Stochastic Gradient Push for Distributed Deep Learning

Mahmoud Assran[1,3]**, Nicolas Loizou**[2,3]**, Nicolas Ballas**[3]**, and Michael Rabbat**[3]
[1] McGill University
[2] The University of Edinburgh
[3] Facebook AI Research

Abstract

Large mini-batch parallel SGD is commonly used for distributed training of deep networks. Approaches that use tightly-coupled exact distributed averaging based on AllReduce are sensitive to slow nodes and high-latency communication. In this work we show the applicability of Stochastic Gradient Push (SGP) for distributed training. SGP uses a gossip algorithm called PushSum for approximate distributed averaging, allowing for much more loosely coupled communications, which can be beneficial in high-latency or high-variability systems. The tradeoff is that approximate distributed averaging injects additional noise in the gradient which can affect the train and test accuracies. We prove that SGP converges to a stationary point of smooth, non-convex objective functions. Furthermore, we validate empirically the potential of SGP. For example, using 32 nodes with 8 GPUs per node to train ResNet-50 on ImageNet, where nodes communicate over 10Gbps Ethernet, SGP completes 90 epochs in around 1.6 hours while AllReduce SGD takes over 5 hours, and the top-1 validation accuracy of SGP remains within 1.2% of that obtained using AllReduce SGD.

1 Introduction

Deep Neural Networks (DNNs) are the state-of-the art machine learning approach in many application areas, including image recognition [4] and natural language processing [15]. Stochastic Gradient Descent (SGD) is the current workhorse for training neural networks. The algorithm optimizes the network parameters, $\boldsymbol{x}$, to minimize a loss function, $f(\cdot)$, through gradient descent, where the loss function's gradients are approximated using a subset of training examples (a mini-batch). DNNs often require large amounts of training data and trainable parameters, necessitating non-trivial computational requirements [16, 9]. There is a need for efficient methods to train DNNs in large-scale computing environments.

A data-parallel version of SGD is often adopted for large-scale, distributed training [3, 7]. Worker nodes compute local mini-batch gradients of the loss function on different subsets of the data and then calculate an exact inter-node average gradient using either the ALLREDUCE communication primitive, in synchronous implementations [3], or using a central parameter server, in asynchronous implementations [2]. Using a parameter server to aggregate gradients introduces a potential bottleneck and a central point of failure [8]. The ALLREDUCE primitive computes the exact average gradient at all workers in a decentralized manner, avoiding issues associated with centralized communication and computation.

However, exact averaging algorithms like ALLREDUCE are not robust in high-latency or high-variability platforms, *e.g.*, where the network bandwidth may be a significant bottleneck, because they involve tightly-coupled, blocking communication (*i.e.*, the call does not return until all nodes have finished aggregating). Moreover, aggregating gradients across all the nodes in the network can

Preprint. Work in progress.

introduce non-trivial computational overhead when there are many nodes, or when the gradients themselves are large. This issue motivates the investigation of a decentralized and inexact version of SGD to reduce the overhead associated with distributed training.

Numerous decentralized optimization algorithms have been studied in the control-systems literature that leverage consensus-based approaches for the computation of aggregate information; see the survey [11] and references therein. Rather than exactly aggregating gradients (as with ALLREDUCE), this line of work uses less-coupled message passing algorithms that compute inexact distributed averages.

Most previous work in this area has focused on theoretical convergence analysis assuming convex objectives. Recent work has begun to investigate their applicability to large-scale training of DNNs [8, 5]. However, these papers study methods based on communication patterns which are static (the same at every iteration) and symmetric (if i sends to j, then i must also receive from j before proceeding). Such methods inherently require blocking communication overhead. State-of-the-art consensus optimization methods build on the PUSHSUM algorithm for approximate distributed averaging [6, 11], which allows for non-blocking, time-varying, and directed (asymmetric) communication. Since SGD already uses stochastic mini-batches, the hope is that an inexact average mini-batch will be as useful as the exact one if the averaging error is sufficiently small relative to the variability in the gradient.

