Stochastic Gradient Push for Distributed Deep Learning

Mahmoud Assran [1 2] **Nicolas Loizou** [1 3] **Nicolas Ballas** [1] **Mike Rabbat** [1]

Abstract

Distributed data-parallel algorithms aim to accelerate the training of deep neural networks by parallelizing the computation of large mini-batch gradient updates across multiple nodes. Approaches that synchronize nodes using exact distributed averaging (e.g., via ALLREDUCE) are sensitive to stragglers and communication delays. The PUSHSUM gossip algorithm is robust to these issues, but only performs approximate distributed averaging. This paper studies Stochastic Gradient Push (SGP), which combines PUSHSUM with stochastic gradient updates. We prove that SGP converges to a stationary point of smooth, non-convex objectives at the same sub-linear rate as SGD, and that all nodes achieve consensus. We empirically validate the performance of SGP on image classification (ResNet-50, ImageNet) and machine translation (Transformer, WMT'16 En-De) workloads.

1. Introduction

Deep Neural Networks (DNNs) are the state-of-the art machine learning approach in many application areas, including computer vision (He et al., 2016) and natural language processing (Vaswani et al., 2017). 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 (Wu et al., 2016; Mahajan et al., 2018).

Large mini-batch parallel SGD is usually adopted for distributed training of deep networks (Goyal et al., 2017; Li et al., 2014). 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 (Goyal et al., 2017; Akiba et al., 2017), or using a central parameter server, in asynchronous implementations (Dean et al., 2012). Using a parameter server to aggregate gradients introduces a potential bottleneck and a central point of failure (Lian et al., 2017). 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 communication-constrained settings, *i.e.*, where the network bandwidth is a significant bottleneck.

This issue motivates the investigation of a decentralized and inexact version of SGD to reduce the communication overhead associated with distributed training. There have been numerous decentralized optimization algorithms proposed and studied in the control-systems literature that leverage gossip-based approaches for the computation of aggregate information; see the survey of Nedić et al. (2018) and references therein. State-of-the-art gossip-based optimization methods build on the PUSHSUM algorithm for distributed averaging (Kempe et al., 2003; Nedić et al., 2018). Rather than computing exact averages (as with ALLREDUCE), this line of work uses less-coupled message passing and computes approximate averages. The tradeoff is that approximate distributed averaging also injects additional noise in the average gradient estimate.

In this work we study Stochastic Gradient Push (SGP), an algorithm blending parallel SGD and PUSHSUM. SGP enables the use of generic communication topologies that may be directed (asymmetric), sparse, and time-varying. In contrast, existing gossip-based approaches explored in the context of training DNNs (Lian et al., 2017; Jiang et al., 2017) are constrained to use symmetric communication (*i.e.*, if node i sends to j, then i must also receive from j before proceeding) and thus inherently require deadlock-avoidance, and more synchronization, making them slower and more sensitive to stragglers. Moreover, SGP can be seen as a generalization of parallel SGD and these previous approaches.

[1]Facebook AI Research, Montréal, QC, Canada [2]Department of Electrical and Computer Engineering, McGill University, Montréal, QC, Canada [3]School of Mathematics, University of Edinburgh, Edinburgh, Scotland. Correspondence to: Mahmoud Assran <mahmoud.assran@mail.mcgill.ca>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

SGP was first proposed in the control systems literature for minimizing the sum of *strongly-convex* functions (Nedić & Olshevsky, 2016). We make three main contributions. 1) We propose and analyze a novel variant of SGP, called Overlap SGP, which overlaps communication and computation to hide communication overhead. 2) We provide novel theoretical guarantees, proving that SGP (and Overlap SGP) converges to a stationary point of smooth *non-convex* functions at an $\mathcal{O}(1/\sqrt{nK})$ rate, for an appropriately chosen step-size, where n is the number of nodes and K is the number of iterations. 3) We conduct experiments on image classification (ResNet50, ImageNet) and neural machine translation tasks (Transformer, WMT16 En-De), demonstrating that SGP and Overlap SGP can substantially accelerate training of deep neural networks, by reducing communication overhead and mitigating the effects of stragglers. Given a fixed runtime budget, we find that SGP and Overlap SGP can train models that achieve better final train/validation accuracies than ALLREDUCE SGD in communication-constrained settings.

