

---

# GRAWA: Gradient-based Weighted Averaging for Distributed Training of Deep Learning Models

---

**Tolga Dimlioglu**  
New York University

**Anna Choromanska**  
New York University

## Abstract

We study distributed training of deep learning models in time-constrained environments. We propose a new algorithm that periodically pulls workers towards the center variable computed as a weighted average of workers, where the weights are inversely proportional to the gradient norms of the workers such that recovering the flat regions in the optimization landscape is prioritized. We develop two asynchronous variants of the proposed algorithm that we call Model-level and Layer-level Gradient-based Weighted Averaging (resp. MGRAWA and LGRAWA), which differ in terms of the weighting scheme that is either done with respect to the entire model or is applied layer-wise. On the theoretical front, we prove the convergence guarantee for the proposed approach in both convex and non-convex settings. We then experimentally demonstrate that our algorithms outperform the competitor methods by achieving faster convergence and recovering better quality and flatter local optima. We also carry out an ablation study to analyze the scalability of the proposed algorithms in more crowded distributed training environments. Finally, we report that our approach requires less frequent communication and fewer distributed updates compared to the state-of-the-art baselines.

the computations in distributed environments, such as data and model parallelization, both of which has its own advantages and use cases (Ben-Nun and Hoeffler, 2019). These parallelization techniques were used in many different machine learning tasks such as image recognition (Deng et al., 2009), machine translation (Yang et al., 2020), language understanding (Kenton and Toutanova, 2019), and more. This paper focuses on data parallelization in which models on different devices are trained with different portions from the dataset.

In data parallelization schemes, many models are simultaneously trained on different devices with different and non-overlapping portions of the data set, i.e., the data portions on different devices are mutually exclusive (Ben-Nun and Hoeffler, 2019). In this way, the effective number of data samples used for training is increased by the number of devices in the parallel computing environment. When training deep learning models, most of the time these devices are Graphical Processing Units (GPUs) or Neural Processing Units (NPU) (Verbraeken et al., 2020). During training, each worker (GPU) on each node runs Stochastic Gradient Descent (SGD) algorithm (Bottou, 1998) or its variant (Kingma and Ba, 2014; Ruder, 2016). The communication between workers and the distributed updates of model parameters done across the nodes assure that the knowledge embedded in the model parameters on different devices is properly shared among all workers.

## 1 Introduction

Training deep learning models in a distributed setting has become a necessity in scenarios involving large models and data sets due to efficiency and practicality. There are different ways of parallelizing

In this paper we introduce two new algorithms for distributed deep learning optimization in data parallel setting. We refer to them as Gradient-based Weighted Averaging (GRAWA) algorithms, Model-level GRAWA (MGRAWA) and Layer-level GRAWA (LGRAWA). By design, both of these variants address the curse of symmetry problem (Teng et al., 2019) which occurs when the model is stuck on the hill in the optimization landscape lying in-between workers that are trapped in the local minima around it, and

simultaneously prevent workers from getting stuck in narrow minima by pushing them towards flatter regions on the optimization loss surface. The proposed methods compare favorably to SOTA approaches in terms of convergence speed, generalization ability, and communication overhead, by considering the geometry of the landscape. Although there exist flat minima seeking deep learning optimizers in the literature, such as Entropy-SGD (Chaudhari et al., 2019), SAM (Foret et al., 2021), and LPF-SGD (Bisla et al., 2022), to the best of our knowledge, a flatness-aware update policy has never been introduced before in the context of distributed deep model training, more specifically for parameter-sharing methods. Our approach is therefore new. Our MGRAWA algorithm encourages flat minima by considering the gradient scores of the workers in the distributed training environment. Finally, the LGRAWA algorithm that we propose prioritizes model layers appropriately, when computing worker averages and model updates, to take advantage of the layers that learn faster than the others. This is motivated by the fact that neural networks are essentially compositions of functions and we seek a robust solution for each function in the composition. In this sense, our treatment of model layers and their effect on the algorithm’s distributed update is also novel.

This paper is organized as follows: Section 2 reviews the related literature, Section 3 formulates the problem, Section 4 motivates, derives the GRAWA algorithm and provides theoretical analysis, Section 5 introduces two GRAWA variants (MGRAWA and LGRAWA), Section 6 presents the experimental results and ablation study on scalability, and Section 7 concludes the paper. The supplement contains experiment details, additional results and proofs.

## 2 Related Work

In this section we review the literature devoted to distributed data parallel deep learning optimization and methods encouraging the recovery of flat minima in a deep learning optimization landscape.

**Distributed data parallel deep learning optimization** There exist only a few algorithms for distributed data parallel deep learning optimization. The DataParallel algorithm (Li et al., 2020), a gradient-sharing method, initiates all of the workers on different devices with the same model parameters that also remain the same throughout the training. The communication between the workers occurs after all the nodes complete processing their data batches.

During the communication, the calculated gradients on different nodes are averaged and used to update the model parameters. Since communication occurs after processing each batch, the method suffers from communication overhead. Furthermore, since the models are kept identical on all the machines throughout the training, the algorithm does not take advantage of having multiple workers performing individual exploration of the loss surface (Choromanska et al., 2015).

Moving on to the parameter sharing methods, in the EASGD algorithm (Zhang et al., 2015), each worker has distinctive model parameters during the course of training since they all run SGD independently on different data shards. During the communication, the algorithm applies an elastic force to each worker pulling it towards the center model. Here, the center model is calculated as a moving average of the model parameters of all the workers, both in space and time. The elastic force, therefore, establishes the link between different workers. The method suffers from the curse of symmetry phenomenon that occurs when the center model gets stuck at the maximum between several local minima that trap the workers. This can happen in the in the symmetric regions of the loss landscape. The workers are then pushed towards the maximum point during the distributed training phase which impedes their generalization abilities. The LSGD algorithm (Teng et al., 2019) addresses this problem by pushing all of the workers towards the leader worker, which is selected as the worker with the smallest training loss. Despite breaking the curse of symmetry and outperforming both DataParallel and EASGD in terms of convergence speed and accuracy, the LSGD algorithm can recover sub-optimal final model when the leader worker is trapped in a narrow region of the loss landscape, as we will show in the motivating example later in the paper.

### Flatness-aware deep learning optimization

It has been shown in the literature that model’s generalization capability in deep learning is firmly tied to the flatness of the loss valley that the model converged to (Jiang\* et al., 2020). In particular, when the model converges to flatter local minima, it yields a smaller generalization gap. There are many flatness measures introduced in the literature such as  $\epsilon$ -sharpness (Keskar et al., 2016), Shannon entropy of the output layer (Pereyra et al., 2017), PAC-Bayes measures (Jiang\* et al., 2020) and, measures based on the Hessian and its spectrum (Jiang\* et al., 2020; MacKay, 1991; Maddox et al., 2020b). However, they all are expensive to compute in a time-constrained environment.

There are also optimizers with mechanisms for seeking good quality (Kawaguchi, 2016), flat-minima. Entropy-SGD (Chaudhari et al., 2019) smooths out the energy landscape of the original loss function using local entropy, while increasing the processing time of a batch by  $L$  folds, where  $L$  is the number of Langevin iterations. The SAM optimizer (Foret et al., 2021) seeks parameters that lie in neighborhoods having uniformly low loss. Their optimization problem is formulated as a min-max problem. They approximate the inner maximization problem in order to derive a closed-form solution, whose computation requires double iteration for one batch. In LPF-SGD (Bisla et al., 2022), the gradient is smoothed by taking the average of the gradient vectors calculated after applying Gaussian noise on the current batch. This process multiplies the single batch processing time by the number of Markov Chain Monte-Carlo (MCMC) sampling iterations.

Notice that it is unclear how to adapt these schemes efficiently to the distributed setting since it would require distributing the computations of flatness measure across multiple workers. Also, since the current flat minima seeking optimizers require extra time to compute flatness measures, they are not well suited for data parallel training, where the goal is to reduce the training time. A flat minima seeking update has to be simple in order to avoid introducing computation overhead and is easy to parallelize. This is challenging and has never been investigated before to the best of our knowledge.

### 3 Problem Formulation

Consider the problem of minimizing a loss function  $F$  with respect to model parameters  $x$  over a large data set  $\mathcal{D}$ . In a parallel computing environment with  $M$  workers ( $x_1, x_2, \dots, x_M$ ), this optimization problem can be written as:

$$\min_x F(x; \mathcal{D}) = \min_{x_1, \dots, x_M} \sum_{m=1}^M \mathbb{E}_{\xi \sim \mathcal{D}_m} f(x_m; \xi) + \frac{\lambda}{2} \|x_m - x_c\|^2 \quad (1)$$

where  $\mathcal{D}$  is partitioned and sent to  $m$  workers and,  $\mathcal{D}_m$  is the sampling distribution of worker  $m$  exclusive to itself.  $x_c$  is the center variable (for example, an average of all the workers), and  $x_m$  stands for the parameters of model  $m$ . The equivalence of the optimization problems captured on the left-hand side and right-hand side of the Equation 1 is investigated in the literature and is commonly referred to as the global variable consensus

optimization problem (Boyd et al., 2011). The penalty term  $\lambda$  in front of the quadratic term in Equation 1 ensures the proximity of the model parameters of different workers by attracting them towards the center model.

## 4 GRAWA Algorithm

### 4.1 Motivating Example

Motivated by the connection between the generalization capability of deep learning models and the flatness of the local minima recovered by deep learning optimizers, we propose an algorithmic family called GRAWA that pushes the workers to flatter regions of the loss surface in the distributed optimization phase. Before formalizing the algorithm, we first simply illustrate its effectiveness in seeking flatter valleys on toy non-convex surface. Let us consider the Vincent function that has the following formulation for a 2-D input vector:  $f(x, y) = -\sin(10 \ln(x)) - \sin(10 \ln(y))$ .

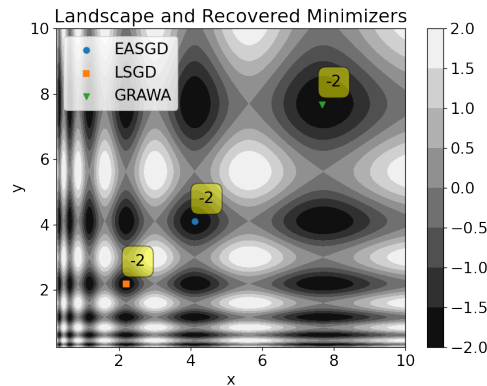


Figure 1: Contour plot of the loss landscape of the Vincent function and the final optima obtained by running LSGD, EASGD, and GRAWA optimizers.

The Vincent function is a non-convex function and it has several global minimizers that all attain a value of  $-2$ . But the flatness of the valleys in which the minimizers reside is different from each other. The contour map of the loss surface and the final solutions obtained by running the EASGD, LSGD, and GRAWA optimizers are provided in the figure below. As can be seen from Figure 1, all algorithms converge to a point that corresponds to the same loss value, but GRAWA recovers the valley in the loss landscape that is flatter than in the case of the other two schemes. The convergence trajectories of different distributed optimizers can be found in Appendix A.

### 4.2 Gradient Norm and Flatness

Our motivation behind finding a mechanism that encourages the recovery of flat loss valleys relies on simple properties of the gradient vector. If the Euclidean norm

of the gradient vector is high, it implies that the point at which the gradient is calculated is still at the steep slope of the loss valley. Otherwise, if the Euclidean norm of the gradient vector is low, the point at which the gradient is calculated is in the flat region of the loss landscape. As mentioned in the related work, flatness and generalization are mirror terms. Therefore, for completeness, we confirm empirically that for the workers residing in different valleys, the valleys with worse generalization capabilities would yield greater gradient norms, i.e., they would be sharper. We train 162 ResNet-18 models with different hyperparameters on the CIFAR-10 dataset in a parallel computing environment using the EASGD distributed training method. For each trained model, we calculate the norm of the gradient vector accumulated on the whole train set, the train error, and the test error. We indeed find out that there is a relation between the generalization gap (i.e., the difference between the test and train errors) and the gradient norm (and thus the flatness).