This paper studies the use of Stochastic Gradient Push (SGP), an algorithm blending SGD and PUSHSUM, for distributed training of deep neural networks. We provide a theoretical analysis of SGP, showing it converges for smooth non-convex objectives. We also evaluate SGP experimentally, training ResNets on ImageNet using up to 32 nodes, each with 8 GPUs (*i.e.*, 256 GPUs in total).

Our main contributions are summarized as follows. We provide the first convergence analysis for Stochastic Gradient Push when the objective function is smooth and non-convex. We show that, for an appropriate choice of the step size, SGP converges to a stationary point at a rate of $\mathcal{O}\left(1/\sqrt{nK}\right)$, where n is the number of nodes and K is the number of iterations. In a high-latency scenario, where nodes communicate over 10Gbps Ethernet, SGP runs up to $3\times$ faster than ALLREDUCE SGD and exhibits 88.6% scaling efficiency over the range from 4–32 nodes. The top-1 validation accuracy of SGP matches that of ALLREDUCE SGD for up to 8 nodes (64 GPUs), and remains within 1.2% of ALLREDUCE SGD for larger networks. In comparison to other decentralized consensus-based approaches that require symmetric messaging, SGP runs faster and it produces models with better validation accuracy.

2 Preliminaries

Problem formulation. We consider the setting where a network of n nodes cooperates to solve the stochastic consensus optimization problem

$$\begin{array}{ll} \min_{\boldsymbol{x}_i \in \mathbb{R}^d, i=1,\dots,n} & \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\xi_i \sim D_i} F_i(\boldsymbol{x}_i;\xi_i) \\ \text{subject to} & \boldsymbol{x}_i = \boldsymbol{x}_j, \forall i,j = 1,\dots,n. \end{array} \tag{1}$$

Each node has local data following a distribution D_i, and the nodes wish to cooperate to find the parameters $\boldsymbol{x}$ of a DNN that minimizes the average loss with respect to their data, where F_i is the loss function at node i. Moreover, the goal codified in the constraints is for the nodes to reach agreement (*i.e.*, consensus) on the solution they report. We assume that nodes can locally evaluate stochastic gradients $\nabla F(\boldsymbol{x}_i;\xi_i)$, $\xi_i \sim D_i$, but they must communicate to access information about the objective functions at other nodes.

Distributed averaging. The problem described above encompasses distributed training based on data parallelism. There a canonical approach is large mini-batch parallel stochastic gradient descent: for an overall mini-batch of size nb, each node computes a local stochastic mini-batch gradient using b samples, and then the nodes use the ALLREDUCE communication primitive to compute the average gradient at every node. Let $f_i(\boldsymbol{x}_i) = \mathbb{E}_{\xi_i \sim D_i} F_i(\boldsymbol{x}_i;\xi_i)$ denote the objective at node i, and let $f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x})$ denote the overall objective. Since $\nabla f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{x})$, averaging gradients via ALLREDUCE provides an exact stochastic gradient of f. Typical implementations of ALLREDUCE have each node send and receive $2\frac{n-1}{n}\mathtt{B}$ bytes, where $\mathtt{B}$ is the size (in bytes) of the tensor being reduced, and involve $2\log_2(n)$ communication steps [13]. Moreover, ALLREDUCE is a

Algorithm 1 Stochastic Gradient Push (SGP)

Require: Initialize $\gamma > 0$, $\boldsymbol{x}_i^{(0)} = \boldsymbol{z}_i^{(0)} \in \mathbb{R}^d$ and $w_i^{(0)} = 1$ for all nodes $i \in \{1, 2, \ldots, n\}$

1: **for** $k = 0, 1, 2, \cdots, K$ **do** at node i

2: Sample new mini-batch $\xi_i^{(k)} \sim \mathcal{D}_i$ from local distribution

3: Compute a local stochastic mini-batch gradient at $\boldsymbol{z}_i^{(k)}$: $\nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

