RCD-SGD: Resource-Constrained Distributed SGD in Heterogeneous Environment Via Submodular Partitioning

Haoze He[*]
New York University, NY, USA
hh2537@nyu.edu

Parijat Dube
IBM Research, NY, USA
pdube@us.ibm.com

Abstract

The convergence of SGD based distributed training algorithms is tied to the data distribution across workers. Standard partitioning techniques try to achieve equal-sized partitions with per-class population distribution in proportion to the total dataset. Partitions having the same overall population size or even the same number of samples per class may still have Non-IID distribution in the feature space. In heterogeneous computing environments, when devices have different computing capabilities, even-sized partitions across devices can lead to the straggler problem in distributed SGD. We develop a framework for distributed SGD in heterogeneous environments based on a novel data partitioning algorithm involving submodular optimization. Our data partitioning algorithm explicitly accounts for resource heterogeneity across workers while achieving similar class-level feature distribution and maintaining class balance. Based on this algorithm, we develop a distributed SGD framework that can accelerate existing SOTA distributed training algorithms by up to 32%.

1. Introduction

Stochastic gradient descent (SGD) is the skeleton of most state-of-the-art (SOTA) machine learning algorithms. Traditional SGD was designed to be run serially at a single worker. However, with the increasing size of deep learning models and dataset, too much time is required to perform training using a single machine [7]. Parallelism in training is inevitably necessary to deal with this magnitude of model, data, and compute requirements [22]. Distributed SGD parallelizes the training across multiple workers, through different types of parallelism (model [27], data [26, 17], pipeline) to speed up training. While parallelism can achieve high training throughput, guaranteeing the stability and convergence of SGD is challenging in distributed training [18, 26]. It is a complex interplay of machine learning hyperparameters and dataset distribution that contributes to convergence of distributed machine learning training. Most of the SOTA decentralized distributed machine learning frameworks [17, 26, 1, 11, 18] randomly partition the original dataset into subsets and assign them to different workers. Their convergence analysis simply assumes the subsets in each worker are independent and identically distributed (IID). However, random partitioning cannot guarantee IID at the feature level and the Non-IID issue still exists.

Submodular optimization, associated with a rich family of submodular functions that can measure the diversity of a given subset, has been used in machine learning in the past decade. Submodular optimization is well known for its strong capability to select a representative subset. Earlier work have tried applying submodular optimization to real-world applications including speech recognition, active learning, and computer vision [10, 13, 21, 12, 20, 32, 31, 29, 34]. To solve the Non-IID issue in distributed machine learning, [30] tries to use submodular optimization to partition the original dataset into IID subsets. General partition on any dataset ahead of training won't give additional computational costs and will lead to faster convergence. However, the submodular partition algorithm GreedMax [30] will lead to different sized subsets after partition. Different sized subsets will lead to straggler problem (the delays in waiting for the learners with large batch size) due to synchronization step in most distributed SGD. Straggler problem" refers to delays caused by the slowest learners in distributed machine learning. This problem is caused by the "synchronization barrier," which is the process of coordinating updates to the model parameters across multiple machines or devices, typically done after each iteration of the training process [24, 18]. Although some recent papers try to explore the possibility of using submodular optimization to partition dataset under certain constraints [28], it cannot be applied to distributed machine learning directly due to non-uniformly sized and imbalanced classes across partitions. To solve these problems, we propose RCD-SGD, a resource-constrained distributed SGD. Our main contribu-

[*]Corresponding author.

tions are:

1. We propose a novel algorithm to partition data for distributed SGD in heterogeneous environments. The proposed algorithm partitions the dataset across workers in proportion to their computational capabilities while achieving similarity in class-level feature distribution and maintaining class balance. (Class-level feature distribution refers to the pattern of specific feature values within different categories or classes in a dataset, providing insights into the data's characteristics and feature importance. Maintaining class data sample number balance means ensuring that the number of data samples in each class or category of a dataset is equal.) Our algorithm reduces the computational complexity of earlier partitioning algorithms by a factor proportional to the number of classes in the dataset [28].

2. By partitioning the original dataset into subsets with the same number of data points, the label-balanced greedy partition algorithm addresses the straggler problem in distributed synchronous SGD.

3. We evaluate the RCD-SGD algorithm using two different submodular functions and two SOTA-distributed SGD algorithms. By achieving IID partitioning of the dataset our algorithm achieves faster convergence than the SOTA baseline. With approximately local IID subsets, we reduce the communication frequency of distributed SGD and achieve up to 32% speedup in wall-clock time when compared with the SOTA algorithms [23]. The final model also achieves slightly better loss and improves the final accuracy by 1.1% ("Approximately local IID subsets" means subsets of data partitioned from original dataset are expected to be similar and follow similar distribution properties).

4. The proposed resource-constrained distributed SGD is a general algorithm that is extendable to most distributed SGD SOTA algorithms. Using any algorithm as a baseline, resource-constrained distributed SGD can achieve faster convergence and shorter wall-clock training time.