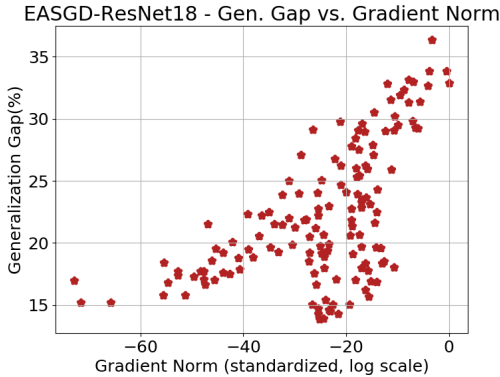


Figure 2: Plot of standardized gradient norms in log scale vs. the generalization gap (%).

As can be seen in Figure 2, when the generalization gap increases (the model’s generalization capability becomes worse), the gradient norm increases as well (the flatness deteriorates). Hence, we can use the gradient norm to obtain flatter valleys with good generalization capabilities. In a distributed setting, we propose to apply a weighted averaging scheme that favors workers with small gradient norms and penalizes the ones with large gradient norms when calculating the center model. Using gradient norm as opposed to common flatness metrics, such as  $\epsilon$ -sharpness (Keskar et al., 2016), Hessian-based measures (Maddox et al., 2020a), or recently introduced LPF measure (Bisla et al., 2022), is essential since these metrics are computationally expensive to periodically calculate in a time-constrained distributed training environment.

### 4.3 GRAWA: Gradient-based Weighted Averaging

We next explain the GRAWA algorithm, from which MGRAWA and LGRAWA schemes originate.

#### 4.3.1 GRAWA Method

Gradient-based Weighted Averaging (GRAWA) algorithm is an asynchronous distributed training algorithm that periodically applies a pulling force to all workers towards a consensus model created by taking weighted average among the workers. The weights in this weighted averaging scheme are determined based on the gradients. The vanilla GRAWA algorithm assigns to each worker  $m$  the weight  $\beta_m$  that is inversely proportional to the Euclidean norm of the flattened gradient vector  $A_m$ . More specifically, the weights are calculated as follows:

$$\beta_m \propto \frac{1}{\|A_m\|_2}, \quad \sum_{m=1}^M \beta_m = 1 \quad \rightarrow \quad \beta_m = \frac{\Theta}{\|\nabla f(x_m)\|}$$

$$\text{where } \Theta = \frac{\prod_{i=1}^M \|\nabla f(x_i)\|}{\sum_{i=1}^M \frac{\prod_{j=1}^M \|\nabla f(x_j)\|}{\|\nabla f(x_i)\|}} \quad (2)$$

Let  $x_m$  be the model parameters of the worker  $m$ . Then, the center model  $x_C$ , is calculated with a weighted average. Thus, in each distributed training phase, a new  $x_C$  is calculated and then all the workers are pushed towards it with a fixed  $\lambda$  coefficient where  $\lambda$  is  $\in [0, 1]$ . This coefficient is the strength of the pulling force applied to the workers. Overall, the calculation of the center variable and the distributed update are as follows:

$$x_C = \sum_{m=1}^M \beta_m x_m \quad \text{and} \quad x_m \leftarrow (1 - \lambda)x_m + \lambda x_C.$$

#### 4.3.2 Convergence Rate of GRAWA in Convex Case

We show that GRAWA algorithm matches the theoretical guarantees of both EASGD and LSGD and enjoys the same convergence rate as SGD. We first prove that center variable ( $x_C$ ) obtained with GRAWA algorithm attains a smaller loss value than the workers in the distributed training environment. We also note that the convergence analysis for MGRAWA would be similar to GRAWA with modifications to the constants in the derivation whereas, for LGRAWA, such analysis would be mathematically intractable since the weighted averaging is done on the layer level.

**Theorem 1.** *Let  $x_C = \sum_{i=1}^M \beta_i x_i$  and  $\beta_i$ ’s are calculated as in equation 8. For an  $L$ -Lipschitz differentiable,*

real-valued, continuous convex function  $f$  with minimizer  $x^*$  that also satisfies  $\|\nabla f(x)\| \geq \mu(f(x) - f(x^*))$  and is also bounded by a cone which has a slope  $k$  and a tip at  $x^*$ ; the GRAWA center variable holds the following property:  $f(x_C) \leq f(x_i)$  for all  $i \in 1, 2, \dots, M$  when  $k\mu \geq L\sqrt{M}$ .

We consider a general update rule where the distributed update is applied at every iteration. Thus, the overall update rule for each iteration becomes  $x_i^{t+1} = x_i^t - \eta(\nabla f(x_i^t) + \lambda(x_i^t - x_C))$ . Note that this update rule is applicable to each worker  $i$ . For simplicity, we drop the worker indices for the following theorem that is derived from (Teng et al., 2019).

**Theorem 2.** *Let  $f$  be a function that satisfies the conditions of 1. Let  $\tilde{g}(x)$  be an unbiased estimator for  $\nabla f(x)$  with  $\text{Var}(\tilde{g}(x)) \leq \sigma^2 + \nu\|\nabla f(x)\|^2$ , and let  $x_C$  be the center variable obtained with the GRAWA algorithm. Suppose that  $\eta, \lambda$  satisfy  $\eta \leq (2L(\nu + 1))^{-1}$  and  $\eta\lambda \leq \frac{m}{2L}$ ,  $\eta\sqrt{\lambda} \leq \frac{\sqrt{m}}{\sqrt{2L}}$ . Then the GRAWA step satisfies:*

$$\begin{aligned} \mathbb{E}[f(x^{t+1})] - f(x^*) &\leq (1 - m\eta)(f(x^t) - f(x^*)) \\ &\quad - \eta\lambda(f(x^t) - f(x_C)) + \frac{\eta^2 L}{2}\sigma^2 \end{aligned}$$

The presence of the new term  $f(x_C)$  due to the GRAWA update rule increases the speed of the convergence since  $f(x_C) \leq f(x)$  as given in 1. Then,  $\limsup_{t \rightarrow \infty} \mathbb{E}[f(x^{t+1})] - f(x^*) \leq \eta \frac{L}{2m} \sigma^2$ . If  $\eta$  decreases at rate  $\eta = O(\frac{1}{t})$ , then  $\mathbb{E}[f(x^{t+1})] - f(x^*) \leq O(\frac{1}{t})$ .

### 4.3.3 Convergence Rate of GRAWA in Non-Convex Case

Here we state the convergence guarantee for GRAWA in the non-convex optimization setting.

**Theorem 3.** *Let  $F$  be an  $L$ -smooth function and let us have the following bounds for the stochastic gradients that satisfy the following variance bounds:  $\mathbb{E}_{\xi \sim \mathcal{D}_m} \|\nabla f(x, \xi) - \nabla f_m(x_m)\|^2 \leq \sigma^2$ ,  $\mathbb{E}_{m \sim U[M]} \|\nabla f_m(x_m) - \nabla F(x_m)\|^2 \leq \zeta^2$ . Also, assume that we have the following bound for local workers and center variable:  $\mathbb{E}\|x_m^t - x_c^t\|^2 \leq \rho^2$  at any time or iteration  $t$ . We show that GRAWA algorithm and its variants satisfy the following convergence rate:*

$$\begin{aligned} \frac{1}{MN} \sum_{m=1}^M \sum_{t=1}^N \mathbb{E} \|\nabla F(x_m^t)\|^2 &\leq \\ O\left(\frac{F(x^0) - F(x^*)}{MN\eta} + (2L\zeta^2 + L\sigma^2 + \lambda^2\rho^2L)\eta\right), \end{aligned}$$

which characterizes the square norm of the gradients averaged with respect to all  $M$  workers and the number of samples.

## 5 GRAWA Algorithm Variants

In this paper we also develop two extensions of the GRAWA algorithm which are more suitable for the training of deep learning models: Model-level GRAWA (MGRWA) and Layer-level (LGRWA). To be more specific, the mechanism of GRAWA that relies on simply vectorizing all the gradients and then taking their norm is less representative than individually calculating the layer gradient norms and summing them due to the triangle inequality. Furthermore, in these variants, we are utilizing the stochastic gradients for norm calculation. The details of MGRWA and LGRWA are provided in the following subsections.

### 5.1 MGRWA Algorithm

In the MGRWA scheme, similar to the GRAWA method, the center model is obtained by taking the weighted average of the model parameters of all the workers. First, before any updates are applied, the gradients are accumulated in each worker using a randomly selected subset from the train set. Unlike the vanilla GRAWA, these accumulated gradients in all of the layers are summed up for each worker. This quantification better represents the gradient norm of each layer individually. Let  $g_k^n$  be the stochastic gradient calculated in the  $k^{\text{th}}$  layer of the model with respect to the  $n^{\text{th}}$  sample. Also, let  $N^1$  be the size of the subset of the train set, and  $K$  be the number of layers in the network. Then, the accumulated gradient  $A_m$  for each worker  $m$  is calculated as follows:

$$G_k = \sum_n^n g_k^n \quad A_m = \sum_k^K \|G_k\|_F, \quad (3)$$

where  $\|\cdot\|_F$  is the Frobenius norm. Let us have  $M$  workers in the distributed training and let  $A_1, A_2, \dots, A_M$  be the sum of accumulated gradients of all layers for worker 1 to worker  $M$  respectively. Then, the weight calculated for worker  $m$ ,  $\beta_m$ , is inversely proportional to its accumulated gradient  $A_m$ . Furthermore, we require that the sum of these weights equals 1.

$$\beta_m \propto \frac{1}{A_m} \quad \text{and} \quad \sum_{m=1}^M \beta_m = 1 \quad (4)$$

The expression to calculate the center model is the same as in GRAWA. Pseudo-code of the MGRWA algorithm is provided in the Algorithm 1 (proximity search step from the pseudo-code will be explained later).

### 5.2 LGRWA Algorithm

In the LGRWA algorithm, the weighted averaging scheme is also used, but it is applied independently

<sup>1</sup>In practice, we choose  $N$  equal to the batch size.

---

**Algorithm 1** MGRAWA
 

---

**Input:** Pulling force  $\lambda$ , communication period  $\tau$ , learning rate  $\eta$ , proximity search strength  $\mu$ , loss function  $f$

**Initialize** workers  $x_1, x_2, \dots, x_M, x_C$  from the same random model, worker-exclusive data shards  $\Psi_1, \Psi_2, \dots, \Psi_M$  and iteration counters for workers  $t_1 = t_2 = \dots = t_M = 0$

At each worker  $m$  **do**

**while** not converged **do**

    Draw a random batch  $\xi_m \in \Psi_m$

$x_m \leftarrow x_m - \eta \nabla f(x_m; \xi_m)$

$x_m \leftarrow (1 - \frac{\mu}{\tau}) x_m + \frac{\mu}{\tau} x_C$  (Prox.)

$t_m \leftarrow t_m + 1$

**if**  $M\tau$  divides  $\sum_{m=1}^M t_m$  **then**

        Draw a random batch from  $\mathcal{D}$

        (this batch is same for all workers)

        Accumulate the gradients

        Calculate the weights  $\beta_m$

$x_C = \sum_{m=1}^M \beta_m x_m$

$x_m \leftarrow (1 - \lambda)x_m + \lambda x_C$

**end if**

**end while**

---

on each layer of the network. Similarly to MGRAWA, first gradients are accumulated for the network layers using the subset of the train data set without updating any network components. The main difference in this version is that, rather than assigning a weight to the whole model parameter vector, weights do vary across different model layers. The motivation behind this approach relies on favoring a worker’s layer that corresponds to a gradient with a smaller norm compared to the same layer of the other workers. Intuitively, a smaller gradient norm means that the layer requires less correction. We refer to these layers as *mature* layers and, in LGRAWA’s weighted averaging scheme, mature layers are granted larger weights. Figure 3 provides an illustration of the difference between MGRAWA and LGRAWA algorithms. In order to describe LGRAWA mathematically, let us define  $A^k$  that corresponds to the norm of the total accumulated gradient at layer  $k$ . We can write  $A^k$  as:

$$G_k = \sum_n^N g_k^n \quad \text{and} \quad A^k = \|G_k\|_F. \quad (5)$$

Let  $A_m^k$  be the norm of the accumulated gradient of worker  $m$  at its  $k^{\text{th}}$  layer. For each worker  $m$ , we form the following list:

$$[A_m^k]_{k=1}^K = [A_m^1, A_m^2, \dots, A_m^K]. \quad (6)$$

Let  $\beta_m^k$  be the coefficient for worker  $m$ ’s  $k^{\text{th}}$  layer. Then, for each layer of each worker, we find the coefficients of the weighted average scheme using inverse proportionality to gradient norm scores. Notice that

the coefficient calculation procedure is applied to each layer  $k$  separately and  $\sum_{k=1}^K \sum_{m=1}^M \beta_m^k = K$ . Let  $x_m^k$  and  $x_C^k$  be the parameters at layer  $k$  of worker  $m$  and the center model respectively. Overall, we write:

$$\beta_m^k \propto \frac{1}{A_m^k} \text{ s.t. } \sum_{m=1}^M \beta_m^k = 1 \longrightarrow x_C^k = \sum_{m=1}^M \beta_m^k x_m^k \quad (7)$$

Equation 7 calculates the parameters of the consensus model at the  $k^{\text{th}}$  layer in the network structure. So, the consensus model is simply the model that is constructed from layers  $k = 1 : K$ , where the parametrization of each layer is calculated according to this equation. We can symbolically write  $x_C = \text{construct}(x_C^1, x_C^2, \dots, x_C^K)$ . Then, the update rule is the same as in MGRAWA and GRAWA updates. Pseudo-code of can be found in Algorithm 2.

