

# Bidirectional Top $_K$ Sparsification for Distributed Learning

by

William Zou

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Mathematics  
in  
Data Science

Waterloo, Ontario, Canada, 2022

© William Zou 2022

### **Author's Declaration**

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Training large neural networks requires a large amount of time. To speed up the process, distributed training is often used. One of the largest bottlenecks in distributed training is communicating gradients across different workers [35]. Different gradient compression techniques have been proposed to alleviate the communication bottleneck, including top $_K$  gradient sparsification, which truncates the gradient to the top  $K$  components before being sent to other nodes [1].

Recently, authors have adopted top $_K$  gradient sparsification to the parameter server framework by applying top $_K$  compression in both the worker-to-server and server-to-worker direction, as opposed to only the worker-to-server direction [23, 32, 36]. Current intuition and analysis suggest that adding extra compression degrades the convergence of the model [21, 32, 36]. We provide a simple counterexample where iterating with bidirectional top $_K$  SGD allows better convergence than iterating with unidirectional top $_K$  SGD. We explain this example with the theoretical framework developed by Alistarh *et al.*, remove a critical assumption the authors' made in their non-convex convergence analysis of the top $_K$  sparsification, and show that bidirectional top $_K$  SGD can achieve the same convergence bound as unidirectional top $_K$  SGD with assumptions that are potentially easier to satisfy [3]. Finally, we experimentally evaluate unidirectional top $_K$  SGD against bidirectional top $_K$  SGD and show that under careful tuning, models trained with bidirectional top $_K$  SGD will perform just as well as models trained with unidirectional top $_K$  SGD.

## **Acknowledgements**

I would like to thank my supervisors Hans de Sterck and Jun Liu whose feedback made this thesis possible.

## **Dedication**

This is dedicated to my parents and my brother.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Thesis Outline . . . . .	2
<b>2</b>	<b>Preliminaries</b>	<b>4</b>
2.1	Optimization Problem Statement . . . . .	4
2.2	Stochastic Gradient Descent . . . . .	5
2.3	Data Parallelism . . . . .	6
2.4	Model Parallelism . . . . .	7
2.5	Parameter Server Framework . . . . .	8
2.6	All-Reduce . . . . .	9
2.7	Non-Convex Analysis . . . . .	10
2.8	Non-Convex Analysis of Distributed SGD . . . . .	12
<b>3</b>	<b>Related Work</b>	<b>14</b>
3.1	Gradient Compression . . . . .	14
3.2	Gradient Quantization . . . . .	16
3.3	Gradient Sparsification . . . . .	16
3.4	Top <sub>K</sub> Sparsification . . . . .	16
3.5	Bidirectional Top <sub>K</sub> Sparsification . . . . .	18

<b>4</b>	<b>Motivating Example of Bidirectional Top<sub>K</sub> Outperforming Unidirectional Top<sub>K</sub></b>	<b>20</b>
4.1	Problem Setup . . . . .	20
4.2	Discussion . . . . .	22
<b>5</b>	<b>Bidirectional Top<sub>K</sub> Convergence Analysis</b>	<b>28</b>
5.1	Assumptions . . . . .	28
5.2	Concept and Intuition . . . . .	30
5.3	Convergence Analysis . . . . .	30
5.4	Non-Convex Convergence of Unidirectional Top <sub>K</sub> Sparsification . . . . .	36
5.5	Discussion . . . . .	39
<b>6</b>	<b>Experimental Results</b>	<b>41</b>
6.1	Testing Framework . . . . .	41
6.2	Differences Between Theoretical and Experimental Setup . . . . .	42
6.3	Experimental Setup . . . . .	42
6.4	Unidirectional vs Bidirectional Training Loss and Test Accuracy . . . . .	43
6.5	Compression Rate of Downlink Top <sub>K</sub> Sparsification . . . . .	46
6.6	Behaviour of Constants in Convergence Analysis . . . . .	48
6.7	Effect of Non-IID Dataset on $\rho$ . . . . .	54
6.7.1	FMNIST MLP Network . . . . .	54
6.7.2	FMNIST CNN Network . . . . .	56
6.7.3	MNIST MLP Network . . . . .	58
6.7.4	MNIST CNN Network . . . . .	60
<b>7</b>	<b>Conclusion</b>	<b>62</b>
7.1	Contributions . . . . .	62
7.2	Future Work . . . . .	63

<b>References</b>	<b>64</b>
<b>APPENDICES</b>	<b>68</b>
<b>A Non-Convex Analysis</b>	<b>69</b>
A.1 Bound on Constant $D$ . . . . .	69
<b>B Learning Rates</b>	<b>72</b>
<b>C Experimental Figures</b>	<b>73</b>
C.1 Testing Accuracy and Training Loss . . . . .	73
C.2 $\hat{\rho}$ Values . . . . .	77
C.3 Percent of Non-Zero Indices in Gradient After Server Aggregation . . . . .	83
C.4 $1 - \hat{\gamma}$ Values . . . . .	84
<b>D Convergence Bound Constants</b>	<b>88</b>



# Chapter 1

## Introduction

In this section we introduce the communication bottleneck associated with the parallelization of gradient descent, some background of the  $\text{top}_K$  gradient compression technique we use to alleviate the bottleneck, and the main concept. We finish this chapter with an outline of the thesis.

### 1.1 Motivation

Deep neural networks trained on large datasets are known to achieve state-of-the-art performance in accuracy. However, the large size of the models and datasets severely impact the training time of the model. To decrease the training time, distributed machine training techniques are often used, which in recent years have involved scaling stochastic gradient descent (SGD) to multiple processes [13, 5].

The standard way to scale SGD on multiple processes is through data parallelism where the training set is split across  $n$  different processes [13]. Each node will calculate a stochastic gradient from their allocated data independently and in parallel. The nodes then communicate the gradient with each other before updating model parameters.

Communication between nodes is one of the largest bottleneck in distributed machine learning [35]. Most efforts in increasing the performance speed of distributed learning comes from reducing this bottleneck. Types of methods to reduce the communication bottleneck include: 1) rearranging the topology of the nodes [25, 27], 2) reducing the size of the message that needs to be communicated [3, 2, 6], and 3) reducing the frequency of communication between nodes [17].

Two types of topologies used in distributed training are centralized (i.e. parameter server framework) and decentralized (i.e. all-reduce) which are further discussed in Chapter 3. Decentralized frameworks were developed to overcome the limitations of having a server bottleneck. However there has been a resurgence of interest in the parameter server framework in recent years [33, 18].

One of the most well-studied compression technique is sparsification, which focuses on reducing communication between worker nodes by sending only a sparse subset of the gradient [5, 35]. The most popular of these methods is  $\text{top}_K$  gradient sparsification, which truncates the gradient to the largest  $K$  components by magnitude [10, 35].  $\text{Top}_K$  gradient sparsification was originally only used during the worker-to-server (uplink) communication of the parameter server framework [23], since the gradient sent by the server is sparse if the number of participating workers is low.

Due to increased number of workers used in distributed training, Sattler *et al.* recommended adding  $\text{top}_K$  gradient sparsification during server-to-worker (downlink) communication as well, and reported that sparsifying the gradients in both uplink and downlink communication (bidirectional) reduced the final accuracy by at most 3% compared to only using uplink [23]. Theoretical frameworks from Tang *et al.* and Zheng *et al.* have been developed to analyze the convergence of bidirectionally compressed error compensated SGD, which can be used to analyze bidirectional  $\text{top}_K$  gradient sparsification [32, 36]. However, these studies suggest that adding downlink sparsification will degrade the performance of the model [21, 32, 36]. In our thesis, we provide a simple example where bidirectional  $\text{top}_K$  SGD has better convergence than unidirectional  $\text{top}_K$  SGD, provide an intuition to why this happens, and construct a convergence bound that captures this intuition. We show that under the theoretical framework provided by Alistarh *et al.* [3], bidirectional sparsification can achieve the same convergence bound as unidirectional sparsification, with assumptions that are potentially easier to satisfy empirically. Finally, we show that through careful tuning bidirectional  $\text{top}_K$  sparsification can perform as well as unidirectional  $\text{top}_K$  sparsification.

The main motivation of our work is to give the reader a better understanding why bidirectional  $\text{top}_K$  SGD can converge as well as, if not better, than unidirectional  $\text{top}_K$  SGD.

## 1.2 Thesis Outline

Chapter 2 will discuss gradient parallelization in more detail, define the optimization problem and stochastic gradient descent, and provide an introduction to non-convex analysis.

Chapter 3 will define biased and unbiased compression, briefly mention how error feedback framework is used in biased compression, and provide more background on gradient compression techniques, such as gradient quantization and gradient sparsification. Finally, we will discuss current research in unidirectional and bidirectional  $\text{top}_K$  SGD.

In Chapter 4 we will provide an example where bidirectional  $\text{top}_K$  SGD outperforms unidirectional  $\text{top}_K$  SGD, and provide some reasoning to why this happens.

The non-convex convergence analysis of bidirectional and unidirectional  $\text{top}_K$  SGD and a comparison between both is provided in Chapter 5.

In Chapter 6, we describe the experimental setup, and provide detailed experimental results comparing the convergence of bidirectional and unidirectional  $\text{top}_K$  SGD on multiple models, datasets and workers. The constants from our convergence bound in Chapter 5 are measured and shown in this section as well.

Finally, we conclude the thesis in Chapter 7 by summarizing our contributions, including, 1) an example where bidirectional  $\text{top}_K$  SGD has better convergence than unidirectional  $\text{top}_K$  SGD, 2) a non-convex analysis of bidirectional  $\text{top}_K$  SGD that shows how it can converge faster than unidirectional  $\text{top}_K$  SGD under certain conditions, 3) experimental results that show that bidirectional  $\text{top}_K$  SGD can perform as well as unidirectional  $\text{top}_K$  SGD, and 4) an estimate of the compression rate achieved by downlink sparsification based on empirical evidence.

# Chapter 2

## Preliminaries

### 2.1 Optimization Problem Statement

For this chapter, we describe the optimization framework, following Bottou *et al.* [9]. When training a neural network, we are concerned in finding a prediction function  $H$ . Assume that  $H$  has a fixed form and is parametrized by a real vector  $w \in \mathbb{R}^d$ . Our goal is to find a parameter for  $H$  such that it minimizes losses from incorrect predictions.

For some  $H(x; w) : \mathbb{R}^{d_x} \times \mathbb{R}^d \rightarrow \mathbb{R}^{d_y}$ , where  $x$  is an input sample, consider the family of prediction functions:

$$\mathcal{H} = \{H(\cdot; w) : w \in \mathbb{R}^d\}.$$

Assume a given loss function  $L(\hat{y}, y) : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \rightarrow \mathbb{R}$ , where  $\hat{y}$  is the predicted label and  $y$  is the actual label, and a set of input-output samples  $\{x_i, y_i\}_{i=1}^n$ . A deep neural network aims to minimize the objective function  $F : \mathbb{R}^d \rightarrow \mathbb{R}$  defined by

$$F(w) = \frac{1}{n} \sum_{i=1}^n L(H(x_i; w), y_i). \quad (2.1)$$

Similar to Bottou *et al.* [9], we use a simplified notation for the loss of our prediction function on a randomly chosen sample  $(x_i, y_i)$

$$L(H(x_i; w), y_i) = f(w, \xi_{[i]}), \quad (2.2)$$

where  $\xi$  is a random variable representing a sample, and  $\{\xi_{[i]}\}_{i=1}^n$  are realizations of  $\xi$  corresponding to the sample set  $\{(x_i, y_i)\}_{i=1}^n$ .

We rewrite the objective function in (2.1) as

$$F(w) = \frac{1}{n} \sum_{i=1}^n f(w, \xi_{[i]}). \quad (2.3)$$

## 2.2 Stochastic Gradient Descent

We want to minimize our objective function  $F : \mathbb{R}^d \rightarrow \mathbb{R}$ . If we have access to the stochastic gradients  $\nabla f(w, \xi)$ , we can approach this problem by the following iterative process given a  $w_0 \in \mathbb{R}^d$  with

$$w_{t+1} = w_t - \alpha_t \nabla f(w_t, \xi_{[i_t]}), \quad (2.4)$$

where  $\alpha_t > 0$  is the step size and  $\xi_{[i_t]}$  corresponds to sample  $(x_{i_t}, y_{i_t})$  from the sample set  $\{(x_i, y_i)_{i \in S}\}$  and  $\{i_t\}$  is a sequence chosen randomly from  $\{1, \dots, n\}$  with replacement. The sequence  $\{w_t\}$  is not deterministic after fixing  $w_0$  and  $\alpha_t$ , and is dependent on the sequence  $\{i_t\}$ .

We might want to consider more than one sample in each SGD iteration. Define  $\xi_{t,i}$  to be the realization of the  $i$ -th sample at iteration  $t$ , and  $m$  to be the number of samples to consider in each SGD iteration. We have

$$w_{t+1} = w_t - \frac{\alpha_t}{m} \sum_{i=1}^m \nabla f(w_t, \xi_{t,i}). \quad (2.5)$$

For simplicity define

$$g(w_t, \xi_t) = \frac{1}{m} \sum_{i=1}^m \nabla f(w_t, \xi_{t,i}), \quad (2.6)$$

where  $\{\xi_t\}_{t \geq 0}$  are independent and identically distributed random variables (i.i.d.).

The algorithm for SGD is described in Algorithm 1. To hide non-essential implementation details, we assume that the stochastic gradient can be obtained by querying an oracle and that there is an automatic mechanism to generate the learning rate  $\alpha_t$ . Specifically,

the stochastic gradient oracle will hide the process of: 1) realizing the random variable  $\xi_t$  and 2) computing the stochastic gradient  $g(w_t, \xi_t)$  given an iterate  $w_t \in R^d$  and the realization of  $\xi_t$ .

---

**Algorithm 1:** Stochastic Gradient Descent

---

**Input:** Stochastic Gradient Oracle  $g(\cdot, \cdot)$ , learning rate sequence  $\{\alpha_t\}_{t \geq 0}$

```

1 Initialize  $w_0 \in \mathbb{R}^d$ ;  $t = 1$ ;
2 while  $t \geq 1$  do
3    $w_t \leftarrow w_{t-1} - \alpha_{t-1} g(w_{t-1}, \xi_{t-1})$ ;
4    $t = t + 1$ ;
5 end
```

---

## 2.3 Data Parallelism

Data parallelism splits the data across  $N$  different nodes. Every node updates a model with stochastic gradients calculated from local data. There are two types of data parallelism: synchronous data parallelism and asynchronous data parallelism. In synchronous data parallelism, the local parameter vector  $w_t$  is kept consistent across all nodes. This requires a locking mechanism to keep all nodes in step with each other [13]. In synchronous data parallelism, we want to minimize the weighted average of  $N$  different functions across  $N$  nodes:

$$\min_{w \in R^d} F(w) \triangleq \sum_{q=1}^N p_q F^q(w), \quad (2.7)$$

and the update step for synchronous data parallelism with  $N$  nodes is described in

$$w_t = w_{t-1} - \alpha_t \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q). \quad (2.8)$$

The probability vector  $(p_0, \dots, p_N)$  is used to assign weighting to the gradients contributed by different nodes. The probability vector is usually defined by the number of samples each node uses to calculate the their stochastic gradient. For example, assume there are 2 workers, and the workers choose  $n_1$  and  $n_2$  samples with replacement respectively, with  $n_1 + n_2 = n$ . Then

$$\begin{aligned}
F(w) &= \frac{1}{n} \sum_{i=1}^n f(w, \xi_{[i]}) \\
&= \frac{n_1}{n} \left( \frac{1}{n_1} \sum_{i=1}^{n_1} f^1(w, \xi_{[i]}) \right) + \frac{n_2}{n} \left( \frac{1}{n_2} \sum_{i=1}^{n_2} f^2(w, \xi_{[i]}) \right) \\
&= p_1 F^1(w) + p_2 F^2(w).
\end{aligned} \tag{2.9}$$

Usually the gradients have equal weighting and the values in the probability vector is  $1/N$ .

The downside of synchronous data parallelism is that computational resources are not fully utilized. The nodes are idle when waiting to receive gradients, and the waiting time increases as the number of nodes increases as well [13]. One way to reduce the waiting time is to remove the lock, and this is done through asynchronous data parallelism. Nodes will pull model parameters from a model stored on the central server and commit gradients asynchronously to central server. The committed gradients are stored in a First-In-First-Out queue on a central server [13].

The problem with asynchronous distributed training is that we are committing old gradients to the central server. Instead of iterating with

$$w_t = w_{t-1} - \alpha_t g(w_{t-1}, \xi_{t-1}), \tag{2.10}$$

to update the central parameter, we are iterating with

$$w_t = w_{t-1} - \alpha_t g(w_{t-1-\tau}, \xi_{t-1-\tau}), \tag{2.11}$$

where  $\tau$  is the delay from the timestep the worker pulls the central parameter vector to the timestep the worker commits the gradient to the central server. The downside of asynchronous distributed training is that when  $\tau$  is large, the stochastic sequence  $\{w_t\}$  may no longer converge to the optimal value. Some implementations of asynchronous distributed training may involve throwing out the gradient without updating the central server if the delay is too large [13].

## 2.4 Model Parallelism

Data parallelism is often mentioned with model parallelism. While model parallelism is not within the scope of this thesis, we will briefly mention it here to contrast it with data parallelism, and how it can be used together with it.

In model parallelism, a single model is split across multiple nodes. The speed-up of model parallelism depends on the model structure. Because of layer interdependence, forward and backward propagation in neural networks are sequential operations. Hence it is difficult to achieve effective parallelization with model parallelism [5]. Sometimes model parallelism is used together with data parallelism with a technique called parameter sharding. In this setup, each  $N$  nodes in the network acts as both a client and a server. The model is split across the servers, and each server is responsible for  $1/N$  of the parameters. The client has a copy of all parameters, and pulls updates from each server. This keeps the bandwidth per node constant, since each node communicates with each of the other nodes for  $1/N$  of the parameters [1].

## 2.5 Parameter Server Framework

Parameter server framework is a common and well studied topology used in data parallelism. The data is distributed over nodes called workers. Another node, called the server, is used to coordinate the workers. The workers will calculate gradients from local data and send it to the server to be aggregated. The server will send the aggregated gradient to the workers. The workers will then update the model stored locally [13]. We describe the parameter server framework in Algorithm 2. Each SEND operation has a matching RECV operation, and both operations block until the send and receive has been completed.

One major bottleneck in performance speed of the parameter server model is the server. The server must wait to receive gradients from all workers. During this time, many worker nodes are idling and we are not making full use of our computational resources [13]. One way to alleviate this bottleneck is to use parameter sharding and split the parameters across multiple servers [25]. However, it becomes difficult to identify the right ratio of servers to workers. Too many servers can saturate the network and too little servers can cause communication bottlenecks [25]. This caused engineers to move away from centralized learning to decentralized learning, though there has been a resurgence of interest in the parameter-server framework recently [23, 33].



---

**Algorithm 2:** SGD Worker Side (Worker  $q$ )

---

**Input:** Stochastic Gradient Oracle  $g^q(\cdot, \cdot)$ , learning rate sequence  $\{\alpha_t\}_{t \geq 0}$

```
1 Initialize  $w_0 \in \mathbb{R}^d$ ;  $t = 1$ ;  
2 while  $t \geq 1$  do  
3   | SEND( $g^q(w_{t-1}, \xi_{t-1}^q)$ , server);  
4   | RECV( $g_t$ , server);  
5   |  $w_t \leftarrow w_{t-1} - \alpha_{t-1} g_t$ ;  
6   |  $t = t + 1$ ;  
7 end
```

---

---

**Algorithm 3:** SGD Server Side

---

**Input:** probability vector  $(p_0, \dots, p_N)$

```
1 Initialize  $t = 1$ ;  
2 while  $t \geq 1$  do  
3   | for every worker  $q$  do  
4   |   | RECV( $g^q(x_{t-1}, \xi_{t-1}^q)$ ,  $q$ );  
5   | end  
6   |  $g_t \leftarrow \sum_{q=1}^N p_q g^q(x_{t-1}, \xi_{t-1}^q)$ ;  
7   | for every worker  $q$  do  
8   |   | SEND( $g_t$ ,  $q$ );  
9   | end  
10  |  $t = t + 1$   
11 end
```

---

## 2.6 All-Reduce

All-reduce is an operation from the Message Passing Interface (MPI) standard that reduces arrays from all nodes to a single array and sends the combined array to all nodes. We describe the all-reduce algorithm with  $N$  nodes in Algorithm 4.

---

**Algorithm 4:** SGD All-Reduce (worker  $q$ )

---

**Input:** Stochastic Gradient Oracle  $g^q(\cdot, \cdot)$ , learning rate sequence  $\{\alpha_t\}_{t \geq 0}$ , probability vector  $(p_0, \dots, p_N)$

```
1 Initialize  $w_0 \in \mathbb{R}^d$ ;  
2 while  $t \geq 1$  do  
3   | BROADCAST( $p_q g^q(w_{t-1}, \xi_{t-1}^q)$ , SUM);  
4   |  $w_t \leftarrow w_{t-1} + \alpha_t \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)$ ;  
5 end
```

---

BROADCAST is a two part operation: 1) worker  $q$  sends its stochastic gradient to all other nodes, and 2) waits to receive a stochastic gradient from all other nodes before adding it to SUM. The BROADCAST operation needs a lock to block progression until all nodes have received all gradients from other nodes.

An example of a distributed framework that can perform all-reduce is Horovod, which uses a ring topology. Experimental results by Sergeev and Balso show that ring all-reduce can use GPU resources more efficiently than the parameter-server framework [25].

However, we point out that all-reduce is not efficient when there is a large number of unreliable machines. An example of this is Federated Learning, which aims to train models on data stored in mobile devices such as personal phones and tablets [23]. These devices are frequently offline and have slow and expensive communication. It is impractical to set up a network that allows participating mobile devices to use MPI all-reduce. Instead, it is easier to have these devices communicate with a central server [18, 23].

Additionally, the recent advancements in cloud networking technologies have also seen a resurgence of parameter server approach in distributed training. Amazon developed a novel communication library Herring, which uses a parameter server topology and demonstrated that it can be twice as fast as all-reduce-based methods [33].

## 2.7 Non-Convex Analysis

We round off the introduction with a discussion of how to analyze the convergence rate of a gradient descent method in the non-convex setting.

To establish convergence guarantees for optimizing a non-convex function  $F : \mathbb{R}^d \rightarrow \mathbb{R}$ , we usually make assumptions similar to the following.

**Assumption 2.1 (Existence of a lower bound).** *There exists some constant  $F^*$  such that  $F(w) \geq F^*$  for all  $w \in \mathbb{R}^d$ .*

This assumption is necessary to ensure there is a minimum value to find.