2. Preliminaries and Related Work

2.1. Problem Formulation

Distributed Stochastic Gradient Descent (SGD) aims to accelerate the training process by parallelizing it across multiple worker nodes, employing various types of parallelism such as model, data, and pipeline parallelism. Let us consider a network with N worker nodes implementing distributed SGD. The model parameters are denoted by x, where $x \in \mathbb{R}^d$. Each worker node i has access only to its own local training data, distributed as D_i. The objective of distributed SGD is to train a model by minimizing the objective function $L(x)$ using N worker nodes. The challenge of distributed SGD can be formulated as follows:

$$\min_{x \in \mathbb{R}^d} L(x) = \min_{x \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} E_{s \sim D_i}[l_i(x; s)], \tag{1}$$

where $l(x)$ is the loss function defined by the learning model, and $E_{s \sim D_i}[l_i(x; s)]$ represents the local objective function at the i-th worker.

2.2. Related Work

Synchronous **centralized SGD** with a parameter server is a form of parallel mini-batch SGD. In this approach, workers compute stochastic gradients of their local objectives in parallel and use the averaged gradient to update model parameters after each iteration. The update rule is as follows:

$$x_{k+1} = x_k - \eta \left[\frac{1}{N} \sum_{i=1}^{N} g_i(x_k; \xi_i) \right], \tag{2}$$

where x_k represents the model parameters of server at the k-th iteration, $g_i(x_k; \xi_i)$ denotes the gradient at learner i in the k-th iteration, η is the learning rate, and ξ_i represents randomly sampled mini-batches from the local data distribution. g_i is the gradient estimate at the ith learner in the kth iteration calculated using x_k. The convergence analysis of this approach has been presented in previous work [6, 2].

Centralized SGD with a parameter server [4, 16, 3, 8, 5] encounters the communication bottleneck problem when the framework has either a large number of workers or low network bandwidth [19, 18, 17, 25]. To overcome this bottleneck, **decentralized SGD** frameworks have been proposed. In the D-PSGD (Decentralized Parallel Stochastic Gradient Descent) approach [30], workers perform one local update and average their models only with neighboring workers. The update rule for D-PSGD is as follows:

$$x_{k+1,i} = \sum_{j=1}^{N} W_{ij} \left[x_{k,j} - \eta g_j(x_{k,j}; \xi_{k,j}) \right], \tag{3}$$

where $x_{k,j}$ denotes the model parameters of worker j at iteration k, $\xi_{k,j}$ represents a batch sampled from the local data distribution of worker j at iteration k, $W \in \mathbb{R}^{N \times N}$, and W_{ij} is the (i, j)-th element of the mixing matrix W, which indicates the adjacency between nodes i and j. W_{ij} is non-zero if and only if nodes i and j are connected.

The convergence of distributed training is tied to the **data distribution** across workers. For efficient distributed training, the data partitions at different workers should have similar data distribution. A simple random partitioning in equal-sized partitions may not preserve class-level distribution across partitions. Class-level random partitioning ensures that the number of samples of different classes in any partition is in the same proportion as in the original dataset. However, this may still not guarantee that the feature distribution of a class is similar across different partitions. Ensuring similar feature distributions across different partitions accelerates convergence because it promotes consistent learning, stable gradients, efficient data usage, and better generalization, allowing the model to learn faster and converge more quickly to an optimal solution.

Some recent works involve the use of submodular functions for data partitioning for efficient distributed machine learning. In [30] using a greedy algorithm involving the use of submodular functions, the dataset was partitioned into subsets with IID features. However, their greedy algorithm has three major problems:

1. **Non-uniform sized partitions** The algorithm can lead to different sized partitions. This will lead to a straggler problem in any distributed training algorithms with a synchronization barrier.

2. **Class imbalance across partitions** The algorithm can lead to a different number of samples per class across the partitions. This imbalance can lead to lower performance of the aggregated model. The reason why this can happen is that if there are two classes with an overlap in feature space (e.g., Figure 1), then since the greedy algorithm is only maintaining similar feature distribution across partitions, we can have uneven number of samples from these two classes in different partitions.

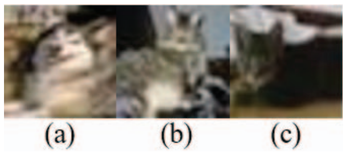


Figure 1. The similarity between dog (a) and cat (b) is higher than similarity between cat (b) and cat (c). Similarity is calculated using cosine similarity and Gaussian kernel with L2 distance. More details about similarity calculation can be found in Section 3.

3. **High computational complexity**: With n samples in the dataset, the greedy algorithm requires $O(n^2)$ computations which can be prohibitive for large datasets.

The approach was generalized in [28] to perform constrained submodular partitioning. The experiment demonstrated that the average performance of models trained individually on different subsets is better than random partitioning. However, this was never demonstrated in distributed training setting with communication between workers. In addition, it cannot handle heterogeneous environments, the issues of class imbalance and high complexity remain.