---

**Algorithm 2** LGRAWA
 

---

**Input:** Pulling force  $\lambda$ , communication period  $\tau$ , learning rate  $\eta$ , proximity search strength  $\mu$ , loss function  $f$

**Initialize** workers  $x_1, x_2, \dots, x_M, x_C$  from the same random model, worker-exclusive data shards  $\Psi_1, \Psi_2, \dots, \Psi_M$  and iteration counters for workers  $t_1 = t_2 = \dots = t_M = 0$

At each worker  $m$  **do**

**while** not converged **do**

    Draw a random batch  $\xi_m \in \Psi_m$

$x_m \leftarrow x_m - \eta \nabla f(x_m; \xi_m)$

$x_m \leftarrow (1 - \frac{\mu}{\tau}) x_m + \frac{\mu}{\tau} x_C$  (Prox.)

$t_m \leftarrow t_m + 1$

**if**  $M\tau$  divides  $\sum_{m=1}^M t_m$  **then**

        Draw a random batch from  $\mathcal{D}$

        (this batch is same for all workers)

        Accumulate the gradients

        Obtain the list  $[A_m^1, A_m^2, \dots, A_m^K]$

        Calculate  $\beta_m^k$  for all  $k = 1 : K$

        Calculate  $x_C^k = \sum_{m=1}^M \beta_m^k x_m^k$

$x_C = \text{construct}(x_C^1, x_C^2, \dots, x_C^K)$

$x_m \leftarrow (1 - \lambda)x_m + \lambda x_C$

**end if**

**end while**

---

In Figure 3, we illustrate the difference between MGRAWA and LGRAWA algorithms. In the figure, blue and red colors represent the parameters of worker 1 ( $x_1$ ) and 2 ( $x_2$ ), respectively. This illustration is provided for a model with 6 layers (6 connected circles). The colors of the layers of the center model  $x_c$  are chosen based on the closeness to the layers of the workers (or in other words the weights of the layers of the workers). In the case of MGRAWA (on the left), the parameters of the center model in different layers have the same color since all layers of the center model are

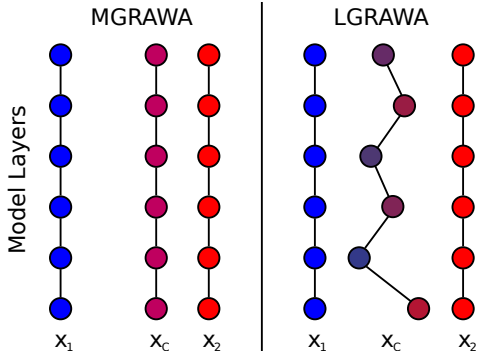


Figure 3: Illustration of MGRAWA and LGRAWA weighted averaging schemes in the case of two workers.

influenced in the same way by each worker (each worker is assigned just a single weight). Furthermore, the layers of the central model are reddish, which implies that the weight coefficient for  $x_2$  was greater than for  $x_1$  and thus the 2<sup>nd</sup> worker had more influence on the center model than the 1<sup>st</sup> one. In LGRAWA (on the right), layers of the center model do not have the same color, unlike in MGRAWA. This is because in LGRAWA the weighted averaging is applied on the layer level, not the model level. Furthermore, the colors in the layers of  $x_C$  are determined by their closeness to the corresponding layers of the workers. The layers that are closer to worker 1 are more blueish-purplish whereas the ones closer to worker 2 are more reddish-pinkish.

### 5.3 Proximity Search Mechanism and Momentum Update

The distributed update for both MGRAWA and LGRAWA requires more time than the other distributed training algorithms due to the gradient accumulation phase. To compensate for that we select higher communication periods in order to balance the time allocated for local and distributed optimization. As a result, the flatness-encouraging updates for MGRAWA and LGRAWA algorithms are applied less frequently, which is detrimental for performance. To overcome this limitation, inspired by a similar mechanism in (Teng et al., 2019), we apply an additional force in the local optimization phase that pulls the workers toward the previously calculated center model. We refer to this additional force as the proximity search mechanism as it ensures that the workers do not completely move apart and encourages the flatter region of the center variable in the loss surface in the local optimization phase. We denote the proximity search coefficient by  $\mu$  and scale it with the multiplicative inverse of the communication period  $\tau$ . To be more specific, the longer the communication period is, the more the worker is pulled towards

an outdated direction. Thus, the effective applied force should be  $\frac{\mu}{\tau}$  so that the larger the  $\tau$  is, the less proximity correction is applied.

Finally, the momentum update applied on the accumulated gradient scores stabilizes the flatness-seeking mechanism and helps to avoid the situation when a worker traversing on an otherwise smooth path, incidentally scores a high gradient norm value for the current selection of the subset of train samples and is therefore granted a small weight in the weighted average of MGRAWA and LGRAWA.

## 6 Experiments

### 6.1 Experiment Details

We consider four parameter-sharing distributed training optimizers: EASGD, LSGD, MGRAWA, and LGRAWA, and one gradient-sharing method DistributedDataParallel (DP) (Li et al., 2020) coupled with the SGD optimizer. To show the performance of the well-known flat-minima-seeking optimizer SAM, we couple it with the DP method, the vanilla distributed training algorithm. We would like to emphasize that we conduct the performance analysis of the listed methods in a time-constrained training environment, that is, the training is limited by the total training time, not the number of epochs since main motivation behind using data-parallel distributed training is to reduce the overall training time. Our code base is publicly available on [github.com/tolgadimli/GRAWA](https://github.com/tolgadimli/GRAWA).

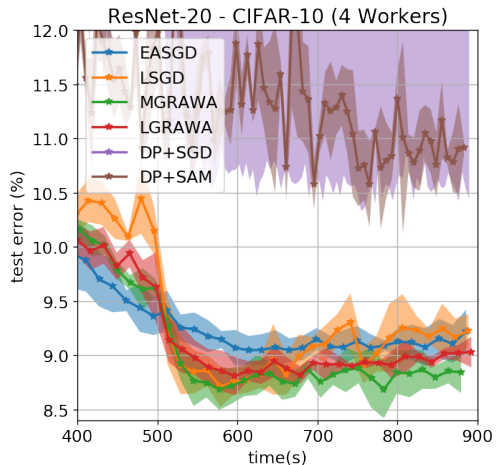


Figure 4: Test error (%) curves of different distributed training methods in 4 workers and ResNet-20 setting.

We carry out experiments in three different settings with varying numbers of workers: 4, 8, and 12. We use the well-known image classification data

Table 1: Mean and standard deviation of the test errors (lower is better).

Experiment	Model	EASGD	LSGD	MGRAWA	LGRAWA	DP+SGD	DP+SAM
CIFAR-10 4 Workers	ResNet-20	9.22±0.20	9.23±0.16	8.99±0.27	<b>8.93±0.20</b>	9.67±0.06	10.91±0.05
	VGG-16	7.68±0.25	7.65±0.13	7.64±0.15	<b>7.52±0.13</b>	8.67±0.06	8.92±0.39
	PyramidNet	4.06±0.16	3.98±0.05	4.04±0.11	<b>3.79±0.08</b>	4.19±0.07	5.54±0.06
CIFAR-100 8 Workers	DenseNet	23.61±0.32	<b>22.55±0.14</b>	22.82±0.28	22.72±0.22	23.42±0.07	28.48±0.09
	WideResNet	20.37±0.15	20.01±0.21	<b>19.68±0.13</b>	19.71±0.02	21.47±0.08	29.54±0.07
	PyramidNet	19.02±0.06	19.25±0.12	19.05±0.14	<b>18.82±0.17</b>	19.59±0.25	29.85±0.96
ImageNet(12W)	ResNet-50	28.59±0.05	26.17±0.06	<b>25.35±0.04</b>	25.68±0.08	25.49±0.04	36.92±0.11

Table 2: Mean and standard deviation of the Frobenius norm of the Hessian matrices (lower is better).

Experiment	Model	EASGD	LSGD	MGRAWA	LGRAWA	DP+SGD	DP+SAM
CIFAR-10 4 Workers	ResNet-20	84.12±5.32	317.96±4.55	84.32±3.72	<b>78.13±2.06</b>	324.97±23.79	113.45±5.41
	VGG-16	258.42±40.50	252.70±17.94	199.18±9.47	227.03±10.36	864.52±73.77	<b>134.81±2.38</b>
	PyramidNet	341.28±35.97	526.61±73.22	297.59±48.31	<b>240.06±59.12</b>	474.26±53.80	318.55±68.38
CIFAR-100 8 Workers	DenseNet	194.04±12.98	194.65±3.67	<b>164.89±25.65</b>	177.29±10.13	435.74±49.55	633.689±47.30
	WideResNet	964.11±83.16	1037.38±126.20	<b>584.13±77.42</b>	708.94±109.86	1382.94±234.10	647.11±56.59
	PyramidNet	241.70±31.38	750.07±52.73	233.98±14.82	255.47±47.52	284.66±26.87	<b>145.57±5.38</b>

sets CIFAR-10 (Krizhevsky et al., a), CIFAR-100 (Krizhevsky et al., b) and ImageNet (Deng et al., 2009). In the experiments with CIFAR-10 and 4 workers, we use ResNet-20 (He et al., 2016), VGG16 (Simonyan and Zisserman, 2014), PyramidNet-110 ( $\alpha = 270$ ) (Han et al., 2017) models, and in the experiments with CIFAR-100 and 8 workers, we use DenseNet-121 (Huang et al., 2017), WideResNet (Zagoruyko and Komodakis, 2016) type WRN28-10 without the bottleneck layer, and PyramidNet again. Finally, in the ImageNet experiments, we utilized all 12 workers (12W) in our distributed training environment and trained ResNet-50 models. The other details regarding the training procedure can be found in Appendix B.

### 6.2 Experiment Results

In this subsection, we present the final test errors and flatness proxy values that are obtained using our proposed LGRAWA and MGRAWA methods and the other competitor methods namely, LSGD, EASGD, and the vanilla algorithm *DistributedDataParallel* which is also coupled with the SAM optimizer. As can be seen from Table 1, GRAWA family algorithms achieve the smallest errors in most settings consistently. Particularly, the LGRAWA method keeps its superiority in the majority of the results.

We also carry out flatness analysis with the eigenvalues of the Hessian matrix. To do this, we utilize the Lanczos algorithm (Simon, 1984; Parlett and Scott, 1979) in the first 100 most representative directions of the Hessian Spectra. Then, we approximate the Frobenius norm of

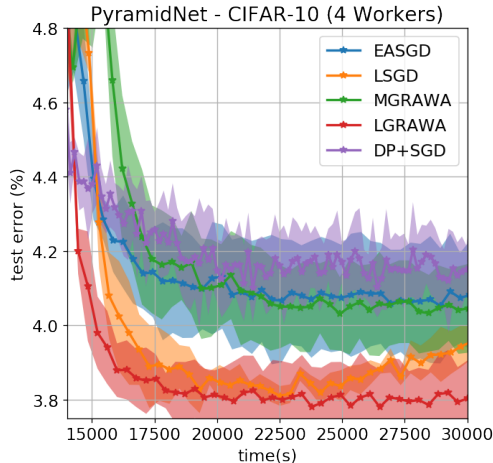


Figure 5: Test error (%) curves of different distributed training methods in 4 workers and PyramidNet setting.

the Hessian matrix, a good surrogate of the sharpness (Jiang\* et al., 2020), by using the extracted 100 pseudo-eigenvalues. The results are provided in the table 2. Note that we exclude the Imagenet experiments from this set of results due to the dataset size. The results in the table show that MGRAWA and LGRAWA algorithms encourage flat minima consistently while ensuring smaller test errors. Notice that in two settings the SAM optimizer yields a flatter minimum but its corresponding test errors are considerably higher than the GRAWA family. Finally in Table 7 from appendix, we show that the GRAWA family algorithms require less amount of inter-worker communication time than its competitors.