**Assumption 2.2 (Lipschitz continuous gradient).**  *$F$  is continuously differentiable and the gradient  $\nabla F : \mathbb{R}^d \rightarrow \mathbb{R}^d$  is Lipschitz continuous with constant  $L > 0$ , i.e.*

$$\|\nabla F(w) - \nabla F(v)\|_2 \leq L\|w - v\|_2 \quad \forall w, v \in \mathbb{R}^d. \quad (2.12)$$

Assumption 2.2 is useful in showing that gradient based optimization converges. Changing the parameter vector from  $w$  to  $v$  will not cause an arbitrary change from  $\nabla F(w)$  to  $\nabla F(v)$  [9]. From Assumption 2.2, we can get

$$F(w) \leq F(v) + \langle \nabla F(v), w - v \rangle + \frac{1}{2}L\|w - v\|_2^2 \quad \forall w, v \in \mathbb{R}^d. \quad (2.13)$$

**Assumption 2.3 (First and second moment).** *The objective function and Algorithm 1 satisfy the following:*

$$\mathbb{E}[g(w, \xi_t)] = \nabla F(w) \quad \forall w \in \mathbb{R}^d, \forall t \in \mathbb{N},$$

and

$$\mathbb{E}[\|g(w, \xi_t)\|_2^2] \leq M \quad \forall w \in \mathbb{R}^d, \forall t \in \mathbb{N}.$$

Generally for a gradient descent method with non-convex objective, we want to show under Assumption 2.1, 2.2 and 2.3,

$$\min_{t \in \{1 \dots T\}} \mathbb{E}[\|\nabla F(w_t)\|_2^2] \xrightarrow{T \rightarrow \infty} 0, \quad (2.14)$$

where  $\mathbb{E}[\cdot]$  here is the expectation taken with respect to the joint distribution of all random variables  $\{\xi_0, \xi_1, \dots, \xi_{t-1}\}$ . Most strategies involve showing

$$\lim_{T \rightarrow \infty} \mathbb{E} \left[ \frac{1}{\sum_{t=1}^T \alpha_t} \sum_{t=1}^T \alpha_t \|\nabla F(w_t)\|_2^2 \right] = 0, \quad (2.15)$$

where  $\{\alpha_t\}$  is a diminishing stepsize sequence.

## 2.8 Non-Convex Analysis of Distributed SGD

For distributed SGD, we want to minimize the weighted average of  $N$  objective functions across  $N$  nodes:

$$\min_{w \in \mathbb{R}^d} F(w) \triangleq \sum_{q=1}^N p_q F^q(w),$$

so we make corresponding adjustments to Assumption 2.3.

**Assumption 2.4 (First and second moment).** *The objective function and Algorithm 4 satisfy the following:*

$$\mathbb{E}\left[\sum_{q=1}^N p_q g^q(w, \xi_t^q)\right] = \sum_{q=1}^N p_q \nabla F^q(w) \quad \forall w \in \mathbb{R}^d, \forall t \in \mathbb{N}, \quad (2.16)$$

and

$$\mathbb{E}\left[\left\|\sum_{q=1}^N p_q g^q(w, \xi_t^q)\right\|_2^2\right] \leq M \quad \forall w \in \mathbb{R}^d, \forall t \in \mathbb{N}. \quad (2.17)$$

where  $\mathbb{E}[\cdot]$  is taken with respect to the joint distribution of  $\{\xi_t^1, \xi_t^2, \dots, \xi_t^N\}$ .

The following theorem for the convergence of distributed SGD is easily adapted from existing analysis [9, 14].

**Theorem 2.1.** *Under Assumptions 2.1, 2.2, and 2.4, running (2.8) for  $T$  iterations gives us*

$$\frac{1}{\sum_{t=0}^T \alpha_t} \sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] \leq \frac{1}{\sum_{t=0}^T \alpha_t} (F(w_0) - F^*) + \frac{LM}{2} \frac{\sum_{t=0}^T \alpha_t^2}{\sum_{t=0}^T \alpha_t}. \quad (2.18)$$

*Proof.* Starting with (2.13), and denote  $\mathbb{E}_{\xi_t}[\cdot]$  to be the expectation taken with respect to the joint distribution of  $\{\xi_t^1, \xi_t^2, \dots, \xi_t^N\}$  given random variables before time  $t$ , we have

$$\begin{aligned}
\mathbb{E}_t[F(w_{t+1})] &\leq F(w_t) + \langle \nabla F(w_t), \mathbb{E}_{\xi_t}[w_{t+1} - w_t] \rangle + \frac{L}{2} \mathbb{E}_{\xi_t}[\|w_{t+1} - w_t\|_2^2] \\
&= F(w_t) - \alpha_t \langle \nabla F(w_t), \mathbb{E}_{\xi_t}[\sum_{i=1}^N p_q g(w_t, \xi_t^q)] \rangle + \\
&\quad \frac{L\alpha_t^2}{2} \mathbb{E}_{\xi_t}[\|\sum_{i=1}^N p_q g(w_t, \xi_t^q)\|_2^2] \\
&\leq F(w_t) - \alpha_t \langle \nabla F(w_t), \nabla F(w_t) \rangle + \frac{L\alpha_t^2 M}{2} \\
&\leq F(w_t) - \alpha_t \|\nabla F(w_t)\|_2^2 + \frac{L\alpha_t^2 M}{2}.
\end{aligned} \tag{2.19}$$

Taking expectation with respect to the joint distribution of all random variables, we get

$$\mathbb{E}[F(w_{t+1})] \leq \mathbb{E}[F(w_t)] - \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] + \frac{L\alpha_t^2 M}{2}. \tag{2.20}$$

Telescoping and rearranging, we have

$$\sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] \leq \sum_{t=0}^T \left( \mathbb{E}[F(w_t)] - \mathbb{E}[F(w_{t+1})] + \frac{L\alpha_t^2 M}{2} \right).$$

or

$$\frac{1}{\sum_{t=0}^T \alpha_t} \sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] \leq \frac{1}{\sum_{t=0}^T \alpha_t} (F(w_0) - F^*) + \frac{LM}{2} \frac{\sum_{t=0}^T \alpha_t^2}{\sum_{t=0}^T \alpha_t}, \tag{2.21}$$

as desired. □

In order for the upper bound in (2.21) to converge to 0, we need

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty, \tag{2.22}$$

so we can choose  $\alpha_t = \frac{1}{(t+1)^\theta}$ , where  $\frac{1}{2} < \theta \leq 1$ .

# Chapter 3

## Related Work

### 3.1 Gradient Compression

We can reduce the communication time of data parallelism through gradient compression. Sending and receiving gradients between nodes is the most costly operation in a distributed setting, and reducing the amount of information that needs to be sent will alleviate the communication bottleneck [35].

Gradient compression is typically divided into biased compression and unbiased compression. We use the definition provided by Beznosikov *et al.* [7].

Let  $\zeta \geq 1$ , a compressor  $C(\cdot)$  is unbiased if

$$\mathbb{E}[C(w)] = w \quad \mathbb{E}[\|C(w)\|_2^2] \leq \zeta \|w\|_2^2 \quad \forall w \in \mathbb{R}^d. \quad (3.1)$$

A compressor  $C(\cdot)$  is biased if there exists  $\gamma \in (0, 1)$  such that

$$\mathbb{E}[\|C(w) - w\|_2^2] \leq (1 - \gamma) \|w\|_2^2 \quad \forall w \in \mathbb{R}^d. \quad (3.2)$$

We briefly mention that there are two other definitions of biased compression [7]. The type that satisfies (3.2) is the one we will use for the rest of the thesis. When  $\gamma = 0$ ,  $C(x) = 0 \in \mathbb{R}^d$  and no information is left after compression. When  $\gamma = 1$ ,  $C(x) = x$

and there is no compression. Intuitively, the inequality implies that a  $\gamma$ -fraction of  $x$  is preserved in the compression.

The expectation in (3.1) and (3.2) is taken with respect to the randomness in the compression operation. To illustrate this, we use an example provided by Beznosikov *et al.* [7]. The unbiased random sparsification compression operator ( $\text{rand}_K$ ) is defined by

$$C(w) := \frac{d}{K} \sum_{i \in S} w_i e_i, \quad (3.3)$$

where  $S$  is a subset of  $\{1, \dots, d\}$  with cardinality  $K$  chosen uniformly at random, and  $e_1, \dots, e_d$  are standard unit basis vectors in  $\mathbb{R}^d$ . The values outputted by the random sparsification operator are measured with respect to a probability distribution, which allows us to take the expectation in (3.1). For some compressors, the output value is deterministic given an input, and we can remove the expectation in (3.1) and (3.2). Experimental results show biased compressors often outperform their unbiased counterparts, and provide better testing accuracy and training loss [7].

Naively using biased gradient compression may cause the algorithm to not generalize or converge. Some sort of correction mechanism is needed to ensure the original gradient is not forgotten [14]. An example of such a mechanism is the error-feedback framework described in Algorithm 5. Here, the node accumulates the difference between the compressed gradient and the actual gradient, then adds the accumulated value into the gradient in the next iteration.

---

**Algorithm 5:** Compressed SGD with Error Feedback (without distributed communication for simplicity)

---

**Input:** Stochastic Gradient Oracle  $g(\cdot, \cdot)$ , compressor  $C(\cdot)$ , learning rate sequence  $\{\alpha_t\}_{t \geq 0}$

---

```

1 Initialize  $w_0 \in \mathbb{R}^d$ ;  $\epsilon_0 = 0 \in \mathbb{R}^d$ ;  $t = 1$ ;
2 while  $t \geq 1$  do
3    $p_t \leftarrow \alpha_{t-1} g(w_{t-1}, \xi_{t-1}) + \epsilon_{t-1}$ ;
4    $w_t \leftarrow w_{t-1} - C(p_t)$ ;
5    $\epsilon_t \leftarrow p_t - C(p_t)$ ;
6    $t = t + 1$ ;
7 end
```

---

## 3.2 Gradient Quantization

Quantization applies lossy compression by mapping each gradient component to a smaller set of values [5]. The most drastic quantization compression proposed is 1-bit quantization, which was proposed by Seide *et al.* and Strom *et al.* [24, 31]. Seide *et al.* suggests using a quantization threshold of 0: positive values are encoded as 1 and negative values as 0 [24]. Convergence for 1-bit quantization using 0 as a threshold was developed by Bernstein *et al.* [6], though unlike Seide *et al.*, they did not consider using an error feedback framework. The authors call the method signSGD. Kamireddy *et al.* later show that signSGD without error feedback generalizes poorly to certain optimization problems, and develops convergence theory for signSGD with error feedback [14]. We also mention that Alistarh *et al.* introduce an unbiased stochastic compression scheme called Quantized SGD, and describes the relationship between bits compressed and convergence time [2].

One problem with gradient quantization is that its compression is limited by the precision of the floating point representation of the gradient component, since 1-bit is the smallest representation of each component. Quantization can only compress gradients represented by 32-bit floating point values to 1/32 of its original size [10]. Because of this, Sattler *et al.* describes 1-bit quantization as “weak” compression [23]. To compress gradients further, we look at gradient sparsification.

## 3.3 Gradient Sparsification

The intuition behind gradient sparsification is that neural network gradients are sparse. There is a large number of parameters in a model that do not necessarily change after each update, and only part of the gradient is needed to achieve convergence [5]. Sparsification has been shown to achieve compression rates up to [0.01, 0.001] without loss in accuracy [10]. Other results have shown that sparsification is able to achieve 54× speedup for 80 nodes [5].

## 3.4 Top<sub>K</sub> Sparsification

Top<sub>K</sub> gradient sparsification truncates the gradient to its top<sub>K</sub> components, sorted by order of increasing magnitude. The nodes communicate this truncated sparse gradient instead of the full gradient [5]. Generally, in top<sub>K</sub> sparsification, updates are delayed and not



discarded. The error from the truncation is accumulated in an error term and added to the gradient before truncation. However, it is important to note that not every update in  $\text{top}_K$  sparsification is guaranteed to be applied [3]. A small update can be delayed forever.  $\text{Top}_K$  sparsification is well studied in literature [35].

Alistarh *et al.* and Stich *et al.* analyze the convergence rate of  $\text{top}_K$  sparsification, and show that the scheme converges at the same rate as vanilla SGD under certain settings [3, 30]. Rengli *et al.* builds a communication library which extends MPI to support sparse communication [22]. Shi *et al.* describes how to implement  $\text{top}_K$  sparsification with all-reduce in a tree topology to reduce the communication complexity even further [27, 28].

Finally, we note that in practice, there are two key disadvantages to  $\text{top}_K$  sparsification: 1) overhead in finding the top  $K$  values in a gradient is a costly operation, since we need to sort the values in tensor, and 2)  $\text{top}_K$  sparsification needs to encode and send the position of each of its non-zero gradient component [10]. These two reasons can make  $\text{rand}_K$  more attractive than  $\text{top}_K$ :  $\text{rand}_K$  eliminates the need to sort, and if all workers generates the positions of the non-zero gradient components with the same seed, there is no need to communicate the position with each other [10]. However,  $\text{rand}_K$  is shown to introduce large compression errors [20]. There has been multiple studies to approximate the  $\text{top}_K$  operator: Shi *et al.* and Lin *et al.* introduces methods to estimate a threshold to estimate which gradient components to keep [26, 16]. Ozfatura *et al.* introduces a sparsification technique that sends a mask to all workers specifying the location of gradient components to keep, with each worker sending a few gradient components that are not part of the mask [20]. However,  $\text{top}_K$  sparsification is still the most common sparsification technique studied in distributed learning [10]. The version of the  $\text{top}_K$  sparsification algorithm implemented in the parameter server framework is described in Algorithms 6 and 7.

---

**Algorithm 6:**  $\text{Top}_K$  SGD Worker Side (Worker  $q$ )

---

**Input:** Stochastic Gradient Oracle  $g^q(\cdot; \cdot)$ , learning rate sequence  $\{\alpha_t\}_{t \geq 0}$

---

```

1 Initialize  $w_0 \in \mathbb{R}^d$ ;  $\epsilon_0^q = 0 \in \mathbb{R}^d$ ;  $t = 1$ ;
2 while  $t \geq 1$  do
3    $a_t^q \leftarrow \epsilon_{t-1}^q + \alpha_{t-1} g^q(w_{t-1}, \xi_{t-1}^q)$ ;
4    $\epsilon_t^q \leftarrow a_t^q - \text{Top}_K(a_t^q)$ ;
5   SEND( $\text{Top}_K(a_t^q)$ , server);
6   RECV( $g_t$ , server);
7    $w_t \leftarrow w_{t-1} - g_t$ ;
8    $t = t + 1$ ;
9 end
```

---

---

**Algorithm 7:** Top<sub>K</sub> SGD Server Side

---

**Input:** probability vector  $(p_0, \dots, p_N)$

```
1 Initialize  $t = 1$ ;  
2 while  $t \geq 1$  do  
3   for every worker  $q$  do  
4     | RECV(TopK( $a_t^q$ ),  $q$ );  
5   end  
6    $g_t \leftarrow \sum_{q=1}^N p_q \text{Top}_K(a_t^q)$ ;  
7   for every client  $q$  do  
8     | SEND( $g_t$ ,  $q$ );  
9   end  
10   $t = t + 1$ ;  
11 end
```

---

### 3.5 Bidirectional Top<sub>K</sub> Sparsification

Early uses of top<sub>K</sub> sparsification did not consider compressing the gradient in the server-to-worker direction, since the sum of gradients received by the server is already a sparse gradient if the number of participating workers is small. However, as the number of workers in the network has increased over the years, it has become important to consider the downlink compression [5, 23]. We describe bidirectional compression in Algorithms 8 and 9.

Bidirectional sparsification has been used in the Federated Learning setting. Sattler *et al.* evaluated communication protocols for Federated Learning, and suggested adding top<sub>K</sub> sparsification in the server-to-worker direction [23]. They reported that using bidirectional top<sub>K</sub> sparsification did not destabilize the training as much as using bidirectional signSGD.

Beyond applications in Federated Learning, theoretical frameworks to evaluate the convergence rate of bidirectional biased compression techniques with error-feedback framework have been developed in [32, 36]. As far as we know, Tang *et al.* is the first to have developed convergence analysis for bidirectional error feedback SGD compatible with any compression technique, and admits the same convergence rate as SGD under certain assumptions [32]. Zheng *et al.* developed a similar theoretical framework to Tang *et al.* independently [36].

Existing work on bidirectional compression frameworks claim that adding server-to-worker compression degrades the model performance [21], while other analysis show a worse

convergence bound [32, 36]. However, we claim that bidirectional  $\text{top}_K$  sparsification can potentially have better convergence guarantees than unidirectional  $\text{top}_K$  sparsification, which to the best of our knowledge is not captured by previous analysis. We motivate this with an example that shows when bidirectional  $\text{top}_K$  SGD can converge closer to the optimum value than unidirectional  $\text{top}_K$  SGD at the start of the next chapter.

---

**Algorithm 8:**  $\text{Top}_K$  Bidirectional SGD Worker Side (Worker  $q$ )

---

**Input:** Stochastic Gradient Oracle  $g^q(\cdot; \cdot)$ , learning rate sequence  $\{\alpha_t\}_{t \geq 0}$

- 1 Initialize  $w_0 \in \mathbb{R}^d$ ;  $\epsilon_0^q = 0 \in \mathbb{R}^d$ ;  $t = 1$ ;
- 2 **while**  $t \geq 1$  **do**
- 3      $a_t^q \leftarrow \epsilon_{t-1}^q + \alpha_{t-1} g^q(w_{t-1}, \xi_{t-1}^q)$ ;
- 4      $\epsilon_t^q \leftarrow a_t^q - \text{Top}_K(a_t^q)$ ;
- 5     SEND( $\text{Top}_K(a_t^q)$ , server);
- 6     RECV( $\text{Top}_K(g_t)$ , server);
- 7      $w_t \leftarrow w_{t-1} - \text{Top}_K(g_t)$ ;
- 8      $t = t + 1$ ;
- 9 **end**

---



---

**Algorithm 9:**  $\text{Top}_K$  Bidirectional SGD Server Side

---

**Input:** probability vector  $(p_0, \dots, p_N)$

- 1 Initialize  $\delta_0 = 0 \in \mathbb{R}^d$ ;  $t = 1$ ;
- 2 **while**  $t \geq 1$  **do**
- 3     **for every worker**  $q$  **do**
- 4         RECV( $\text{Top}_K(a_t^q)$ ,  $q$ );
- 5     **end**
- 6      $g_t \leftarrow \sum_{q=1}^N p_q \text{Top}_K(a_t^q) + \delta_{t-1}$ ;
- 7      $\delta_t \leftarrow g_t - \text{Top}_K(g_t)$ ;
- 8     **for every client**  $q$  **do**
- 9         SEND( $\text{Top}_K(g_t)$ ,  $q$ );
- 10    **end**
- 11     $t = t + 1$ ;
- 12 **end**

---

## Chapter 4

# Motivating Example of Bidirectional Top<sub>K</sub> Outperforming Unidirectional Top<sub>K</sub>

In this section we study a simple case where bidirectional top<sub>K</sub> SGD can outperform unidirectional top<sub>K</sub> SGD, give reasoning to why this happens, and introduce an assumption that can capture this reason.

### 4.1 Problem Setup

Consider solving the distributed problem with 3 workers

$$\min_{w \in R^{100}} F(w) \triangleq \frac{1}{3} \sum_{q=1}^3 F^q(w),$$

where

$$\begin{aligned} F^1(w) &\triangleq \frac{1}{2}(w - \vec{1})^T(w - \vec{1}), \\ F^2(w) &\triangleq \frac{1}{2}(w - \vec{5})^T(w - \vec{5}), \\ F^3(w) &\triangleq \frac{1}{2}(w - \vec{10})^T(w - \vec{10}). \end{aligned}$$

We can find that the minimum value of  $F(w)$  is  $\frac{6100}{9}$ , when  $w = \frac{16}{3}$ . We initialize  $w_0$  as  $2\vec{0}$ , and run bidirectional  $\text{top}_K$  SGD from Algorithms 8 and 9, where each update step is

$$w_t = w_{t-1} - \text{Top}_K(\delta_{t-1} + \frac{1}{3} \sum_{q=1}^3 \text{Top}_K(\epsilon_{t-1}^q + \alpha_{t-1} \nabla F^q(w_{t-1}))),$$

$\text{top}_K$  SGD with uplink compression from Algorithms 6 and 7, where each update step is

$$w_t = w_{t-1} - \frac{1}{3} \sum_{q=1}^3 \text{Top}_K(\epsilon_{t-1}^q + \alpha_{t-1} \nabla F^q(w_{t-1})),$$

and  $\text{top}_K$  SGD with downlink compression from Algorithms 10 and 11, where each update step is

$$w_t = w_{t-1} - \text{Top}_K(\delta_{t-1} + \alpha_{t-1} \frac{1}{3} \sum_{q=1}^3 \nabla F^q(w_{t-1})).$$

Each SGD algorithm is run for 500 iterations on step size  $\alpha = 0.01$  and  $K = 1$ , and  $F(w_t)$  is plotted after each iteration in Figure 4.1. Some initial iterations are removed so  $F(w_t)$  is close to the minimum value  $F^*$ .

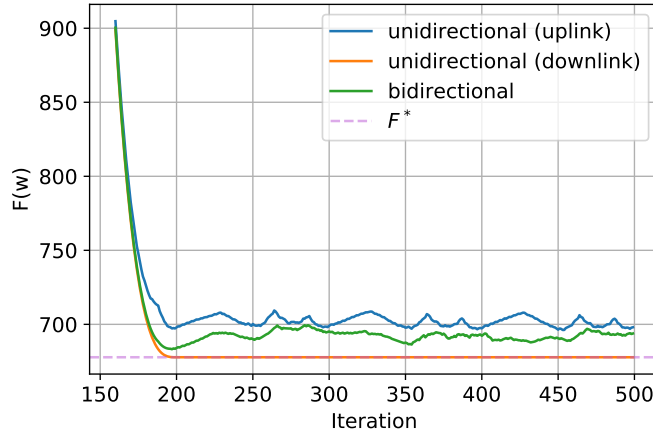


Figure 4.1: Convergence of unidirectional vs bidirectional  $\text{top}_K$  SGD run with  $K = 1$  and  $\alpha = 0.01$ .

