Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices

Max Ryabinin[*]
Yandex, Russia
HSE University, Russia

Eduard Gorbunov[*]
MIPT, Russia
HSE University, Russia
Yandex, Russia

Vsevolod Plokhotnyuk
Yandex, Russia
HSE University, Russia

Gennady Pekhimenko
University of Toronto, Canada
Vector Institute, Canada

Abstract

Training deep neural networks on large datasets can often be accelerated by using multiple compute nodes. This approach, known as distributed training, can utilize hundreds of computers via specialized message-passing protocols such as Ring All-Reduce. However, running these protocols at scale requires reliable high-speed networking that is only available in dedicated clusters. In contrast, many real-world applications, such as federated learning and cloud-based distributed training, operate on unreliable devices with unstable network bandwidth. As a result, these applications are restricted to using parameter servers or gossip-based averaging protocols. In this work, we lift that restriction by proposing Moshpit All-Reduce — an iterative averaging protocol that exponentially converges to the global average. We demonstrate the efficiency of our protocol for distributed optimization with strong theoretical guarantees. The experiments show 1.3x speedup for ResNet-50 training on ImageNet compared to competitive gossip-based strategies and 1.5x speedup when training ALBERT-large on preemptible compute nodes.

1 Introduction

Many recent influential discoveries in deep learning were enabled by the trend of scaling model and dataset size. Over the last decade, computer vision has grown from training models with 60 million parameters [1] on 1.3 million images [2] to 15 times more parameters [3] and 200 times more training data [4]. In natural language processing, the state-of-the-art language models [5] with 175 billion parameters are trained on over 570GB of texts, and even this does not saturate the model quality [6]. Training these large models can take years even with a top-of-the-line GPU server [7]. As a result, researchers and practitioners often have to run distributed training with multiple machines [8].

The dominant approach to distributed deep learning is data-parallel training [9], where each worker processes a fraction of the training batch and then exchanges its gradients with peers. If done naïvely, the gradient exchange step can overload the network as the number of workers increases. To combat this issue, modern distributed training algorithms take advantage of communication-efficient protocols, such as all-reduce [10]. These protocols allow workers to collectively compute the global average gradient with a constant communication overhead, regardless of the total number of peers.

[*]Equal contribution. Correspondence to `mryabinin0@gmail.com`.

However, this efficiency makes the protocols more fragile: if any single participant fails or takes too long to process its batch, all other nodes are stalled. Therefore, scaling all-reduce protocols beyond a couple of servers requires specialized infrastructure with dedicated ultra-high bandwidth networking [8]. This kind of infrastructure is notoriously expensive compared to regular GPU servers or preemptible cloud VMs (see Appendix A for details).

Hence, it is tempting to consider distributed training on cheap unreliable instances as a cost-efficient alternative. A similar scenario arises in federated learning [11], where a single model is trained on heterogeneous devices due to privacy concerns. In both scenarios, workers use a shared network, where both latency and bandwidth can vary drastically due to interference from other users [12]. Furthermore, compute nodes are also subject to failure (or preemption) caused by factors beyond the protocol's control.

Running large-scale distributed training in these circumstances requires fault- and latency-tolerant algorithms [14, 15]. Most of these algorithms replace all-reduce averaging with **gossip**: each participant periodically downloads the latest parameters from their neighbors in a sparsely connected communication graph and averages the results. The updates gradually propagate through the graph over multiple rounds of averaging. However, the communication required to perform gossip grows linearly with the number of neighbors. Hence, when scaling to hundreds of peers, decentralized SGD has to keep the communication graph sparse, slowing down the convergence.

In this work, we propose an alternative approach. Instead of relying on a predefined communication graph, participants dynamically organize themselves into groups using a fully decentralized matchmaking algorithm called **Moshpit All-Reduce**. This strategy allows us to use communication-efficient all-reduce protocols that significantly reduce the network load compared to gossip-based averaging, while still being able to operate in unreliable hardware and network conditions.

Our contributions can be summarized as follows:

- We propose **Moshpit All-Reduce** — a novel decentralized averaging protocol for large-scale training with unreliable communication-constrained devices. According to our analysis, this method has exponential convergence rate independent of network topology and size.
- Armed with this averaging protocol, we develop **Moshpit SGD** for distributed optimization. We derive convergence rates for this algorithm and establish its equivalence to Centralized (Local) SGD in terms of iteration complexity under realistic assumptions.
- Our experiments demonstrate that Moshpit All-Reduce is significantly more efficient under network latency in realistic conditions. In particular, we train ResNet-50 on ImageNet to 75% accuracy 1.3 times faster than existing decentralized training algorithms and pretrain ALBERT-large 1.5 times faster on preemptible cloud VMs.[2]

2 Related Work

2.1 Data parallel training

The most popular way to accelerate neural network training with multiple devices is data-parallel training [9, 16, 17]. On each optimization step, this strategy splits the training batch among participants. Each participant then runs forward and backward passes to obtain gradients of the objective function on their part of the training batch. After that, we can aggregate the gradients from workers and perform an optimization step. There are two main strategies for this aggregation.

Historically, the first solution to gradient aggregation was to use Parameter Server (PS) [18]: a separate process or a dedicated server that keeps track of model parameters and optimizer statistics. After each round, the PS accumulates the gradients from each worker and updates the model parameters using SGD or any other optimizer, such as Adam [19]. Finally, the server distributes the updated model parameters to workers.

This strategy is robust and easy to implement, but it requires the server to regularly download full model gradients from every single worker. As a result, the parameter server can quickly become a bottleneck for large-scale training [20]. Since the original PS, researchers have proposed several modifications that reduce the communication load: accumulating multiple batches [22], compression [23, 24], server sharding [25, 26]. A more detailed overview is given in Appendix B.

[2]Implementation and code of experiments are at `github.com/yandex-research/moshpit-sgd`.

In turn, many practical distributed training systems have instead switched to averaging with All-Reduce [16, 27, 28, 17]. This name refers to a collection of protocols originally developed for HPC applications. Workers can follow these protocols to collectively compute the average[3] gradient more efficiently than with a central server.

2.2 Communication-efficient All-Reduce

There are several all-reduce protocols optimized for different network topologies. The simplest one is known as Butterfly All-Reduce [10]. Each of N participants splits its local vector into N chunks. Then, i-th worker aggregates i-th chunk of data from all peers and sends back the averaged chunk.

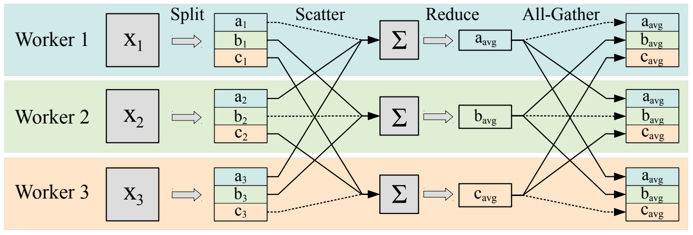

Figure 1: A schematic illustration of Butterfly All-Reduce.