Table 3: Scalability analysis results for MGRWA and LGRWA

Model - Dataset	MGRWA			LGRWA		
	4 Workers	8 Workers	12 Workers	4 Workers	8 Workers	12 Workers
ResNet-20 - CIFAR-10	8.99 $\pm$ 0.27	8.79 $\pm$ 0.19	8.84 $\pm$ 0.13	8.93 $\pm$ 0.20	8.97 $\pm$ 0.16	8.94 $\pm$ 0.11
PyramidNet - CIFAR-100	19.05 $\pm$ 0.14	19.00 $\pm$ 0.08	19.07 $\pm$ 0.05	18.94 $\pm$ 0.07	18.82 $\pm$ 0.17	18.86 $\pm$ 0.11

### 6.3 Ablation Study

We also carry out an ablation study to observe how our algorithm performs in the same setting when we increase the number of workers participating in the distributed training environment. Particularly, we test the scalability of the MGRWA and LGRWA methods by training ResNet-20 models on CIFAR-10 dataset and PyramidNet models on CIFAR-100 with 4, 8 and 12 workers. For each model-dataset-worker combination, the experiments are run for 3 different seeds, the resulting mean and standard deviation values are reported in Table 3.

Our scalability analysis reveals that the proposed algorithms MGRWA and LGRWA do not suffer from performance degradation when the number of workers in the training environment is increased. We also argue that it might be possible to benefit from an increased number of workers in the distributed setting as more information about the loss landscape can be collected by more workers traversing and exploring it. This is left as an open question and it requires further investigation.

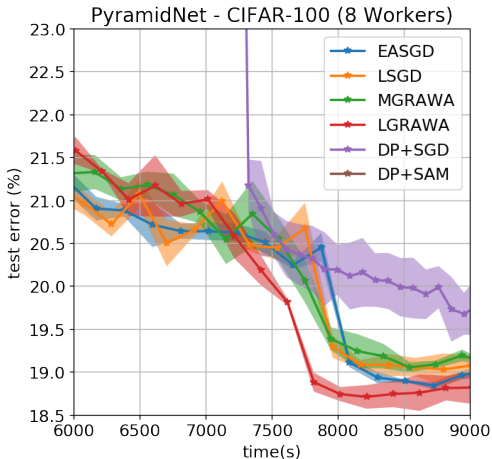


Figure 6: Test error (%) curves of different distributed training methods in 8 workers and PyramidNet setting.

## 7 Conclusion

In this paper, we propose a novel asynchronous algorithm family, which are namely MGRWA and LGRWA, for the distributed training of deep learning models. They fall into the category of knowledge transfer methods with periodic parameter sharing, similar to the LSGD and EASGD. In the distributed update phase, our algorithms employ a weighted averaging scheme. In this scheme, the weights are determined inversely proportional to the accumulated gradients inside the network layers which signals the smoothness of the model’s trajectory on the loss valley and, how mature the model components are. Our motivation behind designing such a weighted averaging scheme is grounded on the importance of the flatness of the loss valley for the generalization capability of the final model, as well as prioritizing the network layers that require less correction, namely mature components. We provide theoretical proof for the convergence rate of the vanilla GRWA algorithm in the convex setting and provide a convergence analysis in the non-convex case that holds for GRWA and both of its variants. We also demonstrate the effectiveness of our algorithms MGRWA and LGRWA empirically by training deep learning models with various architectures and on different data sets. Our experimental results show that MGRWA and LGRWA accelerate the convergence rate and achieve lower error rates than SOTA competitors while yielding flatter minima and requiring less communication. The scalability of the proposed algorithms is also investigated and no performance drop is observed when the number of workers is increased.

## 8 Acknowledgement

The authors acknowledge that the NSF Award #2041872 sponsored the research in this paper.

## References

- T. Ben-Nun and T. Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4): 1–43, 2019.
- D. Bisla, J. Wang, and A. Choromanska. Low-pass filtering sgd for recovering flat optima in the deep

- learning optimization landscape. In *International Conference on Artificial Intelligence and Statistics*, pages 8299–8339. PMLR, 2022.
- L. Bottou. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.
- S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, 2011. doi: 10.1561/22000000016. URL <https://doi.org/10.1561/22000000016>.
- P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina. Entropy-sgd: biasing gradient descent into wide valleys\*. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, dec 2019. doi: 10.1088/1742-5468/ab39d9. URL <https://dx.doi.org/10.1088/1742-5468/ab39d9>.
- A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.
- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=6TmlmposlrM>.
- D. Han, J. Kim, and J. Kim. Deep pyramidal residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5927–5935, 2017.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
- Y. Jiang\*, B. Neyshabur\*, H. Mobahi, D. Krishnan, and S. Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=SJgIPJBFvH>.
- K. Kawaguchi. Deep learning without poor local minima. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL [https://proceedings.neurips.cc/paper\\_files/paper/2016/file/f2fc990265c712c49d51a18a32b39f0c-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2016/file/f2fc990265c712c49d51a18a32b39f0c-Paper.pdf).
- J. D. M.-W. C. Kenton and L. K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2, 2019.
- N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. URL <http://arxiv.org/abs/1609.04836>.
- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). a. URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100 (canadian institute for advanced research). b. URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *CoRR*, abs/2006.15704, 2020. URL <https://arxiv.org/abs/2006.15704>.
- D. MacKay. Bayesian model comparison and backprop nets. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.
- W. J. Maddox, G. W. Benton, and A. G. Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *CoRR*, abs/2003.02139, 2020a. URL <https://arxiv.org/abs/2003.02139>.
- W. J. Maddox, G. W. Benton, and A. G. Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *CoRR*, abs/2003.02139, 2020b. URL <https://arxiv.org/abs/2003.02139>.
- B. N. Parlett and D. S. Scott. The lanczos algorithm with selective orthogonalization. *Mathematics of computation*, 33(145):217–238, 1979.
- G. Pereyra, G. Tucker, J. Chorowski, L. Kaiser, and G. Hinton. Regularizing neural networks by penalizing confident output distributions, 2017. URL <https://openreview.net/forum?id=HkCjNI5ex>.
- N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. ISSN 0893-6080. doi: [https://doi.org/10.1016/S0893-6080\(98](https://doi.org/10.1016/S0893-6080(98)

- 00116-6. URL <https://www.sciencedirect.com/science/article/pii/S0893608098001166>.
- S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL <http://arxiv.org/abs/1609.04747>.
- H. D. Simon. The lanczos algorithm with partial re-orthogonalization. *Mathematics of computation*, 42(165):115–142, 1984.
- K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL <http://arxiv.org/abs/1409.1556>.
- Y. Teng, W. Gao, F. Chalus, A. E. Choromanska, D. Goldfarb, and A. Weller. Leader stochastic gradient descent for distributed training of deep learning models. *Advances in Neural Information Processing Systems*, 32, 2019.
- J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer. A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 53(2):1–33, 2020.
- S. Yang, Y. Wang, and X. Chu. A survey of deep learning techniques for neural machine translation. *CoRR*, abs/2002.07526, 2020. URL <https://arxiv.org/abs/2002.07526>.
- S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016*. British Machine Vision Association, 2016.
- S. Zhang, A. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’15, page 685–693, Cambridge, MA, USA, 2015. MIT Press.
- (b) Complete proofs of all theoretical results. [**Yes**]
- (c) Clear explanations of any assumptions. [**Yes**]
3. For all figures and tables that present empirical results, check if you include:
- (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [**Yes**, we share the experiment results in the CSV format in the supplementary files.]
- (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [**Yes**]
- (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [**Yes**]
- (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [**Yes**]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
- (a) Citations of the creator If your work uses existing assets. [**Yes**]
- (b) The license information of the assets, if applicable. [**Not applicable**]
- (c) New assets either in the supplemental material or as a URL, if applicable. [**Not applicable**]
- (d) Information about consent from data providers/curators. [**Not applicable**]
- (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [**Not applicable**]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
- (a) The full text of instructions given to participants and screenshots. [**Not applicable**]
- (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [**Not applicable**]
- (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [**Not applicable**]

## Checklist

1. For all models and algorithms presented, check if you include:
  - (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [**Yes**]
  - (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [**Yes**]
  - (c) (Optional) Anonymized source code, with the specification of all dependencies, including external libraries. [**Yes**]
2. For any theoretical claim, check if you include:
  - (a) Statements of the full set of assumptions of all theoretical results. [**Yes**]

## A Motivational Example

As mentioned in section the motivational example section, we use the Vincent function to demonstrate the flatness encouraging updates of the GRAWA algorithm and its variants MGRAWA and LGRAWA. Loss surface of this function ( $f(x, y) = -\sin(10 \ln(x)) - \sin(10 \ln(y))$ ) is provided in the figure below.

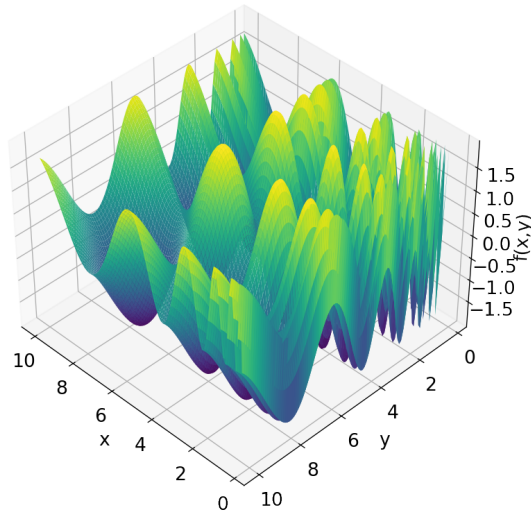


Figure 7: Loss surface of the Vincent function with 2-dimensional input

We initialized 4 workers from the corners of the  $(x, y)$  plane above. More specifically, workers are initialized from the following points:  $(0.25, 0.25)$ ,  $(0.25, 10)$ ,  $(10, 0.25)$ ,  $(10, 10)$ . Each worker runs the vanilla SGD algorithm as its local optimizer with a learning rate of 0.01 and then it is coupled with a distributed optimizer that applies a pulling force after every 4 local updates in accordance with the policy of the distributed training algorithm. We used 5 different distributed optimizers which are EASGD, LSGD, GRAWA, MGRAWA, and LGRAWA. In the following figures, we share the trajectories followed by all of the workers when each of the aforementioned distributed optimizers is used.