Algorithm 1 Ratio-Constrained submodular partitioning for distributed SGD in heterogeneous environment

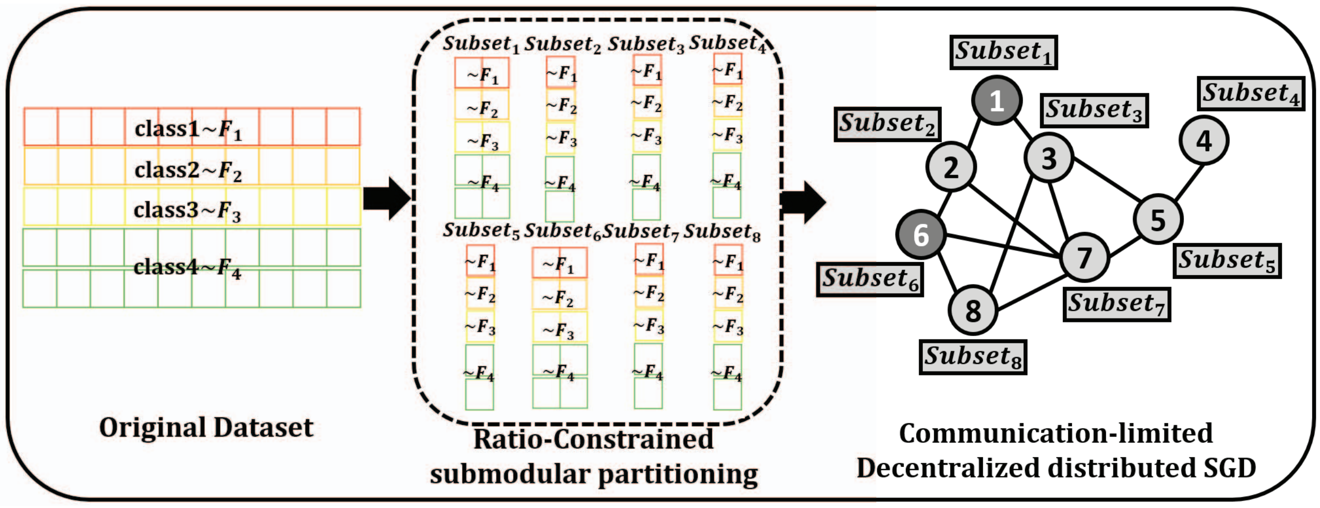
```
1: Initialization: submodular function $f$, ground set $V$ , number of blocks $N$, number of classes $L$
2: Set the ratio of constraint according to the computational performance of different workers $r_1, r_2, \dots, r_N$

3: Let $A_1 = A_2 = ... = A_N = \emptyset$
4: Split the ground set $V$ into $L$ sets $V_1, V_2, ..., V_L$ according to class labels.
5: for $V_l = V_1, V_2, ..., V_L$ do
6:     Constraint $C_{l,j} = \frac{|V_l| r_j}{\sum_{j=1}^{N} r_j}, j \in \{1, \dots, N\}$
7:     Let $A_1^l = A_2^l = ... = A_N^l = \emptyset, J = [N], R = V_l$
8:     while $R \neq \emptyset$ and $J \neq \emptyset$ do
9:         $j^* \in$ argmin $_{j \in J} f(A_j^l)$
10:        if $\exists v \in R$ s.t. $A_{j^*}^n \cup \{v\} \in C_{l,j^*}$ then
11:            $v^* :=$ GreedyStep($R, C_{l,j^*}, A_{j^*}^l$)
12:            $A_{j*}^l := A_{j^*}^l \cup \{v^*\}$, $R := R \backslash \{v^*\}$
13:        else
14:            $J = J \backslash j^*$
15:        end if
16:    end while
17:    Joint subset: $A_1 = A_1 \cup A_1^l, ..., A_N = A_N \cup A_N^l$
18: end for
19: Output:$(A_1, A_2, A_3, ..., A_N)$
```

3. Proposed Method

In this paper, we propose a novel algorithm called RCD-SGD. RCD-SGD is a meta-scheme algorithm that can be applied to most decentralized SGD algorithms. RCD-SGD includes two parts: data partition and distributed machine learning. Both of them are described in Algorithm 1 and Algorithm 2.

3.1. Resource-Constrained submodular partitioning

Algorithm 1 is our proposed algorithm for ratio-constrained submodular partitioning. The similarities between each data point are computed using vectors v, v'. The vectors, which present the features of the encoded image, are extracted as the output of the auto-encoder's bottleneck. In Figure 3, we present the process to calculate similarity between two images. More details of the pre-trained auto-encoder model can be found in Section 4. In heterogeneous environments, there can be a wide gap in the compute capabilities of different workers. To solve this problem, we set different constraints in the submodular optimization algorithm according to compute performance of workers to en-



class1~F_1
class2~F_2
class3~F_3
class4~F_4
Original Dataset
Subset$_1$ Subset$_2$ Subset$_3$ Subset$_4$
Subset$_5$ Subset$_6$ Subset$_7$ Subset$_8$
Ratio-Constrained submodular partitioning
Communication-limited Decentralized distributed SGD

Figure 2. RCD-SGD performs ratio-constrained partitioning of datasets, where the number of samples per class is proportional to the compute capabilities of the workers while maintaining per class feature distribution across partitions. When training multiple epochs of local SGD at workers leads to faster convergence with reduced communication overhead.