For example, we train a ResNet-50 on ImageNet using 256 GPUs spread across 32 compute nodes (8 GPUs / node), where communication between nodes is over 10 Gbps Ethernet. In this setting ALLREDUCE SGD achieves 76.2% top-1 validation accuracy, while Overlap SGP achieves 76.2% accuracy in only 1/3 of the time, and 77.1% accuracy in 1/2 the time. Similarly, when training a Transformer network on the WMT'16 En-De translation task, SGP runs approximately 1.5× faster than ALLREDUCE when using 8 GPUs and achieves a BLEU score that is 0.6 points higher. While our theory focuses on analyzing SGP, which combines PUSH-SUM with SGD, our experiments illustrate that PUSHSUM can similarly be efficiently combined with other optimizers like Nesterov's accelerated gradient method (Nesterov, 1983) and Adam (Kingma & Ba, 2015).

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 $\nabla F_i(\boldsymbol{x}_i;\xi_i)$, $\xi_i \sim D_i$, 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, where the canonical approach is 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.

Approximate distributed averaging. In this work we explore the alternative approach of using a gossip algorithm for approximate distributed averaging—specifically, the PUSH-SUM algorithm (Kempe et al., 2003). Let $\boldsymbol{y}_i^{(0)} \in \mathbb{R}^d$ 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. Concatenate 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, and conforms to the underlying communication topology. 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 sparser matrices $\boldsymbol{P}^{(k)}$ correspond to less communications.

Drawing inspiration from the theory of Markov chains (Seneta, 1981), the mixing matrices $\boldsymbol{P}^{(k)}$ are designed to be column stochastic (each column must sum to 1). Then, under mild conditions (ensuring that information from every node eventually reaches all other nodes) one can show that $\lim_{K\to\infty}\prod_{k=0}^K \boldsymbol{P}^{(k)} = \boldsymbol{\pi}\mathbf{1}^\top$, where $\boldsymbol{\pi}$ is the ergodic limit of the chain and $\mathbf{1}$ is a vector with all entries equal to 1. Consequently, the gossip iterations converge to a limit $\boldsymbol{Y}^{(\infty)} = \boldsymbol{\pi}(\mathbf{1}^\top \boldsymbol{Y}^{(0)})$; *i.e.*, the value at node i converges to $\boldsymbol{y}_i^{(\infty)} = \pi_i \sum_{j=1}^n \boldsymbol{y}_j^{(0)}$.

When the matrices $\boldsymbol{P}^{(k)}$ are symmetric, it is straightforward to design the algorithm so that $\pi_i = 1/n$ for all i by making $\boldsymbol{P}^{(k)}$ doubly-stochastic (each row and each column must sum to 1). However, symmetric $\boldsymbol{P}^{(k)}$ has strong practical ramifications, such as requiring care in the implementation to avoid deadlocks. 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). However, when the matrices $\boldsymbol{P}^{(k)}$ are asymmetric, it is very difficult, and often not possible, to design the algorithm so that $\pi_i = 1/n$. 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 $\boldsymbol{w}^{(\infty)} = \boldsymbol{\pi}(\mathbf{1}^\top \boldsymbol{w}^{(0)})$, or $w_i^{(\infty)} = \pi_i n$ at node i. Thus each node can recover the average of the initial vectors by computing the *de-biased* ratio $\boldsymbol{y}_i^{(\infty)}/w_i^{(\infty)}$.

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 $i \in \{1, 2, \dots, n\}$

1: **for** $k = 0, 1, 2, \cdots, K$, at node i, **do**
2: Sample new mini-batch $\xi_i^{(k)} \sim \mathcal{D}_i$ from local distribution
3: Compute 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; receive $\left(p_{i,j}^{(k)} \boldsymbol{x}_j^{(k+\frac{1}{2})}, p_{i,j}^{(k)} w_j^{(k)}\right)$ from in-neighbors
6: $\boldsymbol{x}_i^{(k+1)} = \sum_j p_{i,j}^{(k)} \boldsymbol{x}_j^{(k+\frac{1}{2})}$
7: $w_i^{(k+1)} = \sum_j p_{i,j}^{(k)} w_j^{(k)}$
8: $\boldsymbol{z}_i^{(k+1)} = \boldsymbol{x}_i^{(k+1)} / w_i^{(k+1)}$
9: **end for**

In practice, we stop after a finite number of gossip iterations K and compute $\boldsymbol{y}_i^{(K)} / w_i^{(K)}$.

PUSHSUM provides a mechanism to approximately synchronize parameters across a network of nodes. In the next section we describe the Stochastic Gradient Push algorithm, where PUSHSUM is used to approximately synchronize (average) parameters across nodes running stochastic gradient descent locally. The same approach can easily be modified to obtain decentralized versions of other popular optimizers such as SGD with momentum or Adam, as illustrated in the experimental section.