---

**Algorithm 10:** Top<sub>K</sub> Unidirectional (Downlink) SGD Worker Side (Worker  $q$ )

---

**Input:** Stochastic Gradient Oracle  $g^q(\cdot; \cdot)$ , learning rate sequence  $\{\alpha_t\}_{t \geq 0}$

```
1 Initialize  $w_0$ ;  $t = 1$ ;  
2 while  $t \geq 1$  do  
3   SEND( $\alpha_{t-1}g^q(w_{t-1}, \xi_{t-1}^q)$ );  
4   RECV(TopK( $g_t$ ), server);  
5    $w_t \leftarrow w_{t-1} - \text{Top}_K(g_t)$ ;  
6    $t = t + 1$ ;  
7 end
```

---

---

**Algorithm 11:** Top<sub>K</sub> Unidirectional (Downlink) SGD Server Side

---

**Input:** probability vector  $(p_0, \dots, p_N)$

```
1 Initialize  $\delta_0 = 0 \in \mathbb{R}^d$ ;  $t = 1$ ;  
2 while  $t \geq 1$  do  
3   for every worker  $q$  do  
4     RECV( $\alpha_{t-1}g^q(w_{t-1}, \xi_{t-1}^q)$ ,  $q$ );  
5   end  
6    $g_t \leftarrow \sum_{q=1}^N p_q \alpha_{t-1} g^q(w_{t-1}, \xi_{t-1}^q) + \delta_{t-1}$ ;  
7    $\delta_t \leftarrow g_t - \text{Top}_K(g_t)$ ;  
8   for every client  $q$  do  
9     SEND(TopK( $g_t$ ),  $q$ );  
10  end  
11   $t = t + 1$ ;  
12 end
```

---

## 4.2 Discussion

We can see that while both unidirectional (uplink) and bidirectional top<sub>K</sub> SGD oscillates periodically at a distance away from the optimal value. However, bidirectional top<sub>K</sub> SGD converges closer to the  $F^*$  than unidirectional uplink top<sub>K</sub> SGD, which is counter-intuitive, since we should be losing information by adding downlink compression.

We provide an explanation of this by observing the gradients from each worker in unidirectional top<sub>K</sub> SGD at  $t = 300$ .  $a_t^q$  from each worker  $q$  in line 3 of Algorithm 6 is shown in Figure 4.2. We note that the non-zero components from the sum of top<sub>K</sub> updates,  $\sum_{i=1}^3 \text{Top}_K(a_t^q)$ , shown in Figure 4.3 is much larger in magnitude than the corresponding

components in the sum of updates,  $\sum_{i=1}^3 a_t^q$ , shown in Figure 4.4. This happens because the components from worker 1 and worker 3 have opposite signs. If the largest gradient component from workers 1 and 3 do not have the same index, then  $\sum_{i=1}^3 \text{Top}_K(a_t^q)$  will be far from  $\sum_{i=1}^3 a_t^q$ . In our example, the norm of the difference between the full update and unidirectional uplink update step is greater than the norm of the difference between the full and the bidirectional update. Specifically, for unidirectional uplink top $_K$  SGD, at  $t = 300$ ,

$$\left\| \sum_{q=1}^3 a_t^q - \sum_{q=1}^3 \text{Top}_K(a_t^q) \right\|_2 = 20.08,$$

while

$$\left\| \sum_{q=1}^3 a_t^q - \text{Top}_K\left(\sum_{q=1}^3 \text{Top}_K(a_t^q)\right) \right\|_2 = 19.93,$$

showing that adding downlink top $_K$  sparsification brings the update closer to the uncompressed update. While applying downlink compression causes the gradient to lose information, it is also causing the gradient to lose “bad” information. This motivates us to split the error into 2 parts for unidirectional top $_K$  SGD,

$$\begin{aligned} & \left\| \sum_{q=1}^N p_q a_t^q - \sum_{q=1}^N p_q \text{Top}_K(a_t^q) \right\|_2 \\ & \approx \left\| \sum_{q=1}^N p_q a_t^q - \text{Top}_K\left(\sum_{q=1}^N p_q a_t^q\right) \right\|_2 + \left\| \text{Top}_K\left(\sum_{q=1}^N p_q a_t^q\right) - \sum_{q=1}^N p_q \text{Top}_K(a_t^q) \right\|_2, \end{aligned} \tag{4.1}$$

and

$$\begin{aligned}
& \|\delta_{t-1} + \sum_{q=1}^N p_q a_t^q - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q \text{Top}_K(a_t^q))\|_2 \\
& \approx \|\delta_{t-1} + \sum_{q=1}^N p_q a_t^q - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q)\|_2 + \\
& \quad \|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q \text{Top}_K(a_t^q))\|_2,
\end{aligned} \tag{4.2}$$

for bidirectional  $\text{top}_K$  SGD.

One benefit of the new representation is that it has nice physical meaning. The first norm is the error inherent to the compressor and can be bounded by  $\gamma$ -approximate compressor defined in (3.2), and the second norm is the error that comes from the distributed system, which occurs when the gradients from the local workers is not representative of the global gradient. Plotting  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N p_q \text{Top}_K(a_t^q)\|_2$  and  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q \text{Top}_K(a_t^q))\|_2$  in Figure 4.5 for bidirectional and unidirectional downlink  $\text{top}_K$  SGD, we see that adding a downlink  $\text{top}_K$  compressor to unidirectional  $\text{top}_K$  SGD can reduce the distributed error.

Alistarh *et al.* consider  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N p_q \text{Top}_K(a_t^q)\|_2$  in their analysis of unidirectional  $\text{top}_K$  SGD, though they divide this value by  $\|\sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2$  [3]. This is present in their analysis as an assumption we describe later in Chapter 5. The authors explain this as a bound on the variance of the local gradients with respect to the global variance. They give an example to illustrate this: consider an instance with two nodes, dimension 2, and  $K = 1$ . Assume that node 1 has gradient vector  $(-1001, 500)$ , and node 2 has gradient vector  $(1001, 500)$ . Applying the  $\text{top}_1$  of the sum of gradients gives  $(0, 1000)$ . However, applying the sum of  $\text{top}_1$  results in  $(0, 0)$ , which is not desirable. Assumption 5.6 limits the variability at the local nodes [3]. We can easily use this example for the bidirectional case. Applying  $\text{top}_1$  to the sum of  $\text{top}_1$  gradients will give us  $(0, 0)$ , which gives us the same problem.

We also mention that iterating with  $\text{top}_K$  SGD with downlink compression causes the  $\{w_t\}$  sequence to converge to the correct value in our example, since the exact weights are fixed point, showing that  $\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q)$  could be a good benchmark for measuring how well an update step will perform. We will use this observation in our analysis in the next section.



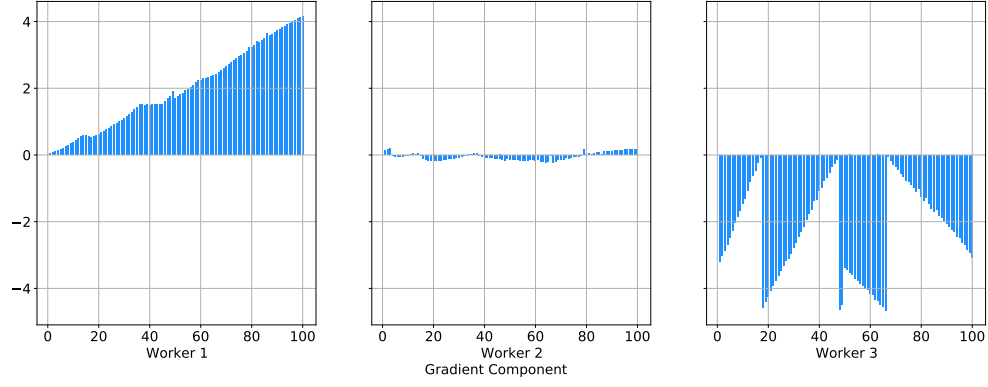


Figure 4.2:  $a_t^q$  from each worker at  $t = 300$  for unidirectional  $\text{top}_K$  SGD.

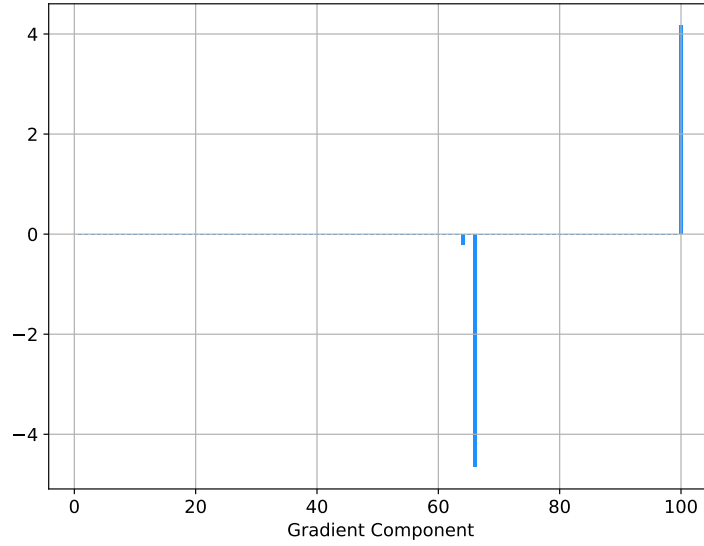


Figure 4.3:  $\sum_{q=1}^N \text{Top}_K(a_t^q)$  at  $t = 300$  for unidirectional  $\text{top}_K$  SGD.

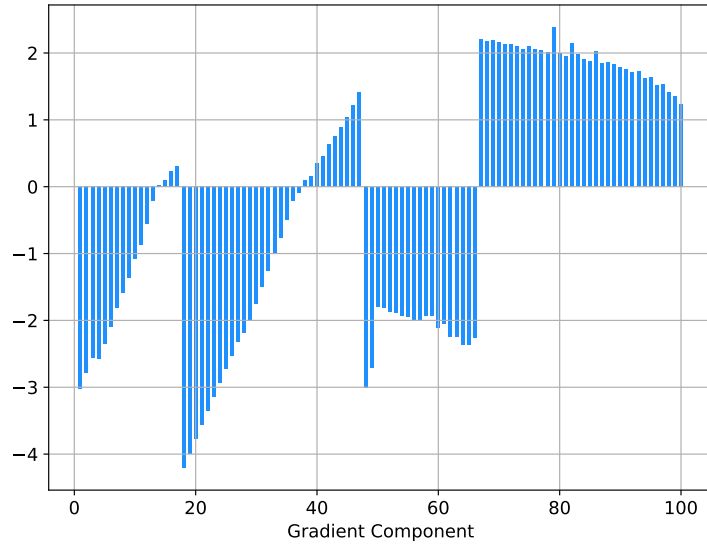


Figure 4.4:  $\sum_{q=1}^N a_t^q$  at  $t = 300$  for unidirectional  $\text{top}_K$  SGD.

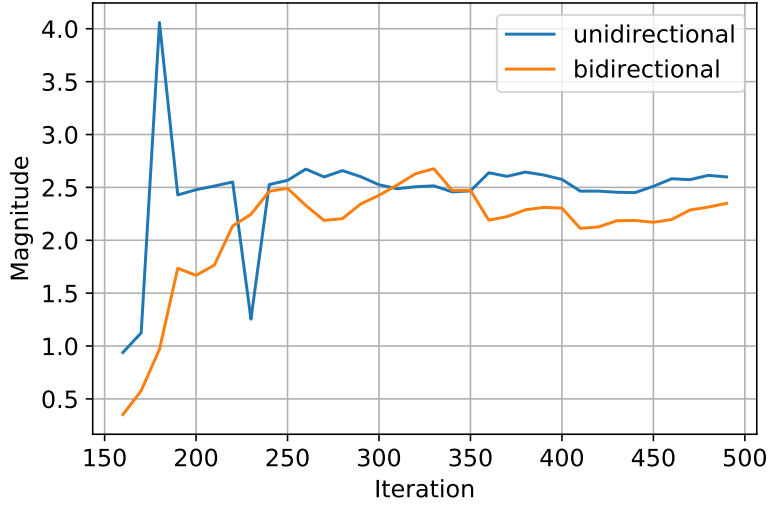


Figure 4.5: Largest  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2$  values every 10 iterations for unidirectional  $\text{top}_K$  SGD and largest  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2$  values every 10 iterations for bidirectional  $\text{top}_K$  SGD.

# Chapter 5

## Bidirectional Top<sub>K</sub> Convergence Analysis

We want to minimize the weighted average of  $N$  different functions across  $N$  nodes:

$$\min_{w \in \mathbb{R}^d} F(w) \triangleq \sum_{q=1}^N p_q F^q(w).$$

### 5.1 Assumptions

Before we focus on the non-convex convergence of Algorithms 8 and 9, we make the following assumptions, most of which we have discussed in Section 2.7 and Section 3.1.

**Assumption 5.1 (Existence of lower bound).** *There exists some constant  $F^*$  such that  $F(w) \geq F^*$  for all  $w \in \mathbb{R}^d$ .*

**Assumption 5.2 (Biased compressor guarantee).** *Since the Top<sub>K</sub> compressor is deterministic, we remove the expectation present in (3.1). There exists some  $\gamma \in (0, 1)$  such that*

$$\|\text{Top}_K(w) - w\|_2^2 \leq (1 - \gamma)\|w\|_2^2 \quad \forall w \in \mathbb{R}^d. \quad (5.1)$$

We mention that we can always find a  $\gamma$  value to satisfy this assumption for  $\text{top}_K$ , since we know

$$\|\text{Top}_K(w) - w\|_2^2 \leq \frac{d-K}{d} \|w\|_2^2, \quad (5.2)$$

where  $d$  is the dimension of the gradient.

**Assumption 5.3 (Lipschitz continuous gradient).**  *$F$  is continuously differentiable and the gradient  $\nabla F : \mathbb{R}^d \rightarrow \mathbb{R}^d$  is Lipschitz continuous with constant  $L > 0$ , i.e.*

$$\|\nabla F(w) - \nabla F(v)\|_2 \leq L\|w - v\|_2 \quad \forall w, v \in \mathbb{R}^d.$$

**Assumption 5.4 (First and second moment).** *The objective function and Algorithm 8 and 9 satisfies the following:*

$$\mathbb{E}\left[\sum_{q=1}^N p_q g^q(w, \xi_t^q)\right] = \sum_{q=1}^N p_q \nabla F^q(w) \quad \forall w \in \mathbb{R}^d, \forall t \in \mathbb{N}. \quad (5.3)$$

and

$$\mathbb{E}\left[\left\|\sum_{q=1}^N p_q g^q(w, \xi_t^q)\right\|_2^2\right] \leq M \quad \forall w \in \mathbb{R}^d, \forall t \in \mathbb{N}. \quad (5.4)$$

where  $\mathbb{E}[\cdot]$  is taken with respect to the joint distribution of  $\{\xi_t^1, \xi_t^2, \dots, \xi_t^N\}$ .

**Assumption 5.5.** *Given a sequence of iterates  $\{w_t\}$ :*

$$\begin{aligned} & \left\| \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q)) \right\|_2 \\ & \leq \rho \left\| \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q) \right\|_2 \quad \forall t \in \mathbb{N}. \end{aligned}$$

Assumption 5.5 is similar to the assumption made by Alistarh *et al* [3], which is discussed in detail in Section 4.2. It's role is to bound the gap between the  $\text{top}_K$  of the gradient sum and the bidirectional  $\text{top}_K$  SGD update step, as opposed to the gap between the  $\text{top}_K$  of the gradient sum and the unidirectional  $\text{top}_K$  SGD update step.

## 5.2 Concept and Intuition

Our proof of non-convex convergence for bidirectional top<sub>K</sub> SGD is similar to the proof for unidirectional top<sub>K</sub> SGD provided by Alistarh *et al* [3]. The proof follows the same intuition used by convergence proofs of gradient descent methods with error feedback framework [3, 14].

We consider an error corrected sequence,

$$\tilde{w}_t = w_t - \sum_{q=1}^N p_q \epsilon_t^q - \delta_t, \quad (5.5)$$

where  $\tilde{w}_t$  is the parameter vector after adjusting the error term stored on all workers and server. To get the non-convex convergence bound, we apply the standard proof of SGD to  $\tilde{w}_t$ , and show that  $\tilde{w}_t \approx w_t$  and  $\nabla F(\tilde{w}_t) \approx \nabla F(w_t)$ .

## 5.3 Convergence Analysis

In this section, we analyze the convergence of the stochastic sequence  $\{w_t\}_{t \geq 0}$  in Algorithm 8 and 9 in the non-convex setting. We follow the proof structure created by Alistarh *et al* [3].

**Lemma 5.1.** *Let  $\{\tilde{w}_t\}_{t \geq 0}$  be defined in 5.5, and  $\{w_t\}_{t \geq 0}$ ,  $\{\alpha_t\}_{t \geq 0}$ , and  $p_i$  for  $1 \leq i \leq N$  be defined in Algorithm 8 and 9. Then*

$$\tilde{w}_t = \tilde{w}_{t-1} - \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q). \quad (5.6)$$

*Proof.* We have

$$\begin{aligned}
\tilde{w}_t &= w_t - \delta_t - \sum_{q=1}^N p_q \epsilon_t^q \\
&= w_{t-1} - \text{Top}_K(g_t) - \delta_t - \sum_{q=1}^N p_q \epsilon_t^q \\
&= w_{t-1} - g_t - \sum_{q=1}^N p_q \epsilon_t^q \\
&= w_{t-1} - \delta_{t-1} - \sum_{q=1}^N p_q \text{Top}_K(a_t^q) - \sum_{q=1}^N p_q \epsilon_t^q \\
&= w_{t-1} - \delta_{t-1} - \sum_{q=1}^N p_q a_t^q \\
&= \tilde{w}_{t-1} + \sum_{q=1}^N p_q \epsilon_{t-1}^q + \delta_{t-1} - \delta_{t-1} - \sum_{q=1}^N p_q a_t^q \\
&= \tilde{w}_{t-1} - \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q).
\end{aligned} \tag{5.7}$$

□

**Lemma 5.2.** *Let  $\{w_t\}_{t \geq 0}$  be defined by Algorithm 8 and 9, and  $\{\tilde{w}_t\}_{t \geq 0}$  be defined by (5.5). Under Assumptions 5.2 and 5.5, we have*

$$\|w_t - \tilde{w}_t\|_2 \leq \frac{1}{\lambda} (\sqrt{1-\gamma} + \rho)^2 \frac{1}{1-\gamma} \sum_{i=1}^t ((1+\lambda)(1-\gamma))^i \|\tilde{w}_{t-i+1} - \tilde{w}_{t-i}\|_2^2, \tag{5.8}$$

where  $\lambda \in (0, \frac{\gamma}{1-\gamma})$  is a constant.

*Proof.* Applying Lemma 5.1 and the iterative relation of sequence  $\{w_t\}_{t \geq 0}$  defined in Algorithms 8 and 9, we get:

$$\begin{aligned}
\|w_t - \tilde{w}_t\|_2 &= \|w_{t-1} - \tilde{w}_{t-1} + \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q) - \\
&\quad \text{Top}_K(\sum_{q=1}^N p_q \text{Top}_K(a_t^q) + \delta_{t-1})\|_2 \\
&= \|\delta_{t-1} + \sum_{q=1}^N p_q \epsilon_{t-1}^q + \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q) - \\
&\quad \text{Top}_K(\sum_{q=1}^N p_q \text{Top}_K(a_t^q) + \delta_{t-1})\|_2 \\
&\leq \|\delta_{t-1} + \sum_{q=1}^N p_q a_t^q - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q)\|_2 + \\
&\quad \|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2.
\end{aligned} \tag{5.9}$$

From Assumption 5.2, we have

$$\|\delta_{t-1} + \sum_{q=1}^N p_q a_t^q - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q)\|_2 \leq \sqrt{1-\gamma} \|\delta_{t-1} + \sum_{q=1}^N p_q a_t^q\|_2. \tag{5.10}$$

From Assumption 5.5, we have

$$\begin{aligned}
&\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2 \\
&\leq \rho \|\alpha_{t-1} \sum_{q=1}^N p_q g(w_{t-1}, \xi_{t-1}^q)\|_2.
\end{aligned} \tag{5.11}$$

Combining (5.10) and (5.11) with (5.9), we get



$$\begin{aligned}
\|w_t - \tilde{w}_t\|_2 &\leq \sqrt{1-\gamma}\|\delta_{t-1}\|_2 + \sum_{q=1}^N p_q a_t^q \|_2 + \rho \|\alpha_{t-1}\|_2 \sum_{q=1}^N p_q g(w_{t-1}, \xi_{t-1}^q) \|_2 \\
&\leq \sqrt{1-\gamma}\|\delta_{t-1}\|_2 + \sum_{q=1}^N p_q \epsilon_{t-1}^q \|_2 + \\
&\quad (\sqrt{1-\gamma} + \rho) \|\alpha_{t-1}\|_2 \sum_{q=1}^N p_q g(w_{t-1}, \xi_{t-1}^q) \|_2 \\
&\leq \sqrt{1-\gamma}\|w_{t-1} - \tilde{w}_{t-1}\|_2 + (\sqrt{1-\gamma} + \rho) \|\tilde{w}_t - \tilde{w}_{t-1}\|_2,
\end{aligned} \tag{5.12}$$

where the final inequality in (5.12) is from Lemma 5.1 and the definition of the error corrected sequence  $\tilde{w}_t$  given in (5.5). Taking the square, we get

$$\begin{aligned}
\|w_t - \tilde{w}_t\|_2^2 &= (\sqrt{1-\gamma}\|w_{t-1} - \tilde{w}_{t-1}\|_2 + (\sqrt{1-\gamma} + \rho) \|\tilde{w}_t - \tilde{w}_{t-1}\|_2)^2 \\
&\leq (1+\lambda)(1-\gamma)\|w_{t-1} - \tilde{w}_{t-1}\|_2^2 + \\
&\quad (1 + \frac{1}{\lambda})(\sqrt{1-\gamma} + \rho)^2 \|\tilde{w}_t - \tilde{w}_{t-1}\|_2^2,
\end{aligned} \tag{5.13}$$

where  $\lambda > 0$  is a constant to be determined later. Iterating downwards on (5.13) gives us

$$\begin{aligned}
\|\tilde{w}_t - w_t\|_2^2 &\leq \sum_{i=1}^t ((1+\lambda)(1-\gamma))^{i-1} (1 + \frac{1}{\lambda})(\sqrt{1-\gamma} + \rho)^2 \|\tilde{w}_{t-i+1} - \tilde{w}_{t-i}\|_2^2 \\
&= \frac{1}{\lambda} (\sqrt{1-\gamma} + \rho)^2 \frac{1}{1-\gamma} \sum_{i=1}^t ((1+\lambda)(1-\gamma))^i \|\tilde{w}_{t-i+1} - \tilde{w}_{t-i}\|_2^2
\end{aligned} \tag{5.14}$$