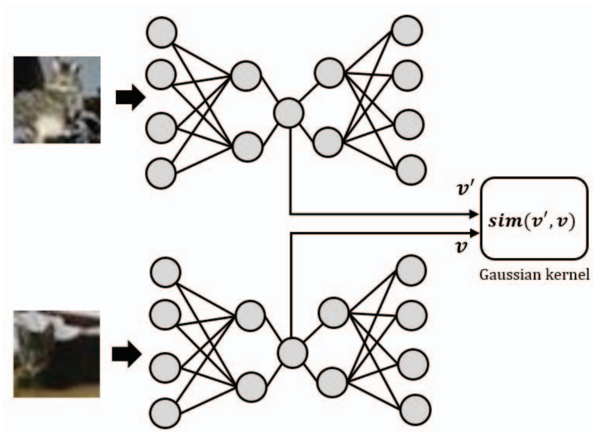


Figure 3. Features of images are extracted from the bottleneck of pre-trained auto-encoder. We use a Gaussian Kernel to measure the similarity.

sure that the number of samples in a subset is proportional to the performance of the worker training with that subset. (The compute performance of different GPU workers can vary based on factors like GPU model, architecture, and the number of cores, with high-end GPUs offering better performance compared to mid-range ones.) Besides, instead of doing partitioning on the whole dataset, we do class-level partitioning of the dataset. By using consistent constraint across all classes for a certain worker, we can get balanced subclasses. The submodular optimization greedy algorithm requires $O(n^2)$ computations, whereas our algorithm will reduce the computation complexity from $O(n^2)$ to $O(\frac{n^2}{L})$ in L classes dataset thereby achieving $100\times$ speedup when partitioning CIFAR-100.

Algorithm 2 Label-balanced submodular partitioning

```
1: **Initialization:** submodular function $f$, ground set $V$, number of blocks $N$, number of classes $L$
2: Let $A_1 = A_2 = ... = A_N = \emptyset$
3: Split the ground set $V$ into $L$ sets $V_1, V_2, ..., V_L$ according to class labels.
4: for $V_l = V_1, V_2, ..., V_L$ do
5:     Constraint $C_l = \frac{|V_l|}{N}$
6:     Let $A_1^l = A_2^l = ... = A_N^l = \emptyset, J = [N], R = V_l$
7:     while $R \neq \emptyset$ and $J \neq \emptyset$ do
8:         $j^* \in \text{argmin}_{j \in J} f(A_j^l)$
9:         if $\exists v \in R$ s.t. $A_{j^*}^l \cup \{v\} \in C_{l,j^*}$ then
10:            $v^* :=$ GreedyStep$(R, C_{l,j^*}, A_{j^*}^l)$
11:            $A_{j^*}^l := A_{j^*}^l \cup \{v^*\}, R := R \backslash \{v^*\}$
12:        else
13:            $J = J \backslash j^*$
14:        end if
15:    end while
16:    Joint subset: $A_1 = A_1 \cup A_1^l, ..., A_N = A_N \cup A_N^l$
17: end for
18: Output:$(A_1, A_2, A_3, ..., A_N)$
```

We formulate the general resources-constrained submodular partitioning optimization task as follows: partition datasets into N groups such that each group contains similar, sufficiently, and approximately-IID strong prediction power. The dataset of class l: $\mathbf{V}_l = \{\mathbf{X}, \boldsymbol{y}\}$ is given, where $\mathbf{X} \in \mathbf{R}^{M \times d}$ and $\boldsymbol{y} \in \{0,1\}^M$. Set the constraint according to the computational capability of different workers: $C_{l,1}, C_{l,2}, ..., C_{l,N}$ so that $\sum_{i=1}^{N} C_{l,i} = M$. Define a partition of V_l as $\{A_1^l, A_2^l, \ldots, A_N^l\}$ where $A_n^l \in V_l$ for