4: $\boldsymbol{x}_i^{(k+\frac{1}{2})} = \boldsymbol{x}_i^{(k)} - \gamma \nabla \boldsymbol{F}_i(\boldsymbol{z}_i^{(k)}; \xi_i^{(k)})$

5: Send $\left(p_{j,i}^{(k)} \boldsymbol{x}_i^{(k+\frac{1}{2})}, p_{j,i}^{(k)} w_i^{(k)}\right)$ to out-neighbors $j \in \mathcal{N}_i^{\text{out}(k)}$;
receive $\left(p_{i,j}^{(k)} \boldsymbol{x}_j^{(k+\frac{1}{2})}, p_{i,j}^{(k)} w_j^{(k)}\right)$ from in-neighbors $j \in \mathcal{N}_i^{\text{in}(k)}$

6: $\boldsymbol{x}_i^{(k+1)} = \sum_{j \in \mathcal{N}_i^{\text{in}(k)}} p_{i,j}^{(k)} \boldsymbol{x}_j^{(k+\frac{1}{2})}$

7: $w_i^{(k+1)} = \sum_{j \in \mathcal{N}_i^{\text{in}(k)}} p_{i,j}^{(k)} w_j^{(k)}$

8: $\boldsymbol{z}_i^{(k+1)} = \boldsymbol{x}_i^{(k+1)} / w_i^{(k+1)}$

9: **end for**

blocking primitive, meaning that no node will proceed with local computations until the primitive returns.

Approximate distributed averaging. In this work we explore the alternative approach of using a gossip algorithm for approximate distributed averaging—specifically, the PUSHSUM algorithm. Gossip algorithms typically use linear iterations for averaging. For example, let $\boldsymbol{y}_i^{(0)} \in \mathbb{R}^n$ be a vector at node i, and consider the goal of computing the average vector $\frac{1}{n}\sum_{i=1}^n \boldsymbol{y}_i^{(0)}$ at all nodes. Stack the initial vectors into a matrix $\boldsymbol{Y}^{(0)} \in \mathbb{R}^{n \times d}$ with one row per node. Typical gossip iterations have the form $\boldsymbol{Y}^{(k+1)} = \boldsymbol{P}^{(k)} \boldsymbol{Y}^{(k)}$ where $\boldsymbol{P}^{(k)} \in \mathbb{R}^{n \times n}$ is referred to as the mixing matrix. This corresponds to the update $\boldsymbol{y}_i^{(k+1)} = \sum_{j=1}^n p_{i,j}^{(k)} \boldsymbol{y}_j^{(k)}$ at node i. To implement this update, node i only needs to receive messages from other nodes j for which $p_{i,j}^{(k)} \neq 0$, so it will be appealing to use sparse $\boldsymbol{P}^{(k)}$ to reduce communications.

The PUSHSUM algorithm only requires that $\boldsymbol{P}^{(k)}$ be column-stochastic, and not necessarily symmetric (so node i may send to node j, but not necessarily vice versa). Instead, one additional scalar parameter $w_i^{(k)}$ is maintained at each node. The parameter is initialized to $w_i^{(0)} = 1$ for all i, and updated using the same linear iteration, $\boldsymbol{w}^{(k+1)} = \boldsymbol{P}^{(k)} \boldsymbol{w}^{(k)}$. Consequently, the parameter converges to $\boldsymbol{w}^{(\infty)} = \boldsymbol{\pi}(\boldsymbol{1}^\top \boldsymbol{w}^{(0)})$, or $w_i^{(\infty)} = \pi_i n$ at node i, where $\boldsymbol{\pi}$ is the ergodic limit of the product of the $\boldsymbol{P}^{(k)}$'s. Thus each node can recover the average of the initial vectors by computing the *de-biased* ratio $\boldsymbol{y}_i^{(\infty)} / w_i^{(\infty)}$. In practice, we stop after a finite number of gossip iterations K and compute $\boldsymbol{y}_i^{(K)} / w_i^{(K)}$. The distance of the de-biased ratio to the exact average can be quantified in terms of properties of the matrices $\{\boldsymbol{P}^{(k)}\}_{k=0}^{K-1}$. Let $\mathcal{N}_i^{\text{out}(k)} = \{j \colon p_{j,i}^{(k)} > 0\}$ and $\mathcal{N}_i^{\text{in}(k)} = \{j \colon p_{i,j}^{(k)} > 0\}$ denote the sets of nodes that i transmits to and receives from, respectively, at iteration k. If we use $\mathtt{B}$ bytes to represent the vector $\boldsymbol{y}_i^{(k)}$, then node i sends and receives $\left|\mathcal{N}_i^{\text{out}(k)}\right| \mathtt{B}$ and $\left|\mathcal{N}_i^{\text{in}(k)}\right| \mathtt{B}$ bytes, respectively, per iteration. In our experiments we use graph sequences with $\left|\mathcal{N}_i^{\text{out}(k)}\right| = \left|\mathcal{N}_i^{\text{in}(k)}\right| = 1$ or 2, and find that approximate averaging is both fast and still facilitates training.