□

**Lemma 5.3.** *Under the same setting as Lemma 5.2, assume that*

$$\frac{1}{1-\gamma} \sum_{i=1}^t ((1+\lambda)(1-\gamma))^i \frac{\alpha_{t-i}^2}{\alpha_t} \leq D,$$

*for some constant  $\lambda > 0$  and  $D > 0$  and denote then we have*

$$\mathbb{E}[\|w_t - \tilde{w}_t\|_2^2] \leq \frac{M}{\lambda}(\sqrt{1-\gamma} + \rho)^2 \alpha_t D.$$

*Proof.* Using Lemma 5.1 and Assumption 5.4, for  $t \in \mathbb{N}$ ,

$$\begin{aligned} \mathbb{E}[\|\tilde{w}_t - \tilde{w}_{t-1}\|_2^2] &\leq \alpha_{t-1}^2 \mathbb{E}[\|g(w_{t-1}, \xi_{t-1}^q)\|_2^2] \\ &\leq \alpha_{t-1}^2 M. \end{aligned} \tag{5.15}$$

Take expectation to the inequality in Lemma 5.1 and combine with (5.15):

$$\begin{aligned} \mathbb{E}[\|w_t - \tilde{w}_t\|_2^2] &\leq \frac{1}{\lambda}(\sqrt{1-\gamma} + \rho)^2 \frac{1}{1-\gamma} \sum_{i=1}^t ((1+\lambda)(1-\gamma))^i \mathbb{E}[\|\tilde{w}_{t-i+1} - \tilde{w}_{t-i}\|_2^2] \\ &\leq \frac{1}{\lambda}(\sqrt{1-\gamma} + \rho)^2 \alpha_t \frac{1}{1-\gamma} \sum_{i=1}^t ((1+\lambda)(1-\gamma))^i M \frac{\alpha_{t-i}^2}{\alpha_t} \\ &\leq \frac{M}{\lambda}(\sqrt{1-\gamma} + \rho)^2 \alpha_t D. \end{aligned} \tag{5.16}$$

□

**Theorem 5.4.** *Under Assumptions 5.1, 5.2, 5.3, 5.4, and 5.5, if a learning rate sequence  $\{\alpha_t\}_{t \geq 0}$  and constant  $\lambda \in (0, \frac{1-\gamma}{\gamma})$  is chosen such that  $\forall t > 0$  there exists constant  $D > 0$*

$$\frac{1}{1-\gamma} \sum_{i=1}^t ((1+\lambda)(1-\gamma))^i \frac{\alpha_{t-i}^2}{\alpha_t} \leq D, \tag{5.17}$$

*running Algorithm 8 and 9 for  $T$  iterations will give us*

$$\begin{aligned} \frac{1}{\sum_{t=0}^T \alpha_t} \sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] &\leq \frac{2}{\sum_{t=0}^T \alpha_t} (F(w_0) - F^*) \\ &\quad + \left( LM + \frac{L^2 MD(\sqrt{1-\gamma} + \rho)^2}{\lambda} \right) \frac{\sum_{t=0}^T \alpha_t^2}{\sum_{t=0}^T \alpha_t}. \end{aligned} \tag{5.18}$$

*Proof.* Denote  $\mathbb{E}_{\xi_t}[\cdot]$  to be the expectation taken with respect to the joint distribution of  $\{\xi_t^1, \xi_t^2, \dots, \xi_t^N\}$  given all random variables before time  $t$ . Starting with Assumption 5.3 and taking  $\mathbb{E}_{\xi_t}[\cdot]$  to both sides of the inequality, we get

$$\begin{aligned}
\mathbb{E}_{\xi_t}[F(\tilde{w}_{t+1})] &\leq F(\tilde{w}_t) + \langle \nabla F(\tilde{w}_t), \mathbb{E}_{\xi_t}[\tilde{w}_{t+1} - \tilde{w}_t] \rangle + \frac{L}{2} \mathbb{E}_{\xi_t}[\|\tilde{w}_{t+1} - \tilde{w}_t\|_2^2] \\
&= F(\tilde{w}_t) - \alpha_t \langle \nabla F(\tilde{w}_t), \mathbb{E}_{\xi_t}[\sum_{i=1}^N p_q g(w_t, \xi_t^q)] \rangle + \\
&\quad \frac{L\alpha_t^2}{2} \mathbb{E}_{\xi_t}[\|\sum_{i=1}^N p_q g(w_t, \xi_t^q)\|_2^2] \\
&\leq F(\tilde{w}_t) - \alpha_t \langle \nabla F(\tilde{w}_t), \nabla F(w_t) \rangle + \frac{L\alpha_t^2 M}{2} \\
&\leq F(\tilde{w}_t) - \alpha_t \langle \nabla F(w_t), \nabla F(w_t) \rangle + \frac{L\alpha_t^2 M}{2} + \\
&\quad \alpha_t \langle \nabla F(w_t) - \nabla F(\tilde{w}_t), \nabla F(w_t) \rangle \\
&\leq F(\tilde{w}_t) - \alpha_t \|\nabla F(w_t)\|_2^2 + \frac{L\alpha_t^2 M}{2} + \\
&\quad \frac{\alpha_t}{2} \|\nabla F(w_t)\|_2^2 + \frac{\alpha_t}{2} \|\nabla F(w_t) - \nabla F(\tilde{w}_t)\|_2^2 \\
&\leq F(\tilde{w}_t) - \frac{\alpha_t}{2} \|\nabla F(w_t)\|_2^2 + \frac{L\alpha_t^2 M}{2} + \frac{\alpha_t L^2}{2} \|w_t - \tilde{w}_t\|_2^2.
\end{aligned} \tag{5.19}$$

Taking expectation to the joint distribution of all random variables to both sides both the inequality, we get

$$\mathbb{E}[F(\tilde{w}_{t+1})] \leq \mathbb{E}[F(\tilde{w}_t)] - \frac{\alpha_t}{2} \mathbb{E}[\|\nabla F(w_t)\|_2^2] + \frac{L\alpha_t^2 M}{2} + \frac{\alpha_t L^2}{2} \mathbb{E}[\|w_t - \tilde{w}_t\|_2^2]. \tag{5.20}$$

Applying Lemma 5.3,

$$\mathbb{E}[F(\tilde{w}_{t+1})] \leq \mathbb{E}[F(\tilde{w}_t)] - \frac{\alpha_t}{2} \mathbb{E}[\|\nabla F(w_t)\|_2^2] + \frac{L\alpha_t^2 M}{2} + \frac{\alpha_t^2 L^2}{2} \frac{M}{\lambda} (\sqrt{1-\gamma} + \rho)^2 D.$$

Telescoping and rearranging, we get

$$\sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] \leq \sum_{t=0}^T 2(\mathbb{E}[F(\tilde{w}_t)] - \mathbb{E}[F(\tilde{w}_{t+1})]) + L\alpha_t^2 M + \frac{\alpha_t^2 L^2 M D (\sqrt{1-\gamma} + \rho)^2}{\lambda},$$

or

$$\begin{aligned} \frac{1}{\sum_{t=0}^T \alpha_t} \sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] &\leq \frac{2}{\sum_{t=0}^T \alpha_t} (F(w_0) - F^*) \\ &\quad + \left( LM + \frac{L^2 M D (\sqrt{1-\gamma} + \rho)^2}{\lambda} \right) \frac{\sum_{t=0}^T \alpha_t^2}{\sum_{t=0}^T \alpha_t}, \end{aligned} \tag{5.21}$$

as desired.  $\square$

In order for the upper bound in (5.21) to converge to zero we need  $\sum_{t=1}^{\infty} \alpha_t = \infty$  and  $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$ . We can choose the stepsize sequence  $\alpha_t = \frac{1}{(t+1)^\theta}$ ,  $0 < \theta \leq 1$ . Finally, we need to check if the sequence satisfies (5.17).

A sufficient condition for (5.17) to be bounded is  $(1+\lambda)(1-\gamma) < 1$ . Therefore  $\lambda$  must be chosen so that  $\lambda \in (0, \frac{\gamma}{1-\gamma})$ . We can always find a  $\lambda$  that satisfies the inequality, since  $\gamma \in (0, 1)$  from Assumption 5.2. It is then easy to see that (5.17) is satisfied, since the exponential term dominates. We include a derivation of  $D$  in Appendix A.

## 5.4 Non-Convex Convergence of Unidirectional Top<sub>K</sub> Sparsification

In this section, we provide the proof of convergence for unidirectional top<sub>K</sub> sparsification described in Algorithm 6 and 7. The proof of convergence for unidirectional convergence follows closely to the non-convex proof provided by Alistarh *et al* with some adjustments to remove the limitation that  $K > \frac{1}{2}d$ , where  $d$  is the size of the gradient [3].

We use Assumption 5.1, 5.2, 5.3, and 5.4 for our non-convex analysis of unidirectional top<sub>K</sub> sparsification. However, instead of Assumption 5.5, we have

**Assumption 5.6.** Given a sequence of iterates  $\{w_t\}$ :

$$\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2 \leq \rho \|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2, \quad \forall t \in \mathbb{N}. \quad (5.22)$$

Similar to bidirectional  $\text{top}_K$ , we consider an error corrected sequence  $\tilde{w}_t$ .

$$\tilde{w}_t = w_t - \sum_{q=1}^N p_q \epsilon_t^q. \quad (5.23)$$

**Lemma 5.5.** Let  $\{\tilde{w}_t\}_{t \geq 0}$  be defined in 5.23, and  $\{w_t\}_{t \geq 0}$ ,  $\{\alpha_t\}_{t \geq 0}$ , and  $p_i$  for  $1 \leq i \leq N$  be defined in Algorithm 6 and 7. Then

$$\tilde{w}_t = \tilde{w}_{t-1} - \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q). \quad (5.24)$$

*Proof.* We have

$$\begin{aligned} \tilde{w}_t &= w_t - \sum_{q=1}^N p_q \epsilon_t^q \\ &= w_{t-1} - \sum_{q=1}^N p_q \text{Top}_K(a_t^q) - \sum_{q=1}^N p_q \epsilon_t^q \\ &= w_{t-1} - \sum_{q=1}^N p_q a_t^q \\ &= \tilde{w}_{t-1} - \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q). \end{aligned} \quad (5.25)$$

□

**Lemma 5.6.** Let  $\{w_t\}_{t \geq 0}$  be defined by Algorithm 6 and 7, let  $\{\tilde{w}_t\}_{t \geq 0}$  be defined by 5.23 and define a constant  $\lambda > 0$ . Under Assumptions 5.2 and 5.6

$$\|w_t - \tilde{w}_t\|_2 \leq \sqrt{1 - \gamma} \|w_{t-1} - \tilde{w}_{t-1}\|_2 + (\sqrt{1 - \gamma} + \rho) \|\tilde{w}_t - \tilde{w}_{t-1}\|_2. \quad (5.26)$$

*Proof.* Applying Lemma 5.1 and the iterative relation of sequence  $\{w_t\}_{t \geq 0}$  defined in Algorithm 8 and 9 we get:

$$\begin{aligned}
\|w_t - \tilde{w}_t\|_2 &= \|w_{t-1} - \tilde{w}_{t-1} + \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q) - \sum_{q=1}^N p_q \text{Top}_K(a_t^q)\|_2 \\
&= \left\| \sum_{q=1}^N p_q \epsilon_{t-1}^q + \alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q) - \sum_{q=1}^N p_q \text{Top}_K(a_t^q) \right\|_2 \\
&\leq \left\| \sum_{q=1}^N p_q a_t^q - \text{Top}_K\left(\sum_{q=1}^N p_q a_t^q\right) \right\|_2 + \\
&\quad \left\| \text{Top}_K\left(\sum_{q=1}^N p_q a_t^q\right) - \sum_{q=1}^N p_q \text{Top}_K(a_t^q) \right\|_2.
\end{aligned} \tag{5.27}$$

From Assumption 5.2, we have

$$\left\| \sum_{q=1}^N p_q a_t^q - \text{Top}_K\left(\sum_{q=1}^N p_q a_t^q\right) \right\|_2 \leq \sqrt{1-\gamma} \left\| \sum_{q=1}^N p_q a_t^q \right\|_2. \tag{5.28}$$

And from Assumption 5.6, we have

$$\left\| \text{Top}_K\left(\sum_{q=1}^N p_q a_t^q\right) - \text{Top}_K\left(\sum_{q=1}^N \text{Top}_K(p_q a_t^q)\right) \right\|_2 \leq \rho \left\| \alpha_{t-1} \sum_{q=1}^N p_q g(w_{t-1}, \xi_{t-1}^q) \right\|_2. \tag{5.29}$$

Combining 5.28 and 5.29 with 5.27, we get

$$\begin{aligned}
\|w_t - \tilde{w}_t\|_2 &\leq \sqrt{1-\gamma} \left\| \sum_{q=1}^N p_q a_t^q \right\|_2 + \rho \left\| \alpha_{t-1} \sum_{q=1}^N p_q g(w_{t-1}, \xi_{t-1}^q) \right\|_2 \\
&\leq \sqrt{1-\gamma} \left\| \sum_{q=1}^N p_q \epsilon_{t-1}^q \right\|_2 + (\sqrt{1-\gamma} + \rho) \left\| \alpha_{t-1} \sum_{q=1}^N p_q g(w_{t-1}, \xi_{t-1}^q) \right\|_2 \\
&\leq \sqrt{1-\gamma} \|w_{t-1} - \tilde{w}_{t-1}\|_2 + (\sqrt{1-\gamma} + \rho) \|\tilde{w}_t - \tilde{w}_{t-1}\|_2.
\end{aligned} \tag{5.30}$$

as desired.  $\square$

The bound for the norm of the difference between parameter vector  $w_t$  and the error-corrected vector  $\tilde{w}_t$  from Algorithm 6, 7 and (5.23) is the same as the bound for the norm of the difference between parameter vector  $w_t$  and the error-corrected vector  $\tilde{w}_t$  from Algorithm 8, 9 and (5.5). From this point, the proof for unidirectional non-convex convergence is the same as the proof for bidirectional non-convex convergence. Following the convergence analysis of Section 5.3, we get the following result.

**Theorem 5.7.** *Under Assumptions 5.1, 5.2, 5.3, 5.4, and 5.6, if a learning rate sequence  $\{\alpha_t\}_{t \geq 0}$  and constant  $\lambda \in (0, \frac{1-\gamma}{\gamma})$  is chosen such that  $\forall t > 0$  there exists constant  $D > 0$*

$$\frac{1}{1-\gamma} \sum_{i=1}^t ((1+\lambda)(1-\gamma))^i \frac{\alpha_{t-i}^2}{\alpha_t} \leq D, \quad (5.31)$$

running Algorithm 6 and 7 for  $T$  iterations gives us

$$\begin{aligned} \frac{1}{\sum_{t=0}^T \alpha_t} \sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] &\leq \frac{2}{\sum_{t=0}^T \alpha_t} (F(w_0) - F^*) \\ &\quad + \left( LM + \frac{L^2 MD (\sqrt{1-\gamma} + \rho)^2}{\lambda} \right) \frac{\sum_{t=0}^T \alpha_t^2}{\sum_{t=0}^T \alpha_t}. \end{aligned} \quad (5.32)$$

## 5.5 Discussion

We remark that with a slight change in assumption, bidirectional  $\text{top}_K$  SGD has the same convergence bound as unidirectional  $\text{top}_K$  SGD. If we choose the step size to be  $\alpha_t = \frac{1}{(t+1)^{\frac{1}{2}+\epsilon}}$ , where  $\epsilon$  is an arbitrarily small number, the bound in (5.4) and (5.7) will approach 0 at  $O(1/\sqrt{T})$ . Both have the same asymptotic convergence rate as SGD. We briefly comment that there is a slight mistake in the convergence analysis of Alistarh *et al*, and the  $\rho$  value in the upper bound should not be dampened by the number of workers [3]. We compare the performance of bidirectional to unidirectional  $\text{top}_K$  SGD in Chapter 6, and estimate the values of the constants in their convergence bound.

The main advantage of our analysis over existing ones [32, 36] is that it can capture the degree to which the compressed gradients from each worker are not representative of the global gradient, which is necessary for explaining when bidirectional  $\text{top}_K$  SGD can outperform unidirectional  $\text{top}_K$  SGD. We also comment that the analysis for bidirectional

$\text{top}_K$  can be applied to other bidirectional compression algorithms with error feedback, and can give the reader an intuition of how to choose a downlink compressor  $C_2$  given an uplink compressor  $C_1$ . Specifically, we want to choose a compressor  $C_2$  such that,

$$\begin{aligned} & \|C_1(\delta_{t-1} + \sum_{q=1}^N g^q(w_{t-1}, \xi_{t-1}^q) + \epsilon_{t-1}^q) - \\ & C_2(\delta_{t-1} + \sum_{q=1}^N p_q C_1(\epsilon_{t-1}^q + \alpha_{t-1} g^q(w_{t-1}, \xi_{t-1}^q)))\|_2 \approx 0. \end{aligned} \tag{5.33}$$

We can use this method to evaluate potential hybrid strategies. For example, if  $C_1$  is the  $\text{top}_K$  compressor we could potentially choose  $C_2$  to be the  $\text{rand}_K$  compressor, where the gradient components chosen by  $\text{rand}_K$  correspond to the non-zero indices after aggregating individual gradients from each worker. We could use (5.33) as a criteria for deciding whether to use  $\text{rand}_K$  or  $\text{top}_K$  as the downlink compressor.



# Chapter 6

## Experimental Results

This chapter describes the setup of experiments, the results, and some discussion towards their implication.

### 6.1 Testing Framework

The testing framework used to evaluate bidirectional  $\text{top}_K$  SGD against unidirectional  $\text{top}_K$  SGD is written with the PyTorch library. To circumvent the difficulty of acquiring a large number of workers to test on, we build on Jadhav’s GitHub repository Federated-Learning-PyTorch [4]. The framework simulates distributed training on a single machine. It does this by instantiating a number of models locally, dividing the dataset among the models, training the model on the divided dataset, and synchronizing models at appropriate times. Specifically, we use Jadhav’s method of assigning data to workers as well their general idea behind synchronizing parameters across local models [4].

Our main contributions to the repository are as follows: 1) sparsifying gradients in both the uplink and downlink direction, 2) adding error feedback memory terms, 3) synchronizing after each batch instead of after each epoch.

## 6.2 Differences Between Theoretical and Experimental Setup

We mention experimental settings that are not reflected in the theory presented in Chapter 5. The first is that the data chosen in each epoch is sampled without replacement using a technique called random reshuffling (RR) [8]. For our convergence analysis, this means that the variables  $\xi$  are not iid across time  $t$  and across workers. As far as we know, this difference is common in gradient compression research [3, 6, 14, 32, 36], as RR has been shown to converge faster than SGD on many experiments, but the convergence property of SGD is more well understood [19].

Another difference is the choice of step size. In our analysis, the stepsize is decreased in each iteration or batch. In practice, the stepsize for  $\text{top}_K$  sparsification is decayed after a fixed number of epochs or kept constant. Tang *et al* trains bidirectional  $\text{top}_K$  SGD for 320 epochs on a learning rate of 0.1 for the first 160 epochs and a learning rate of 0.01 for the second 160 epochs [32]. Sattler *et al* uses a constant learning rate to train their models with bidirectional  $\text{top}_K$  sparsification [23]. For our experiments the learning rates are kept constant.

## 6.3 Experimental Setup

Experiments are run on MNIST [11] and Fashion-MNIST [34] datasets with multilayer perceptrons and convolution neural networks with 20, 50 and 100 workers. We randomly split the 60000 training set into equal size sets and assign each to a worker. The models are trained with SGD, unidirectional  $\text{top}_K$  SGD, and bidirectional  $\text{top}_K$  SGD and evaluated on a test set of 10000. We set the minibatch size to 10 for all models regardless of number of workers participating, and train the models for 100 epochs. The learning rate for SGD, unidirectional  $\text{top}_K$  SGD, and bidirectional  $\text{top}_K$  SGD is tuned separately from 0.01 to 0.25 with step increase of 0.01, unlike previous bidirectional compression experiments, which use the same learning rate [23, 32]. We include the optimum learning rate of all models in Table B.1. There are models where the optimum learning rate differs drastically when run with unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$  SGD, such as the MLP model trained on the MNIST dataset and the CNN model trained on the Fashion-MNIST dataset. This suggests, contrary to the Sattler *et al*'s suggestion [23], that it is unfair to compare unidirectional and bidirectional  $\text{top}_K$  SGD under the same learning rate.

We also perform experiments on a VGG19 network [29] trained on the CIFAR10 [15]

dataset with 20 workers for SGD, unidirectional top<sub>K</sub> SGD, and bidirectional top<sub>K</sub> SGD. These models are trained with batch size 100 for 200 epochs. We tune the learning rate of the VGG19 network on learning rates 0.01, 0.02, 0.05 and 0.1.

For all models we choose  $K \approx 0.001d$  for the Top<sub>K</sub> operator, where  $d$  is the size of the gradient. Due to PyTorch specifics, we calculate  $K$  in the following manner,

$$K = d - \lfloor (1 - 0.001)d \rfloor,$$

and we include the gradient size and the corresponding  $K$  value of each model we tested on in Table 6.1.

Dataset	Model	$d$	$K$
MNIST	MLP	50890	51
	CNN	1199882	1200
Fashion-MNIST	MLP	242762	243
	CNN	29034	30
CIFAR10	VGG19	20040522	20041

Table 6.1: Model and gradient size used in experiments.

## 6.4 Unidirectional vs Bidirectional Training Loss and Test Accuracy

We compare the results of unidirectional and bidirectional compression in this section. With careful tuning, both unidirectional and bidirectional top<sub>K</sub> SGD achieved the same test accuracy as SGD. We include the results for the Fashion-MNIST MLP network in Figure 6.1 and 6.2, and the results for the CIFAR10 VGG19 network in Figure 6.3 and 6.4. The figures for the rest of the models are located in Appendix C.1. The results show that bidirectional top<sub>K</sub> SGD and unidirectional top<sub>K</sub> SGD can achieve similar test accuracy and train loss in the same number of epochs. We also mention that aside from the VGG19 model trained on CIFAR10 dataset, both bidirectional top<sub>K</sub> SGD and unidirectional top<sub>K</sub> SGD were able to achieve similar test accuracy and train loss to vanilla SGD.