3. Stochastic Gradient Push

The *Stochastic Gradient Push* (SGP) method (Nedić & Olshevsky, 2016) for solving equation 1 is obtained by interleaving one local stochastic gradient descent update at each node with one iteration of PUSHSUM. Pseudocode is shown in Alg. 1. 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 $\boldsymbol{z}_i^{(k)} = \boldsymbol{x}_i^{(k)} / w_i^{(k)}$. The vector $\boldsymbol{x}_i^{(0)}$ can be initialized arbitrarily. At each iteration, every 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 $\boldsymbol{x}_i^{(k)}$, in line 4. All communication takes place in line 5, and each message contains two parts, the PUSHSUM numerator $\boldsymbol{x}_i^{(k)}$ and PUSHSUM weight $w_i^{(k)}$.

The non-zero entries in the mixing matrix $\boldsymbol{P}^{(k)}$ define the communication topology at each iteration k. SGP can leverage various communication topologies including sparse, asymmetric or time-varying networks. 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.

Individual nodes do not need to know the entire mixing matrix at each time step. Each node i must only know/choose its outgoing mixing weights, which correspond to the i^{th} column of $\boldsymbol{P}^{(k)}$. Each node can consequently choose its mixing weights independently of the other nodes in the network. In Appendix A we describe one way to design a sequence of mixing matrices that satisfies the requirements of our theory (described in the next section) and for which each node sends and receives exactly one message at every iteration; all appendices are in the Supplementary Material.

Overlapping communication and computation. Although SGP does not use network-wide collective communication primitives like ALLREDUCE, the implementation of Alg. 1 requires using blocking sends and receives; *i.e.*, nodes do not proceed to line 6 until they have received messages from all in-neighbors at that iteration. To hide the communication overhead, we can overlap gradient computation with communication. For a given $\tau \geq 0$, nodes send messages to their out-neighbours every τ iterations (non-blocking), and can receive incoming messages at any time in-between communication intervals. If a node hasn't received messages from its in-neighbors after τ iterations, then it blocks and waits to receive the messages.

Specifically, the communication in line 5 in Alg. 1 is made non-blocking, and each node may perform τ gradient update steps while it occurs. This may result in the gossip updates in lines 6 and 7 incorporating outdated messages, $(p_{j,i}^{(k')} \boldsymbol{x}_i^{(k'+\frac{1}{2})}, p_{j,i}^{(k')} w_i^{(k')})$, where $k - k' \leq \tau$. However, as long as the delay $k - k'$ remains bounded, SGP is still guaranteed to converge (see Theorem 2 below). We refer to this method, with delay bound τ as τ-overlap SGP (τ-OSGP). SGP is equivalent to τ-OSGP with $\tau = 0$. In practice we find that taking τ to be 1 or 2 is sufficient to hide effectively all of the communication overhead. Pseudocode for τ-OSGP is provided in Appendix B.

4. Theoretical guarantees.

SGP was first proposed and analyzed in (Nedić & Olshevsky, 2016) 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 with arbitrary, but bounded, message delays. We make the following four assumptions:

1. (L-smooth) There exists a constant $L > 0$ such that $\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\| \leq L \|\boldsymbol{x} - \boldsymbol{y}\|$.

2. (Bounded variance) There exist finite positive constants

σ^2 and ζ^2 such that

$$\mathbb{E}_{\xi\sim D_i}\|\nabla F_i(\boldsymbol{x};\xi)-\nabla f_i(\boldsymbol{x})\|^2\le\sigma^2\quad\forall i,\forall\boldsymbol{x},\text{ and}$$
$$\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(\boldsymbol{x})-\nabla f(\boldsymbol{x})\|^2\le\zeta^2\quad\forall\boldsymbol{x}.$$

Thus σ^2 bounds the variance of stochastic gradients at each node, and ζ^2 quantifies the similarity of data distributions at different nodes.

3. (Bounded delay) There exists a finite constant $\tau\in\mathbb{Z}_+$, such that the delay, if overlapping communication and computation, satisfies $k'-k\le\tau$.

4. (Mixing connectivity) To each mixing matrix $\boldsymbol{P}^{(k)}$ we can associate a graph with vertex set $\{1,\dots,n\}$ 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. By convention, we take each node to be an in-neighbor of itself (each node in the graph has a self-loop), and we assume that there exists finite, positive integers, B and Δ, such 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 $l\ge0$. [1]

Let $\overline{\boldsymbol{x}}^{(k)}=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i^{(k)}$. Lian et al. (2017) define that a decentralized algorithm for solving equation 1 converges if, for any $\epsilon>0$, it eventually satisfies

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\|\nabla f(\overline{\boldsymbol{x}}^{(k)})\|^2\le\epsilon. \tag{2}$$

We show that SGP converges in this sense.