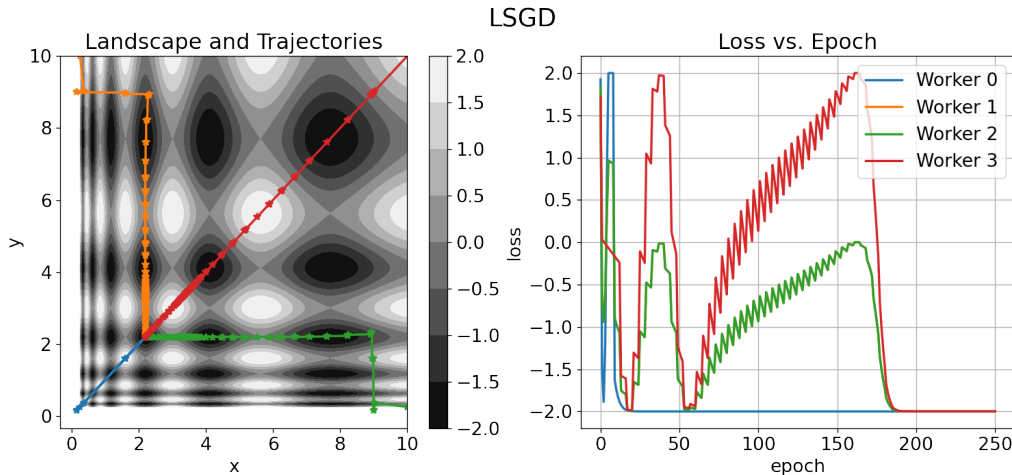


Figure 8: Trajectories of the workers and when the distributed optimizer is LSGD

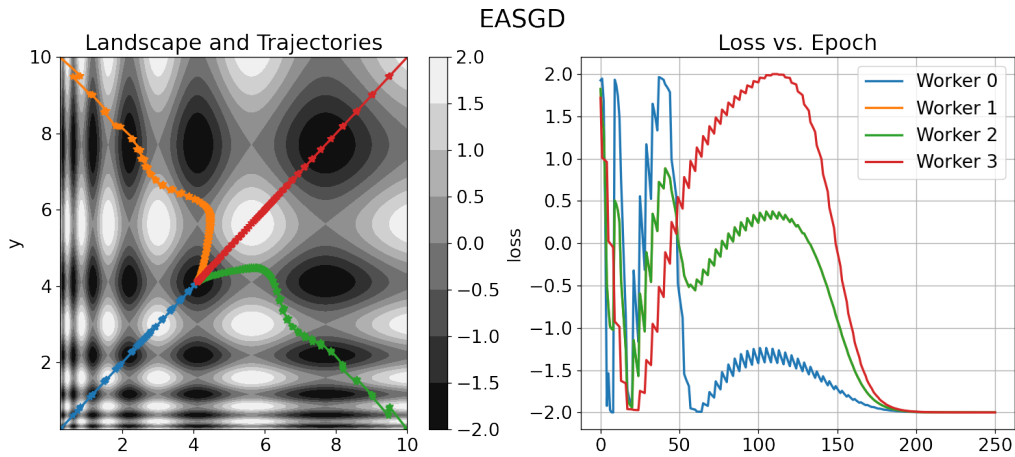


Figure 9: Trajectories of the workers and when the distributed optimizer is EASGD

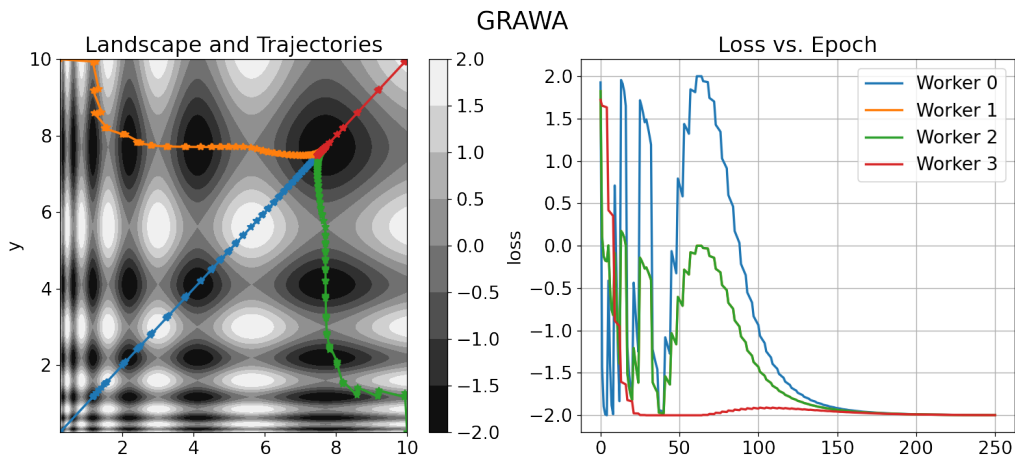


Figure 10: Trajectories of the workers and when the distributed optimizer is GRAWA

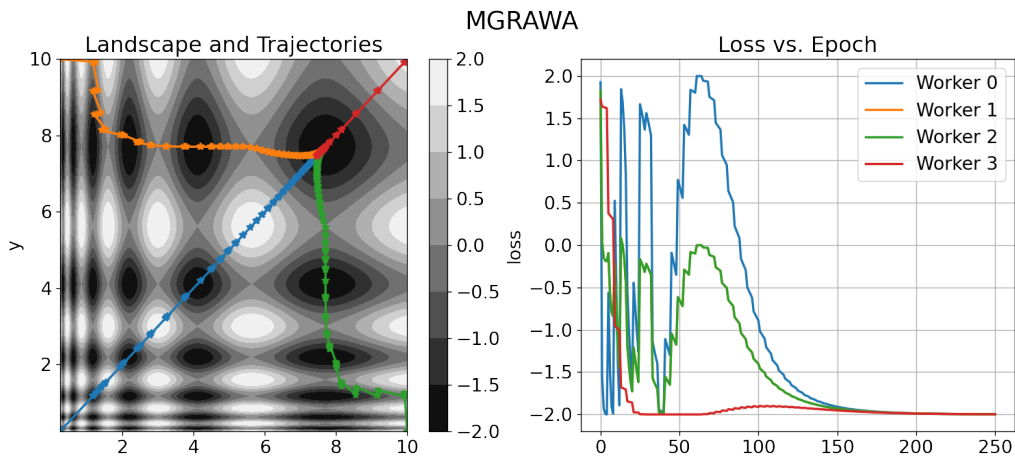


Figure 11: Trajectories of the workers and when the distributed optimizer is MGRAWA

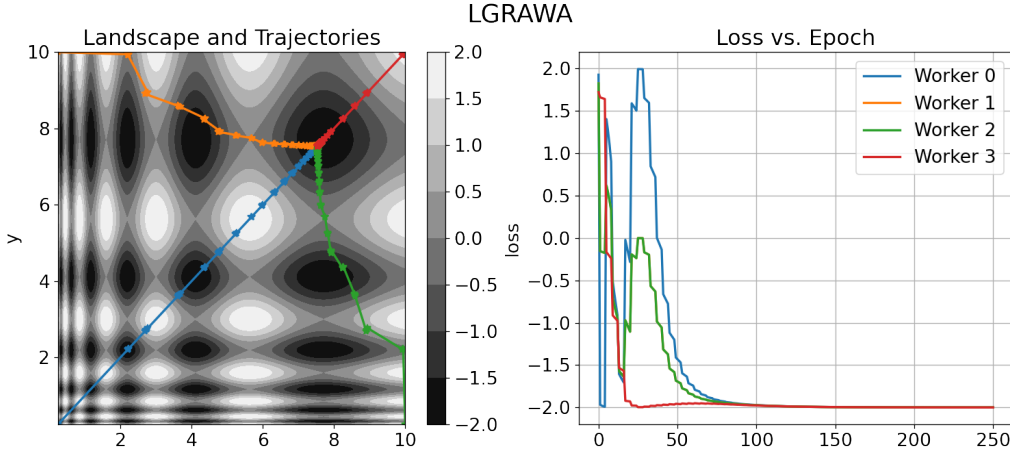


Figure 12: Trajectories of the workers and when the distributed optimizer is LGRAWA

As can be seen from the trajectory figures above, although LSGD manages to spot the minimizer early but with a sharp valley, which results in a low generalization capability. It constantly pulls all of the workers into that valley because the leader is trapped there. As for the EASGD algorithm, independent of the geometry of the loss surface and performance of the other workers, it constantly pulls all the workers towards the average of them and finally converges to a valley with a slightly wider valley than LSGD. In the case GRAWA and its variants (MGRAWA and LGRAWA), the distributed optimization process constantly pushed the workers toward the flatter region of the valley, converging to a point with a small loss value and good generalization capability.

## B Experiment Details

### B.1 Gradient Norm Experiments

In order to validate our hypothesis about the correlation between the gradient norm calculated on the final model parameters and the generalization gap (test error(%) - train error(%)), we first train 162 different ResNet-18 models with varying hyperparameters and architectures. The hyperparameters that are varied and their ranges can be found in the table below.

Table 4: Hyperparameter search space used in the Gradient Norm experiments

Hyperparameter	Search Grid
Number of Residual Layers	[4, 6, 8]
Momentum Coefficient	[0.1, 0.5, 0.9]
Learning Rate	[0.01, 0.0075, 0.005]
Batch Size	[32, 128, 512]
Weight Regularization	[0, 0.0001]

In total there are 162 different models trained with different hyperparameters. For training the models, no augmentation technique is applied on the train set. Only normalization is applied as pre-processing with the mean vector (0.4914, 0.4822, 0.4465) and the standard deviation vector (0.2023, 0.1994, 0.2010). Also note that both train errors and the gradient norms are calculated on the whole train set of the CIFAR-10 data without any augmentation. Similarly, test errors are calculated on the whole test set.

### B.2 Distributed Training Experiments

#### B.2.1 Data Preprocessing and Loading

Since we are using the PyTorch-based distributed training library introduced with the LSGD paper, we stick to the same data preprocessing steps. In all the experiments with the CIFAR-10 and CIFAR-100 dataset, images

are normalized with the mean vectors  $(0.4914, 0.4822, 0.4465)$ ,  $(0.5071, 0.4867, 0.4408)$  and the standard deviation vectors  $(0.2023, 0.1994, 0.2010)$ ,  $(0.2675, 0.2565, 0.2761)$  respectively. To augment the training set, a horizontal flip with a probability of 0.5 is used. For the ResNet-20 model, the original  $3 \times 32 \times 32$  CIFAR-10 images are randomly cropped to be presented as images of size  $3 \times 28 \times 28$  to the network. Note that, for the test samples, the  $3 \times 28 \times 28$  part in the center of the image is extracted.

For the VGG-16, PyramidNet, DenseNet, and WideResNet models, train set samples are first padded to the size  $3 \times 40 \times 40$ , then  $3 \times 32 \times 32$  portion of the padded image is randomly cropped and presented to the network for augmentation. The testing is carried out with the original  $3 \times 32 \times 32$  images without any preprocessing except the normalization.

In the ImageNet experiment with ResNet-50 model, we normalize samples with the mean vector  $(0.485, 0.456, 0.406)$  and the standard deviation vector  $(0.229, 0.224, 0.225)$ . For the augmentation of train set, the image portions are randomly crop such that the crop accounts an area between 8% or 100% of the original image size and then it is resized to the shape  $3 \times 224 \times 224$ . We also randomly apply a horizontal flip. For the test samples, the images are resized to  $3 \times 256 \times 256$ , and the center  $3 \times 224 \times 224$  part is cropped.

To load the data, PyTorch’s distributed data sampler is used. It automatically splits the data into non-overlapping data shards and each worker is only trained with its own shard. Notice that this only applies to the train portion of the dataset and the number of data shards is equal to the number of workers in the parallel computing environment.

### B.2.2 Local Optimizer Details

Unless otherwise stated, in all of the experiments we use SGD with a learning rate of 0.1, a momentum of 0.9 and Nesterov acceleration as the local optimizer Qian (1999). The weight decay values are selected and learning rate drops are applied in accordance with the training details in the aforementioned papers describing the vision models we use. For the SAM optimizer, we select the base optimizer as SGD and we use the suggested value in the paper for the neighborhood search parameter, that is  $\rho = 0.05$ .

### B.2.3 Hyperparameter Search Grids

In the tables below, the hyperparameter search grid for different distributed training optimizers is provided for the experiments on the CIFAR-10 and CIFAR-100 datasets.

Table 5: Hyperparameter search space for LSGD, EASGD, MGRAWA, and LGRWA

	LSGD	EASGD	MGRAWA & LGRWA
Comm. Period( $\tau$ )	4, 8, 16	4, 8, 16	16, 32
Pulling Force( $\lambda$ )	0.01, 0.1	0.1, 0.3, 0.43*	0.3, 0.5, 0.7
Prox. Search( $\mu$ )	0.01, 0.1	NA	0.01, 0.05

For each of the optimizers mentioned in the table above, the learning rate of the local SGD optimizer is set to 0.1 and its momentum value is 0.9. Also, a fixed batch size of 128 is used. Note that, for EASGD, pulling force of 0.43 is specifically added to the hyperparameter search space since it is mentioned in the original EASGD paper.

Table 6: Hyperparameter search space for DataParallel

	DataParallel
Learning Rate	0.01, 0.1
Batch Size	32, 64, 128

Because the DataParallel algorithm processes the gradient from each worker and applies the same update to the all workers, it does not transfer the knowledge with parameter sharing. Hence, we do not have a communication

period or a pulling force towards the center model. Instead, we vary the local learning rate of the optimizer and the batch size.