$n = 1, 2, \ldots, N$ such that

$$A_j^l \cap A_i^l = \emptyset \quad \forall i, j \in \{1, 2, \ldots, N\}, i \neq j$$

$$\bigcup_{n=1}^{N} A_n^l = V_l$$

The proposed Resource-Constrained submodular partitioning algorithm (Algorithm 1) addresses the partition task. The $GreedyStep(R, C_l, A_{j*}^l)$ in Algorithm 1 utilizes the submodular function to measure the value. The GreedyStep is defined as:

$$v^* = \arg \max_{i \in V_l \setminus \hat{V}_l} f\left(A_j^l \cup \{v^i\}\right)$$

where $\bigcup_{n=1}^{N} A_n^l = \hat{V}_l \subset V_l$. After partition the subsets will finally achieve: $\bigcup_{n=1}^{N} A_n^l = V_l$. We define a discrete set function $f \colon \mathbf{V} \to \mathbf{R}$ as a submodular function if

$$f\left(\hat{A}_j^l \cup \{v^*\}\right) - f\left(\hat{A}_j^l\right) \geq f\left(A_j^l \cup \{v^*\}\right) - f\left(A_j^l\right)$$

where $\hat{A}_j^l \subset A_j^l, \forall j \in \{V_l \setminus \cup_{n=1}^{N} A_j^l\}$. A submodular function f must be monotone non-decreasing, which is the basic requirment:

$$f\left(A_j^l \cup \{v^*\}\right) - f\left(A_j^l\right) \geq 0$$

When the performance of the workers are similar, the Algorithm 1 can also be simplified to label-balanced submodular partitioning, which is presented in Algorithm 2.

3.2. Distributed SGD

We next propose a synchronous distributed training algorithm using our proposed partitioning algorithm. The main steps are depicted in Figure 2. Partitioning the data as in Algorithm 1 ensures that each worker gets total (and per class) samples proportional to their processing speed. This will achieve approximate IID partitioning across workers, both at the feature level and label level. After partitioning the data, we perform synchronous distributed training with several rounds of local SGD before each synchronization step. This will not be detrimental to our convergence as we are ensuring IID partitions.

Our training algorithm is a modification of Distributed-Parallel SGD (D-PSGD) [30] where the data partitions can be Non-IID due to random partitioning. The Non-IIDness can be in feature and/or label space. Non-IIDness in feature refers that features or labels of the data used in a machine learning model are not independently and identically distributed. Thus convergence can be poor with D-PSGD, especially for datasets with an overlap in feature space across classes. Further D-PSGD has a communication barrier after every step of local training and hence takes a longer time to converge. During training, the communication between workers happens over a computational graph where the nodes represent the workers and the edges denote the connection between the workers. Thus ith, jth component of W, W_{ij}, is non-zero if and only if node i and node j are connected. The workers only communicate after F iterations of local SGD. Since the batch size and the dataset shard per worker is proportional to its processing speed, our partitioning ensures that all the workers take approximately same amount of time to finish F iterations locally thereby reducing the straggler problem with synchronous SGD.

Algorithm 3 Decentralized distributed SGD algorithm

```
1: Initialization: initialize local models $\{x_0^i\}_{i=1}^N$ with the same initialization, the ratio of computational performance for different workers is $r_1, r_2, ..., r_N$, learning rate $\gamma$, batch size $B\frac{r_1}{r_i}$, communication frequency $F$, weight matrix $W$, and the total number of iterations $K$. Import the local subset $\xi_i$ from subsets $\{A_1, A_2, ..., A_N\}$, which is partitioned by Resource-Constrained submodular partitioning algorithm
2: while $k = 0, 1, 2, ...K - 1$ do
3:     Compute the local stochastic gradient $\nabla F_i(x_{k,i}; \xi_{k,i})$ on all nodes
4:     if $k\%F == 0$ then
5:         Compute the neighborhood weighted average by fetching neighbor models: $\hat{x}_{k,i} = \sum_{j=1}^{N} W_{ij} x_{k,j}^b$
6:     else
7:         $\hat{x}_{k,i} = x_{k,i}$
8:     end if
9:     Update local model: $x_{k+1,i} = \hat{x}_{k,i} - \gamma \nabla F_i(x_{k,i}; \xi_{k,i})$
10: end while
11: Output: Average of all workers $\frac{1}{N} \sum_{i=1}^{N} x_{K-1,i}$
```

4. Experiments

In Fig. 4 and Fig. 5, we report the results of RCD-SGD using D-PSGD and MATCHA as baselines. Performance of RCD-SGD using facility location and graph cut as submodular functions are included. Results are compared with baseline D-PSGD and MATCHA. MATCHA, like D-PSGD, begins with a predetermined communication network topology. However, unlike D-PSGD, MATCHA allows the system designer to define a flexible communication budget c_b, representing the average communication frequency over network links. The dynamic construction of the weight matrices $W^{(k)}$ is dependent on the chosen budget c_b. When c_b = 1, MATCHA is equivalent to the vanilla D-PSGD algorithm. If c_b is less than 1, MATCHA effectively decreases the communication frequency over each link, taking into consideration the link's significance