Theorem 1. *Suppose that Assumptions 1–4 hold, and run SGP for K iterations with step-size $\gamma=\sqrt{n/K}$. Let $f^*=\inf_{\boldsymbol{x}}f(\boldsymbol{x})$ and assume that $f^*>-\infty$. There exist constants C and $q\in[0,1)$, which depend on the diameter of the network Δ, the upper bound on the delays τ, and the sequence of mixing matrices $\boldsymbol{P}^{(k)}$, such that when the total number of iterations satisfies*

$$K\ge\max\left\{\frac{nL^4C^4 60^2}{(1-q)^4},\frac{L^4C^4P_1^2n}{(1-q)^4(f(\overline{\boldsymbol{x}}^{(0)})-f^*+\frac{L\sigma^2}{2})^2},\right.$$
$$\left.\frac{L^2C^2nP_2}{(1-q)^2(f(\overline{\boldsymbol{x}}^{(0)})-f^*+\frac{L\sigma^2}{2})},n\right\} \tag{3}$$

where $P_1=4(\sigma^2+3\zeta^2)n+\frac{\sum_{i=1}^{n}\|\boldsymbol{x}_i^{(0)}\|^2}{n}$ and $P_2=\sigma^2+3\zeta^2L^2C^2+2\frac{\sum_{i=1}^{n}\|\boldsymbol{x}_i^{(0)}\|^2}{n}$, then

$$\frac{\sum_{k=0}^{K-1}\mathbb{E}\left\|\nabla f(\overline{\boldsymbol{x}}^{(k)})\right\|^2}{K}\le\frac{12(f(\overline{\boldsymbol{x}}^{(0)})-f^*+\frac{L\sigma^2}{2})}{\sqrt{nK}}.$$

The proof is given in Appendix E, where we also provide precise expressions for the constants C and q.

Theorem 1 shows that the average of the nodes' parameters, $\overline{\boldsymbol{x}}^{(k)}$, converges, but it does not directly say anything about the parameters at each node. In fact, we can show that:

Theorem 2. *Under the same assumptions as in Theorem 1,*

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

and

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

The proof is also given in Appendix E. This result shows that as K grows, the de-biased variables $\boldsymbol{z}_i^{(k)}$ converge to the node-wise average $\overline{\boldsymbol{x}}^{(k)}$, and hence the de-biased variables at each node also converge to a stationary point. Note that for fixed n and large K, the $1/\sqrt{nK}$ term will dominate the other factors.

5. Related Work

A variety of approaches have been proposed to accelerate distributed training of DNNs in the communcation bound setting, including quantizing gradients (Alistarh et al., 2007; Wen et al., 2007; Jia et al., 2018) and performing multiple local SGD steps at each node before averaging (McMahan et al., 2017).These approaches are complementary to the approach considered in this paper, which advocates for using approximate rather than exact distributed averaging. Quantizing gradients, performing multiple local SGD steps between averaging, and using approximate distributed averaging can all be seen as injecting additional noise (due to approximations) into SGD, leading to a tradeoff between reducing communication (towards training faster) and potentially obtaining worse predictive accuracy (due to approximations). Combining these approaches (quantized, infrequent, and inexact averaging) is an interesting direction for future work.

Blot et al. (2016) report initial experimental results on small-scale experiments with an SGP-like algorithm. Jin et al. (2016) make a theoretical connection between PUSHSUM-based methods and Elastic Averaging SGD (Zhang et al.,

[1] For the purpose of analysis, we model delays by augmenting the mixing-matrices $\boldsymbol{P}^{(k)}$, and the corresponding graph topologies, with virtual nodes and edges that store the state of information that was transmitted, but not yet received. We omit this description from the main text to simplify the discussion, and relegate this discussion to the supplementary material *Modeling message delays* in Appendix E.

2015). Relative to those previous works, we provide the first convergence analysis for a PUSHSUM-based method in the smooth non-convex case. Moreover, our analysis also holds in the presence of bounded message delays.

Lian et al. (2017) and Jiang et al. (2017) study synchronous gossip-based versions of SGD. Those methods involve symmetric message passing which inherently involves blocking (if i sends to j at iteration k, then j also sends to i before both nodes update). Consequently, they are slower in communication-constrained settings, in comparison to PUSHSUM-based SGP which may use directed message passing (i can send to j without needing a response). The work of Jakovetić et al. (2014) studies gossip-based versions of the Nesterov gradient method for smooth strongly-convex functions with deterministic gradients. The method requires performing two rounds of symmetric message passing per gradient update, and consequently is also slower in communication-constrained settings.