For the ImageNet experiments using the ResNet-50 model, we have used the same hyperparameter search grids for the parameter sharing methods as before. In the 12 worker setting, LSGD has global and local communication periods. Following the convention in the paper LSGD paper, the local communication period is picked as a quarter of the global communication period.  $\tau_L = \frac{\tau_G}{4}$ . For the DataParallel algorithm, the fixed batch size of 32 is used and only the learning rate is varied among the values 0.01, 0.1.

### C Hardware Details

We run both the flatness experiments and carry out the distributed training of deep learning models on 3 machines, each of which has  $4 \times$  NVIDIA GTX 1080 GPU cards (in total 12 GPUs). The machines are connected over Ethernet.

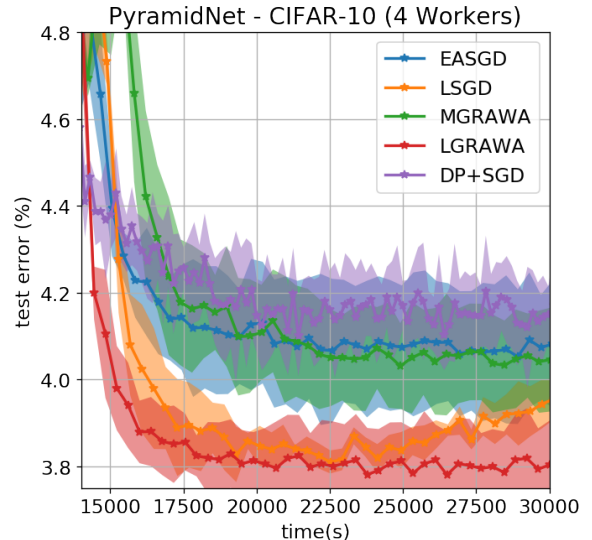
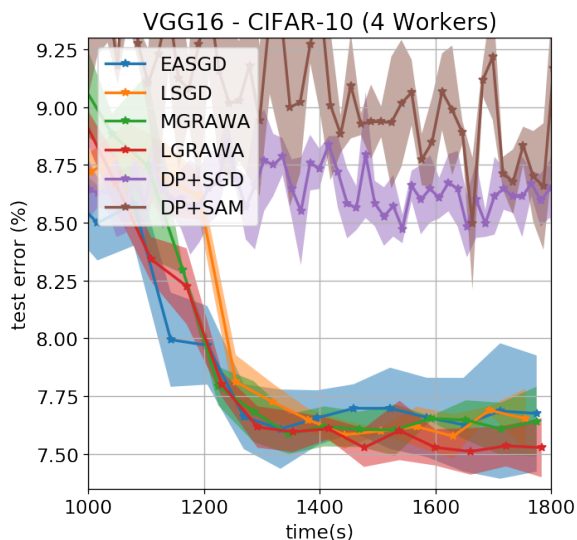
### D Additional Experiment Results

In the table below, we report the total time spent on the communication for each parameter-sharing distributed optimizer, in their hyperparameter setting that yields the lowest test error. As can be seen, the GRAWA family requires the least amount of time spent on inter-worker communication.

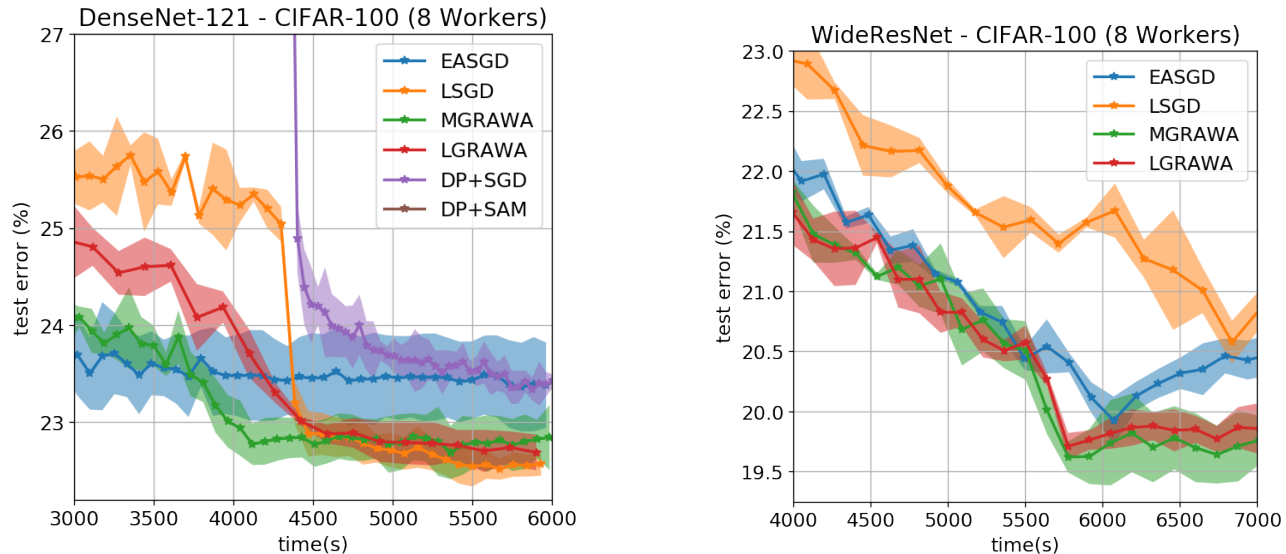
Table 7: Time spent on inter-worker communication in seconds(s) and total number of inter-worker communications (in parenthesis) for the parameter sharing distributed training methods (lower is better).

Experiment	Model	EASGD	LSGD	MGRAWA	LGRAWA
CIFAR-10 4 Workers	ResNet-20	57s (2539)	29s (1319)	<b>26s (167)</b>	50s (328)
	VGG-16	100s (2747)	95s (2783)	81s (353)	<b>75s (351)</b>
	PyramidNet	3673s (15053)	3252s (14842)	<b>1602s (1858)</b>	1608s (1886)
CIFAR-100 8 Workers	DenseNet	590s (7557)	438s (6761)	<b>291s (927)</b>	317s (923)
	WideResNet	880s (2991)	2268s (2469)	531s (392)	<b>435s (393)</b>
	PyramidNet	1884s (7381)	1177s (7496)	<b>730s (979)</b>	767s (971)

We also provide additional plots that include convergence curves of the other four settings in Table 1 that are not presented in the main body due to page limit.







We also report the generalization gap that is simply calculated as test error (%) - train error (%). We only show cases when the generalization gap is not equal or very close to the test error and we observe that LGRAWA attains the smallest gap overall.

Experiment	Model	EASGD	LSGD	MGRAWA	LGRAWA	DP+SGD	DP+SAM
CIFAR-10	ResNet-20	6.75 $\pm$ 0.32	6.83 $\pm$ 0.27	6.50 $\pm$ 0.24	<b>6.39<math>\pm</math>0.16</b>	7.45 $\pm$ 0.59	9.35 $\pm$ 0.52
4 Workers	VGG-16	7.08 $\pm$ 0.26	7.19 $\pm$ 0.17	6.86 $\pm$ 0.15	<b>6.22<math>\pm</math>0.34</b>	8.38 $\pm$ 0.30	8.59 $\pm$ 0.54
CIFAR-100 (8W)	DenseNet	23.53 $\pm$ 0.31	22.49 $\pm$ 0.18	22.49 $\pm$ 0.45	<b>22.11<math>\pm</math>0.59</b>	23.42 $\pm$ 0.07	28.35 $\pm$ 0.26

## E Details of the Theoretical Results

### E.1 Technical Preliminaries

Recall the problem setting in a distributed training environment. There are  $M$  workers  $(x_1, x_2, \dots, x_M)$ , and there is a function  $f$  (here we use  $f$  instead of  $F$ ) which is to be collectively minimized by all of the workers. In the GRAWA algorithm, there is a center variable  $x_C$  that is obtained by a weighted average of all of the workers. Let  $\beta_1, \beta_2, \dots, \beta_M$  be the weights associated with the workers  $x_1, x_2, \dots, x_M$  respectively in this weighted averaging scheme. Also, recall that these weights are inversely proportional to the gradient norms of the function at points  $x_1, x_2, \dots, x_M$  and,  $\sum_{i=1}^M \beta_i = 1$ . This corresponds to the following expression for each weight  $\beta_i$ :

$$\beta_i = \frac{\Theta}{\|\nabla f(x_i)\|} \quad \text{where} \quad \Theta = \frac{\prod_{i=1}^M \|\nabla f(x_i)\|}{\sum_{i=1}^M \frac{\prod_{j=1}^M \|\nabla f(x_j)\|}{\|\nabla f(x_i)\|}} \quad (8)$$

### E.2 Converge Analysis in Convex Case

**Assumption:** Let us have a function  $f$  with minimizer  $x^*$  such that the following conditions are satisfied  $\forall x, y \in \text{dom } f$ :

- $L$ -smooth:  $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$
- $m$ -strongly convex:  $f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{m}{2}\|y - x\|^2$
- $k$ -cone:  $|f(x) - f(x^*)| \geq k\|x - x^*\|$  i.e. lower bounded by a cone which has a tip at  $x^*$  and a slope  $k$

- $\mu$ -sPL:  $\|\nabla f(x)\| \geq \mu(f(x) - f(x^*))$

We would like to highlight that in the last precondition, we use a modified version of the well-known Polyak inequality, with a stricter upper bound (without the square) which we refer to as *sPL*.

**Theorem 4.** *For a  $\mu$ -sPL,  $L$ -smooth and  $k$ -cone function  $f$  with minimizer  $x^*$ , if  $f(a) \leq f(b)$  and  $k\mu \geq L$  then  $\|\nabla f(a)\| \leq \|\nabla f(b)\| \forall a, b \in \text{dom}f$ . In other words, the function  $G(y) = \|\nabla y\|$  is a monotonously increasing function when  $y$  is a  $\mu$ -sPL,  $k$ -cone and  $L$ -smooth function and when  $k\mu \geq L$ .*

*Proof.* From the definition of  $\mu$ -sPL, we can write:  $\|\nabla f(b)\| \geq \mu(f(b) - f(x^*))$ . Because  $f(a) \leq f(b)$ , we can also write:  $f(a) - f(x^*) \leq f(b) - f(x^*)$ . Overall, this implies the following:

$$\frac{1}{\mu} \|\nabla f(b)\| \geq f(a) - f(x^*)$$

Now, using the cone assumption, we can write  $f(a) - f(x^*) \geq k\|a - x^*\|$  which implies:

$$\frac{1}{\mu} \|\nabla f(b)\| \geq k\|a - x^*\|$$

Then, from  $L$ -Lipschitz differentiable property, we have  $\|a - x^*\| \geq L^{-1}\|\nabla f(a)\|$  which leads us to:

$$\frac{1}{\mu} \|\nabla f(b)\| \geq \frac{k}{L} \|\nabla f(a)\|$$

So, for  $k\mu \geq L$ , we have:

$$\|\nabla f(b)\| \geq \frac{k\mu}{L} \|\nabla f(a)\| \geq \|\nabla f(a)\| \quad \text{which implies} \quad \|\nabla f(a)\| \leq \|\nabla f(b)\|.$$

□

**Theorem 5.** *Let  $x_1, x_2, \dots, x_M \in \text{dom}f$  and,  $f$  is a convex,  $\mu$ -sPL, real-valued, continuous and  $L$ -Lipschitz differentiable function with. Let  $x_C = \sum_{i=1}^M \beta_i x_i$  where each  $\beta_i$  is calculated as in equation 8 so that  $\sum_{i=1}^M \beta_i = 1$ . Then,  $\|\nabla f(x_C)\| \leq \sqrt{M} \|\nabla f(x_i)\|$  holds  $\forall i \in 1, 2, \dots, M$ .*

*Proof.* Since  $f$  is a continuous, real, convex function and  $\sum_{i=1}^M \beta_i = 1$ , we can write the Jensen's inequality as follows:

$$f\left(\sum_{i=1}^M \beta_i x_i\right) \leq \sum_{i=1}^M \beta_i f(x_i) \quad \text{that is} \quad f(x_C) \leq \sum_{i=1}^M \beta_i f(x_i)$$

Then, by applying theorem 4, we can write  $\|\nabla f(x_C)\|^2 \leq \|\sum_{i=1}^M \beta_i \nabla f(x_i)\|^2$ . Using the triangle-inequality of the norm, we can also write the following upper bound for  $\|\nabla f(x_C)\|^2$ .

$$\|\nabla f(x_C)\|^2 \leq \sum_{i=1}^M \|\beta_i \nabla f(x_i)\|^2 = \sum_{i=1}^M \beta_i^2 \|\nabla f(x_i)\|^2$$

Plugging in the definition of  $\beta_i$  in equation 8, we obtain:

$$\|\nabla f(x_C)\|^2 \leq \sum_{i=1}^M \frac{\Theta^2}{\|\nabla f(x_i)\|^2} \|\nabla f(x_i)\|^2 = M\Theta^2$$

Taking square root of both sides, we obtain:

$$\|\nabla f(x_C)\| \leq \sqrt{M}\Theta \quad (9)$$

Because, each weight  $\beta_i \in [0, 1]$ , we have:

$$\beta_i = \frac{\Theta}{\|\nabla f(x_i)\|} \leq 1 \quad \longrightarrow \quad \Theta \leq \|\nabla f(x_i)\| \quad \forall i \in 1, 2, \dots, M$$

By combining this with inequality 9, we can write:

$$\|\nabla f(x_C)\| \leq \sqrt{M}\|\nabla f(x_i)\| \quad \forall i \in 1, 2, \dots, M$$

□

**Theorem 6.** Let  $x_C = \sum_{i=1}^M \beta_i x_i$  and  $\beta_i$ 's are calculated as in equation 8. For an  $L$ -Lipschitz differentiable,  $\mu$ -sPL,  $k$ -cone real-valued continuous function  $f$  with minimizer  $x^*$  satisfies  $f(x_C) \leq f(x_i)$  for all  $i \in 1, 2, \dots, M$  provided that  $k\mu \geq L\sqrt{M}$ .