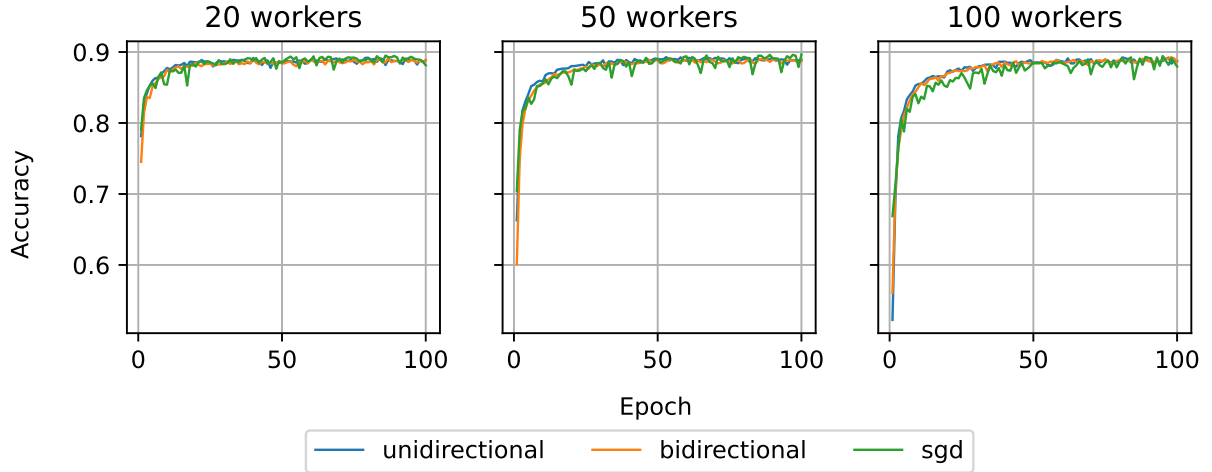


Figure 6.1: Comparison of testing accuracy for MLP model trained on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

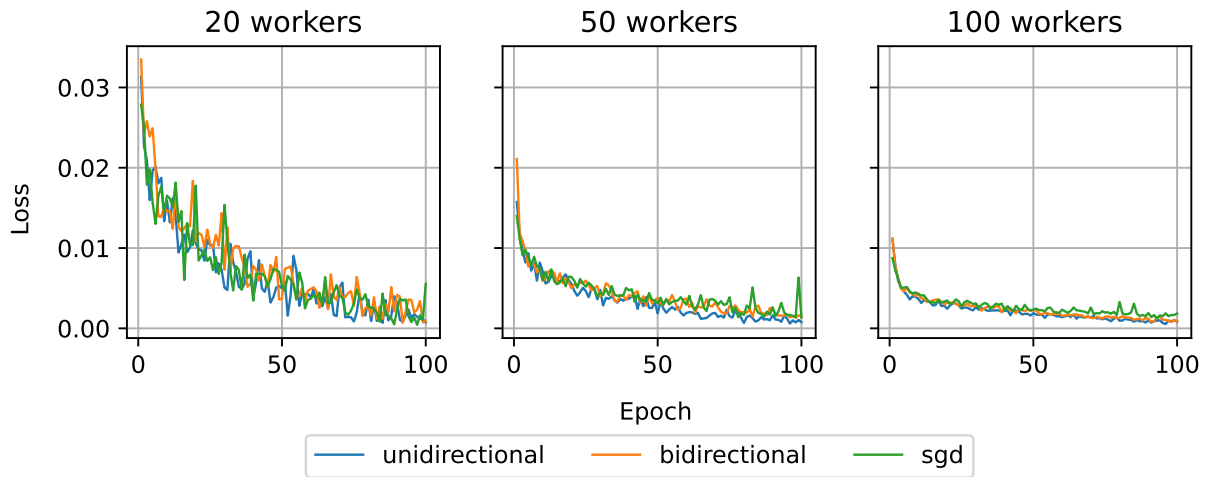


Figure 6.2: Comparison of testing loss for CNN model trained on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

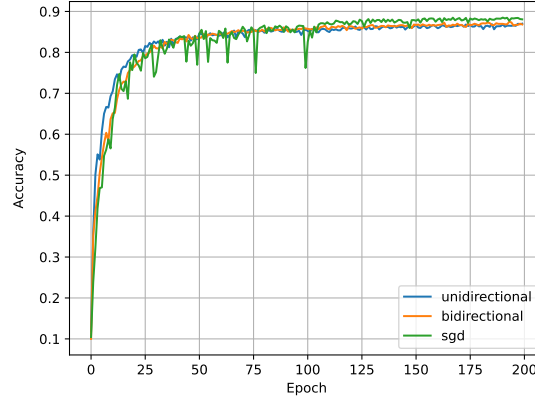


Figure 6.3: Comparison of testing accuracy for VGG19 model trained on CIFAR10 data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD on 20 workers. Trained on batch size 100.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

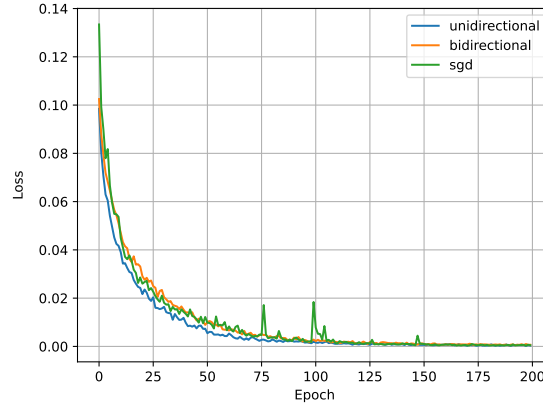


Figure 6.4: Comparison of training loss for VGG19 model trained on CIFAR10 data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD on 20 workers. Trained on batch size 100.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

## 6.5 Compression Rate of Downlink Top<sub>K</sub> Sparsification

Unlike uplink compression, the number of bits saved from downlink compression is related to the number of participating workers. For example, if  $N$  workers contributed sparse gradients of size  $K_{\text{uplink}}$ , then the gradient sent back by the server after aggregation is a sparse gradient with size at most  $K_{\text{uplink}}N$ .<sup>1</sup> Applying top<sub>K</sub> sparsification in the downlink will compress the gradient to at most  $K_{\text{downlink}}/K_{\text{uplink}}N$  of its size. If the indices of the gradient components contributed by different workers overlap, then the compression rate would be less than  $K_{\text{downlink}}/K_{\text{uplink}}$ . We measure the percentage of non-zero indices in the sum of gradients from the workers, which we show in Figure 6.5 and 6.6, and Appendix C.3. We see that the percentage approaches  $K_{\text{uplink}}N/d$  the number of iterations increase, showing that the downlink compressor achieves close to the largest compression rate. If  $K_{\text{downlink}} = K_{\text{uplink}}$ , downlink top<sub>K</sub> sparsification compresses the gradient to almost  $1/N$  of its original size.

Using our findings, the time it takes to transfer a gradient from worker-to-server then server-to-worker for unidirectional compression is

$$\alpha_1 + 2K_{\text{uplink}}\beta_1 + \alpha_2 + 2NK_{\text{uplink}}\beta_2, \quad (6.1)$$

and the time it takes to transfer a message in bidirectional compression is

$$\alpha_1 + 2K_{\text{uplink}}\beta_1 + \alpha_2 + 2K_{\text{downlink}}\beta_2, \quad (6.2)$$

where  $\alpha_1$  is the latency of a message transfer between worker-to-server,  $\alpha_2$  is latency of a message transfer from server-to-worker,  $\beta_1$  is the uplink transfer time for a 32 bit float, and  $\beta_2$  is the downlink transfer time for a 32 bit float.

We need to double the communication time between nodes since the system needs to communicate the indices of the non-zero elements of the gradient in addition to the value of each gradient component.

---

<sup>1</sup>In this section, the size of the gradient refers to the number of non-zero components in the gradient, not the magnitude of the gradient.

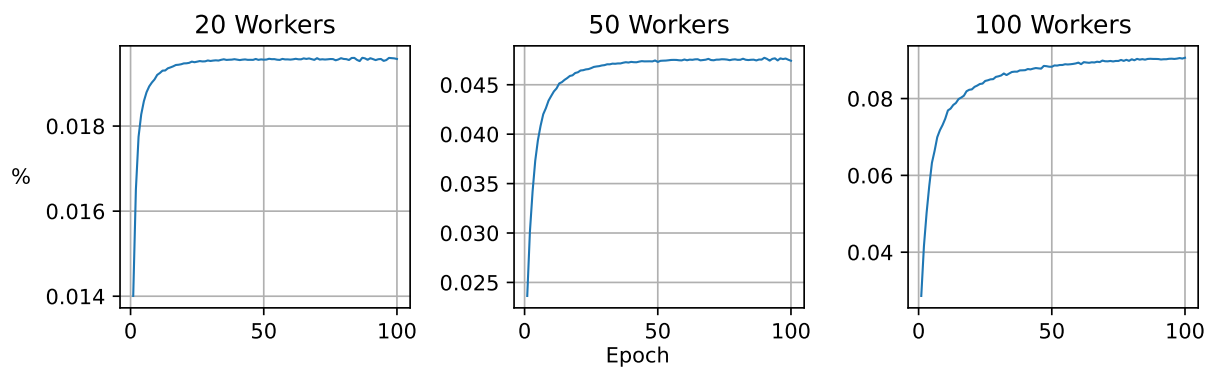


Figure 6.5: Percent of non-zero indices after aggregating sparse gradients from workers. Trained on a MLP network using Fashion-MNIST dataset.

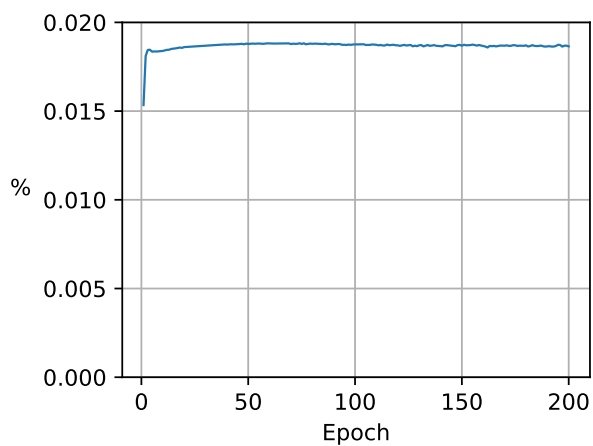


Figure 6.6: Percent of non-zero indices after aggregating sparse gradients from workers. Trained on a VGG19 network using CIFAR10 dataset.

## 6.6 Behaviour of Constants in Convergence Analysis

We analyze the constants  $\rho$  and  $\gamma$  from the convergence bound of unidirectional and bidirectional  $\text{top}_K$  SGD. For SGD with bidirectional sparsification, using the notation from Algorithms 8 and 9, we record the sequences  $\{\hat{\rho}_t\}$ ,  $\{1 - \hat{\gamma}_t^{\text{downward}}\}$ , and  $\{1 - \hat{\gamma}_t^{\text{uplink}}\}$  for each batch, where

$$\hat{\rho}_t = \frac{\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2}{\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2}, \quad (6.3)$$

$$1 - \hat{\gamma}_t^{\text{downlink}} = \frac{\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q)) - \delta_{t-1} - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2}{\|\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2}, \quad (6.4)$$

and

$$1 - \hat{\gamma}_t^{\text{uplink}} = \max \left\{ \frac{\|\text{Top}_K(p_q a_t^q) - p_q a_t^q\|}{\|p_q a_t^q\|} : 1 \leq q \leq N \right\}. \quad (6.5)$$

For SGD with unidirectional sparsification, using notation from Algorithms 6 and 7, we record the sequences  $\{\hat{\rho}_t\}$ ,  $\{1 - \hat{\gamma}_t^{\text{uplink}}\}$  for each minibatch, where

$$\hat{\rho}_t = \frac{\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2}{\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2}, \quad (6.6)$$

and

$$1 - \hat{\gamma}_t^{\text{uplink}} = \max \left\{ \frac{\|\text{Top}_K(p_q a_t^q) - p_q a_t^q\|}{\|p_q a_t^q\|} : 1 \leq q \leq N \right\}. \quad (6.7)$$

We estimate the constants  $\gamma$  and  $\rho$  from the maximum of these sequence of values. Comparisons of  $\hat{\rho}$  values for unidirectional and bidirectional  $\text{top}_K$  SGD are shown in Figure 6.7 and 6.9, and Appendix C.4. The maximum value of the sequence  $\{\hat{\rho}_t\}$  is recorded in Table D.1. We can see that the  $\hat{\rho}$  value for bidirectional  $\text{top}_K$  SGD is consistently smaller than  $\hat{\rho}$  value for unidirectional  $\text{top}_K$  SGD across all experiments. We plot the numerator and denominator of  $\hat{\rho}$  from (6.3) and (6.6) separately in Figure 6.8 and 6.10, and Appendix C.4. We observe the minimum  $\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2$  value in each



epoch is approximately the same for unidirectional and bidirectional sparsification. The difference comes from the numerator, showing that adding downlink top $_K$  compressor can reduce the distributed error we defined in (4.1).

We plot the the sequence  $\{1 - \hat{\gamma}_t^{\text{downlink}}\}$ , and  $\{1 - \hat{\gamma}_t^{\text{uplink}}\}$  in Figure 6.11. For bidirectional compression, when  $K_{\text{downlink}} = K_{\text{uplink}}$ , the maximum  $\{1 - \hat{\gamma}_t^{\text{downlink}}\}$  value in each epoch is consistently smaller than the maximum  $\{1 - \hat{\gamma}_t^{\text{uplink}}\}$  value in each epoch, and the sequence  $\{1 - \hat{\gamma}_t^{\text{uplink}}\}$  is similar for both unidirectional and bidirectional compression. This implies that the  $\gamma$  constant in the convergence bound of Theorem 5.4 and 5.7 is approximately equal.

The smaller  $\rho$  value and equal  $\gamma$  value suggests that the convergence bound of bidirectional top $_K$  SGD can potentially be tighter than the convergence bound of unidirectional top $_K$  SGD.

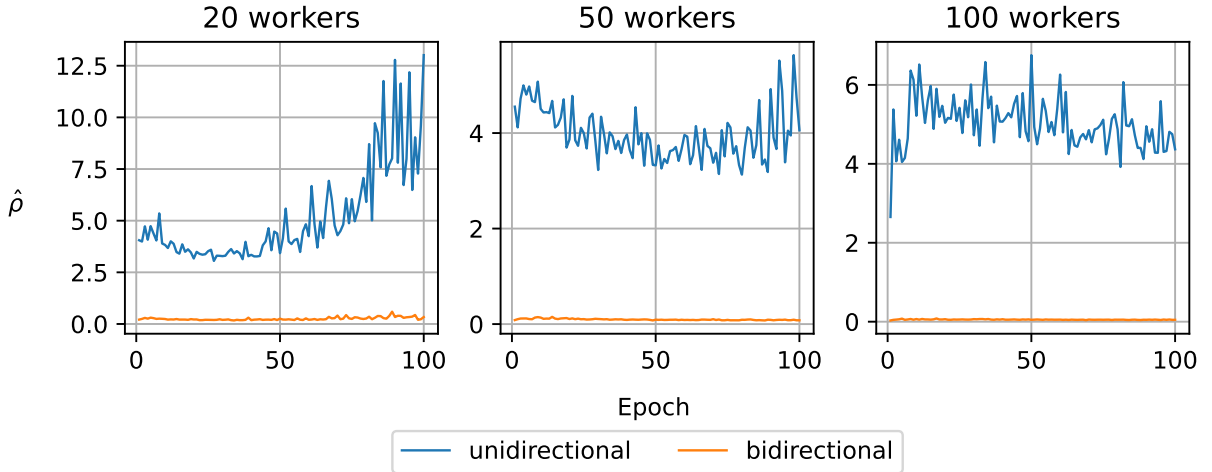


Figure 6.7: Largest  $\hat{\rho}$  value in each epoch of a MLP model trained with unidirectional and bidirectional top $_K$  SGD on Fashion-MNIST dataset. Batch size 10.

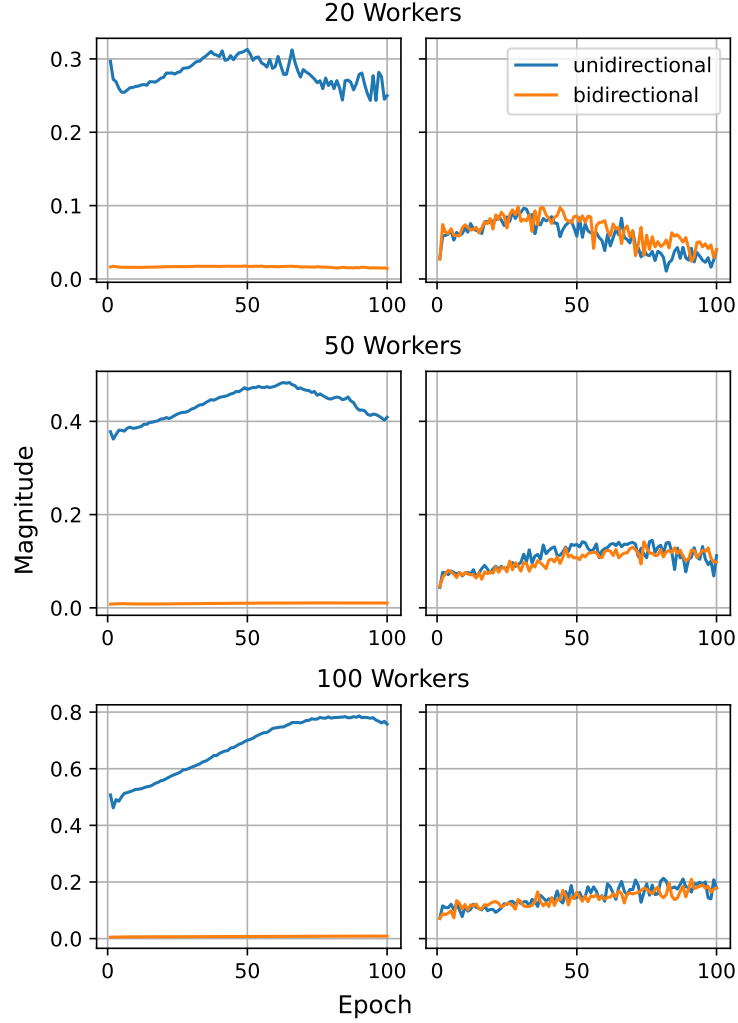


Figure 6.8: Results from training MLP model on Fashion-MNIST data. Left side graphs are the largest  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2$  values in each epoch for unidirectional top<sub>K</sub> SGD and the largest  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2$  values in each epoch for bidirectional top<sub>K</sub> SGD. Right side graphs are the smallest  $\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2$  value in each epoch.

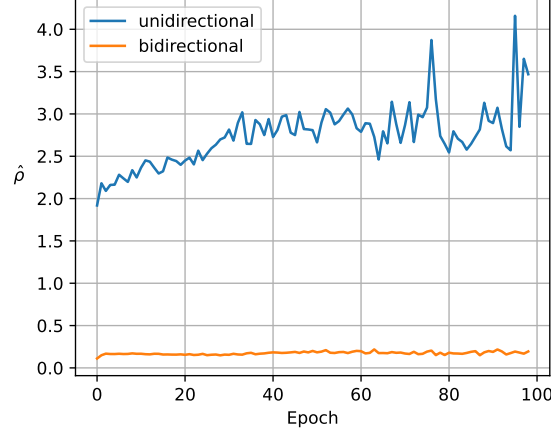


Figure 6.9: Largest  $\hat{\rho}$  value in each epoch of a VGG19 model trained with unidirectional and bidirectional  $\text{top}_K$  SGD on CIFAR10 dataset with batch size 100.

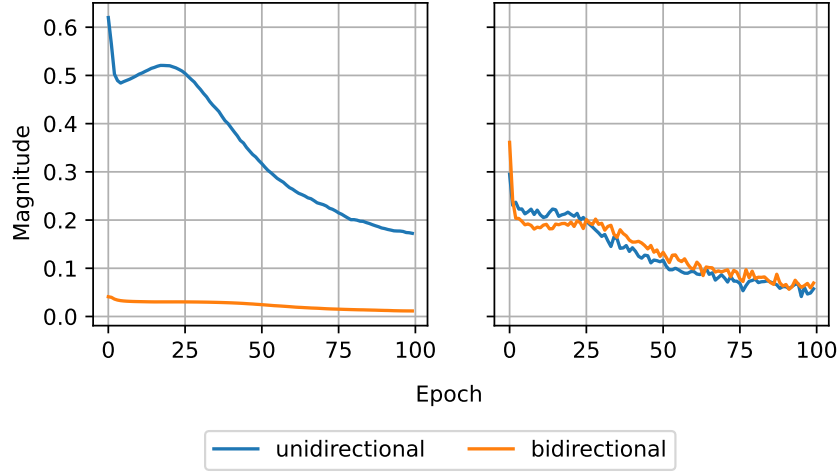


Figure 6.10: Results from training VGG19 model on CIFAR10 data. Left side graphs are the largest  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2$  values in each epoch for unidirectional  $\text{top}_K$  SGD and the largest  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2$  values in each epoch for bidirectional  $\text{top}_K$  SGD. Right side graphs are the smallest  $\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2$  value in each epoch.

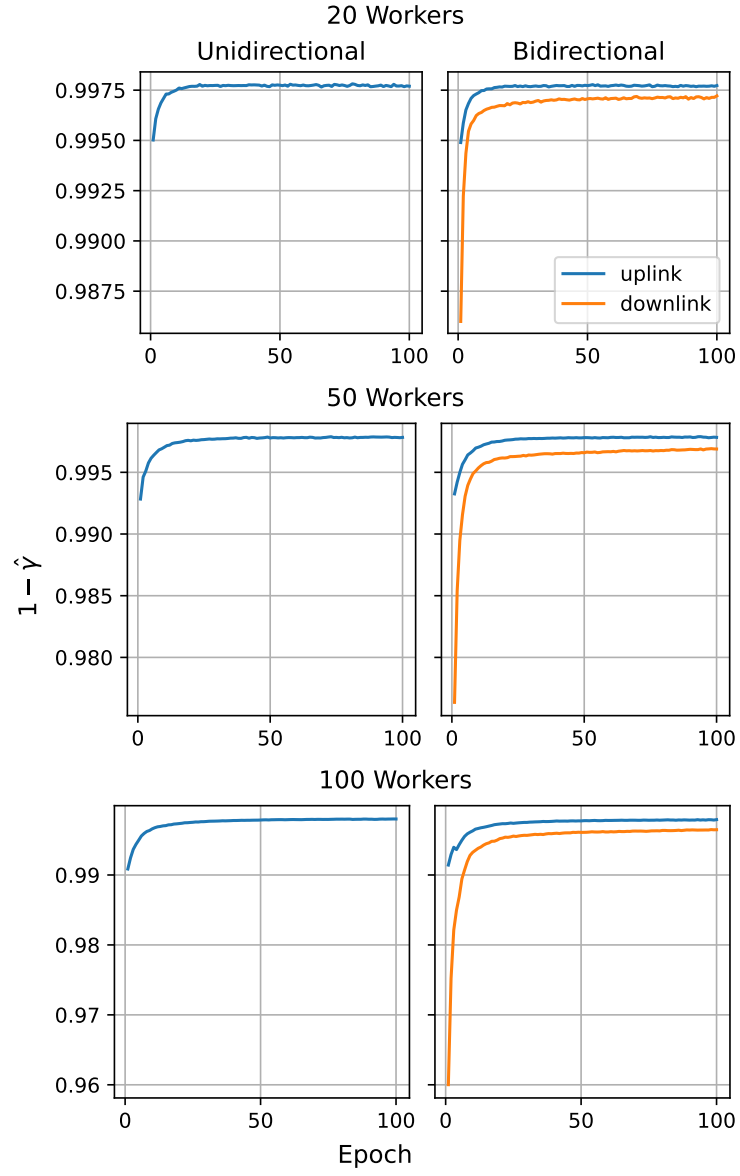


Figure 6.11: Largest  $1 - \hat{\gamma}$  value in each epoch of a MLP model trained with unidirectional and bidirectional  $\text{top}_K$  SGD on Fashion-MNIST dataset.