Decentralized parallel SGD (D-PSGD) (Lian et al., 2017) produces iterates whose node-wise average, $\overline{\boldsymbol{x}}^{(k)}$, converges in the sense of equation 2. Our results in Sec. 3 show that SGP converges in the same sense and go beyond to show that the individual values at each node also converge to a stationary point, since the values at each node converge to the network-wide average. In fact, SGP is a generalization of D-PSGD: when the communication topology is static, undirected, and connected at every iteration, and when nodes use symmetric mixing weights ($p_{j,i}^{(k)} = p_{i,j}^{(k)}$ for all (i, j)), then the push-sum weights $w_i^{(k)}$ are always equal to 1 and SGP is mathematically equivalent to D-PSGD. We compare SGP with D-PSGD experimentally in Section 6 and find that SGP is consistently faster and the two methods find solutions of comparable accuracy.

Jin et al. (2016) and Lian et al. (2018) study asynchronous gossip-based methods for training DNNs. Lian et al. (2018) analyzes an asynchronous version of D-PSGD and proves that its node-wise averages also converge to a stationary point. In general, these contributions focusing on asynchrony can be seen as orthogonal to the use of a PUSHSUM based protocol for approximate distributed averaging. Moreover, we find that synchronous Overlap SGP runs faster than asynchronous state-of-the-art AD-PSGD, and produces models with better training/validation performance.

6. Experiments

Next we experimentally compare SGP with ALLREDUCE SGD (AR-SGD), D-PSGD, and asynchronous D-PSGD (AD-PSGD). We aim to study the relationship between runtime and predictive accuracy as a function of the number of nodes used and the network bandwidth. In low-bandwidth experiments the servers communicate over 10 Gbps Ethernet links (typical in data centers) and in high-bandwidth experiments they communicate over 100 Gbps InfiniBand (typical in high-performance computing clusters). To illustrate the versatility of SGP, we consider two typical workloads: image classification and machine translation. All algorithms are implemented in PyTorch (Paszke et al., 2017).

While our SGP analysis focuses solely on the combination of PUSHSUM with SGD, we leverage Nesterov momentum or Adam in practice. Each node sends and receives one message per iteration in our SGP baseline implementation, and the destination and source of these messages changes from one iteration to the next. We refer the reader to Appendix A for implementation details, including how we design the sequence of mixing matrices $\boldsymbol{P}^{(k)}$. Our code is available at [https://github.com/facebookresearch/stochastic_gradient_push].

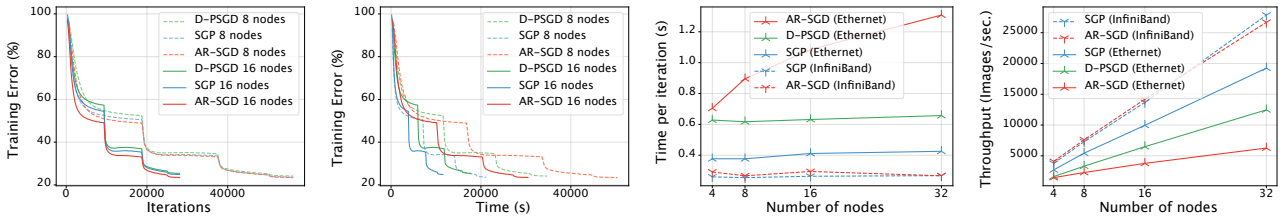
6.1. Image classification

We train a ResNet-50 (He et al., 2016) on the ImageNet classification task (Russakovsky et al., 2015). [2] Our experiments use 32 NVIDIA DGX-1 servers. Each server has 8 V100 GPUs. To investigate scaling we run experiments using 4, 8, 16, and 32 servers (*i.e.*, 32, 64, 128, and 256 GPUs). We follow the experimental protocol of Goyal et al. (2017). Every node uses a mini-batch size of 256, so using more nodes corresponds to larger effective mini-batch size. Unless indicated otherwise, all experiments are run for 90 epochs, the learning rate warms up to $n \times 0.1$ during the first five epochs following Goyal et al. (2017) and is decayed by a factor of 10 at epochs 30, 60, and 80. All methods use Nesterov momentum.

Scaling and convergence. The first set of experiments studies the scaling and convergence properties of our baseline SGP implementation, where every node sends and receives one message at every iteration (1-peer) and we do not overlap communication and computation (*i.e.*, $\tau = 0$).

Figure 1 (a) shows the iteration-wise training convergence of SGP, ALLREDUCE-SGD, and D-PSGD. Note that when we increase the number of nodes by a factor of 2, we also decrease the total number of iterations by the same factor. Figure 1 (b) shows the time-wise training convergence over 8 and 16 node Ethernet networks. In all cases, SGP completes 90 epochs in less time than ALLREDUCE-SGD and D-PSGD. Figures 1 (c) and (d) show the scaling efficiency of the methods on both 10 Gbps Ethernet and 100 Gbps InfiniBand networks. In the case of the InfiniBand network, all methods exhibit a near linear scaling (constant time per iteration), which is expected since communication is not a bottleneck in this setting. On the other hand, over the

[2] ImageNet was only used for the non-commercial research purposes of this paper and not for training networks deployed in production or for other commercial purposes.