*Proof.* using the  $k$ -cone property, we can write:

$$f(x_i) - f(x^*) \geq k\|x_i - x^*\| \quad \forall i \in 1, 2, \dots, M$$

From  $L$ -Lipschitz property, we know that  $L\|x_i - x^*\| \geq \|\nabla f(x_i)\|$  and by combining with the above inequality we can write:

$$f(x_i) - f(x^*) \geq \frac{k}{L}\|\nabla f(x_i)\| \quad \forall i \in 1, 2, \dots, M$$

By making use of 5, we can write:

$$f(x_i) - f(x^*) \geq \frac{k}{L\sqrt{M}}\|\nabla f(x_C)\| \quad \forall i \in 1, 2, \dots, M$$

Finally, by utilizing the  $\mu$ -sPL property, we reach the following inequality.

$$f(x_i) - f(x^*) \geq \frac{k}{L}\|\nabla f(x_C)\| \geq \frac{k\mu}{L\sqrt{M}}(f(x_C) - f(x^*)) \quad \forall i \in 1, 2, \dots, M$$

When  $k\mu \geq L\sqrt{M}$ , we can write:  $f(x_i) \geq f(x_C)$ .

□

**Theorem 7.** Let  $f$  be a function that holds 6. Let  $\tilde{g}(x)$  be an unbiased estimator for  $\nabla f(x)$  with  $\text{Var}(\tilde{g}(x)) \leq \sigma^2 + \nu\|\nabla f(x)\|^2$ , and let  $x_C$  be the center variable obtained with the GRAWA algorithm. Suppose that  $\eta, \lambda$  satisfy  $\eta \leq (2L(\nu + 1))^{-1}$  and  $\eta\lambda \leq \frac{m}{2L}$ ,  $\eta\sqrt{\lambda} \leq \frac{\sqrt{m}}{\sqrt{2L}}$ . Then the GRAWA step satisfies:

$$\mathbb{E}[f(x^{t+1}) - f(x^*)] \leq (1 - m\eta)(f(x^t) - f(x^*)) - \eta\lambda(f(x^t) - f(x_C)) + \frac{\eta^2 L}{2}\sigma^2$$

The presence of the new term due to the GRAWA update increases the speed of the convergence since  $f(x_C) \leq f(x)$  from 6. Then,  $\limsup_{t \rightarrow \infty} \mathbb{E}[f(x^{t+1}) - f(x^*)] \leq \eta \frac{L}{2m} \sigma^2$ . If  $\eta$  decreases at rate  $\eta = O(\frac{1}{t})$ , then  $\mathbb{E}[f(x^{t+1}) - f(x^*)] \leq O(\frac{1}{t})$ .

*Proof.* The proof can be found in section 7.4 of the appendix of the LSGD paper.

□

### E.3 Convergence Analysis in Non-Convex Scenario

**Theorem 8.** Recall the main objective of the distributed training objective we have:

$$\min_x F(x; \mathcal{D}) = \min_{x_1, \dots, x_M} \sum_{m=1}^M \mathbb{E}_{\xi \sim \mathcal{D}_m} f(x_m; \xi) + \frac{\lambda}{2} \|x_m - x_c\|^2$$

In our non-convex analysis, we make the following assumptions that impose  $L$ -smoothness condition on function  $F$ , bounds the variance of the calculated gradients using  $\xi$  sampled from  $\mathcal{D}_m$  for  $m = 1, 2, \dots, M$  and the norm of the Euclidean distance between the center variable and workers at any iteration or time  $t$ . Also, define  $f_m(x_m) = \mathbb{E}_{\xi \sim \mathcal{D}_m} f(x_m; \xi)$ .

- Function  $F$  is  $L$ -smooth.
- $\mathbb{E}_{\xi \sim \mathcal{D}_m} \|\nabla f(x, \xi) - \nabla f_m(x_m)\|^2 \leq \sigma^2$
- $\mathbb{E}_{m \sim U[M]} \|\nabla f_m(x_m) - \nabla F(x_m)\|^2 \leq \zeta^2$
- $\mathbb{E} \|x_m^t - x_c^t\|^2 \leq \rho^2$  at any time or iteration  $t$ .

*Proof.* We start by using the  $L$ -smooth property of the function  $F$  on the points  $x_m^{t+1}$  and  $x_m^t$ .

$$\mathbb{E}[F(x_m^{t+1})] \leq \mathbb{E}[F(x_m^t)] + \mathbb{E}\langle \nabla F(x_m^t), x_m^{t+1} - x_m^t \rangle + \frac{L}{2} \mathbb{E} \|x_m^{t+1} - x_m^t\|^2 \quad (10)$$

Using the update rule:  $x_m^{t+1} = x_m^t - \eta \nabla f(x_m^t, \xi) - \eta \lambda (x_m^t - x_c^t)$ , we can write:

$$\begin{aligned} \mathbb{E}[F(x_m^{t+1})] &\leq \mathbb{E}[F(x_m^t)] - \eta \mathbb{E}\langle \nabla F(x_m^t), \nabla f(x_m^t, \xi) \rangle - \lambda \mathbb{E}\langle \nabla F(x_m^t), x_m^t - x_c^t \rangle \\ &\quad + \frac{L\eta^2}{2} \mathbb{E} \|\nabla f(x_m^t, \xi)\|^2 + \frac{L\eta^2 \lambda^2}{2} \mathbb{E} \|x_m^t - x_c^t\|^2 \end{aligned}$$

With further manipulation of the terms, we obtain the following:

$$\begin{aligned} \mathbb{E}[F(x_m^{t+1})] &\leq \mathbb{E}[F(x_m^t)] - (\eta - L\eta^2) \mathbb{E}\langle \nabla F(x_m^t), \nabla f(x_m^t, \xi) \rangle + \frac{L\eta^2}{2} \mathbb{E} \|\nabla F(x_m^t) - \nabla f(x_m^t)\|^2 \\ &\quad - \eta \lambda \mathbb{E}\langle \nabla F(x_m^t), x_m^t - x_c^t \rangle + \frac{L\eta^2 \lambda^2}{2} \mathbb{E} \|x_m^t - x_c^t\|^2 \end{aligned} \quad (11)$$

We will derive upper bounds for each term in 11. We start with  $-\mathbb{E}\langle \nabla F(x_m^t), \nabla f(x_m^t, \xi) \rangle$ .

$$\begin{aligned} -\mathbb{E}\langle \nabla F(x_m^t), \nabla f(x_m^t, \xi) \rangle &= \frac{1}{2} \mathbb{E} \|\nabla F(x_m^t) - \nabla f(x_m^t, \xi)\|^2 - \frac{1}{2} \mathbb{E} \|\nabla F(x_m^t)\|^2 - \frac{1}{2} \mathbb{E} \|\nabla f(x_m^t, \xi)\|^2 \\ &\leq \frac{1}{2} \mathbb{E} \|\nabla F(x_m^t) - \nabla f(x_m^t, \xi)\|^2 - \frac{1}{2} \mathbb{E} \|\nabla F(x_m^t)\|^2 \\ &\leq \frac{1}{2} (\zeta^2 - \mathbb{E} \|\nabla F(x_m^t)\|^2) \end{aligned} \quad (12)$$

First, we derive a bound for the term  $\mathbb{E} \|\nabla F(x_m^t) - \nabla f(x_m^t, \xi)\|^2$ :

$$\begin{aligned} \mathbb{E} \|\nabla f(x_m^t, \xi) - \nabla F(x_m^t)\|^2 &= \mathbb{E} \|\nabla f(x_m^t, \xi) - \mathbb{E}_{\xi \sim \mathcal{D}_m} \nabla f(x_m; \xi) - (\nabla F(x_m^t) - \mathbb{E}_{\xi \sim \mathcal{D}_m} \nabla f(x_m; \xi))\|^2 \\ &\leq \mathbb{E} \|\nabla f(x_m^t, \xi) - \mathbb{E}_{\xi \sim \mathcal{D}_m} \nabla f(x_m; \xi)\|^2 + \|\nabla F(x_m^t) - \mathbb{E}_{\xi \sim \mathcal{D}_m} \nabla f(x_m; \xi)\|^2 \\ &= \mathbb{E} \|\nabla f(x_m^t, \xi) - \nabla f_m(x_m)\|^2 + \|\nabla F(x_m^t) - \nabla f_m(x_m; \xi)\|^2 \leq \sigma^2 + \zeta^2 \end{aligned} \quad (13)$$

Lastly, we derive a bound for the term  $-\mathbb{E}\langle \nabla F(x_m^t), x_m^t - x_c^t \rangle$ :

$$\begin{aligned}
 -\mathbb{E}\langle \nabla F(x_m^t), x_m^t - x_c^t \rangle &= \frac{1}{2} \mathbb{E} \|\nabla F(x_m^t)\|^2 + \frac{1}{2} \|x_m^t - x_c^t\|^2 - \frac{1}{2} \|\nabla F(x_m^t) + (x_m^t - x_c^t)\|^2 \\
 &\leq \frac{1}{2} \mathbb{E} (\|\nabla F(x_m^t)\|^2 + \|x_m^t - x_c^t\|^2) \\
 &\leq \frac{1}{2} \mathbb{E} \|\nabla F(x_m^t)\|^2 + \frac{\rho^2}{2}
 \end{aligned} \tag{14}$$

By making use of the expressions in 12, 13 and 14; we can re-write the inequality in 11 as follows:

$$\begin{aligned}
 \mathbb{E}F(x_m^{t+1}) &\leq \mathbb{E}F(x_m^t) + \frac{1}{2} (\eta - L\eta^2) (\zeta^2 - \mathbb{E}\|\nabla F(x_m^t)\|^2) + \frac{L\eta^2}{2} (\zeta^2 + \sigma^2) \\
 &\quad + \frac{\eta\lambda}{2} \mathbb{E}\|\nabla F(x_m^t)\|^2 + \frac{\eta\lambda}{2} \rho^2 + \frac{\eta^2\lambda^2L}{r} \rho^2
 \end{aligned} \tag{15}$$

$$\left( \frac{\eta - \lambda - L\eta^2}{2} \right) \mathbb{E}\|\nabla F(x_m^t)\|^2 \leq F(x_m^t) - F(x_m^{t+1}) + \left( \frac{\eta}{2} + L\eta^2 \right) \zeta^2 + \frac{L\eta^2}{2} \sigma^2 + \left( \frac{\eta\lambda}{2} + \frac{\eta^2\lambda^2L}{2} \right) \rho^2 \tag{16}$$

We set  $\eta \leq \frac{\frac{1}{3} - \lambda}{L}$  so that we satisfy  $\frac{\eta(1-\lambda-L\eta)}{2} \geq \frac{\eta}{3}$ . As a result, we can write:

$$\mathbb{E}\|\nabla F(x_m^t)\|^2 \leq \frac{3}{\eta} \left( F(x_m^t) - F(x_m^{t+1}) + \left( \frac{\eta}{2} + L\eta^2 \right) \zeta^2 + \frac{L\eta^2}{2} \sigma^2 + \left( \frac{\eta\lambda}{2} + \frac{\eta^2\lambda^2L}{2} \right) \rho^2 \right) \tag{17}$$

We then take the average of the sum of the inequality from  $t = 1, \dots, N$  and  $m = 1, \dots, M$ ; we ultimately get the following inequality where  $x^0$  and  $x^*$  are the initial model parameters and minimizer of  $F$  respectively:

$$\frac{1}{MN} \sum_{m=1}^M \sum_{t=1}^N \mathbb{E}\|\nabla F(x_m^t)\|^2 \leq \frac{3(F(x^0) - F(x^*))}{MN\eta} + \frac{3}{2} (\zeta^2 + \lambda\rho^2) + \frac{3}{2} (2L\zeta^2 + L\sigma^2 + \lambda^2\rho^2L) \eta \tag{18}$$

Note that this convergence rate analysis characterizes the average of the gradient norm square obtained by all local variables on the function  $F$ .