3 Stochastic Gradient Push

Algorithm description. The *stochastic gradient push* (SGP) method for solving (1) is obtained by interleaving one local stochastic gradient descent update at each node with one iteration of PUSHSUM. Each node maintains three variables: the model parameters $\boldsymbol{x}_i^{(k)}$ at node i, the scalar PUSHSUM weight $w_i^{(k)}$, and the de-biased parameters $\boldsymbol{z}_i^{(k)} = \left(w_i^{(k)}\right)^{-1} \boldsymbol{x}_i^{(k)}$. The initial $\boldsymbol{x}_i^{(0)}$ and $\boldsymbol{z}_i^{(0)}$ can be initialized to any arbitrary value as long as $\boldsymbol{x}_i^{(0)} = \boldsymbol{z}_i^{(0)}$. Pseudocode is shown in Alg. 1. Each node

performs a local SGD step (lines 2–4) followed by one step of PUSHSUM for approximate distributed averaging (lines 5–8).

Note that the gradients are evaluated at the de-biased parameters $\boldsymbol{z}_i^{(k)}$ in line 3, and they are then used to update $\boldsymbol{x}_i^{(k)}$, the PUSHSUM numerator, in line 4. All communication takes place in line 5, and each message contains two parts, the PUSHSUM numerator and denominator. In particular, node i controls the values $p_{j,i}^{(k)}$ used to weight the values in messages it sends.

We are mainly interested in the case where the mixing matrices $\boldsymbol{P}^{(k)}$ are sparse in order to have low communication overhead. However, we point out that when the nodes' initial values are identical, $\boldsymbol{x}_i^{(0)} = \boldsymbol{x}_j^{(0)}$ for all $i, j \in [n]$, and every entry of $\boldsymbol{P}^{(k)}$ is equal to $1/n$, then SGP is mathematically equivalent to parallel SGD using ALLREDUCE.

Theoretical guarantees. SGP was first proposed and analyzed in [10] assuming the local objectives $f_i(\boldsymbol{x})$ are strongly convex. Here we provide convergence results in the more general setting of smooth, non-convex objectives. We make the following three assumptions:

1. (L-smooth) There exists a constant $L > 0$ such that $\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|$, or equivalently $f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y}) + \nabla f_i(\boldsymbol{y})^\top(\boldsymbol{x} - \boldsymbol{y}) + \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2$. Note that this assumption implies that function $f(x)$ is also L-smooth.

2. (Bounded variance) There exist finite positive constants σ^2 and ζ^2 such that $\mathbb{E}_{\xi \sim D_i}\|\nabla F_i(\boldsymbol{x}; \xi) - \nabla f_i(\boldsymbol{x})\|^2 \leq \sigma^2 \quad \forall i, \forall \boldsymbol{x}$, and $\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(\boldsymbol{x}) - \nabla f(\boldsymbol{x})\|^2 \leq \zeta^2 \quad \forall \boldsymbol{x}$.