(a) (b) (c) (d)

Figure 1: Scaling and convergence results on 4–32 nodes interconnected via 10 Gbps Ethernet and 100Gbps-InfiniBand for ALLREDUCE-SGD (AR-SGD), SGP and D-PSGD. (a)/(b): Iteration-wise convergences and time-wise convergence over 10 Gbps Ethernet. (c)/(d): Time-wise scaling efficiency over Ethernet and InfiniBand networks.

	4 nodes (32 GPUs)		8 nodes (64 GPUs)		16 nodes (128 GPUs)		32 nodes (256 GPUs)	
AR-SGD	76.2%	22.0 hrs.	76.4%	14.0 hrs.	76.3%	8.5 hrs.	76.2%	5.1 hrs.
D-PSGD	76.4%	19.7 hrs.	76.1%	9.7 hrs.	75.9%	5.0 hrs.	74.4%	2.6 hrs.
SGP	76.3%	11.8 hrs.	76.4%	5.9 hrs.	75.9%	3.2 hrs.	75.0%	1.7 hrs.

Table 1: Top-1 validation accuracy (%) and training time (hours), when communicating over 10 Gbps Ethernet for ALLREDUCE-SGD (AR-SGD), SGP and D-PSGD. SGP and D-PSGD are using 1-peer communication topologies.

10 Gbps Ethernet network, as we increase the number of nodes, the average iteration time stays almost constant for SGP and D-PSGD, while the per-iteration time of ALLREDUCE-SGD significantly increases, resulting in an overall slower training time. Moreover, although D-PSGD and SGP both exhibit strong scaling, SGP is roughly 1.5× faster over 10 Gbps Ethernet, supporting the claim that it involves less communication overhead.

Table 1 shows the total training time and top-1 validation accuracy of the different runs over the 10 Gbps Ethernet network using the baseline 1-peer topologies. 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 particular, for 32 node networks (256 GPUs), SGP training takes approximately 1.7 hours, while D-PSGD and ALLREDUCE-SGD require roughly 2.6 and 5.1 hours respectively.

To get a sense of the difference in runtimes between Ethernet and InfiniBand, and also to illustrate the variability, Table 2 shows the mean training time and top-1 validation accuracy along with the maximum absolute deviation from the mean, for 4- and 16-node experiments run on 100 Gbps InfiniBand networks. The max. absolute deviations are calculated based on five runs of each algorithm, using five different seeds. Even with the high-bandwidth InfiniBand network, SGP exhibits less variation in training time across different runs, supporting the claim that SGP helps reduce the effects of stragglers or other sources of latency.

All methods achieve roughly the same validation accuracy

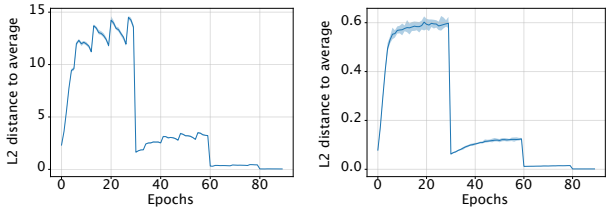
	4 nodes (32 GPUs)	16 nodes (128 GPUs)
AR-SGD	$76.3 \pm 0.2\%$ 8.8 ± 0.4 hrs.	$76.2 \pm 0.2\%$ 2.5 ± 0.3 hrs.
SGP	$76.3 \pm 0.2\%$ 8.2 ± 0.1 hrs.	$75.8 \pm 0.2\%$ 2.2 ± 0.1 hrs.

Table 2: Top-1 validation accuracy and training time statistics (mean $\pm$ max. abs. deviation), across 5 different runs of each algorithm using 5 different seeds. All methods complete 90 epochs, communicating over 100 Gbps InfiniBand.

for smaller 1-peer topologies (4 and 8 nodes), and the accuracy of D-PSGD and SGP degrades for larger topologies (16 and 32 nodes). We hypothesize that this is due to the error introduced by approximate distributed averaging—models at different nodes are only approximately identical, and for larger networks the divergence between models at different nodes is larger. We investigate this further next.

Parameter deviations. We track the deviations between the model parameters at different nodes while training with SGP. Our convergence result (cf. Lemma 3 in Appendix E) states that each node converges exponentially fast to a neighborhood of the average, where the size of the neighborhood is proportional to the step-size and the connectivity of the graph topology.