□

#### E.4 Relating GRAWA and MGRAWA Coefficients to Non-Convex Convergence Analysis

In this subsection, we directly relate the  $\beta_1, \beta_2, \dots, \beta_M$  coefficients that appear in the vanilla GRAWA and MGRAWA to one of the conditions imposed in Theorem 8. Notice that, as mentioned before, it is mathematically intractable to find a relation between LGRAWA weights and the convergence rate due to its layer-wise weighted averaging.

We keep the first three assumptions of Theorem 8, remove the last one, and add two more. Overall, we have the following set of assumptions:

- Function  $F$  is  $L$ -smooth.
- $\mathbb{E}_{\xi \sim \mathcal{D}_m} \|\nabla f(x, \xi) - \nabla f_m(x_m)\|^2 \leq \sigma^2$
- $\mathbb{E}_{m \sim U[M]} \|\nabla f_m(x_m) - \nabla F(x_m)\|^2 \leq \zeta^2$
- $\mathbb{E} \left[ x_i^t x_j^t \right] \leq y^2$  at any time or iteration  $t$  for all  $i, j \in [1, M]$ .
- $\mathbb{E} \left[ \beta_i^t \beta_j^t \right] \leq \frac{\rho^2 + y^2}{y^2 M^2}$  at any time or iteration  $t$  for all  $i, j \in [1, M]$ .

The last two assumptions are conditioned on the value of the dot product between any two workers at any iteration and, the value of the multiplication between any two GRAWA/MGRAWA coefficients at any iteration. In the extension, we show that having these two assumptions is equivalent to the last assumption of Theorem 8.

*Proof.* We start by writing  $x_m^t - x_c^t = x_m^t - \sum_{i=1}^M \beta_i x_i^t$  and expressing  $\|x_m^t - x_c^t\|^2$  as:

$$\begin{aligned} \|x_m^t - x_c^t\|^2 &= (x_m^t - x_c^t)^T (x_m^t - x_c^t) = x_m^{tT} x_m^t - 2 \sum_{i=1}^M \beta_i x_m^{tT} x_i^t + \left( \sum_{i=1}^M \beta_i x_i^t \right)^T \left( \sum_{i=1}^M \beta_i x_i^t \right) \\ &= x_m^{tT} x_m^t - 2 \sum_{i=1}^M \beta_i x_m^{tT} x_i^t + \sum_{i=1}^M \sum_{j=1}^M \beta_i \beta_j x_i^{tT} x_j^t \end{aligned}$$

Taking the expectation of the last expression by also using independence between  $\beta_m$  and  $x_m$ , we arrive at the following inequality:

$$\begin{aligned} \mathbb{E} [\|x_m^t - x_c^t\|^2] &= \mathbb{E} [x_m^{tT} x_m^t] - 2 \sum_{i=1}^M \beta_i \mathbb{E} [x_m^{tT} x_i^t] + \sum_{i=1}^M \sum_{j=1}^M \mathbb{E} [\beta_i \beta_j] \mathbb{E} [x_i^{tT} x_j^t] \\ &\leq y^2 - 2y^2 \sum_{i=1}^M \beta_i + \sum_{i=1}^M \sum_{j=1}^M y^2 \frac{\rho^2 + y^2}{y^2 M^2} \\ &\leq y^2 - 2y^2 \sum_{i=1}^M \beta_i + y^2 M^2 \frac{\rho^2 + y^2}{y^2 M^2} \end{aligned}$$

By definition we have  $\sum_{i=1}^M \beta_i = 1$ , so we can also write:

$$\mathbb{E} [\|x_m^t - x_c^t\|^2] \leq y^2 (1 - 2) + \rho^2 + y^2 = -y^2 + \rho^2 + y^2$$

which means that overall we have  $\mathbb{E} [\|x_m^t - x_c^t\|^2] \leq \rho^2$  that is the same as the last assumption specified in Theorem 8.  $\square$

## F Pseudo-Codes of the Competing Algorithms

Below are the pseudo-codes of two other parameter-sharing-based distributed training algorithms that compete with MGRAWA and LGRAWA. To form the center variable, the EASGD takes the average of the workers in the distributed environment both with respect to space and time whereas, the LSGD chooses the leader worker with the smallest loss value as the center variable.

**Algorithm 3** EASGD

**Input:** Pulling force  $\lambda$ , moving average strength  $\rho$ , communication period  $\tau$ , learning rate  $\eta$ , loss function  $f$

**Initialize** workers  $x_1, x_2, \dots, x_M, x_C$  from the same random model, worker-exclusive data shards  $\Psi_1, \Psi_2, \dots, \Psi_M$  and iteration counters for workers  $t_1 = t_2 = \dots = t_M = 0$

At each worker  $m$  **do**

**while** not converged **do**

    Draw a random batch  $\xi_m \in \Psi_m$

$x_m \leftarrow x_m - \eta \nabla f(x_m; \xi_m)$

$t_m \leftarrow t_m + 1$

**if**  $M\tau$  divides  $\sum_{m=1}^M t_m$  **then**

$x_a = \sum_{m=1}^M \beta_m x_m$

$x_C \leftarrow (1 - \rho)x_a + \rho x_C$

$x_m \leftarrow (1 - \lambda)x_m + \lambda x_C$

**end if**

**end while**

**Algorithm 4** LSGD

**Input:** Pulling force  $\lambda$ , proximity force strength  $\mu$ , communication period  $\tau$ , learning rate  $\eta$ , loss function  $f$

**Initialize** workers  $x_1, x_2, \dots, x_M, x_C$  from the same random model, worker-exclusive data shards  $\Psi_1, \Psi_2, \dots, \Psi_M$  and iteration counters for workers  $t_1 = t_2 = \dots = t_M = 0$

At each worker  $m$  **do**

**while** not converged **do**

    Draw a random batch  $\xi_m \in \Psi_m$

$x_m \leftarrow x_m - \eta \nabla f(x_m; \xi_m)$

$x_m \leftarrow (1 - \frac{\mu}{\tau})x_m + \frac{\mu}{\tau}x_C$  (Prox.)

$t_m \leftarrow t_m + 1$

**if**  $M\tau$  divides  $\sum_{m=1}^M t_m$  **then**

$x_C \leftarrow \operatorname{argmin}_{x_i} f(x_i, \xi_i)$

$x_m \leftarrow (1 - \lambda)x_m + \lambda x_C$

**end if**

**end while**

We also share the pseudo-codes of DP+SGD and DP+SAM configurations. The execution of the SAM optimizer in the parallel environment is not as straightforward. Per the official implementation suggestion, the ascend step is not synchronized across the distributed processes but the descend step is synchronized.

**Algorithm 5** DP+SGD

**Input:** learning rate  $\eta$ , loss function  $f$

**Initialize** workers  $x_1, x_2, \dots, x_M, x_C$  from the same random model and worker-exclusive data shards  $\Psi_1, \Psi_2, \dots, \Psi_M$

At each worker  $m$  **do**

**while** not converged **do**

    Draw a random batch  $\xi_m \in \Psi_m$

$g_m \leftarrow \nabla f(x_m; \xi_m)$

**Initiate** distributed communication

$g = \frac{1}{M} \sum_{m=1}^M g_m$

$x_m \leftarrow x_m - \eta g$

**end while**

**Algorithm 6** DP+SAM

**Input:** Pulling force  $\lambda$ , moving average strength  $\rho$ , communication period  $\tau$ , learning rate  $\eta$ , loss function  $f$

**Initialize** workers  $x_1, x_2, \dots, x_M, x_C$  from the same random model and worker-exclusive data shards  $\Psi_1, \Psi_2, \dots, \Psi_M$

At each worker  $m$  **do**

**while** not converged **do**

    Draw a random batch  $\xi_m \in \Psi_m$

$x_{m,adv} \leftarrow x_m + \rho \frac{\nabla f(x_m; \xi_m)}{\|\nabla f(x_m; \xi_m)\|}$

$g_{m,adv} \leftarrow \nabla f(x_{m,adv}; \xi_m)$

**Initiate** distributed communication

$g = \frac{1}{M} \sum_{m=1}^M g_{m,adv}$

$x_m \leftarrow x_m - \eta g$

**end while**

**G Local Optimization Step Aware GRAWA: Local GRAWA**

We also propose versions of MGRAWA and LGRAWA that use the gradient information from the local optimizer rather than accumulating the gradient on a separate batch that is exclusively used for this purpose. We refer to these variants Local-MGRAWA and Local-LGRAWA respectively. Particularly, the gradient accumulation step is not present and instead, the gradient norms are calculated and kept track of from the batches encountered by the local optimizer such as SGD.

At the implementation level, we wrap the local optimizer to access the gradient information from its state and use it to calculate gradient norms for each batch suitable to MGRAWA or LGRAWA, depending on the chosen type. Note that the model-level and layer-level gradient norm aware flat minima-seeking mechanisms are still present for MGRAWA and LGRAWA. The difference comes from directly using local optimizer gradients rather than

dedicating a phase to selecting another random batch and calculating gradients with it. Until the communication phase, each worker keeps track of the running estimate of the gradient norm encountered by its data shard and then the estimate is used for assigning weights to the worker’s model or layers depending on MGRAWA and LGRAWA selection. The pseudo-code can be found below.

---

**Algorithm 7** Local MGRAWA/LGRAWA

---

**Input:** Pulling force  $\lambda$ , grad. norm momentum  $\gamma$ , communication period  $\tau$ , learning rate  $\eta$ , loss function  $f$   
**Initialize** workers  $x_1, x_2, \dots, x_M, x_C$  from the same random model and worker-exclusive data shards  $\Psi_1, \Psi_2, \dots, \Psi_M$   
 At each worker  $m$  **do**  
    $g_{mvg} = \text{concat}[0, 0, \dots, 0]$   
**while** not converged **do**  
   Draw a random batch  $\xi_m \in \Psi_m$  and form  $g_{cur} = \text{concat}[||g_1||, ||g_2||, \dots, ||g_K||]$   
    $t_m \leftarrow t_m + 1$   
    $g_{mvg} \leftarrow \gamma g_{mvg} + (1 - \gamma)g_{cur}$   
    $g_{est} \leftarrow \gamma g_{mvg} / (1 - \gamma^{t_m})$   
    $x_m \leftarrow x_m - \eta \nabla f(x_m; \xi_m)$   
    $x_m \leftarrow (1 - \frac{\mu}{\tau})x_m + \frac{\mu}{\tau}x_C$  (Prox.)  
   **if**  $M\tau$  divides  $\sum_{m=1}^M t_m$  **then**  
     form  $x_C$  using the running gradient norm estimate  $g_{est}$  based on MGRAWA/LGRAWA policy  
      $x_m \leftarrow (1 - \lambda)x_m + \lambda x_C$   
      $g_{mvg} \leftarrow \text{concat}[0, 0, \dots, 0]$   
      $t_m \leftarrow 0$   
   **end if**  
**end while**

---