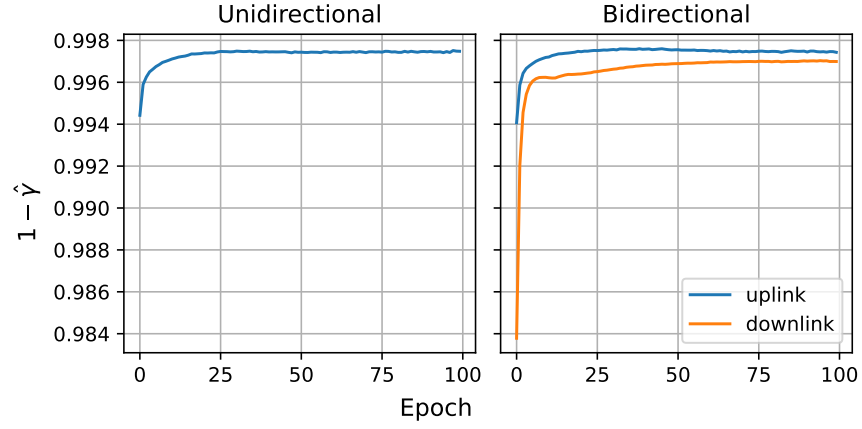


Figure 6.12: Largest  $1 - \hat{\gamma}$  value in each epoch of a VGG19 model trained with unidirectional and bidirectional  $\text{top}_K$  SGD on CIFAR10 dataset with batch size 100.

## 6.7 Effect of Non-IID Dataset on $\rho$

In this section, we briefly discuss testing on non-iid datasets. We use the experimental setup proposed by McMahan *et al.* [18] and implemented by [4]. Data is sorted by digit labels, then divided into 200 shards of 300 data points with the same label. Each shard is assigned to a random worker. Since the experiment is run with 100 workers, and MNIST and Fashion-MNIST each have 60000 data points, each worker has 2 shards with 300 data points. We plot the

### 6.7.1 FMNIST MLP Network

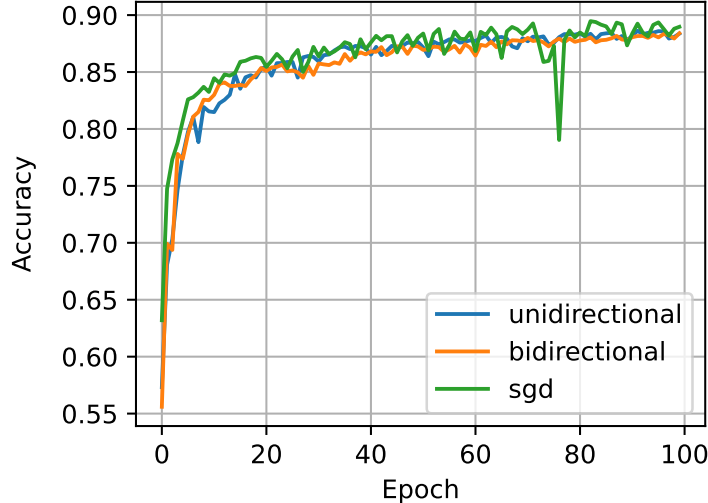


Figure 6.13: Comparison of testing accuracy from training MLP model on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

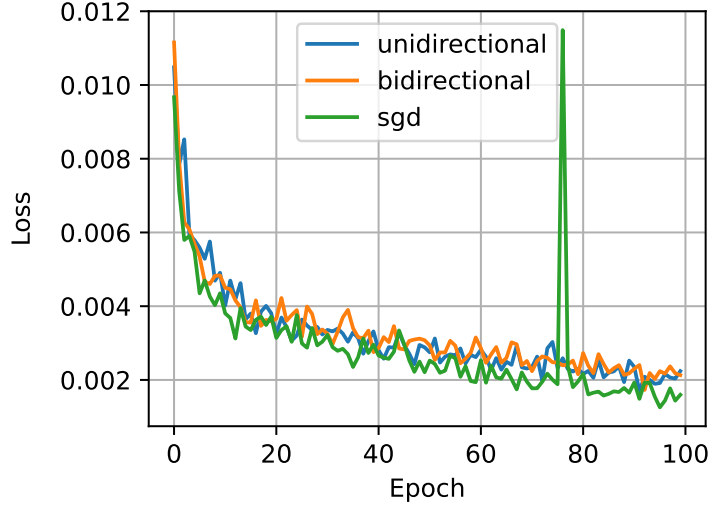


Figure 6.14: Comparison of training loss from training MLP model on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

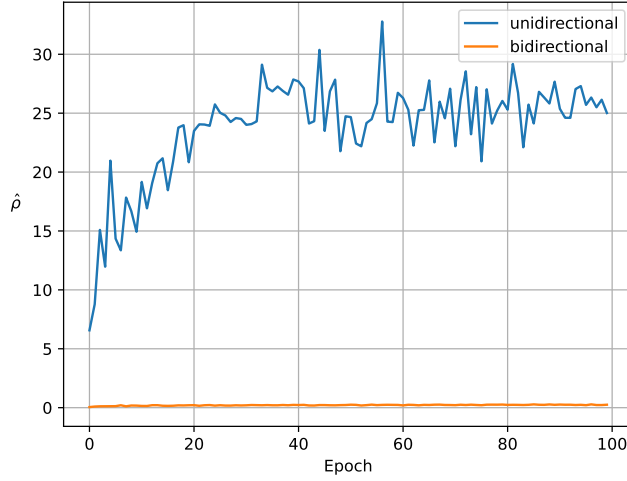


Figure 6.15: Comparison of  $\hat{\rho}$  values from training MLP model on Fashion-MNIST data for unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$ . Models trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

### 6.7.2 FMNIST CNN Network

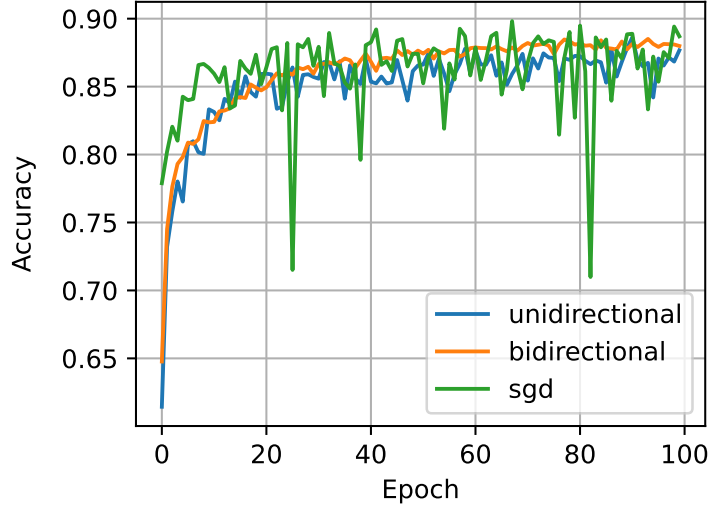


Figure 6.16: Comparison of testing accuracy from training CNN model on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.



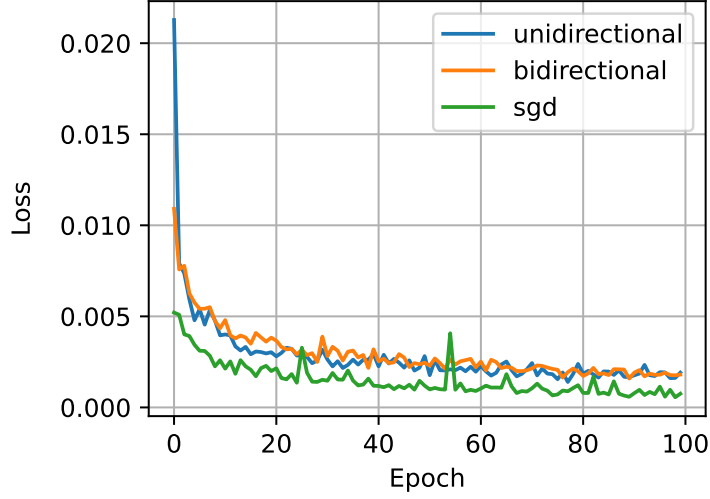


Figure 6.17: Comparison of training loss from training CNN model on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

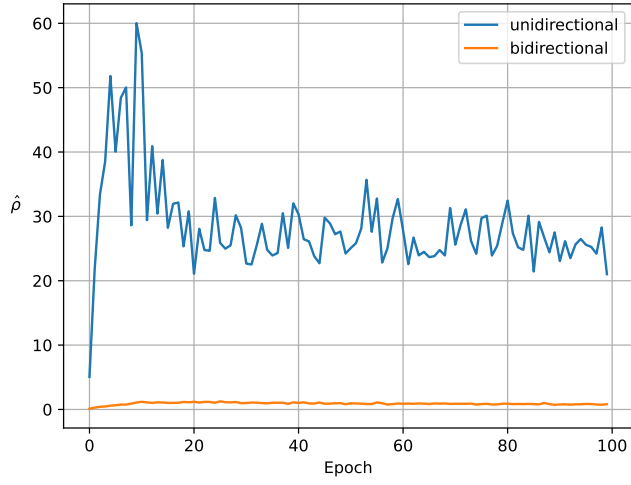


Figure 6.18: Comparison of  $\hat{\rho}$  values from training CNN model on Fashion-MNIST data for unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$ . Models trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

### 6.7.3 MNIST MLP Network

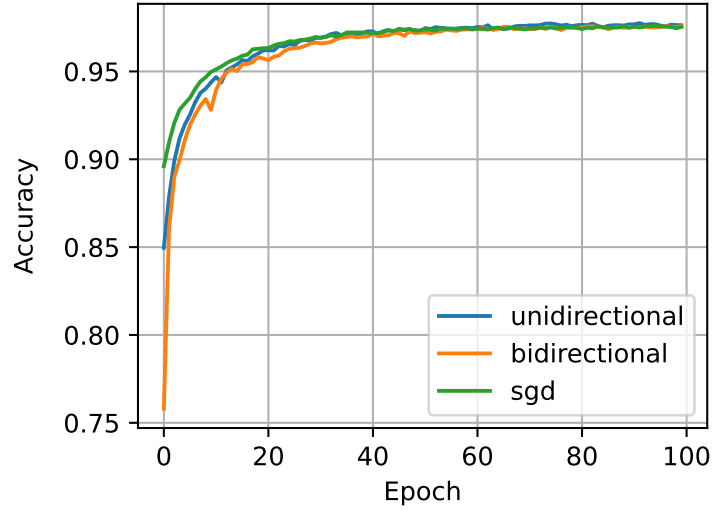


Figure 6.19: Comparison of testing accuracy from training MLP model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

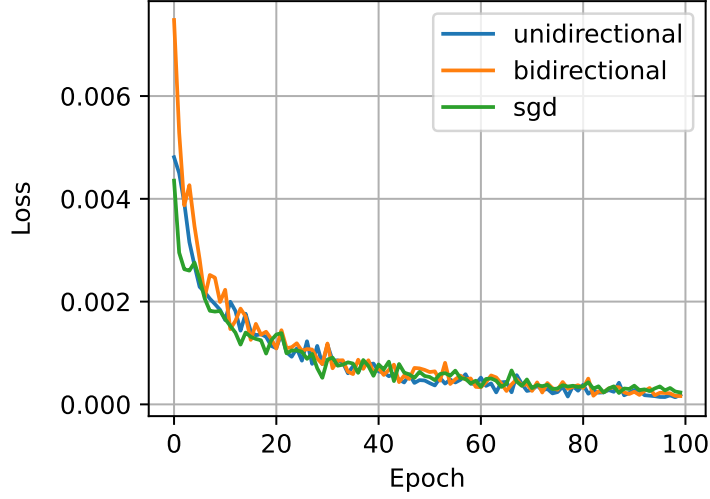


Figure 6.20: Comparison of training loss from training MLP model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

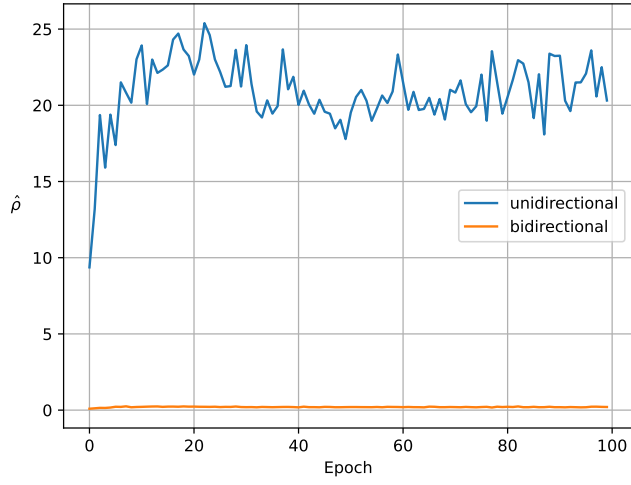


Figure 6.21: Comparison of  $\hat{\rho}$  values from training MLP model on MNIST data for unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$ . Models trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

#### 6.7.4 MNIST CNN Network

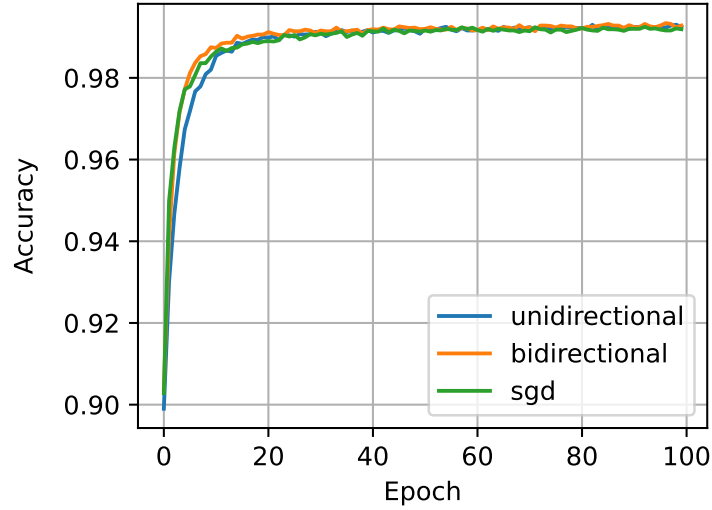


Figure 6.22: Comparison of testing accuracy from training CNN model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

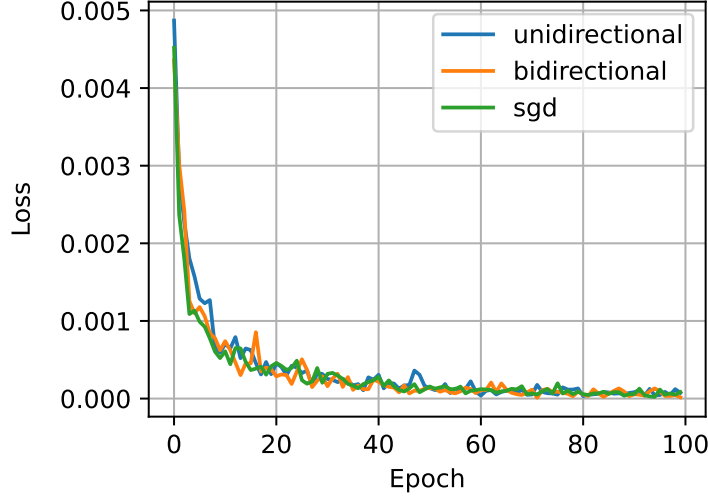


Figure 6.23: Comparison of training loss from training CNN model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

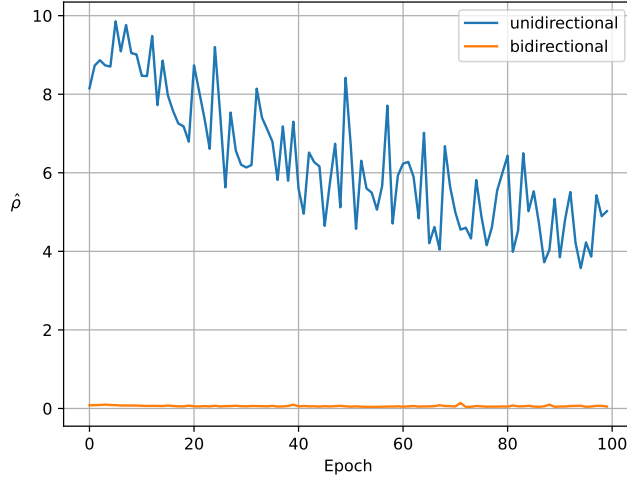


Figure 6.24: Comparison of  $\hat{\rho}$  values from training MLP model on MNIST data for unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$ . Models trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional. Dataset is non-iid.

# Chapter 7

## Conclusion

This thesis studies bidirectional  $\text{top}_K$  SGD in distributed learning. We started in Chapter 2, where we provided some background into SGD and distributed learning. In Chapter 3 we discussed gradient compression techniques, and gave a brief introduction to current research in unidirectional and bidirectional  $\text{top}_K$  sparsification. In Chapter 4 we introduced an example where bidirectional  $\text{top}_K$  sparsification outperforms unidirectional  $\text{top}_K$  sparsification and provided some justification to why this is the case. We then use this justification to build our non-convex convergence analysis in Chapter 5. Finally, in Chapter 6, we provide experimental results showing that bidirectional  $\text{top}_K$  SGD can perform just as well as unidirectional SGD in different settings, and give the reader an idea of what the constants in the convergence bound are.

### 7.1 Contributions

The main contributions of our thesis are as follows:

- We provide an example where bidirectional  $\text{top}_K$  SGD has better convergence than unidirectional  $\text{top}_K$  SGD.
- We provide a non-convex convergence analysis for bidirectional  $\text{top}_K$  SGD that has the same convergence bound as unidirectional  $\text{top}_K$  SGD with assumptions that can be empirically easier to satisfy. We argue that the assumptions made in the analysis is natural, and can be used to explain our example where bidirectional  $\text{top}_K$

SGD outperforms unidirectional  $\text{top}_K$  SGD, which previous analysis are unable to do [32, 36].

- We provide detailed testing across different models, datasets, and number of workers at state-of-the-art sparsification levels [10] to show that bidirectional  $\text{top}_K$  SGD can converge as well as unidirectional  $\text{top}_K$  SGD. We comment that for optimum performance, the learning rate for unidirectional and bidirectional  $\text{top}_K$  SGD need to be tuned separately, which is not done by representative works of bidirectional compression [23, 32].
- We modify Alistarh *et al*'s non convex proof of unidirectional  $\text{top}_K$  SGD so that it works for  $K < \frac{1}{2}d$ , where  $d$  is the size of the gradient [3].
- We provide empirical evidence that the communication saved by downlink  $\text{top}_K$  sparsification is almost linear to the number of workers when  $K \ll d$ .

## 7.2 Future Work

A natural step would be to try to extend our results for bidirectional  $\text{top}_K$  SGD to the Federated Learning setting, where the data allocated to each worker is non-i.i.d., and only a random fraction of workers is capable of updating the central model at a given time [18]. However, it has been shown experimentally that it is difficult for models to converge to high accuracy when the number of participating workers is low [23]. Some extra mechanism may need to be added to ensure better convergence.

In Chapter 4, we discussed that we want to minimize our distributed error  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N a_t^q) - \text{Top}_K(\delta_{t-1} + \text{Top}_K(\sum_{q=1}^N a_t^q))\|_2$ . Therefore instead of sending the  $\text{top}_K$  values from each worker, we could try to predict the indices of the non-zero components of  $\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N a_t^q)$ , and send those instead. A good place to start would be to adapt the theory for bidirectional  $\text{top}_K$  SGD to explain the Time-Correlated Sparsification (TCS) scheme proposed by Ozfatura *et al.*, which identifies significant weights at time  $t$ , and then has all workers update those significant weights [20]. To the best of our knowledge, no convergence theory for the TCS scheme has been offered.

# References

- [1] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.
- [2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 30, 2017.
- [3] Dan Alistarh, Torsten Hoefer, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. The convergence of sparsified gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.
- [4] Jadhav Ashwin. Federated-learning-pytorch. <https://github.com/AshwinRJ/Federated-Learning-PyTorch>, 2019.
- [5] Tal Ben-Nun and Torsten Hoefer. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.
- [6] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR, 2018.
- [7] Aleksandr Beznosikov, Samuel Horváth, Peter Richtárik, and Mher Safaryan. On biased compression for distributed learning. *arXiv preprint arXiv:2002.12410*, 2020.
- [8] Léon Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the symposium on learning and data science, Paris*, volume 8, pages 2624–2633, 2009.
- [9] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.



- [10] Mingzhe Chen, Deniz Gündüz, Kaibin Huang, Walid Saad, Mehdi Bennis, Aneta Vulgarakis Feljan, and H Vincent Poor. Distributed learning in wireless networks: Recent progress and future challenges. *IEEE Journal on Selected Areas in Communications*, 2021.
- [11] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.
- [12] Pengchao Han, Shiqiang Wang, and Kin K Leung. Adaptive gradient sparsification for efficient federated learning: An online learning approach. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 300–310. IEEE, 2020.
- [13] Joeri Hermans. *On scalable deep learning and parallelizing gradient descent*. PhD thesis, Maastricht U., 2017.
- [14] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pages 3252–3261. PMLR, 2019.
- [15] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [16] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.
- [17] Wang Luping, Wang Wei, and Li Bo. Cmfl: Mitigating communication overhead for federated learning. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*, pages 954–964. IEEE, 2019.
- [18] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agueray Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.
- [19] Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. Random reshuffling: Simple analysis with vast improvements. *Advances in Neural Information Processing Systems*, 33:17309–17320, 2020.
- [20] Emre Ozfatura, Kerem Ozfatura, and Deniz Gündüz. Time-correlated sparsification for communication-efficient federated learning. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 461–466. IEEE, 2021.