As long as the vector size s is greater than N, this protocol uses $\mathcal{O}\left(s \times \frac{N-1}{N}\right)$ total bandwidth on each worker. However, it requires all-to-all communication, which is not always practical for the HPC infrastructure due to network contention [10]. As a result, real-world systems typically use Ring or Tree All-Reduce, where each worker only communicates with a small subset of its peers.

These protocols enable highly efficient and scalable averaging with $\mathcal{O}(1)$ or $\mathcal{O}(\log N)$ total communication per worker, but they also share a common drawback: they cannot tolerate node failures or network instability. If any single participant fails to execute its part or takes long to respond, this paralyzes all other workers.

2.3 Distributed training in unstable conditions

Some distributed training applications must deal with unstable network bandwidth and/or unreliable workers. This issue is most prevalent in federated learning [11, 29, 30]. When dealing with privacy-sensitive data distributed across multiple actors, such as hospital servers [31, 32] or mobile phones [33, 34], one must train the model using whichever hardware and network available to those actors.

Another important motivational factor is cost: HPC-grade infrastructure can be prohibitively expensive, pushing researchers and practitioners towards commodity servers or preemptible cloud VMs that are significantly cheaper (see Appendix A). Another solution is to use volunteer computing [35, 36] with abundant, but even less reliable, compute resources.

Training under these conditions requires specialized strategies. At a small scale, one can deploy one or a few reliable parameter servers to aggregate the updates from workers. This strategy can tolerate individual node failures [37], but scales poorly due to the reasons discussed in Section 2.1.

2.4 Decentralized training

If there are too many participants for PS, it can be advantageous to use decentralized SGD via **gossip-based** averaging [38, 39, 14]. In this scenario, participants form a sparse graph: each worker periodically downloads parameters from its neighbors and mixes them with local parameters.

In essence, gossip-based averaging removes the communication bottlenecks of PS at the cost of using different local parameters on each peer. That said, gossip-based optimization algorithms can match, and sometimes even outperform, their centralized counterparts in terms of training speed [40, 41, 42, 14, 43]. However, the convergence properties of gossip averaging and gossip-based optimization methods significantly depend on the communication graph through the spectral properties of the mixing matrix [44, 42] or the Laplacian matrix of the network [45, 46].

[3]All-Reduce works with any commutative associative operation, such as min, max, or product.

Consequently, as the number of peers increases, gossip-based averaging has to either increase the number of neighbors (hence more communication) or accept slower convergence speed. Because of this, gossip is less communication-efficient than all-reduce algorithms reviewed in Section 2.2. However, gossip-based algorithms are more robust to changes, which makes them applicable to time-varying networks [47, 48, 49, 50] and federated learning [51, 52, 53].

3 Moshpit SGD

Large-scale training with unreliable participants requires a protocol that is both communication-efficient and fault-tolerant. Unfortunately, existing methods have only provide one of these properties. To better address our conditions, we propose Moshpit All-Reduce — a fully decentralized averaging protocol that combines the efficiency of all-reduce and the fault tolerance of gossip-based averaging.

The rest of this section is organized as follows:

- Section 3.1 describes the protocol and proves its correctness and communication efficiency;
- Section 3.2 provides the analysis of the protocol and proves exponential convergence rate for averaging and the rate matching the one of centralized Local-SGD for optimization;
- Section 3.3 contains implementation details for training with heterogeneous compute nodes.

3.1 Moshpit All-Reduce

The core idea of Moshpit All-Reduce is that workers perform averaging in small independent groups. That way, a single failed participant would only affect his current group. In turn, the composition of each group should be chosen dynamically to converge in the least number of steps. Ideally, if there are 9 peers with local parameters θ, we can average them in 2 rounds, as demonstrated in Figure 2:

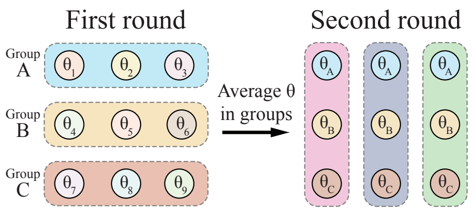

Figure 2: Example averaging order for 9 peers in 2 rounds. On each round, peers are split into 3 groups that run All-Reduce in parallel.

Algorithm 1 Moshpit All-Reduce (for i-th peer)

```
Input: parameters $\{\theta_j\}_{j=1}^N$, number of peers $N$, $d$, $M$, number of iterations $T$, peer index $i$
$\theta_i^0 := \theta_i$
$C_i^0 :=$ get_initial_index(i)
for $t \in 1 \dots T$ do
    DHT$[C_i^{t-1}, t]$.add(address$_i$)
    Matchmaking() // wait for peers to assemble
    peers$_t :=$ DHT.get($[C_i^{t-1}, t]$)
    $\theta_i^t, c_i^t :=$ AllReduce($\theta_i^{t-1}$, peers$_t$)
    $C_i^t := (C_i^{t-1}$[1:]$, c_i^t)$ // same as eq. (1)
end for
Return $\theta_i^T$
```

To achieve this in a decentralized system, we use Distributed Hash Tables (DHT) — a decentralized key-value storage; Appendix B contains its more detailed description. On each averaging round:

- Each worker computes its group key C_i;
- Workers add their network addresses to the DHT key corresponding to C_i;
- Each worker can now fetch a full list of peers that have the same C_i and run All-Reduce with those peers.

Unfortunately, the averaging structure from Figure 2 is impossible to maintain when participants are constantly joining, leaving, and failing. However, we can achieve equivalent results without global structure using a simple rule: *if two peers were in the same group in round t, they must choose different groups in round* t+1.

A natural way to enforce this rule is to take advantage of the chunk indices from Butterfly All-Reduce (see Figure 1). Recall that each worker accumulates a *unique* chunk of parameters defined by an index c_i. By setting $C_i := c_i$, we can guarantee that any workers that were in the same group at a round t will have different group indices in round t+1.

This averaging scheme can be generalized to more than two dimensions in order to fit a larger number of peers or reduce the group size. For a d-dimensional hypercube, nodes should find groups of peers that they have not communicated with during $d{-}1$ previous rounds. To that end, we define C_i as tuples containing chunk indices from $d{-}1$ previous rounds (t denotes the communication round):

$$C_i^t := (c_i^{t-d+1}, c_i^{t-d+2}, \dots, c_i^t). \tag{1}$$

The above intuition can be formalized with Algorithm 1. Here, N peers form a virtual d-dimensional grid with M peers per row and average their parameters θ_i over T rounds. `DHT`$[\cdot]$ is a shortcut for using the DHT to add or retrieve values for a given key. The `Matchmaking` step corresponds to the decentralized matchmaking procedure that organizes active workers with the same index into groups, described in detail in Appendix E. In turn, `AllReduce` denotes running all-reduce to compute the average θ in a given group. The `get_initial_index` function takes the peer index i and returns $d{-}1$ integers in range $[0, M)$ such that the size of initial groups does not exceed M. This way, the groups formed on subsequent rounds will also have at most M participants. One possible strategy is:

$$\texttt{get\_initial\_index}(i) = \left(\lfloor i/M^{d-1} \rfloor \bmod M\right)_{j \in \{1,\ \dots,\ d\}} \tag{2}$$