Figure 2 shows the average Euclidean distance between the parameters at individual nodes and the node-wise average, at the end of each epoch, for two different network configu-



(a) Sparse Graph Topology (b) Dense Graph Topology

Figure 2: Parameter Deviations for ResNet-50 trained on ImageNet (using SGP) over 16 node. Figures show the average Euclidean distance between the parameters at individual nodes and the node-wise average; recorded at the end of each training epoch, after the last gradient step, but before the last gossip step. Shaded regions show the max. and min. parameter deviations across all nodes. Nodes' parameters exhibit much greater deviation from the average when communicating over sparse topologies, than when communicating over dense topologies.

rations. The sparse graph corresponds to the time-varying 1-peer communication strategy used in the experiments above. The dense graph is fully-connected (all-to-all), and the deviations are computed just after line 4 in Alg. 1, before communicating.

Indeed, we see that distance from the mean model is proportional to the magnitude of the step-size, and each node's parameters are approximately equidistant from the average. At epochs 30, 60, and 80, the parameter deviations drop by an order of magnitude, in concert with the learning rate. We also see the parameter deviations gradually increase in the first five epochs, following the learning rate warm-up. The relation between the parameter deviations and the communication topology is also evident. Indeed, the dense topology exhibits significantly less parameter deviations than the time-varying sparsely connected topology (1-peer communication topology). Therefore, one can directly control the parameter deviations by adjusting the learning rate and/or graph topology, as predicted by Lemma 3.

Communication and the speed-accuracy tradeoff. Next we explore the effect of the communication topology on the speed-accuracy tradeoff when training with 16 and 32 nodes (128 and 256 GPUs) over 10 Gbps Ethernet. Table 3 shows the validation accuracy and wall-clock time for SGP using a 1-peer topology (1P-SGP), and SGP using a 2-peer topology (2P-SGP), *i.e.*, each node sends and receives to two peers at each iteration. Using just this one additional neighbor improves the validation accuracy of SGP to 76.2% in the 16 nodes case and to 75.7% for 32 nodes, while retaining much of the speed advantages. We also experiment with hybrid communication schemes that use more communication

	16 nodes (128 GPUs)		32 nodes (256 GPUs)	
AR-SGD	76.3%	8.5 hrs.	76.2%	5.2 hrs.
2P-SGP	76.2%	5.1 hrs.	75.7%	2.5 hrs.
1P-SGP	75.9%	3.2 hrs.	75.0%	1.7 hrs.
AR/1P-SGP	76.2%	4.8 hrs.	75.4%	2.8 hrs.
2P/1P-SGP	76.0%	3.5 hrs.	75.1%	1.8 hrs.

Table 3: Top-1 validation accuracies (%) and training time (hours) for 1P-SGP (1-peer topology); 2P-SGP (2-peer topology), AR-SGD (ALLREDUCE SGD), AR/1P-SGP (ALLREDUCE first 30 epochs, 1-peer topology last 60 epochs), and 2P/1P-SGP (2-peer topology first 30 epochs, 1-peer topology last 60 epochs), all communicating over 10 Gbps Ethernet.

	Train Acc.	Val. Acc.	Train Time
AR-SGD	76.9%	76.3%	8.5 hrs.
D-PSGD	75.6%	75.9%	4.9 hrs.
AD-PSGD	74.7%	75.5%	2.9 hrs.
SGP	75.6%	75.9%	3.2 hrs.
biased 1-OSGP	75.4%	75.3%	1.8 hrs.
1-OSGP	77.1%	75.7%	1.8 hrs.

Table 4: Comparing state-of-the-art synchronous and asynchronous gossip-based approaches to 1-OSGP, an implementation of synchronous SGP where communication is overlapped with 1 gradient step (all messages are always received with 1-iteration of staleness). 1-OSGP is also compared with a biased implementation of 1-OSGP that directly incorporates delayed messages without accounting for the bias in the push-sum weight. Experiments are run for 90 epochs over 16 nodes (128 GPUs) interconnected via 10 Gbps Ethernet.

at the start of training, to mitigate the parameter deviations when they are largest (cf. Figure 2). Table 3 compares 1P-SGP, 2P-SGP, and ALLREDUCE SGD (mathematically equivalent to running SGP with a fully-connected topology) with two hybrid methods: AR/1P-SGP, which uses ALLREDUCE for the first 30 epochs and 1-peer SGP for the last 60 epochs, and 2P/1P-SGP, which uses 2-peer SGP for the first 30-epochs and 1-peer SGP for the remainder. We find that these hybrid communication schemes provide a balance between speed and accuracy, and that communicating more during the first few epochs of training can mitigate accuracy tradeoffs.