3. (Mixing connectivity) To each mixing matrix $\boldsymbol{P}^{(k)}$ we can associate a graph with vertex set $\{1, \ldots, n\}$ and edge set $E^{(k)} = \{(i, j) \colon p_{i,j}^{(k)} > 0\}$; *i.e.*, with edges (i, j) from j to i if i receives a message from j at iteration k. Assume that the graph with edge set $\bigcup_{k=lB}^{(l+1)B-1} E^{(k)}$ is strongly connected and has diameter at most Δ for every $l \geq 0$. To simplify the discussion, we assume that every column of the mixing matrices $\boldsymbol{P}^{(k)}$ has at most D non-zero entries.

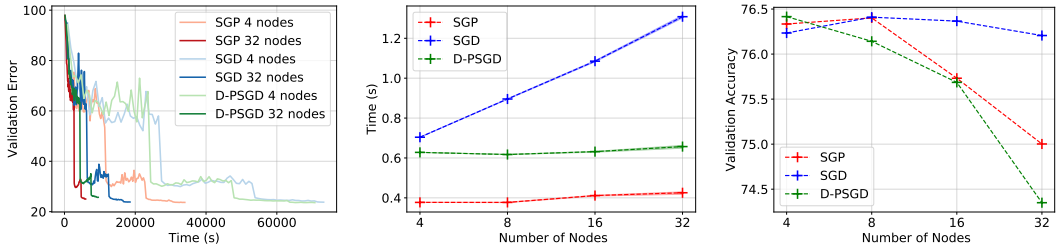
Theorem 1. *Suppose that Assumptions 1–3 hold, and run SGP for K iterations with step-size $\gamma = \sqrt{n/K}$. Let $f^* = \min_{\boldsymbol{x}} f(\boldsymbol{x})$ and assume that $f^* > -\infty$. There exist constants $C > 0$ and $q \in (0, 1)$ which depend on B, n, and Δ such that if the total number of iterations K is sufficiently large. Then*

$$\frac{1}{nK}\sum_{k=0}^{K-1}\sum_{i=1}^{n}\mathbb{E}\left\|\nabla f(\boldsymbol{z}_i^k)\right\|^2 \leq \mathcal{O}\left(\frac{1}{\sqrt{nK}} + \frac{1}{K} + \frac{1}{K^{3/2}}\right)$$

A more detailed statement of the theorem, along with proofs, are provided in the extended version of this paper [1]. There we give a precise expression for how large is sufficiently large, regarding the number of iterations K, and we also give a detailed expression, showing the constants which are buried in the $\mathcal{O}$ above. Generally speaking, the number of iterations needs to be large enough to ensure that the SGP has time to diffuse information across the network (roughly speaking, K should be at least proportional to n). This is reasonable for training DNNs, where the number of iterations (not just epochs) is typically much larger than the number of nodes used for training. In short, our theoretical result shows that the magnitude of the gradient at every worker's version of model becomes arbitrarily small as K increases. The $\mathcal{O}(1/\sqrt{nK})$ term dominates asymptotically, and thus, to drive the error to $\mathcal{O}(\epsilon)$ we need $K = \Omega(\frac{1}{n\epsilon^2})$ iterations; in otherwise, theoretically we obtain a linear speedup. In the extended version we also show that the quantity $\left\|\boldsymbol{x}_i^k - \overline{\boldsymbol{x}}^k\right\|^2$ obeys a similar bound, where $\overline{\boldsymbol{x}}^k = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i^k$. Thus, as K increases, the values at each node also remain close.