- [21] Constantin Philippenko and Aymeric Dieuleveut. Preserved central model for faster bidirectional compression in distributed settings. *Advances in Neural Information Processing Systems*, 34, 2021.
- [22] Cédric Renggli, Saleh Ashkboos, Mehdi Aghagolzadeh, Dan Alistarh, and Torsten Hoefer. Sparcml: High-performance sparse communication for machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2019.
- [23] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 31(9):3400–3413, 2019.
- [24] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*. Citeseer, 2014.
- [25] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.
- [26] Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.
- [27] Shaohuai Shi, Qiang Wang, Kaiyong Zhao, Zhenheng Tang, Yuxin Wang, Xiang Huang, and Xiaowen Chu. A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 2238–2247. IEEE, 2019.
- [28] Shaohuai Shi, Kaiyong Zhao, Qiang Wang, Zhenheng Tang, and Xiaowen Chu. A convergence analysis of distributed sgd with communication-efficient gradient sparsification. In *IJCAI*, pages 3411–3417, 2019.
- [29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [30] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. *Advances in Neural Information Processing Systems*, 31, 2018.
- [31] Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

- [32] Hanlin Tang, Chen Yu, Xiangru Lian, Tong Zhang, and Ji Liu. Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. In *International Conference on Machine Learning*, pages 6155–6165. PMLR, 2019.
- [33] Indu Thangakrishnan, Derya Cavdar, Can Karakus, Piyush Ghai, Yauheni Selivonchyk, and Cory Pruce. Herring: Rethinking the parameter server at scale for the cloud. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13. IEEE, 2020.
- [34] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.
- [35] Hang Xu, Chen-Yu Ho, Ahmed M Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, Marco Canini, and Panos Kalnis. Compressed communication for distributed deep learning: Survey and quantitative evaluation. Technical report, 2020.
- [36] Shuai Zheng, Ziyue Huang, and James Kwok. Communication-efficient distributed blockwise momentum sgd with error-feedback. *Advances in Neural Information Processing Systems*, 32, 2019.

# APPENDICES

# Appendix A

## Non-Convex Analysis

### A.1 Bound on Constant $D$

**Lemma A.1.** *Let  $0 < \beta < 1$ ,  $\theta \geq 0$  and let  $m = t + 1 - \lceil \sqrt{t+1} \rceil$ . For  $t \geq 1$*

$$\sum_{i=1}^t \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} \leq D,$$

*Proof.* When  $t = 1$ ,

$$\sum_{i=1}^t \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} = 2\beta.$$

When  $t = 2$ ,

$$\sum_{i=1}^t \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} = \beta 3^\theta + \beta^2 \left(\frac{3}{4}\right)^\theta.$$

We want to check if the condition holds for  $t \geq 3$ . To show this, we will use  $m \geq \sqrt{t+1}$  for  $t \geq 3$ .

$$\begin{aligned}
\sum_{i=1}^t \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} &= \sum_{i=1}^m \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} + \sum_{i=m}^t \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} \\
&\leq \sum_{i=1}^m \beta^i + \sum_{i=m}^t \beta^m (t+1)^\theta \\
&\leq \frac{1 - \beta^{m+1}}{1 - \beta} + \beta^m (t+1)^\theta (t-m+1) \\
&\leq \frac{1 - \beta^{\sqrt{t+1}+1}}{1 - \beta} + \beta^{\sqrt{t+1}} (t+1)^\theta \lceil \sqrt{t+1} \rceil \\
&\leq \frac{1 - \beta^{\sqrt{t+1}+1}}{1 - \beta} + \beta^{\sqrt{t+1}} (t+1)^{\theta+1}.
\end{aligned}$$

We have

$$\beta^{\sqrt{t+1}} (t+1)^{\theta+1} < \max \left\{ \beta^2 4^{\theta+1}, \beta^{\frac{2(\theta+1)}{\ln \beta}} \left( \frac{2\theta+2}{\ln \beta} \right)^{2(\theta+1)} \right\}.$$

from checking the endpoints and stationary points of  $\beta^{\sqrt{t+1}} (t+1)^{\theta+1}$ . We also know

$$\frac{1 - \beta^{\sqrt{t+1}+1}}{1 - \beta} < \frac{1}{1 - \beta},$$

from taking  $t \rightarrow \infty$ . So, for  $t \geq 3$

$$\sum_{i=1}^t \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} \leq \frac{1}{1 - \beta} + \max \left\{ \beta^2 4^{\theta+1}, \beta^{\frac{2(\theta+1)}{\ln \beta}} \left( \frac{2\theta+2}{\ln \beta} \right)^{2(\theta+1)} \right\}.$$

For all  $t \geq 1$ , we have

$$\begin{aligned}
\sum_{i=1}^t \beta^i \frac{(t+1)^\theta}{(t-i+1)^{2\theta}} &\leq \max \left\{ 2\beta, \beta 3^\theta + \beta^2 \left( \frac{3}{4} \right)^\theta, \right. \\
&\quad \left. \frac{1}{1 - \beta} + \max \left\{ \beta^2 4^{\theta+1}, \beta^{\frac{2(\theta+1)}{\ln \beta}} \left( \frac{2\theta+2}{\ln \beta} \right)^{2(\theta+1)} \right\} \right\}.
\end{aligned}$$

□

From Theorem 5.4 and Lemma A.1 it immediately follows that

**Corollary A.1.1.** *Under Assumptions 5.1, 5.2, 5.3, 5.4, and 5.5, if Algorithm 8 and 9 is run with learning rate sequence  $\alpha_t = (t+1)^{-1}$  for  $T$  iterations, we get*

$$\begin{aligned} \frac{1}{\sum_{t=0}^T \alpha_t} \sum_{t=0}^T \alpha_t \mathbb{E}[\|\nabla F(w_t)\|_2^2] &\leq \frac{2}{\sum_{t=0}^T \alpha_t} (F(w_0) - F^*) \\ &\quad + \left( LM + \frac{L^2 MD(\sqrt{1-\gamma} + \rho)^2}{\lambda} \right) \frac{\sum_{t=0}^T \alpha_t^2}{\sum_{t=0}^T \alpha_t}. \end{aligned} \quad (\text{A.1})$$

where  $\lambda \in (0, \frac{1-\gamma}{\gamma})$  and  $D = \frac{1}{1-\gamma} \max \left\{ 2\beta, \beta 3^\theta + \beta^2 \left(\frac{3}{4}\right)^\theta, \max \left\{ \beta^2 4^{\theta+1}, \beta^{\frac{2(\theta+1)}{\ln \beta}} \left(\frac{2\theta+2}{\ln \beta}\right)^{2(\theta+1)} \right\} \right\}$ ,  
 where  $\beta = (1+\lambda)(1-\gamma)$ .

# Appendix B

## Learning Rates

Dataset	Model	Workers	unidirectional lr	bidirectional lr	SGD lr
Fashion-MNIST	MLP	20	0.08	0.08	0.06
		50	0.13	0.12	0.07
		100	0.22	0.22	0.08
	CNN	20	0.09	0.08	0.06
		50	0.12	0.11	0.07
		100	0.14	0.2	0.08
MNIST	MLP	20	0.06	0.09	0.03
		50	0.17	0.18	0.1
		100	0.17	0.24	0.1
	CNN	20	0.08	0.09	0.05
		50	0.14	0.16	0.07
		100	0.09	0.16	0.07
CIFAR10	VGG19	20	0.05	0.05	0.05

Table B.1: Learning rate chosen for all models.



# Appendix C

## Experimental Figures

### C.1 Testing Accuracy and Training Loss

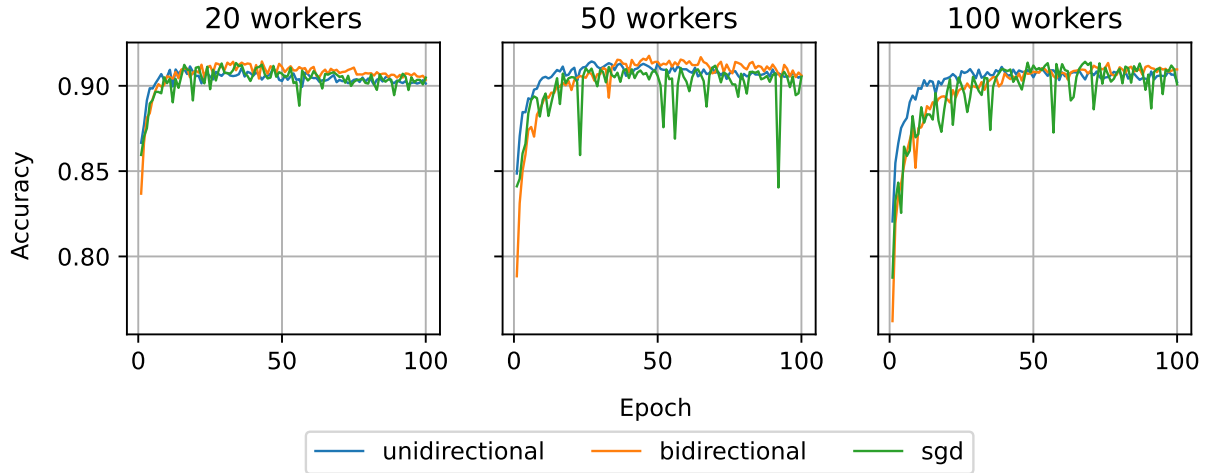


Figure C.1: Comparison of testing accuracy from training CNN model on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.



Figure C.2: Comparison of training loss from training CNN model on Fashion-MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

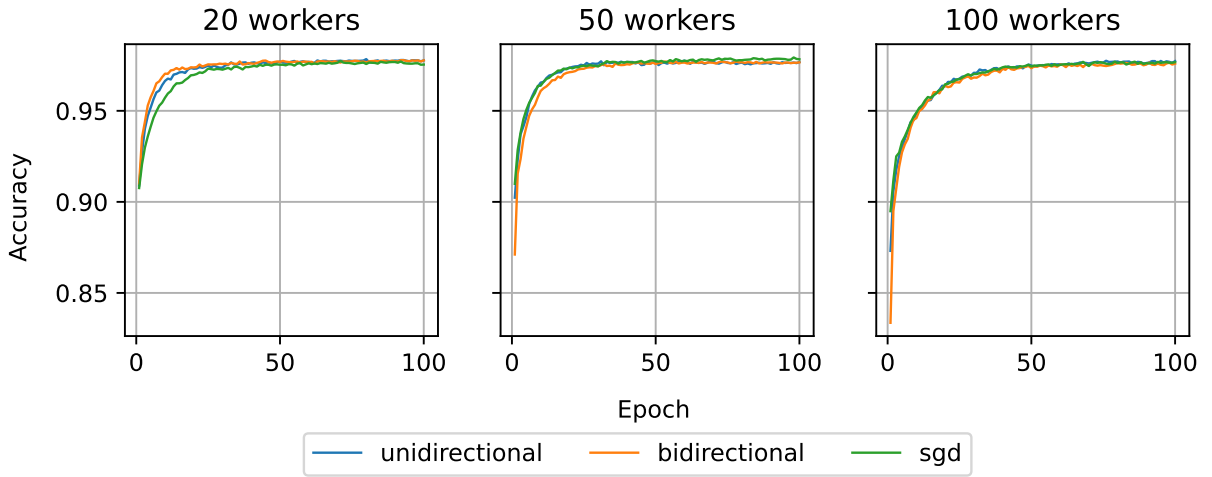


Figure C.3: Comparison of testing accuracy from training MLP model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

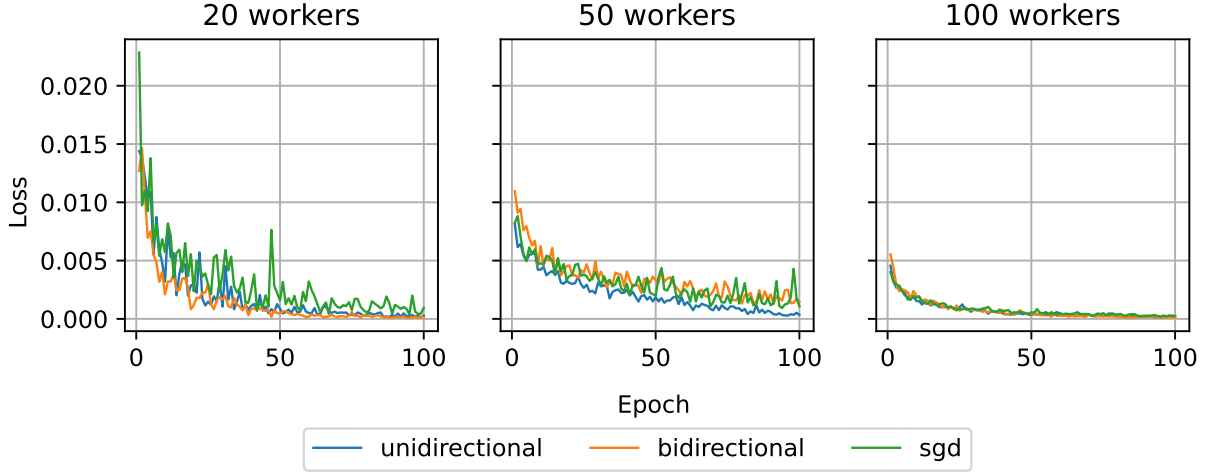


Figure C.4: Comparison of training loss from training MLP model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

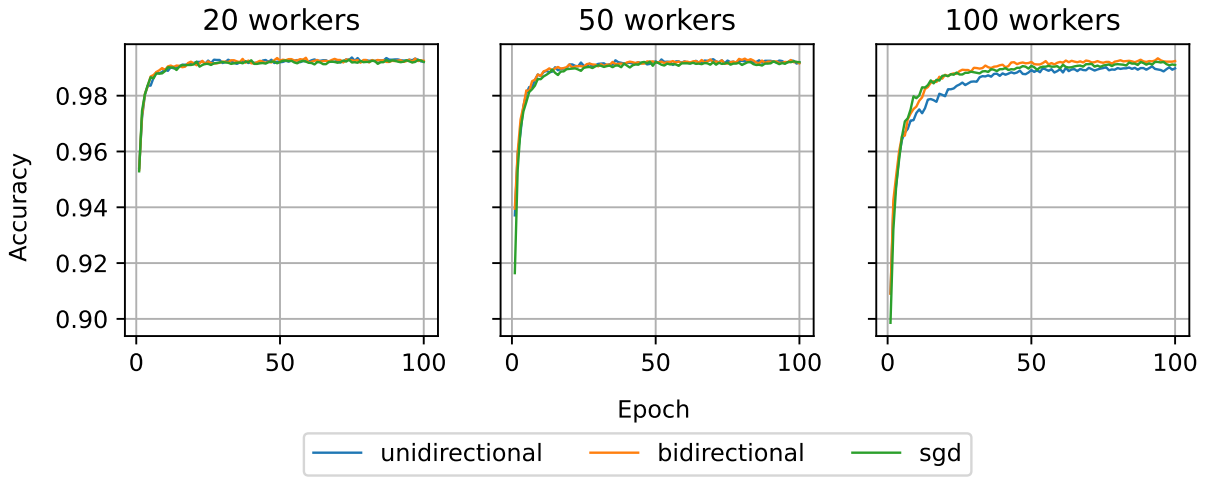


Figure C.5: Comparison of testing accuracy from training CNN model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

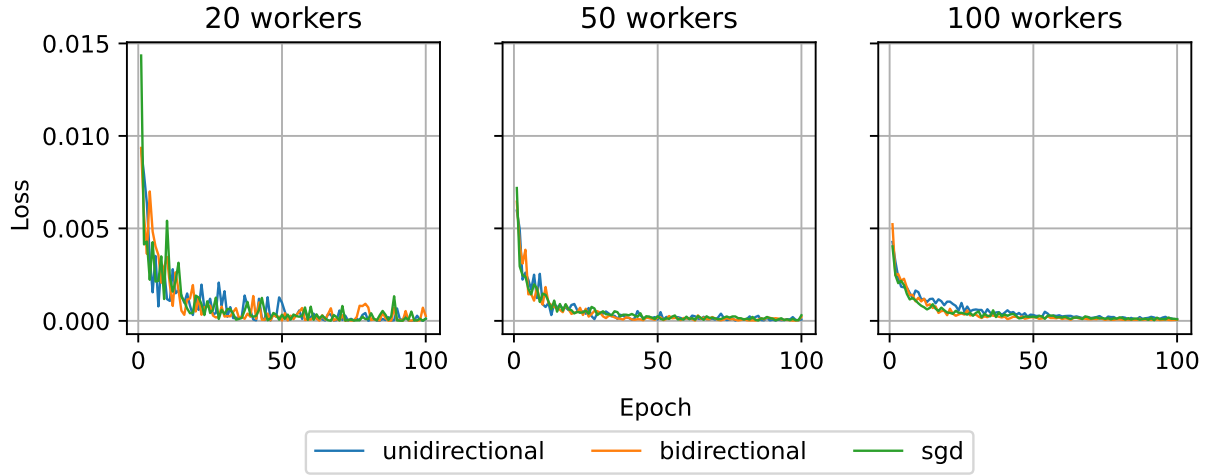


Figure C.6: Comparison of training loss from training CNN model on MNIST data for unidirectional  $\text{top}_K$ , bidirectional  $\text{top}_K$  and vanilla SGD. Trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

## C.2 $\hat{\rho}$ Values



Figure C.7: Comparison of  $\hat{\rho}$  values from training CNN model on Fashion-MNIST data for unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$ . Models trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

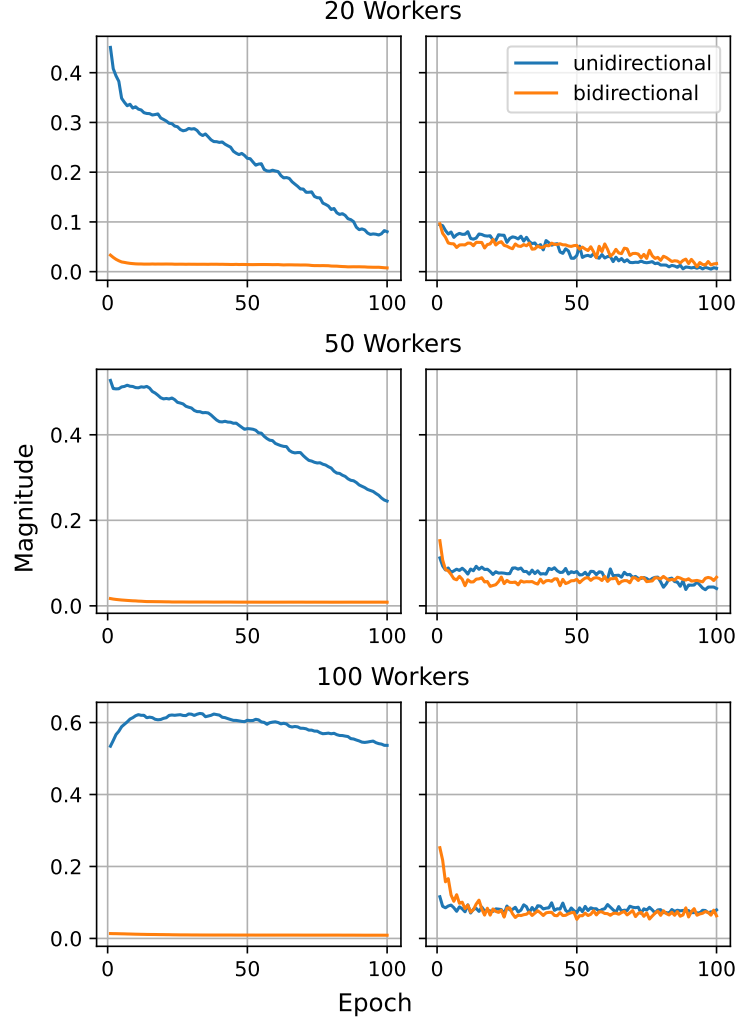


Figure C.8: Results from training CNN model on Fashion-MNIST data. Left side graphs are the largest  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2$  values in each epoch for unidirectional top<sub>K</sub> SGD and the largest  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2$  values in each epoch for bidirectional top<sub>K</sub> SGD. Right side graphs are the smallest  $\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2$  value in each epoch.