If $N{=}M^d$ and there are no node/network failures, Algorithm 1 is equivalent to Torus All-Reduce [54], achieving the exact average after d rounds of communication (see Appendix C.1). However, our typical use case is far from this perfect scenario; for example, some groups can have less than M members. Furthermore, a peer might fail during all-reduce, causing its groupmates to skip a round of averaging. Still, Moshpit All-Reduce is applicable even in these conditions:

Theorem 3.1 (Correctness). *If all workers have a non-zero probability of successfully running a communication round and the order of* $\texttt{peers}_t$ *is random, then all local vectors* θ_i^t *converge to the global average with probability 1:*

$$\forall i, \left|\left|\theta_i^t - \frac{1}{N}\sum_i \theta_i^0\right|\right|_2^2 \xrightarrow[t\to\infty]{} 0. \tag{3}$$

Proof (sketch, complete in Appendix C.2). Running all-reduce with a subset of peers preserves the invariant $\frac{1}{N}\sum_i \theta_i^t = \frac{1}{N}\sum_i \theta_i^{t-1}$ and reduces the deviation of θ_i^t from the overall average. □

Complexity. The matchmaking protocol is implemented over Kademlia DHT [55], meaning that each read and write operation needs at most $\mathcal{O}(\log N)$ requests and $\mathcal{O}(M)$ bandwidth to load $\texttt{peers}_t$.

After the matchmaking is over, each group runs a single all-reduce round to compute the average. In principle, Moshpit Averaging can use any general-purpose all-reduce protocol. We opted for a butterfly-like version (Figure 1), as it is simpler than Ring All-Reduce while still being communication-efficient. The communication complexity of this algorithm is $\mathcal{O}\left(\max(s, M) \times \frac{M-1}{M}\right)$, where s is the size of vector θ. Thus, the total time complexity of Algorithm 1 becomes:

$$\mathcal{O}\left(T \times \left[\log_2 N + M + \max(s, M) \times \frac{M-1}{M}\right]\right). \tag{4}$$

This compares favorably to gossip, where network load grows linearly with the number of neighbors.

3.2 Convergence analysis

3.2.1 Mixing properties of Moshpit Averaging

As stated in the previous section, Moshpit All-Reduce computes the exact average when $N = M^d$, which cannot be guaranteed in practice. Therefore, additional analysis is needed to establish how quickly Moshpit Averaging approximates the actual average of N vectors stored on peers.

In the following theorem, we provide such analysis for a simplified version of Moshpit Averaging. One can find the full proof in Appendix C.3.

Theorem 3.2. *Consider a modification of Moshpit All-Reduce that works as follows: at each iteration* $k \geq 1$, *1) peers are randomly split in* r *disjoint groups of sizes* $M_1^k, \dots, M_r^k$ *in such a way that* $\sum_{i=1}^r M_i^k = N$ *and* $M_i^k \geq 1$ *for all* $i = 1, \dots, r$ *and 2) peers from each group compute their group average via All-Reduce. Let* $\theta_1, \dots, \theta_N$ *be the input vectors of this procedure and* $\theta_1^T, \dots, \theta_N^T$ *be the outputs after* T *iterations. Also, let* $\overline{\theta} = \frac{1}{N}\sum_{i=1}^N \theta_i$ *Then,*

$$\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^N \|\theta_i^T - \overline{\theta}\|^2\right] = \left(\frac{r-1}{N} + \frac{r}{N^2}\right)^T \frac{1}{N}\sum_{i=1}^N \|\theta_i - \overline{\theta}\|^2. \tag{5}$$

Algorithm 2 Moshpit SGD

```
1: Input: starting point $\theta^0$, learning rate $\gamma > 0$, communication period $\tau \geq 1$
2: for $k = 0, 1, \ldots$ do
3:     for each peer $i \in P_{k+1}$ in parallel do
4:         Compute the stochastic gradient $g_i^k$ at the current point $\theta_i^k$
5:         if $k + 1 \mod \tau = 0$ then
6:             $\theta_i^{k+1} =$ Moshpit All-Reduce$_{j \in P_{k+1}}(\theta_j^k - \gamma g_j^k)$ for $i$-th peer (Algorithm 1)
7:         else
8:             $\theta_i^{k+1} = \theta_i^k - \gamma g_i^k$
9:         end if
10:    end for
11: end for
```

In particular, this result implies that even if workers are randomly split into pairs at each iteration, the simplified version of Moshpit Averaging makes the average distortion (the left-hand side of Equation 5) less than ε in expectation after $\mathcal{O}\left(\log(1/\varepsilon)\right)$ iterations. That is, this algorithm finds ε-accurate average on each node with the rate that *does not* depend on the spectral properties of the communication graph like gossip and its variants (see Section 2.4 and Appendix B.1). Since Moshpit Averaging prevents two peers from participating in the same groups during successive iterations, the actual algorithm should find ε-accurate averages on participating peers even faster than Equation 5 predicts. Moreover, in Appendix C.3 we explain how this result can be generalized to the case when $\{M_i^k\}_{i=1}^N$ and r depends on k or even is random. In Appendix C.4, we also provide the guarantees measuring how fast Algorithm 1 reduces the variance when averaging random vectors.

3.2.2 Moshpit SGD

We consider a classical distributed optimization problem

$$\min_{\theta \in \mathbb{R}^n} \left\{ f(\theta) = \frac{1}{N} \sum_{i=1}^N f_i(\theta) \right\}, \tag{6}$$

where N is the number of workers and worker i has access only to the function f_i.
We propose a new algorithm called Moshpit SGD to solve this problem (see Algorithm 2). In this algorithm, workers perform independent local SGD steps and periodically synchronize their parameters θ_i^k with other peers using Moshpit All-Reduce. Moreover, we define the indices of participating nodes at iteration k as P_{k+1} ($P_0 = \{1, \ldots, N\}$) allowing peers to vanish.

First of all, we list the key assumptions that we use in the convergence analysis of Moshpit SGD.

Assumption 3.1 (Bounded variance)**.** *We assume that for all $k \geq 0$ and $i = 1, \ldots, N$ stochastic gradients g_i^k satisfy $\mathbb{E}\left[g_i^k \mid \theta_i^k\right] = \nabla f_i(\theta_i^k)$ and*

$$\mathbb{E}\left[\|g_i^k - \nabla f_i(\theta_i^k)\|^2 \mid \theta_i^k\right] \quad \leq \quad \sigma^2. \tag{7}$$

This assumption is classical in the stochastic optimization literature [56, 57]. We notice that our analysis can be generalized to the settings when the stochastic gradients satisfy less restrictive assumptions such as expected smoothness [58] or have more sophisticated structure similar to [59] using the theoretical framework from [60].

The following assumption controls the averaging properties and the effect of the peers' vanishing.