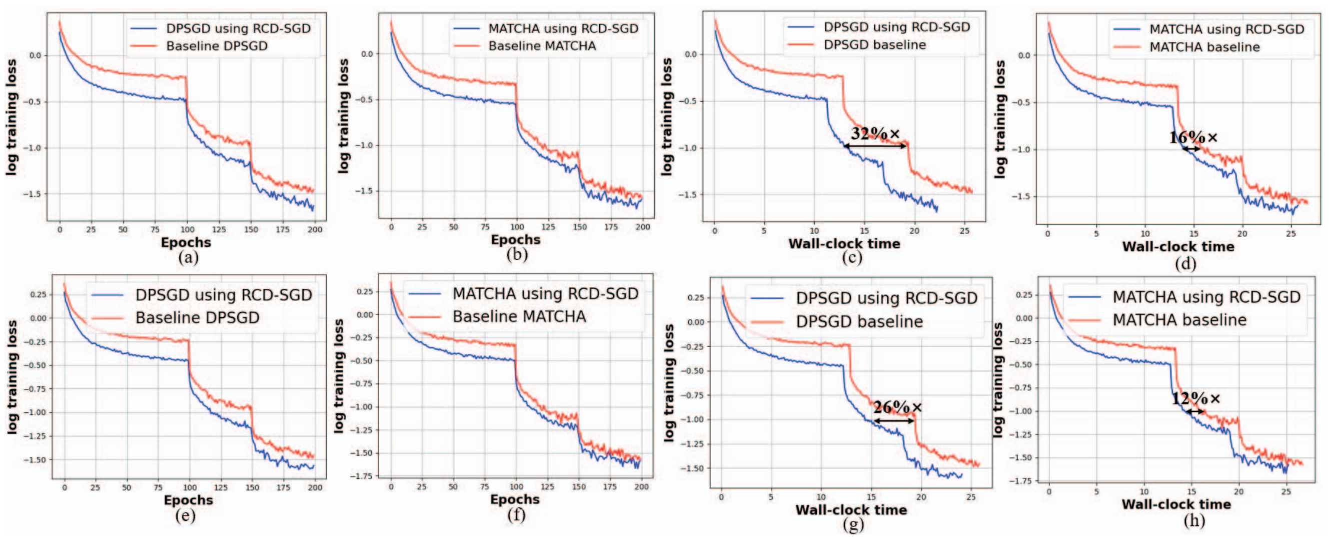


Figure 4. RCD-SGD use facility location in *greedystep*: (a, b, c, d); use graph cut *greedystep*: (e, f, g, h). Results were obtained on CIFAR-10 data set using ResNet-50. (a), (b) and (e), (f) show convergence with number of epochs while (c), (d) and (g), (h) show convergence with wall clock time.

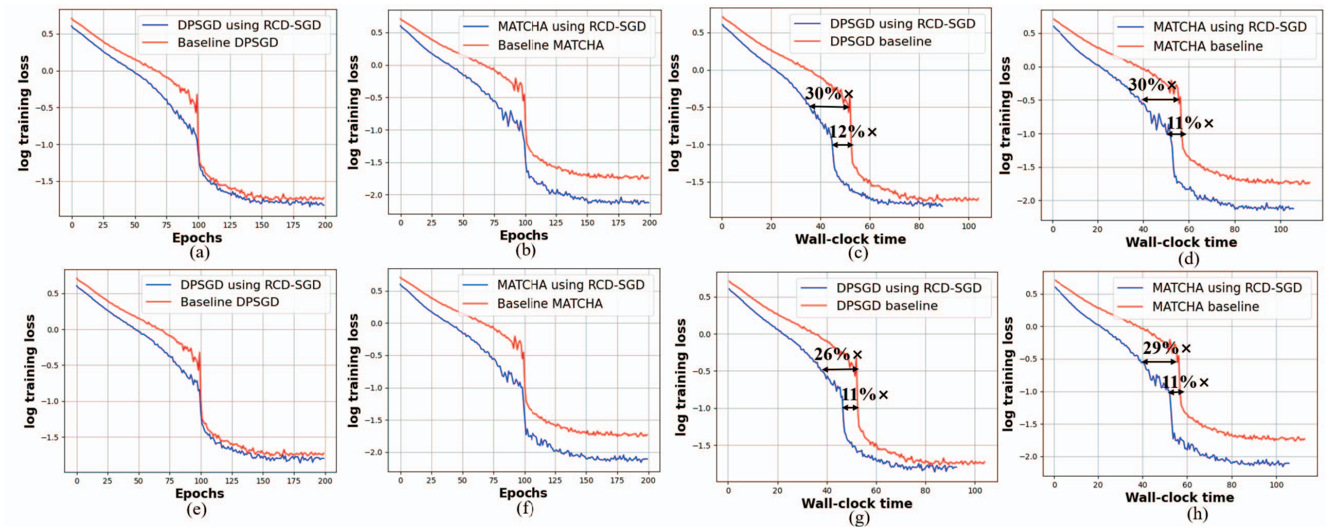


Figure 5. RCD-SGD use facility location in *greedystep*: (a, b, c, d); use graph cut *greedystep*: (e, f, g, h). Results were obtained on CIFAR-100 data set using WideResNet. (a), (b) and (e), (f) show convergence with number of epochs while (c), (d) and (g), (h) show convergence with wall clock time.

in maintaining the overall graph connectivity. Furthermore, MATCHA assigns probabilities to worker connections, enabling their activation in certain iterations. Experiments use the following setting:

1. **Models, dataset, and compared algorithms:** The performance of all algorithms is evaluated on image classification task using CIFAR-10 and CIFAR-100 ($|V|$ = 500000) [14]. In our experiments, we employ ResNet-50 [9] and Wide ResNet models [33]. We implement RCD-SGD as modification of D-PSGD and MATCHA with a communication budget $c_b = 0.5$.