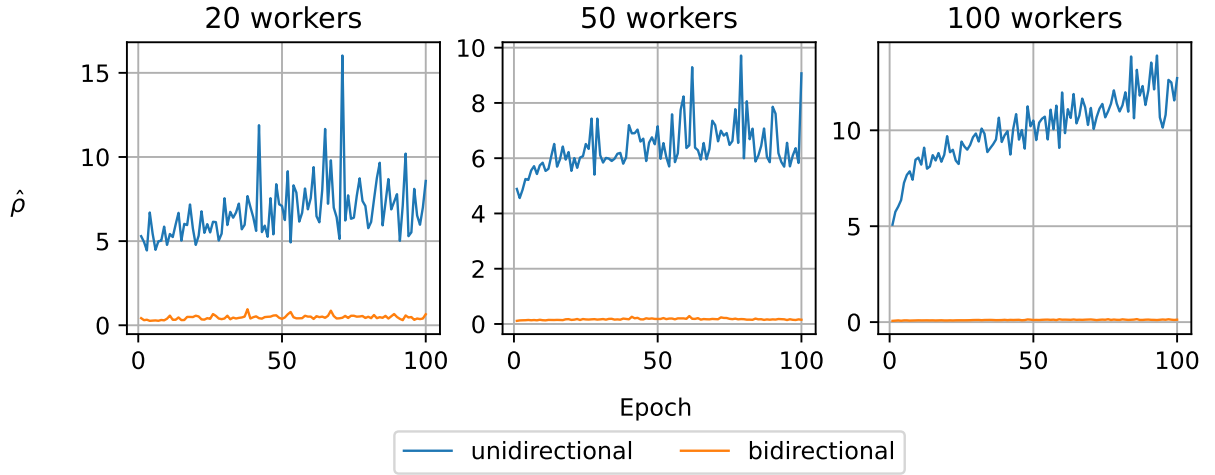


Figure C.9: Comparison of  $\hat{\rho}$  values from MLP Model trained on MNIST data for unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$ . Models trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

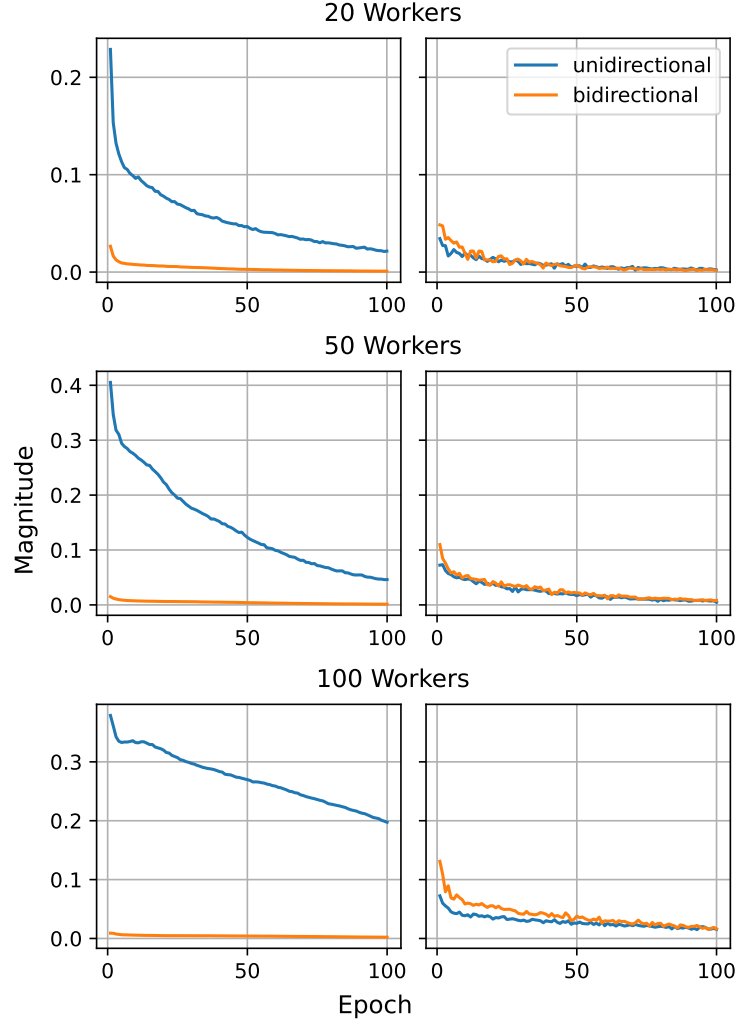


Figure C.10: Results from training MLP model on MNIST data. Left side graphs are the largest  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2$  values in each epoch for unidirectional top<sub>K</sub> SGD and the largest  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2$  values in each epoch for bidirectional top<sub>K</sub> SGD. Right side graphs are the smallest  $\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2$  value in each epoch.



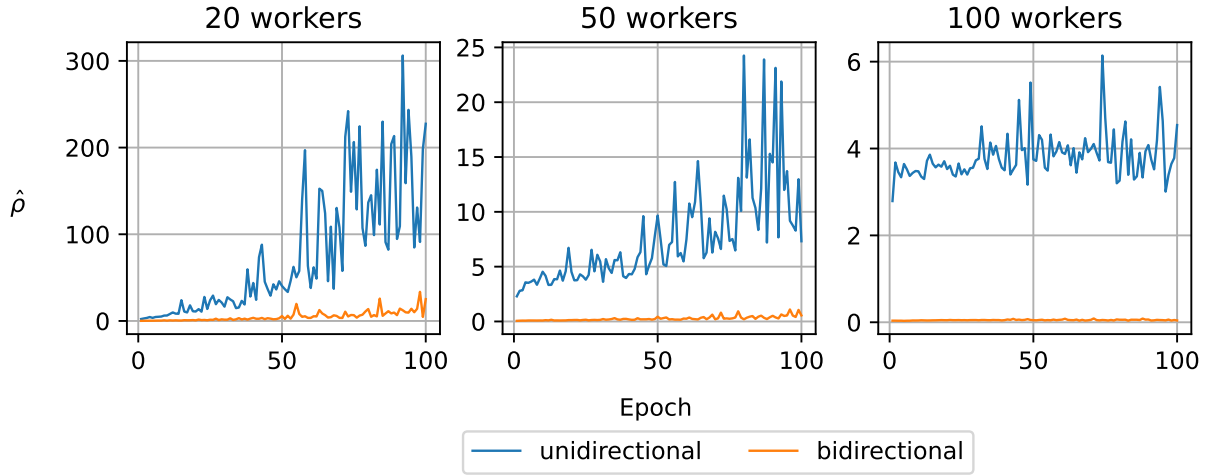


Figure C.11: Comparison of  $\hat{\rho}$  values from CNN Model trained on MNIST data for unidirectional  $\text{top}_K$  and bidirectional  $\text{top}_K$ . Models trained on batch size 10.  $K_{\text{downlink}} = K_{\text{uplink}} \approx 0.001d$  for bidirectional.  $K_{\text{uplink}} \approx 0.001d$  for unidirectional.

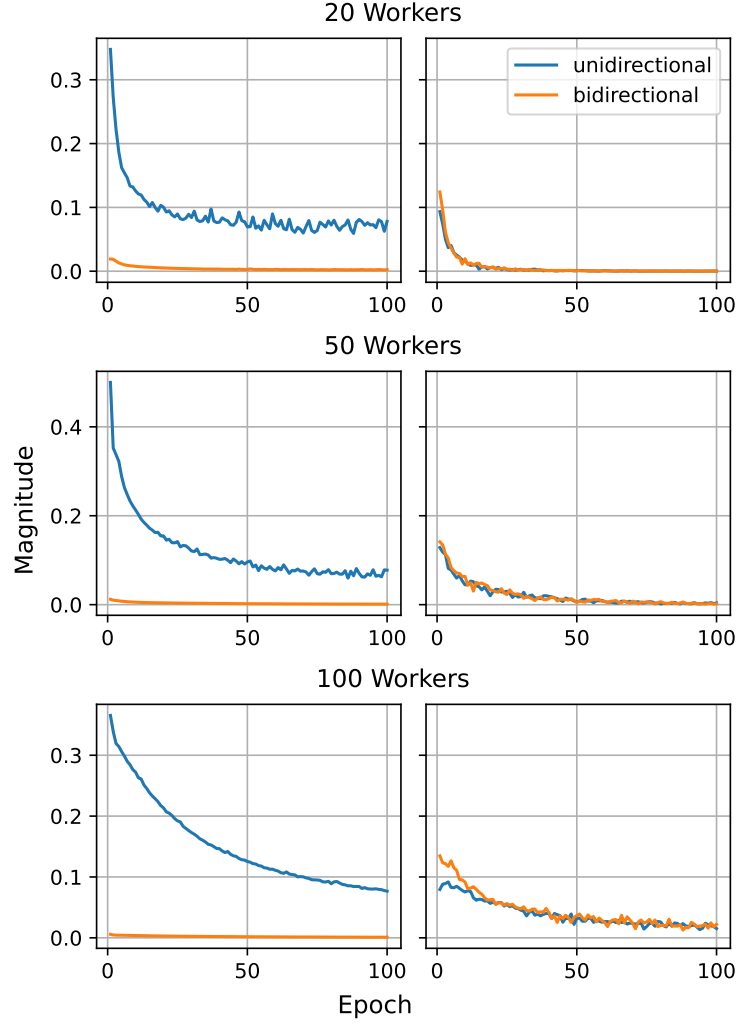


Figure C.12: Results from training CNN model on MNIST data. Left side graphs are the largest  $\|\text{Top}_K(\sum_{q=1}^N p_q a_t^q) - \sum_{q=1}^N \text{Top}_K(p_q a_t^q)\|_2$  values in each epoch for unidirectional top<sub>K</sub> SGD and the largest  $\|\text{Top}_K(\delta_{t-1} + \sum_{q=1}^N p_q a_t^q) - \text{Top}_K(\delta_{t-1} + \sum_{q=1}^N \text{Top}_K(p_q a_t^q))\|_2$  values in each epoch for bidirectional top<sub>K</sub> SGD. Right side graphs are the smallest  $\|\alpha_{t-1} \sum_{q=1}^N p_q g^q(w_{t-1}, \xi_{t-1}^q)\|_2$  value in each epoch.

### C.3 Percent of Non-Zero Indices in Gradient After Server Aggregation

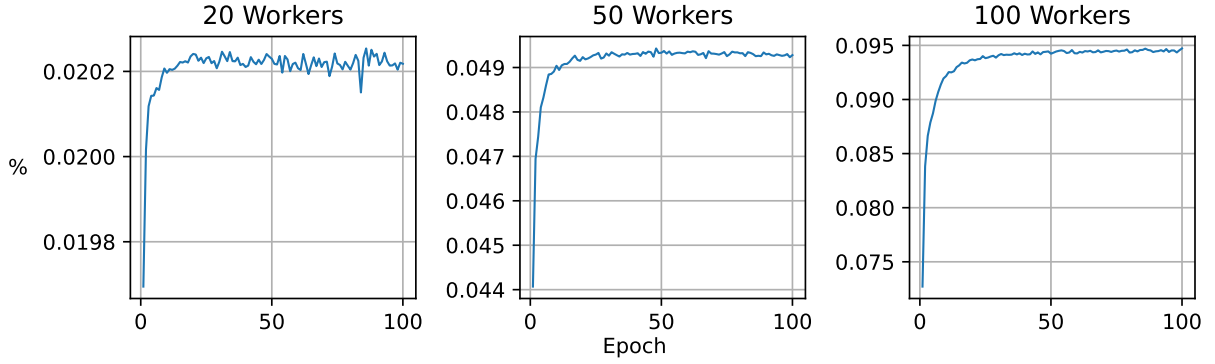


Figure C.13: Percent of non-zero indices after aggregating sparse gradients from workers. Trained on a CNN network using Fashion-MNIST dataset.

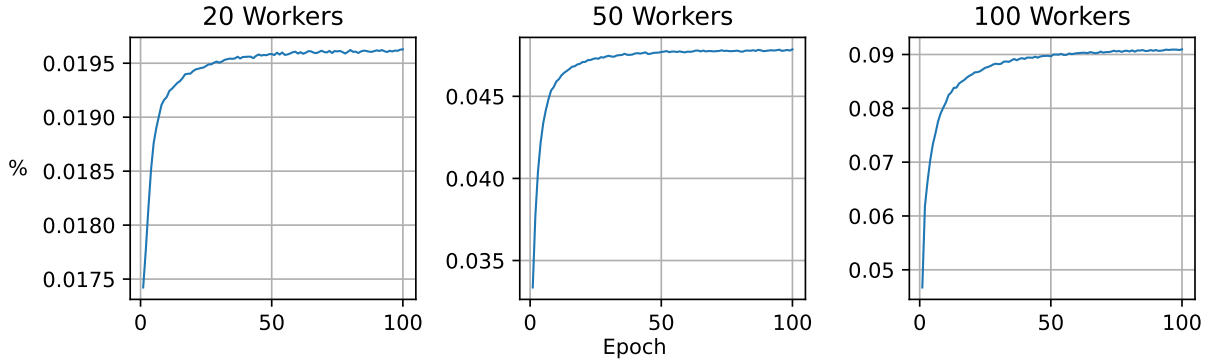


Figure C.14: Percent of non-zero indices after aggregating sparse gradients from workers. Trained on a MLP network using MNIST dataset.

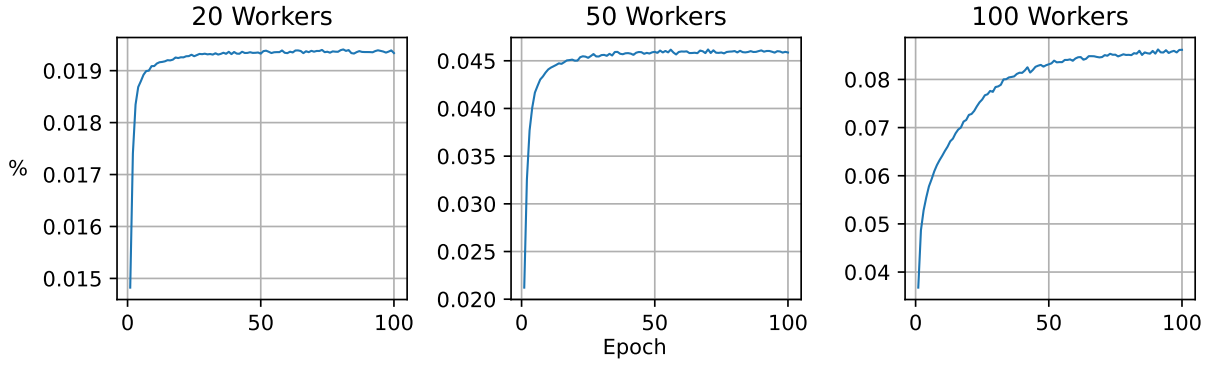


Figure C.15: Percent of non-zero indices after aggregating sparse gradients from workers. Trained on a CNN network using MNIST dataset.

## C.4 $1 - \hat{\gamma}$ Values

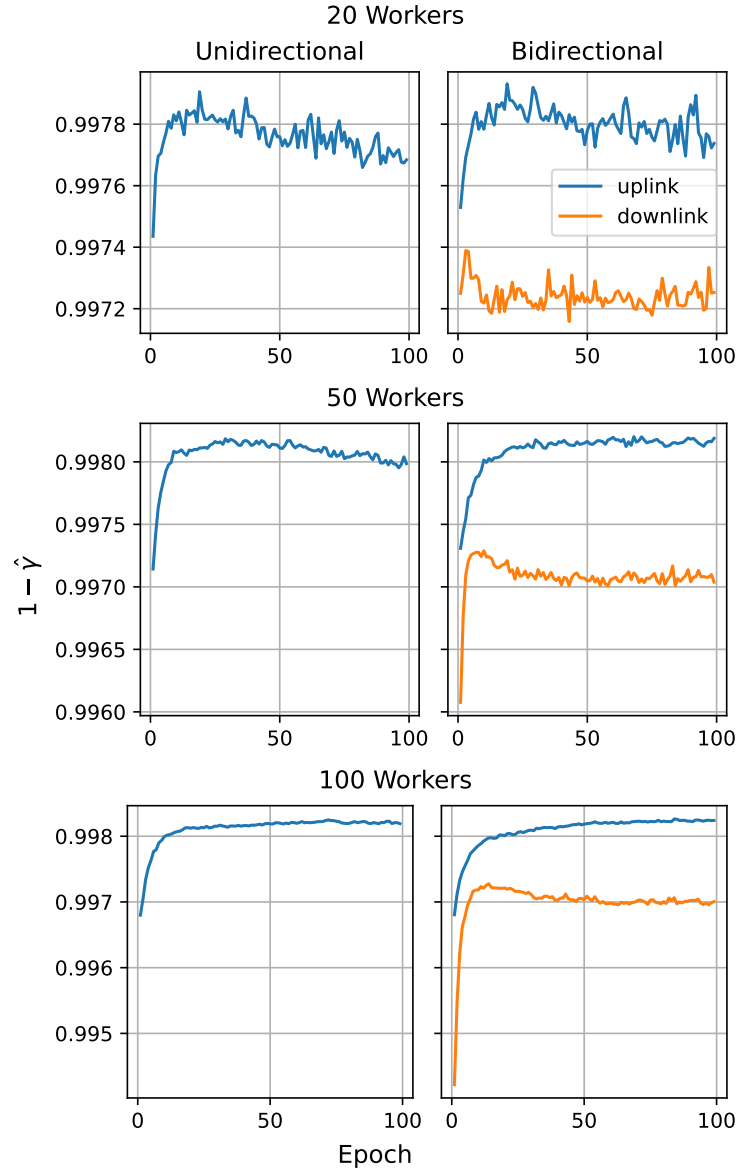


Figure C.16: Largest  $1 - \hat{\gamma}$  value in each epoch of a CNN model trained with unidirectional and bidirectional  $\text{top}_K$  SGD on a Fashion-MNIST dataset.

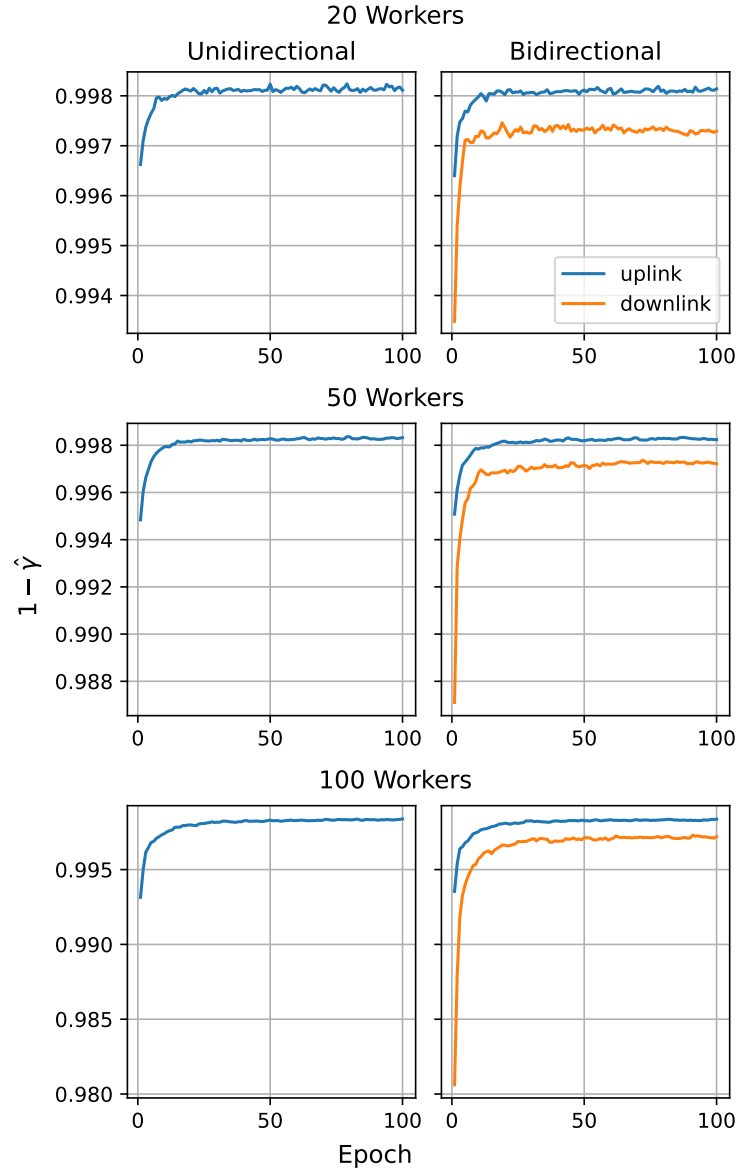


Figure C.17: Largest  $1 - \hat{\gamma}$  value in each epoch of a MLP model trained with unidirectional and bidirectional  $\text{top}_K$  SGD on a MNIST dataset.

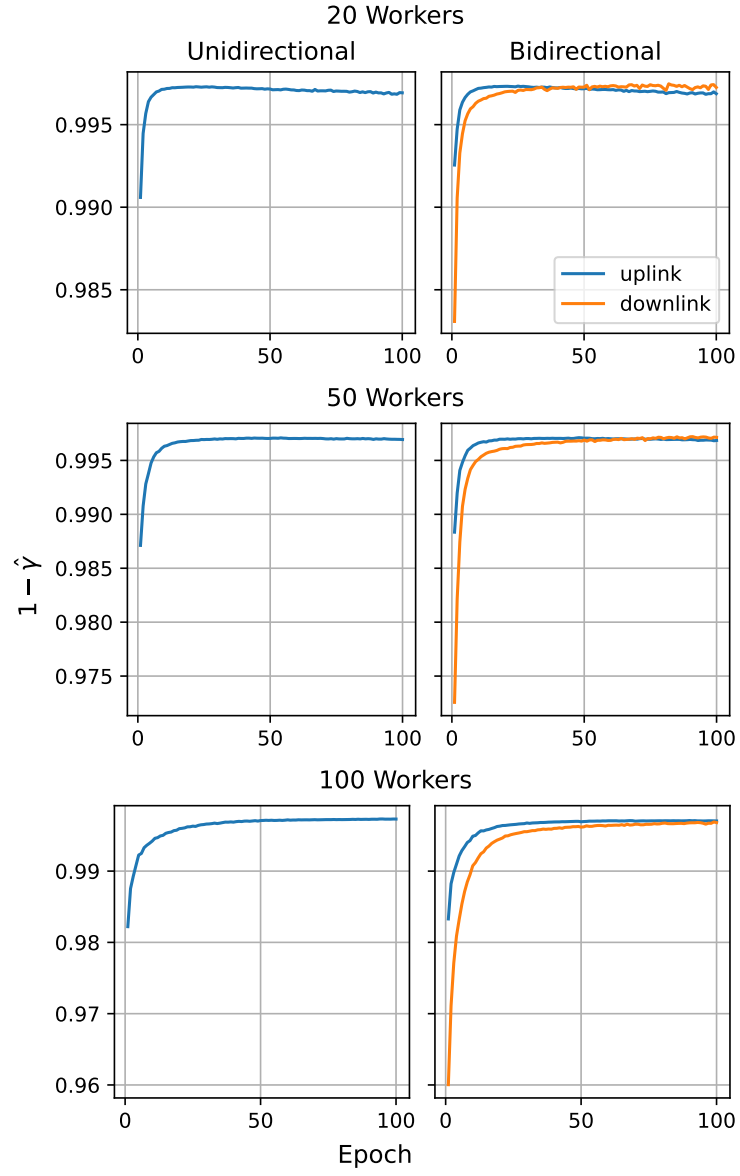


Figure C.18: Largest  $1 - \hat{\gamma}$  value in each epoch of a CNN model trained with unidirectional and bidirectional  $\text{top}_K$  SGD on a MNIST dataset.

# Appendix D

## Convergence Bound Constants

Dataset	Model	Workers	$\hat{\rho}^{\text{unidirectional}}$	$\hat{\rho}^{\text{bidirectional}}$
Fashion-MNIST	MLP	20	21.78	0.60
		50	5.86	0.14
		100	6.65	0.08
	CNN	20	15.35	0.92
		50	7.48	0.24
		100	9.05	0.18
MNIST	MLP	20	16.03	0.95
		50	9.72	0.28
		100	13.90	0.15
	CNN	20	306.19	33.60
		50	24.26	1.09
		100	6.14	0.09
CIFAR10	VGG19	20	4.16	0.21

Table D.1: Maximum  $\hat{\rho}$  value of trained models across all epochs.