Assumption 3.2 (Averaging quality & peers' vanishing)**.** *We assume that the vanishing of peers does not change the global average of the iterates of Moshpit SGD too much, i.e., $P_{k+1} \subseteq P_k$ and $|P_k| \geq N_{\min}$ for all $k \geq 0$, $|P_{a\tau}| \leq 2|P_{a(\tau+1)}|$ for all non-negative integers $a \geq 0$, and there exist such $\widetilde{\theta} \in \mathbb{R}^n$ and a sequence of non-negative numbers $\{\Delta_{pv}^k\}_{k \geq 0}$ that $\forall k \geq 0$*

$$\mathbb{E}\left[\langle \theta^{k+1} - \widehat{\theta}^{k+1}, \theta^{k+1} + \widehat{\theta}^{k+1} - 2\widetilde{\theta} \rangle\right] \leq \Delta_{pv}^k, f \text{ convex;} \tag{8}$$

$$\mathbb{E}\left[\langle \nabla f(\theta^k), \theta^{k+1} - \widehat{\theta}^{k+1} \rangle + L\|\widehat{\theta}^{k+1} - \theta^{k+1}\|^2\right] \leq \Delta_{pv}^k, f \text{ non-convex, } L\text{-smooth, (Def. D.1)} \tag{9}$$

where $N_k = |P_k|$, $\theta^{k+1} = \frac{1}{N_{k+1}} \sum_{i \in P_{k+1}} \theta_i^{k+1}$, and $\widehat{\theta}^{k+1} = \frac{1}{N_k} \sum_{i \in P_k} (\theta_i^k - \gamma g_i^k)$ for $k \geq 0$.

Moreover, we assume that for some $\delta_{aq} \geq 0$ and for all non-negative integers $a \geq 0$,

$$\mathbb{E}\left[\frac{1}{N_{a\tau}}\sum_{i\in P_{a\tau}}\|\theta_i^{a\tau}-\theta^{a\tau}\|^2\right] \quad\leq\quad \gamma^2\delta_{aq}^2. \tag{10}$$

If $P_k = P_{k+1} = \{1,\ldots,N\}$ for all $k \geq 0$, i.e., peers do not vanish, then $\theta^k = \widehat{\theta}^k$ and properties (8, 9) hold with $\Delta_{pv}^k \equiv 0$ for all $k \geq 0$. Moreover, according to the mixing properties of Moshpit Averaging established in Theorem 3.2, inequality 10 holds after $\mathcal{O}\left(\log\left({}^1\!/\!{}_{\gamma^2\delta_{aq}^2}\right)\right)$ iterations of Algorithm 1. Therefore, the assumption above is natural and well-motivated.

Under these assumptions, we derive the convergence rates both for convex and non-convex problems. The full statements and complete proofs are deferred to Appendix D.

Theorem 3.3 (Convex case)**.** *Let $f_1 = \ldots = f_N = f$, function f be μ-strongly convex (Def. D.2) and L-smooth (see Def. D.1), and Assumptions 3.1 and 3.2 hold with $\Delta_{pv}^k = \delta_{pv,1}\gamma\mu\mathbb{E}[\|\theta^k-\theta^*\|^2] + \gamma^2\delta_{pv,2}^2$ and $\widetilde{\theta} = \theta^*$, where $\theta^* \in \operatorname{argmin}_{\theta\in\mathbb{R}^n} f(\theta)$ and $\delta_{pv,1} \in [0,1)$, $\delta_{pv,2} \geq 0$. Then there exists a choice of γ such that $\mathbb{E}\left[f(\overline{\theta}^K) - f(\theta^*)\right] \leq \varepsilon$ after K iterations of Moshpit SGD, where K equals*

$$\widetilde{\mathcal{O}}\left(\frac{L}{(1-\delta_{pv,1})\mu} + \frac{\delta_{pv,2}^2 + {}^{\sigma^2}\!/\!{}_{N_{\min}}}{(1-\delta_{pv,1})\mu\varepsilon} + \sqrt{\frac{L((\tau-1)\sigma^2+\delta_{aq}^2)}{(1-\delta_{pv,1})^2\mu^2\varepsilon}}\right),\ \mu > 0; \tag{11}$$

$$\mathcal{O}\left(\frac{LR_0^2}{\varepsilon} + \frac{R_0^2(\delta_{pv,2}^2 + {}^{\sigma^2}\!/\!{}_{N_{\min}})}{\varepsilon^2} + \frac{R_0^2\sqrt{L((\tau-1)\sigma^2+\delta_{aq}^2)}}{\varepsilon^{3/2}}\right),\ \mu = 0, \tag{12}$$

where $\overline{\theta}^K = \frac{1}{W_K}\sum\limits_{k=0}^{K}\frac{1}{N_k}\sum\limits_{i\in P_k} w_k\theta_i^k$, $w_k = (1-\gamma\mu)^{-(k+1)}$, $W_K = \sum_{k=0}^{K} w_k$, $R_0 = \|\theta^0-\theta^*\|$ and $\widetilde{\mathcal{O}}(\cdot)$ hides constant and $\log({}^1\!/\!{}_\varepsilon)$ factors.

That is, if $\delta_{pv,1} \leq {}^1\!/\!{}_2$, $N_{\min} = \Omega(N)$, $\delta_{pv,2}^2 = \mathcal{O}({}^{\sigma^2}\!/\!{}_{N_{\min}})$, and $\delta_{aq}^2 = \mathcal{O}((\tau-1)\sigma^2)$, then Moshpit SGD has the same iteration complexity as Local-SGD in the homogeneous case [61, 62]. However, the averaging steps of Moshpit SGD are much faster than those of the parameter-server architecture when the number of peers is large. Also, unlike the state-of-the-art convergence guarantees for Decentralized Local-SGD [63], our bounds do not depend on the spectral properties of the communication graph (see Appendix B.1 for the details).

Theorem 3.4 (Non-convex case)**.** *Let $f_1 = \ldots = f_N = f$, function f be L-smooth and bounded from below by f_*, and Assumptions 3.1 and 3.2 hold with $\Delta_{pv}^k = \delta_{pv,1}\gamma\mathbb{E}[\|\nabla f(\theta^k)\|^2] + L\gamma^2\delta_{pv,2}^2$, $\delta_{pv,1} \in [0, {}^1\!/\!{}_2)$, $\delta_{pv,2} \geq 0$. Then there exists such choice of γ that $\mathbb{E}\left[\|\nabla f(\theta_{rand}^K)\|^2\right] \leq \varepsilon^2$ after K iterations of Moshpit SGD, where K equals*

$$\mathcal{O}\left(\tfrac{L\Delta_0}{(1-2\delta_{pv,1})^2\varepsilon^2}\left[1+\tau\sqrt{1-2\delta_{pv,1}} + \tfrac{\delta_{pv,2}^2+{}^{\sigma^2}\!/\!{}_{N_{\min}}}{\varepsilon^2} + \tfrac{\sqrt{(1-2\delta_{pv,1})(\delta_{aq}^2+(\tau-1)\sigma^2)}}{\varepsilon}\right]\right),$$

$\Delta_0 = f(\theta^0) - f(\theta^*)$ and θ_{rand}^K is chosen uniformly from $\{\theta^0,\theta^1,\ldots,\theta^{K-1}\}$ defined in As. 3.2.