4 Experiments

Next, we compare SGP with ALLREDUCE SGD, and D-PSGD [8], an approximate distributed averaging baseline relying on doubly-stochastic gossip. We run experiments on a large-scale distributed computing environment using up to 256 GPUs. Our results show that when communication is the bottleneck, SGP is faster than both SGD and D-PSGD. SGP also outperforms D-PSGD in terms of



(a) Validation Curve (b) Time Per Iteration (c) Validation Accuracy

Figure 1: Results on Ethernet 10Gbits. (a): Validation performance w.r.t. training time (in seconds) for model trained on 4 and 32 nodes. (b): Average time per training iteration (in seconds) (c): Best validation accuracy. Stochastic Gradient Push (SGP) is faster than both Decentralized-Parallel SGD (D-PSGD) and ALLREDUCE SGD while decreasing validation accuracy by 1.2%.

validation accuracy, while achieving a slightly worse accuracy compared to SGD when using a large number of compute nodes.

We run experiments on 32 DGX-1 GPU servers in a high-performance computing cluster. Each server contains eight NVIDIA Volta V100 GPUs. Nodes communicate over a 10 Gbit/s Ethernet network. To investigate how each algorithm scales, we run experiments with 4, 8, 16, and 32 nodes (*i.e.*, 32, 64, 128, and 256 GPUs).

We adopt the 1000-way ImageNet classification task [14] as our experimental benchmark. We train a ResNet-50 [4] following the experimental protocol of [3], using the same hyperparameters with the exception of the learning rate schedule in the 32 node experiment for SGP and D-PSGD. In the experiments, we also modify SGP to use Nesterov momentum. In our default implementation of SGP, each node sends and receives to one other node at each iteration, and this destination changes from one iteration to the next. Please refer to [1] for more implementation details.

All algorithms are implemented in PyTorch v0.5 [12]. To leverage the highly efficient NVLink interconnect within each server, we treat each DGX-1 as one node in all of our experiments. In our implementation of SGP, each node computes a local mini-batch in parallel using all eight GPUs using a local ALLREDUCE, which is efficiently implemented via the NVIDIA Collective Communications Library. Then inter-node averaging is accomplished using PUSHSUM either over Ethernet.

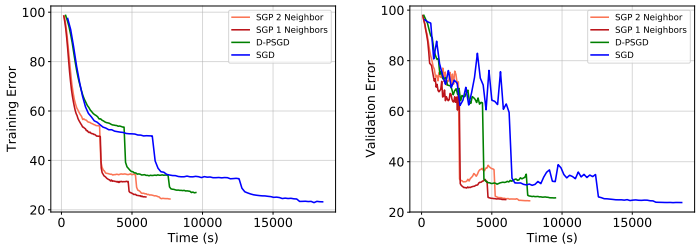
4.1 Evaluation on High-Latency Interconnect

We consider the high-latency scenario where nodes communicate over 10Gbit/s Ethernet. With a local mini-batch size of 256 samples per node (32 samples per GPU), a single Volta DGX-1 server can perform roughly 4.384 mini-batches per second. Since the ResNet-50 model size is roughly 100MBytes, transmitting one copy of the model per iteration requires 3.5 Gbit/s. Thus in the high-latency scenario the problem, if a single 10 Gbit/s link must carry the traffic between more than two pairs of nodes, then communication clearly becomes a bottleneck.

Figure 1 (a) shows the validation curves when training on 4 and 32 nodes.For any number of nodes used in our experiments, we observe that SGP consistently outperforms D-PSGD and ALLREDUCE SGD in terms of total training time in this scenario. In particular for 32 nodes, SGP training time takes less than 1.6 hours while D-PSGD and ALLREDUCE SGD require roughly 2.6 and 5.1 hours.

Figure 1 (b) shows the average time per iteration for the different training runs. As we increase the number of nodes, the average iteration time stays almost constant for SGP and D-PSGD, while we observe a significant time-increase in the case of ALLREDUCE SGD, resulting in an overall slower training time. Moreover, although D-SGD and SGP both exhibit strong scaling, SGP is roughly 200ms faster per iteration, supporting the claim that it involves less communication overhead.