2. **Submodular functions:** We use facility location, i.e., $f(A_n^l) = \sum_{v \in V_l} max_{v' \in A_n^l} sim(v, v')$ and Graph Cut, i.e., $f(A_n^l) = \sum_{v \in V_l \setminus A_n^l} \sum_{v' \in A_n^l} sim(v, v')$ in $Greedystep$ (algorithm 1) separately to run the experiments. $sim(v, v')$ is the similarity between (v, v'). We use a Gaussian kernel with L2 distance to measure the similarity. We set Gaussian Kernel

as $\sigma = \sum_{v,v' \in V} ||v - v'||_2/n^2$, where σ is the bandwidth of the kernel. Similarities between each data point are computed by vectors v, v'. In CIFAR-10 and CIFAR-100, the vectors are obtained from the bottleneck layer's outputs of a deep auto-encoder model. The partition step is not included in the comparison of wall-clock times, since the partition only needs to be done once to generate subsets.

3. **Implementations:** All algorithms are trained for a sufficiently long time until convergence or onset of overfitting. The learning rate is fine-tuned for the D-PSGD baseline and then used for all other algorithms. We set the initial learning rate as 0.8 and it decays by 10 after 100 and 150 epochs. The batch size per worker node is 64. RCD-SGD uses $F = 2$ and reduces the communication frequency to 50%. The auto-encoder is trained using ReLU non-linearity and batch normalization. The network is trained in PyTorch using the procedure described in [15, 28]. The auto-encoder is pre-trained using image reconstruction task. The neural network architecture of auto-encoder can be found in table 1. The auto-encoder utilizes ADAM as the optimization method. The initial learning rate of 5e-3, with a weight decay of 5e-4 and a minibatch size of 100.

Group	Block Type (kernel size, stride, channels)	Blocks
conv1	$[3 \times 3], 2, 64$	1
conv1 (residual)	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix}, 1, 64$	2
conv2	$[3 \times 3], 2, 16$	1
conv2 (residual)	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix}, 1, 16$	2
conv3	$[3 \times 3], 2, 8$	1
conv3 (residual)	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix}, 1, 8$	2
conv4	$[3 \times 3], 1, 4$	1
conv4 (residual)	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix}, 1, 4$	1
deconv4 (residual)	$[3 \times 3], 1, 4$	1
deconv3	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix}, 1, 8$	1
deconv3 (residual)	$[3 \times 3], 1, 9$	2
deconv2	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix}, 2, 16$	1
deconv2 (residual)	$[3 \times 3], 2, 16$	2
deconv1	$[3 \times 3], 2, 64$	1
deconv1 (residual)	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix}, 2, 64$	2
deconv0	$[3 \times 3], 2, 3$	1

Table 1. Neural network architecture of the Auto-encoder.

4. **Machines and clusters:** All the implementations are compiled with PyTorch and OpenMPI within mpi4py and rtx8000 GPUs as workers. We conduct experiments on a HPC cluster with 100Gbit/s Infini-band network.

We conducted an analysis of the results obtained with RCD-SGD. In both Figure 4 and Figure 5, we observed that RCD-SGD, utilizing either facility location or graph cut as submodular functions, consistently outperformed the baseline methods. Notably, improvements were evident in both the convergence rate over epochs and the convergence speed with respect to wall-clock time.

When training the results from CIFAR-10 and ResNet-50, RCD-SGD achieved a significant time saving of up to 30% (Fig.4 c). Similarly, when utilizing CIFAR-100 and WideResNet, substantial performance improvements were observed, with a time saving of up to 12% (Fig.5 c), measured until the log of training loss reached -1.0.

By analyzing the results, it becomes evident that RCD-SGD not only accelerates the convergence speed of both baselines but also demonstrates a remarkable improvement in convergence. The use of IID subsets aids in achieving faster convergence for RCD-SGD, as the localized training process enables more efficient model updates and fosters enhanced learning across the dataset. This advantage can be attributed to the network's ability to focus on specific patterns and features within each subset, leading to better training dynamics and overall performance.

An interesting aspect of RCD-SGD is that it maintains convergence even after the baselines have already converged, thanks to the independent and identically distributed (IID) partitioned subsets. At the 100th epoch, before the learning rate decay, RCD-SGD exhibited up to 30% wall-clock time saving (Fig. 5 c) when measured until the log of training loss reached -0.05.