Again, if $\delta_{pv,1} \leq {}^1\!/\!{}_3$, $N_{\min} = \Omega(N)$, $\delta_{pv,2}^2 = \mathcal{O}({}^{\sigma^2}\!/\!{}_{N_{\min}})$, and $\delta_{aq}^2 = \mathcal{O}((\tau-1)\sigma^2)$, then the above theorem recovers the state-of-the-art results in the non-convex case for Local-SGD [64, 63].

3.3 Implementation details

Training on heterogeneous unreliable hardware also poses a number of engineering challenges. The most obvious one is that the system must be able to recover from node failures. To address this challenge, we use a fully decentralized infrastructure where all information is replicated in a Distributed Hash Table; see Appendix B.5 for details. When a new worker joins midway through training, it can download the latest model parameters and metadata from any other peer (see Appendix F). Another challenge arises when devices in a group have uneven network bandwidth. In that case, we dynamically adjust the communication load of each peer to avoid being bottlenecked. More information on this procedure can be found in Appendix G.

4 Experiments

In this section, we conduct empirical evaluation of the proposed averaging protocol and its corresponding optimization algorithm. First, we check the theoretical properties of Moshpit All-Reduce in a controlled setup (Section 4.1). Then, we compare Moshpit SGD with other distributed methods on practical tasks of image classification and masked language model pretraining (Sections 4.2 and 4.3).

4.1 Decentralized averaging

In this series of experiments, we aim to empirically verify the convergence and fault tolerance properties proven in Section 3.2. To measure this in a controlled setting, we create peers with parameters that are scalar values drawn from the standard Gaussian distribution. We study the convergence of different distributed methods with respect to the number of workers N and their individual failure rate for a single iteration of averaging p (failed peers return in the next round).

We compare Moshpit Averaging with the following algorithms from prior work: All-Reduce (with restarts in case of node failures), Gossip, PushSum (equivalent to the method described in [15]). Also, we provide the results of averaging in random groups as a simpler version of our approach. However, the implementation of group averaging maintains approximately the same group size across all iterations: this property might be hard to achieve in a decentralized setting, and as a result, the estimate of this method's performance should be considered highly optimistic.

We report the average squared difference between the worker parameters and the actual average of all values; the results are averaged across 100 restarts from different random initializations. We compare the convergence for 512–1024 peers and consider failure probabilities ranging from 0 to 0.01. For Moshpit Averaging and random group averaging, we use groups of size 32, which corresponds to $M = 32$ and $d = 2$ for Algorithm 1.

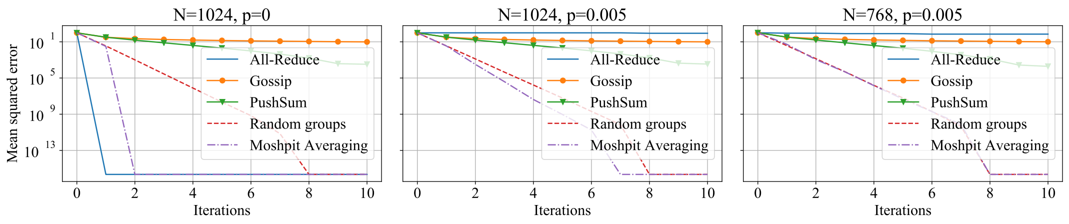

Figure 3: Convergence of averaging algorithms in different configurations.

Figure 3 displays the results of experiments for several combinations of N and p; the complete results with additional grid configurations are available in Appendix I. We make several key observations:

1. When the failure rate of each peer is zero, standard All-Reduce predictably computes the average faster than all other methods. However, as soon as p reaches a value of at least 0.005, the number of retries needed for the success becomes prohibitively high.

2. Previous decentralized averaging methods, such as Gossip or PushSum, require significantly more iterations for convergence to the global average than Moshpit All-Reduce, likely due to the structure of their communication graphs.

3. As discussed in Section 3.1, when the total number of peers is equal to the grid capacity and there are no failures, Moshpit All-Reduce matches the result of regular All-Reduce with the number of steps equal to the number of grid dimensions (2 in this case).

4. Averaging in random groups can perform comparably to Moshpit Averaging when the number of peers is less than half of the grid capacity. The reason for this behavior is that when the workers do not fully occupy the grid, the group sizes are no longer guaranteed to be equal across groups and across iterations. In the worst case, there can be groups of only one peer for certain grid coordinates, which may significantly affect the convergence. However, as the grid utilization grows, Moshpit Averaging starts to outperform random group averaging. Moreover, even if we use 512 peers, arranging them in a proper 8x8x8 grid leads to faster convergence.

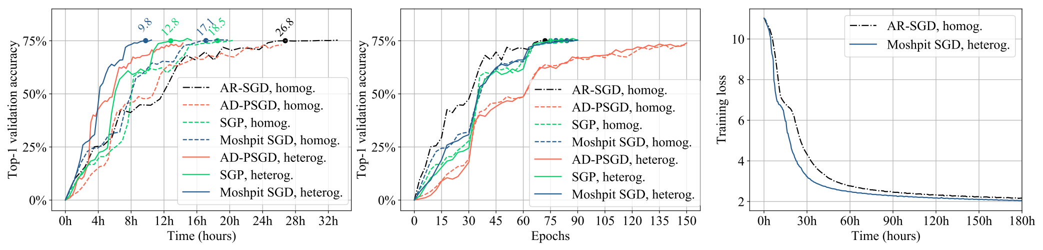

Figure 4: **(Left, Middle)** ResNet-50 top-1 validation accuracy for ImageNet as a function of training time (left) and epochs (middle). **(Right)** Full training objective (MLM + SOP) of ALBERT-large on BookCorpus as a function of training time.

4.2 ImageNet training

Here, we evaluate the performance of Moshpit SGD in distributed training. More specifically, we train ResNet-50 [65] on the ILSVRC [2] dataset, following the training protocol of [16]. Trainers use SGD with Nesterov momentum with a batch size of 256 and 32-bit precision regardless of the GPU type[4]. We evaluate the following training strategies:

- **All-Reduce SGD (AR-SGD)** — traditional distributed training with all-reduce gradient averaging;
- **Asynchronous Decentralized Parallel SGD (AD-PSGD)** — parallel SGD that runs gossip communication in a cycle: each worker averages parameters with 2 neighbors [66]. Communication rounds are overlapped with computation;
- **Stochastic Gradient Push (SGP)** — a more advanced algorithm with an exponential communication graph and push-based communication [15];
- **Moshpit SGD** — similar to **SGP**, but with 1 round of Moshpit Averaging instead of PushSum.

We report top-1 validation accuracy as a function of training time in two experimental setups:

- **Homogeneous**: 16 servers with a single Tesla V100-PCIe GPU, 6 CPU cores, and 64GB RAM.
- **Heterogeneous**: a total of 81 GPUs (V100, 1080Ti, and P40) across 64 servers and workstations.[5]

All servers and workstations communicate over the network with 1Gb/s Ethernet (non-dedicated symmetric bandwidth). The machines are located in two data centers and one office within 300 km of one another. The communication latency is 1–6ms depending on the location. To simulate shared usage, at the beginning of each communication round we inject additional latency sampled from the exponential distribution [67] with the mean of 100ms.