Figure 1 (c) reports the best validation accuracy for the different training runs. While they all start around the same value, the accuracy of D-PSGD and SGP decreases as we increase the number of nodes. In the case of SGP, we see its performance decrease by 1.2% relative to SGD on 32 nodes. We



(a) Train (b) Valid

Figure 2: Comparison of SGP using a communication graph with 1-neighbor, SGP using a graph with 2-neighbors, D-PSGD and SGD on 32 nodes communicating over 10 Gbit/s Ethernet. Using one additional neighbor improves the validation performance of SGD (from 75.0 to 75.4) while retaining most of the computational benefits.

hypothesize that this decrease is due to the noise introduced by approximate distributed averaging. We will see below than changing the connectivity between the nodes can ameliorate this issue. We also note that the SGP validation accuracy is better than D-PSGD for larger networks.

In the extended version [1], we report additional experiments on a low-latency Infiniband interconnect. There, the per-iteration time of SGP scales essentially as well as AllReduce, but the accuracy of SGP is still degraded for larger numbers of nodes, so AllReduce SGD remains the preferred choice for parallel optimization in low-latency settings.

4.2 Impact of Graph Topology

Next we investigate the impact of the communication graph topology on the SGP validation performance using Ethernet 10Gbit/s. In the limit of a fully-connected communication graph, SGD and SGP are strictly equivalent (see section 3). By increasing the number of neighbors in the graph, we expect the accuracy of SGP to improve (approximate averages are more accurate) but the communication time required for training will increase.

In figure 2, we compare the training and validation accuracy for SGP using a communication graph with 1-neighbor and 2-neighbors with D-PSGD and SGD on 32 nodes. By increasing the number of neighbors to two, SGP achieves better training/validation accuracy (from 74.8/75.0 to 75.6/75.4) and gets closer to final validation achieves by SGD (77.0/76.2). Increasing the number of neighbors also to increases the communication, hence the overall training time. SGP with 2 neighbors completes training in 2.1 hours and its average time per iteration increases by 27% relative to SGP with one neighbor. Nevertheless, SGP 2-neighbors is still faster than SGD and D-PSGD, while achieving better accuracy than SGP 1-neighbor.

5 Conclusion

DNN training often necessitates non-trivial computational requirements leveraging distributed computing resources. Traditional parallel versions of SGD use exact averaging algorithms to parallelize the computation between nodes, and induce additional parallelization overhead as the model and network sizes grow. This paper proposes the use of Stochastic Gradient Push for distributed deep learning. The proposed method computes in-exact averages at each iteration in order to improve scaling efficiency and reduce the dependency on the underlying network topology. SGP converges to a stationary point at an $\mathcal{O}\left(1/\sqrt{nK}\right)$ rate in the smooth and non-convex case, and provably achieves a linear speedup (in iterations) with respect to the number of nodes. Empirical results show that SGP can be up to $3\times$ times faster than traditional ALLREDUCE SGD over high-latency interconnect, matches the top-1 validation accuracy up to 8 nodes (64GPUs), and remains within 1.2% top-1 validation accuracy for larger-networks.

References

[1] M. Assran, N. Loizou, N. Ballas, and M. Rabbat. Stochastic gradient push for distributed deep learning [extended version]. Preprint, available at arxiv.org/abs/1811.10792, Oct. 2018.

[2] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[3] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[5] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar. Collaborative deep learning in fixed topology networks. In *Advances in Neural Information Processing Systems*, pages 5904–5914, 2017.

[6] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science.*, pages 482–491. IEEE, 2003.

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 14, pages 583–598, 2014.

[8] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. 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.

[9] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. Exploring the limits of weakly supervised pretraining. *arXiv preprint arXiv:1805.00932*, 2018.

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

[11] A. Nedić, A. Olshevsky, and M. G. Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, (5):953–976, 2018.

[12] A. Paszke, S. Chintala, R. Collobert, K. Kavukcuoglu, C. Farabet, S. Bengio, I. Melvin, J. Weston, and J. Mariethoz. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration, may 2017.

[13] R. Rabenseifner. Optimization of collective reduction operations. In *Proc. Intl. Conf. Computational Science*, Krakow, Poland, Jun. 2004.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.