accuracy curve w.r.t. the overall time for Slim-DP (0.2, 0.1), Slim-DP (0.3, 0.15) (the one that we used in the former experiments) and Slim-DP (0.5, 0.25) in Figure 5. We take GoogLeNet and $K = 4$ as an example and the observations on VGG-16 and $K = 8$ are similar. We fix the ratio of the size of the explorer to the size of communicated set (i.e., fix ϵ/α) to avoid extra influence introduced by the trade-off between exploration and exploitation.

From Figure 5, we observe that Slim-DP (0.3,0.15) achieves both best speedup and accuracy. For Slim-DP (0.2,0.1), it cannot achieve the same accuracy as Plump-DP since it communicates too few parameters and cannot cover enough important gradients (or parameters). For Slim-DP (0.5,0.25), although it achieves the similar accuracy as Slim-DP (0.3, 0.15), it does not achieve the similar speedup as Slim-DP (0.3, 0.15) because it transfers more gradients (or parameters).

5 RELATED WORKS

Many works improve the parallel training of DNN by designing new local training algorithms, new model aggregation methods, and new global model update rules. For example, to improve the local training, NG-SGD [16] implements an approximate and efficient algorithm for Natural Gradient SGD; large mini-batch methods [6, 29] increase the learning rate and the mini-batch size to accelerate the convergence. To design new model aggregation methods, EASGD [30] introduces an elastic force which takes the weighted combination of the local model and the global model as the new local model; EC-DNN [23] uses the output-average instead of the parameter-average to aggregate local models. To improve the global model update rules, BMUF [3] designs block-wise model-update filtering and utilizes the momentum of the global model to improve the speedup of Plump-DP. All of these methods do not aim at the same problem as our method, and can be used simultaneously with our method to further improve the performance of distributed deep learning.

Some works consider to improve distributed deep learning by reducing the communication cost. First, system-level technique makes the computation of one layer overlap with the communication of gradients of another layer [2, 6]. Unfortunately, there is no system-level technique can perfectly hide all the communication time without loss of accuracy [2], and thus it is necessary to employ algorithm-level methods. Second, for NLP tasks, sampling method [28] only transfers the gradients of the parameters that corresponds to the most frequent words in the vocabulary in the RNN model. However, such method cannot be generalized to the parallel training of general DNN models. Furthermore, quantization method [1, 19, 27] quantizes each gradient to a small number of bits (less than 32 bits) during the communication.

A few works consider to describe the distributed machine learning as a multi-agent system, but they target at different task with us. For example, job assignment problem is studied in [26] and distributed data mining is combined with multi-agent system in [17]. To best of our knowledge, it is the first time to apply techniques about multi-agent system to communication-efficient distributed deep learning.

6 CONCLUSION AND FUTURE WORK

In this paper, we describe the distributed training of DNN as a multi-agent system, and propose a novel MAS approach to communication-efficient data parallelism, called Slim-DP. Specifically, we only transfer a subset of the gradients or parameters during the communication, which consists of one set that is chosen to maximize the expected utilities and another randomly picked set. Experimental results demonstrate that Slim-DP reduces more communication cost and achieves better speedup without loss of accuracy than standard data parallelism and its quantization version. This paper is a successful attempt to leverage the knowledge about multi-agent system to solve the problem in the distributed deep learning. In the future, we plan to give some theoretical justifications for Slim-DP and propose more algorithms from the MAS view to improve the performance of distributed deep learning.

REFERENCES

[1] Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. QSGD: Randomized Quantization for Communication-Optimal Stochastic Gradient Descent. *arXiv preprint arXiv:1610.02132* (2016).

[2] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting Distributed Synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).

[3] Kai Chen and Qiang Huo. 2016. 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*. IEEE, 5880–5884.

[4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. 1223–1231.

[5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[6] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

[7] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28*. 1135–1143.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385* (2015).

[9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[10] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[12] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. 598–605.

[13] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation*. 583–598.

[14] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*. 19–27.

[15] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[16] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. 2014. Parallel training of DNNs with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455* (2014).

[17] Vuda Sreenivasa Rao. 2009. Multi agent-based distributed data mining: An overview. *International Journal of Reviews in Computing* 3 (2009), 83–92.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[19] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs.. In *Interspeech*. 1058–1062.

[20] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[21] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[22] Rupesh K Srivastava, Klaus Greff, and Juergen Schmidhuber. 2015. Training Very Deep Networks. In *Advances in Neural Information Processing Systems 28*. 2368–2376.

[23] Shizhao Sun, Wei Chen, Jiang Bian, Xiaoguang Liu, and Tie-Yan Liu. 2017. Ensemble-Compression: A New Method for Parallel Training of Deep Neural Networks. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[25] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 1058–1066.

[26] Gerhard Weiß. 1998. A multiagent perspective of parallel and distributed machine learning. In *International Conference on Autonomous Agents: Proceedings of the second international conference on Autonomous agents*, Vol. 10. 226–230.

[27] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *Advances in Neural Information Processing Systems 28*.

[28] Tong Xiao, Jingbo Zhu, Tongran Liu, and Chunliang Zhang. 2017. Fast Parallel Training of Neural Language Models. *International Joint Conference on Artificial Intelligence* (2017).

[29] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Scaling SGD Batch Size to 32K for ImageNet Training. *arXiv preprint arXiv:1708.03888* (2017).

[30] Sixin Zhang, Anna E Choromanska, and Yann LeCun. 2015. Deep learning with Elastic Averaging SGD. In *Advances in Neural Information Processing Systems 28*. 685–693.