Furthermore, the final loss and test accuracy were consistently improved when employing RCD-SGD, as demonstrated in Table 2. It is important to note that all the results presented in Table 2 are the average of ten experiments, ensuring robustness and reliability.

Dataset	Model	Algorithms	Accuracy
CIFAR-100	**WideResNet**	D-PSGD	0.718
		D-PSGD based RCD-SGD	**0.752**
		MATCHA	0.755
		MATCHA based RCD-SGD	**0.762**
CIFAR-10	**ResNet-50**	D-PSGD	0.925
		D-PSGD based RCD-SGD	**0.937**
		MATCHA	0.931
		MATCHA based RCD-SGD	**0.939**

Table 2. Test accuracy obtained with MATCHA and MATCHA-based AL-DSGD for ResNet-50 model trained on CIFAR-10 and WideResNet model trained on CIFAR-100.

5. Conclusion

Distributed training in heterogeneous clusters requires efficient data partitioning for faster convergence. RCD-SGD achieves IID partitioning with similar per-class feature distribution across workers having different compute capabilities. The training can be performed with increased epochs of local training leading to reduced synchronization overhead. We are exploring use of other submodular functions and the sensitivity of distributed SGD convergence on their choice.

References

[1] Michael Blot, David Picard, Matthieu Cord, and Nicolas Thome. Gossip training for deep learning. *arXiv preprint arXiv:1611.09726*, 2016.

[2] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[3] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R Ganger, Phillip B Gibbons, et al. Exploiting bounded staleness to speed up big data analytics. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 37–48, 2014.

[4] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[6] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1), 2012.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. *Advances In Neural Information Processing Systems*, 29, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Stefanie Jegelka and Jeff Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *CVPR 2011*, pages 1897–1904. IEEE, 2011.

[11] Peter H Jin, Qiaochu Yuan, Forrest Iandola, and Kurt Keutzer. How to scale distributed deep learning? *arXiv preprint arXiv:1611.04581*, 2016.

[12] Andreas Krause, H Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9(12), 2008.

[13] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(2), 2008.

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[15] Chandrashekhar Lavania and Jeff Bilmes. Auto-summarization: A step towards unsupervised learning of a submodular mixture. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 396–404. SIAM, 2019.

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

[17] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

[18] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[19] Mingrui Liu, Wei Zhang, Youssef Mroueh, Xiaodong Cui, Jarret Ross, Tianbao Yang, and Payel Das. A decentralized parallel algorithm for training generative adversarial nets. *Advances in Neural Information Processing Systems*, 33:11056–11070, 2020.

[20] Yuzong Liu, Kai Wei, Katrin Kirchhoff, Yisong Song, and Jeff Bilmes. Submodular feature selection for high-dimensional acoustic score spaces. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7184–7188. IEEE, 2013.

[21] Kiyohito Nagano, Yoshinobu Kawahara, and Satoru Iwata. Minimum average cost clustering. *Advances in Neural Information Processing Systems*, 23, 2010.

[22] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[23] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[24] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376. PMLR, 2017.

[25] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms. *The Journal of Machine Learning Research*, 22(1):9709–9758, 2021.

[26] Jianyu Wang, Anit Kumar Sahu, Zhouyi Yang, Gauri Joshi, and Soummya Kar. Matcha: Speeding up decentralized sgd via matching decomposition sampling. In *2019 Sixth Indian Control Conference (ICC)*, pages 299–300. IEEE, 2019.

[27] Jue Wang, Binhang Yuan, Luka Rimanic, Yongjun He, Tri Dao, Beidi Chen, Christopher Re, and Ce Zhang. Fine-tuning language models over slow networks using activation compression with guarantees. *arXiv preprint arXiv:2206.01299*, 2022.

[28] Shengjie Wang, Tianyi Zhou, Chandrashekhar Lavania, and Jeff A Bilmes. Constrained robust submodular partitioning. *Advances in Neural Information Processing Systems*, 34:2721–2732, 2021.

[29] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International conference on machine learning*, pages 1954–1963. PMLR, 2015.

[30] Kai Wei, Rishabh Iyer, Shengjie Wang, Wenruo Bai, and Jeff Bilmes. How to intelligently distribute training data to multiple compute nodes: Distributed machine learning via submodular partitioning. In *Neural Information Processing Society (NIPS) Workshop, Montreal, Canada*, 2015.

[31] Kai Wei, Yuzong Liu, Katrin Kirchhoff, Chris Bartels, and Jeff Bilmes. Submodular subset selection for large-scale speech training data. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3311–3315. IEEE, 2014.

[32] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 721–726, 2013.

[33] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[34] Jingjing Zheng, Zhuolin Jiang, Rama Chellappa, and Jonathon P Phillips. Submodular attribute selection for action recognition in video. *Advances in Neural Information Processing Systems*, 27, 2014.