For Moshpit SGD, we use a two-dimensional "grid" with 4 and 8 groups for homogeneous and heterogeneous setups respectively. For AD-PSGD, we attempt to compensate for slow convergence by training for 60 more epochs without changing the learning rate schedule. Finally, we only report AR-SGD in the first setup, as it is unsuitable for heterogeneous hardware.

The results in Figure 4 (Left) demonstrate that the two most efficient strategies for our setting are Moshpit SGD and SGP. In the **homogeneous** setup, Moshpit is only slightly more efficient than SGP, likely due to higher efficiency of all-reduce. This advantage increases to over 30% for the **heterogeneous** setup with 64 servers. In turn, AR-SGD demonstrates the best performance per iteration, but its training time is by far the longest due to network latency ($1.5\times$ of Moshpit SGD). Finally, AD-PSGD predictably shows the best throughput (time per epoch), but achieves lower accuracy even after training for 150 epochs. We report results for smaller setups in Appendix J.

4.3 Masked Language Model training

Finally, we evaluate Moshpit All-Reduce training performance in the wild with preemptible cloud instances. For this experiment, we perform one of the most resource-demanding tasks in modern deep learning — unsupervised pretraining of Transformers [68, 69, 70, 5]. We opt for the ALBERT model [71] to make better use of communication-constrained devices. This model has fewer trainable parameters due to layer-wise weight sharing.

[4]For GPUs that cannot fit this into memory, we accumulate gradients over 2 batches of 128 examples.
[5]We provide a detailed configuration in Appendix H.

Specifically, we train ALBERT-large (18M parameters) on the BookCorpus [72] dataset, following the training setup from the original paper. We minimize the masked language modeling loss (MLM) along with the sentence order prediction loss (SOP) using the LAMB optimizer [17] with a global batch size of 4096 and sequence length 512. We measure convergence in terms of full training loss [73, 74]. Similarly to Section 4.2, we use two training setups:

- **Homogeneous:** a single cloud instance with 8 Tesla V100-PCIe GPUs and 56 vCPUs;
- **Heterogeneous:** a total of 66 preemptible GPUs, 32 of which are cloud T4, and the remaining 34 are various devices rented on a public marketplace.

Despite the fact that the latter setup has almost $3\times$ more raw compute[6], its hourly rent costs less than the homogeneous setup due to relying on preemptible instances[7]. This instance type is much cheaper than regular cloud instances, but it can be interrupted at any time. As a side-effect, the participants in **heterogeneous** setup are also spread across 3 continents with uneven network bandwidth, ranging from 100Mb/s to 1500Mb/s per worker. These limitations make it impractical to deploy conventional all-reduce protocols. By contrast, the fully decentralized nature of Moshpit SGD allows it to operate on unreliable nodes.

In this setup, the participants accumulate gradients over multiple local batches and use DHT to track the global batch size. Once the swarm collectively accumulates gradients over 4096 training samples, it runs 2 rounds of Moshpit All-Reduce with $M{=}8$ and $d{=}2$. Unfortunately, training with simple parameter averaging does not converge, likely due to diverging LAMB statistics. To mitigate this issue, workers recover "pseudo-gradients" [76, 77] after averaging to update the optimizer statistics.

Figure 4 (right) demonstrates that Moshpit SGD with a fully preemptible fleet of machines trains 1.5 times faster than the traditional data-parallel setup. The final loss achieved by two training strategies is the same within the margin of error. A closer investigation reveals that this speedup is entirely explained by the reduced iteration time. An interesting observation is that the iteration time of Moshpit SGD varies between 10–22 seconds, while AR-SGD consistently spends 25s per step. This can be explained by natural variation in the preemptible fleet size: there were 30–66 active participants depending on the resource availability.

5 Conclusion and future work

In this work, we propose Moshpit All-Reduce, a decentralized averaging protocol intended for distributed optimization in unstable and network-constrained environments. It has favorable theoretical properties when compared to gossip-based approaches and achieves considerable speedups in distributed training for image classification and masked language modeling.

Our approach was primarily designed for cloud-based training and federated learning, as well as for distributed training on unreliable instances; future work might explore additional settings, such as collaborative training of neural networks. Another potential research direction is to study the interactions of Moshpit All-Reduce with other methods that improve communication efficiency of distributed optimization, such as gradient compression. Finally, the idea of arranging All-Reduce nodes into groups can be improved to address specific issues that may arise in practice, such as the varying number of workers and their geographical distribution.

Acknowledgements

We would like to thank Anastasia Koloskova, Liudmila Prokhorenkova and Anton Osokin for helpful feedback and discussions. We are also grateful to the anonymous reviewers for their suggestions on improving the paper. Finally, we would like to thank Dmitry Afanasiev, Vladimir Aliev, Anand Jayarajan and Michael Solotky for their suggestions on the technical aspects of our study. This project was supported in part by the Canada Foundation for Innovation JELF grant, NSERC Discovery grant, AWS Machine Learning Research Award, and Facebook Faculty Research Award. The paper was also partially supported by by a grant for research centers in the field of artificial intelligence, provided by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730321P5Q0002) and the agreement with the Moscow Institute of Physics and Technology dated November 1, 2021 No. 70-2021-00138. The computational resources for the experiments were provided by the Amazon Research Awards program and Yandex.

[6]Based on official performance benchmarks [75].

[7]Please refer to Appendix H for full experimental setups.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, S. Gelly, and N. Houlsby. Big transfer (bit): General visual representation learning. In *ECCV*, 2020.

[4] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017.

[5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[6] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

[7] Chuan Li. Demystifying gpt-3 language model: A technical overview, 2020. "`https://lambdalabs.com/blog/demystifying-gpt-3`".

[8] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. MLPerf Training Benchmark. In *Proceedings of the 3rd Conference on Machine Learning and Systems (MLSys'20)*, 2020.

[9] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[10] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, February 2009.

[11] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

[12] V. Persico, P. Marchetta, A. Botta, and A. Pescape. On network throughput variability in microsoft azure cloud. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2015.

[13] Valerio Persico, Pietro Marchetta, Alessio Botta, and Antonio Pescapè. Measuring network throughput in the cloud: The case of amazon ec2. *Computer Networks*, 93:408 – 422, 2015. Cloud Networking and Communications II.

[14] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5330–5340, 2017.

[15] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. Stochastic gradient push for distributed deep learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 344–353. PMLR, 09–15 Jun 2019.

[16] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2017.

[17] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020.

[18] Mu Li. Scaling distributed machine learning with the parameter server. In *Proceedings of the 2014 International Conference on Big Data Science and Computing*, BigDataScience '14, New York, NY, USA, 2014. Association for Computing Machinery.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[20] Salem Alqahtani and Murat Demirbas. Performance analysis and comparison of distributed machine learning systems, 2019.

[21] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. A survey on distributed machine learning. *ACM Comput. Surv.*, 53(2), March 2020.

[22] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 2595–2603. Curran Associates, Inc., 2010.

[23] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018.

[24] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3478–3487. PMLR, 09–15 Jun 2019.

[25] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1223–1231. Curran Associates, Inc., 2012.

[26] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed DNN training in heterogeneous gpu/cpu clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479. USENIX Association, November 2020.

[27] Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka, and Yuichi Kageyama. Massively distributed sgd: Imagenet/resnet-50 training in a flash, 2019.

[28] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[29] Aaron Segal, Antonio Marcedone, Benjamin Kreuter, Daniel Ramage, H. Brendan McMahan, Karn Seth, K. A. Bonawitz, Sarvar Patel, and Vladimir Ivanov. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, 2017.

[30] K. A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé M Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *SysML 2019*, 2019. To appear.

[31] Micah J. Sheller, Brandon Edwards, G. Anthony Reina, Jason Martin, Sarthak Pati, Aikaterini Kotrotsou, Mikhail Milchenko, Weilin Xu, Daniel Marcus, Rivka R. Colen, and Spyridon Bakas. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific Reports*, 10(1):12598, Jul 2020.

[32] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M. Jorge Cardoso, and Andrew Feng. *Privacy-Preserving Federated Brain Tumour Segmentation*, pages 133–141. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). SPRINGER, January 2019. 10th International Workshop on Machine Learning in Medical Imaging, MLMI 2019 held in conjunction with the 22nd International Conference on Medical Image Computing and Computer-Assisted Intervention, MICCAI 2019 ; Conference date: 13-10-2019 Through 13-10-2019.

[33] Andrew Hard, Chloé M Kiddon, Daniel Ramage, Francoise Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, and Sean Augenstein. Federated learning for mobile keyboard prediction, 2018.

[34] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions, 2018.

[35] Ekasit Kijsipongse, Apivadee Piyatumrong, and Suriya U-ruekolan. A hybrid gpu cluster and volunteer computing platform for scalable deep learning. *The Journal of Supercomputing*, 04 2018.

[36] Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. In *Advances in Neural Information Processing Systems*, 2020.

[37] Aaron Harlap, Alexey Tumanov, Andrew Chung, Gregory R. Ganger, and Phillip B. Gibbons. Proteus: Agile ml elasticity through tiered reliability in dynamic resource markets. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, page 589–604, New York, NY, USA, 2017. Association for Computing Machinery.

[38] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6):2508–2530, 2006.

[39] John Nikolas Tsitsiklis. Problems in decentralized decision making and computation. Technical report, Massachusetts Inst of Tech Cambridge Lab for Information and Decision Systems, 1984.

[40] Kevin Scaman, Francis Bach, Sébastien Bubeck, Yin Tat Lee, and Laurent Massoulié. Optimal algorithms for smooth and strongly convex distributed optimization in networks. In *International Conference on Machine Learning*, pages 3027–3036, 2017.

[41] Kevin Scaman, Francis Bach, Sébastien Bubeck, Laurent Massoulié, and Yin Tat Lee. Optimal algorithms for non-smooth distributed optimization in networks. In *Advances in Neural Information Processing Systems*, pages 2740–2749, 2018.

[42] Kevin Scaman, Francis Bach, Sébastien Bubeck, Yin Lee, and Laurent Massoulié. Optimal convergence rates for convex distributed optimization in networks. *Journal of Machine Learning Research*, 20:1–31, 2019.

[43] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2019.

[44] Lin Xiao and Stephen Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

[45] Russell Merris. Laplacian matrices of graphs: a survey. *Linear algebra and its applications*, 197:143–176, 1994.

[46] César A Uribe, Soomin Lee, Alexander Gasnikov, and Angelia Nedić. A dual approach for optimal algorithms in distributed optimization over networks. *Optimization Methods and Software*, pages 1–40, 2020.

[47] Angelia Nedić and Alex Olshevsky. Distributed optimization over time-varying directed graphs. *IEEE Transactions on Automatic Control*, 60(3):601–615, 2014.

[48] Angelia Nedić and Alex Olshevsky. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. *IEEE Transactions on Automatic Control*, 61(12):3936–3947, 2016.

[49] Angelia Nedić, Alex Olshevsky, and Michael G Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, 106(5):953–976, 2018.

[50] Alexander Rogozin and Alexander Gasnikov. Projected gradient method for decentralized optimization over time-varying networks. *arXiv preprint arXiv:1911.08527*, 2019.

[51] S Sundhar Ram, A Nedić, and Venugopal V Veeravalli. Asynchronous gossip algorithms for stochastic optimization. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 3581–3586. IEEE, 2009.

[52] Feng Yan, Shreyas Sundaram, SVN Vishwanathan, and Yuan Qi. Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2483–2493, 2012.

[53] Kun Yuan, Qing Ling, and Wotao Yin. On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 26(3):1835–1854, 2016.

[54] Paul Sack and William Gropp. Collective algorithms for multiported torus networks. *ACM Trans. Parallel Comput.*, 1(2), February 2015.

[55] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[56] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

[57] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[58] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. Sgd: General analysis and improved rates. In *International Conference on Machine Learning*, pages 5200–5209. PMLR, 2019.

[59] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[60] Eduard Gorbunov, Filip Hanzely, and Peter Richtarik. Local sgd: Unified theory and new efficient methods. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3556–3564. PMLR, 13–15 Apr 2021.

[61] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. Tighter theory for local sgd on identical and heterogeneous data. In *International Conference on Artificial Intelligence and Statistics*, pages 4519–4529. PMLR, 2020.

[62] Blake Woodworth, Kumar Kshitij Patel, Sebastian Stich, Zhen Dai, Brian Bullins, Brendan Mcmahan, Ohad Shamir, and Nathan Srebro. Is local sgd better than minibatch sgd? In *International Conference on Machine Learning*, pages 10334–10343. PMLR, 2020.

[63] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

[64] Xiang Li, Wenhao Yang, Shusen Wang, and Zhihua Zhang. Communication efficient decentralized training with multiple local updates. *arXiv preprint arXiv:1910.09126*, 5, 2019.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[66] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3043–3052. PMLR, 10–15 Jul 2018.

[67] Andrei M Sukhov, MA Astrakhantseva, AK Pervitsky, SS Boldyrev, and AA Bukatov. Generating a function for network delay. *Journal of High Speed Networks*, 22(4):321–333, 2016.

[68] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[69] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019.

[70] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[71] Zhen-Zhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

[72] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

[73] Jiahuang Lin, Xin Li, and Gennady Pekhimenko. Multi-node bert-pretraining: Cost-efficient approach, 2020.

[74] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

[75] NVIDIA. Nvidia data center deep learning product performance. `"https://developer.nvidia.com/deep-learning-performance-training-inference"`, accessed at 2021.02.03.

[76] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[77] Xiangyi Chen, Xiaoyun Li, and Ping Li. Toward communication efficient adaptive gradient method. In *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*, FODS '20, page 119–128, New York, NY, USA, 2020. Association for Computing Machinery.

[78] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, 2016.

[79] David Aldous and James Allen Fill. Reversible markov chains and random walks on graphs, 2002. unfinished monograph, recompiled 2014, 2002.

[80] Jinming Xu, Ye Tian, Ying Sun, and Gesualdo Scutari. Distributed algorithms for composite optimization: Unified and tight convergence analysis. *arXiv preprint arXiv:2002.11534*, 2020.

[81] Alireza Fallah, Mert Gurbuzbalaban, Asu Ozdaglar, Umut Simsekli, and Lingjiong Zhu. Robust distributed accelerated stochastic gradient methods for multi-agent networks. *arXiv preprint arXiv:1910.08701*, 2019.

[82] Dmitry Kovalev, Adil Salim, and Peter Richtárik. Optimal and practical algorithms for smooth and strongly convex decentralized optimization. *Advances in Neural Information Processing Systems*, 33, 2020.

[83] Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. *Advances in neural information processing systems*, 28:1756–1764, 2015.

[84] 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*, 2014.

[85] Dan Alistarh, Demjan Grubic, Jerry Z Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: communication-efficient sgd via gradient quantization and encoding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1707–1718, 2017.

[86] Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. Distributed mean estimation with limited communication. In *International Conference on Machine Learning*, pages 3329–3337. PMLR, 2017.

[87] Ali Ramezani-Kebrya, Fartash Faghri, Ilya Markov, Vitalii Aksenov, Dan Alistarh, and Daniel M Roy. Nuqsgd: Provably communication-efficient data-parallel sgd via nonuniform quantization. *Journal of Machine Learning Research*, 22(114):1–43, 2021.

[88] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel M Roy, and Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd. *Advances in Neural Information Processing Systems*, 33:3174–3185, 2020.

[89] Samuel Horvath, Chen-Yu Ho, Ludovit Horvath, Atal Narayan Sahu, Marco Canini, and Peter Richtarik. Natural compression for distributed deep learning. *arXiv preprint arXiv:1905.10988*, 2019.

[90] Aleksandr Beznosikov, Samuel Horváth, Peter Richtárik, and Mher Safaryan. On biased compression for distributed learning. *arXiv preprint arXiv:2002.12410*, 2020.

[91] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: ternary gradients to reduce communication in distributed deep learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1508–1518, 2017.

[92] Konstantin Mishchenko, Eduard Gorbunov, Martin Takáč, and Peter Richtárik. Distributed learning with compressed gradient differences. *arXiv preprint arXiv:1901.09269*, 2019.

[93] Samuel Horváth, Dmitry Kovalev, Konstantin Mishchenko, Sebastian Stich, and Peter Richtárik. Stochastic distributed learning with gradient quantization and variance reduction. *arXiv preprint arXiv:1904.05115*, 2019.

[94] Zhize Li, Dmitry Kovalev, Xun Qian, and Peter Richtarik. Acceleration for compressed gradient descent in distributed and federated optimization. In *International Conference on Machine Learning*, pages 5895–5904. PMLR, 2020.

[95] Eduard Gorbunov, Dmitry Kovalev, Dmitry Makarenko, and Peter Richtarik. Linearly converging error compensated sgd. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20889–20900. Curran Associates, Inc., 2020.

[96] Constantin Philippenko and Aymeric Dieuleveut. Artemis: tight convergence guarantees for bidirectional compression in federated learning. *arXiv preprint arXiv:2006.14591*, 2020.

[97] Zhize Li and Peter Richtárik. A unified analysis of stochastic gradient methods for nonconvex federated optimization. *arXiv preprint arXiv:2006.07013*, 2020.

[98] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. *arXiv preprint arXiv:2007.01154*, 2020.

[99] Rudrajit Das, Abolfazl Hashemi, Sujay Sanghavi, and Inderjit S Dhillon. Improved convergence rates for non-convex federated learning with compression. *arXiv preprint arXiv:2012.04061*, 2020.

[100] Eduard Gorbunov, Konstantin P. Burlachenko, Zhize Li, and Peter Richtarik. Marina: Faster non-convex distributed learning with compression. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3788–3798. PMLR, 18–24 Jul 2021.

[101] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4452–4463, 2018.

[102] 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.

[103] Xun Qian, Peter Richtárik, and Tong Zhang. Error compensated distributed sgd can be accelerated. *arXiv preprint arXiv:2010.00091*, 2020.

[104] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, and Ramtin Pedarsani. An exact quantized decentralized gradient descent algorithm. *IEEE Transactions on Signal Processing*, 67(19):4934–4947, 2019.

[105] Dmitry Kovalev, Anastasia Koloskova, Martin Jaggi, Peter Richtarik, and Sebastian Stich. A linearly convergent algorithm for decentralized optimization: Sending less bits for free! In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 4087–4095. PMLR, 13–15 Apr 2021.

[106] Anastasia Koloskova, Tao Lin, Sebastian U Stich, and Martin Jaggi. Decentralized deep learning with arbitrary communication compression. In *International Conference on Learning Representations*, 2020.

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

[108] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[109] Sebastian Urban Stich. Local SGD converges fast and communicates little. *International Conference on Learning Representations (ICLR)*, page arXiv:1805.09767, 2019.

[110] Tao Lin, Sebastian Urban Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local SGD. *ICLR*, page arXiv:1808.07217, 2020.

[111] Blake Woodworth, Kumar Kshitij Patel, and Nathan Srebro. Minibatch vs local sgd for heterogeneous distributed learning. *arXiv preprint arXiv:2006.04735*, 2020.

[112] Honglin Yuan and Tengyu Ma. Federated accelerated stochastic gradient descent. *Advances in Neural Information Processing Systems*, 33, 2020.

[113] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations. In *Advances in Neural Information Processing Systems*, pages 14668–14679, 2019.

[114] Honglin Yuan, Manzil Zaheer, and Sashank Reddi. Federated composite optimization. *arXiv preprint arXiv:2011.08474*, 2020.

[115] Mahmoud Assran, Arda Aytekin, Hamid Reza Feyzmahdavian, Mikael Johansson, and Michael G Rabbat. Advances in asynchronous parallel and distributed optimization. *Proceedings of the IEEE*, 108(11):2013–2031, 2020.

[116] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[117] Shen-Yi Zhao and Wu-Jun Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[118] Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Asaga: asynchronous parallel saga. In *Artificial Intelligence and Statistics*, pages 46–54. PMLR, 2017.

[119] Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin. Arock: an algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 38(5):A2851–A2879, 2016.

[120] Konstantin Mishchenko, Franck Iutzeler, Jérôme Malick, and Massih-Reza Amini. A delay-tolerant proximal-gradient algorithm for distributed learning. In *International Conference on Machine Learning*, pages 3587–3595. PMLR, 2018.

[121] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 873–881, 2011.

[122] Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 61(12):3740–3754, 2016.

[123] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In *Algorithmic Learning Theory*, pages 111–132. PMLR, 2020.

[124] Hari Balakrishnan, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Communications of the ACM*, 46(2):43–48, 2003.

[125] Seymour Kaplan. Application of programs with maximin objective functions to problems of optimal resource allocation. *Operations Research*, 22(4):802–807, 1974.

[126] Erling D. Andersen and Knud D. Andersen. The mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm. In *Applied Optimization*, pages 197–232. Springer US, 2000.

[127] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed dnn training. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 132–145, 2019.