Overlap SGP. Table 4 compares 1-OSGP using a 1-peer communication topology, with AD-PSGD, D-PSGD, and a biased implementation of 1-OSGP that directly incorporates delayed messages without accounting for the bias in the push-sum weight. Overlapping communication and computation greatly speeds up training and results in no accuracy

	Train Acc.	Val. Acc.	Train Time
AR-SGD	76.9%	76.2%	5.1 hrs. (90 epochs)
AD-PSGD	80.3%	76.9%	4.7 hrs. (270 epochs)
SGP	80.0%	77.1%	4.6 hrs. (270 epochs)
1-OSGP	81.8%	77.1%	2.7 hrs. (270 epochs)

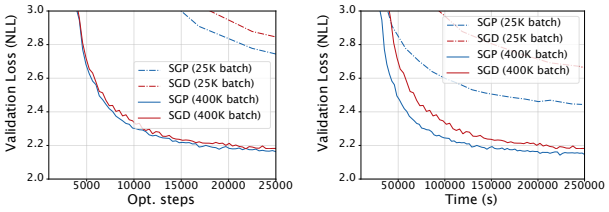
Table 5: Comparing ALLREDUCE SGD (AR-SGD) and SGP under a fix runtime budget. Given a similar runtime, SGP outperforms SGD for both training and validation accuracy. Running 1-OSGP for the same number of epochs than SGP outperforms SGD while improving the overal training efficiency. Experiments are run over 1-peer graph topologies, using 32 nodes (256 GPUs) interconnected via 10 Gbps Ethernet.

degradation relative to non-overlap SGP. In contrast, the biased implementation of 1-OSGP (not using the PUSHSUM weight) has significantly worse accuracy, supporting our theoretical development that the bias tracked in the push-sum weight facilitates convergence. Moreover, synchronous 1-OSGP runs faster than the state-of-the-art asynchronous AD-PSGD, and achieves better training and validation accuracy.

Fixed runtime budget. The results so far highlight that SGP completes 90 epochs of training faster than ALLREDUCE SGD, especially in communication-constrained settings, and this comes at the cost of reduced accuracy. However, if we compare the algorithms based on a runtime budget rather than an epoch budget, SGP achieves superior results. Specifically, since the baseline SGP is roughly $3\times$ faster than ALLREDUCE SGD when run on 32 nodes over 10 Gbps Ethernet, we run SGP for 270 epochs (instead of 90 epochs) using a scaled learning rate schedule (similar warm-up, but decaying the learning rate by a factor of 10 at epochs 90, 180, and 240). With this setup (32 nodes/256 GPUs and 10 Gbps Ethernet), SGP surpasses the best ALLREDUCE SGD accuracy (76.2% after 90 epochs/5.1 hrs.), and achieves a top-1 validation accuracy of 77.1% at then end of the 270 epochs (4.6 hrs.). Similarly, overlap SGP achieves a top-1 validation accuracy of 77.1% at then end of the 270 epochs (2.7 hrs.). When run longer, AD-PSGD achieves better accuracy than ALLREDUCE SGD, but not as fast nor as high of an accuracy as achieved by 1-OSGP.

6.2. Neural Machine Translation

We train a transformer network (Vaswani et al., 2017) on WMT16-En-De using our baseline implementation of SGP, and utilizing the same hyperparameters as (Vaswani et al., 2017). We train models using both $25K$ token batches (Vaswani et al., 2017), and $400K$ token batches (Ott et al., 2018). All methods use Adam. Training is performed



(a) Iteration-wise convergence (b) Time-wise convergence

Figure 3: Neural Machine Translations experiments run over $8\times$V100 GPUs located on 8 different machines, interconnected via 10 Gbps Ethernet. Adam-SGP and ALLREDUCE Adam-SGD iteration- and time-wise convergence in both small- and large-batch settings. SGP makes slightly more progress per iteration in both small- and large-batch settings, and runs approximately $1.5\times$ faster in the large-batch setting and $2\times$ faster in the small batch setting.

on 8 NVIDIA V100 GPUs, located on 8 different machines, interconnected via a 10 Gbps Ethernet link. This setup is similar to the AWS P3.2XLARGE machines commonly used for distributed training (Bernstein et al., 2019).

Figures 3 (a) and (b) show the iteration-wise and time-wise validation curves (respectively) of (Adam) SGP and ALLREDUCE (Adam) SGD using small- and large-batch training. We find that SGP makes slightly more progress per iteration in both small- and large-batch settings, and runs approximately $1.5\times$ faster than ALLREDUCE SGD in the large-batch setting, and $2\times$ faster in the small-batch setting. We also evaluate the test-set BLEU scores for models trained using 400K tokens batch. We investigate the performance obtained after the same number of iterations ($\sim$14K), and after training for the same amount of time (3 days). We evaluate BLEU scores using a beam search of 4, and a length penalty of 0.6, following (Vaswani et al., 2017). SGP achieves a superior BLEU score to ALLREDUCE SGD, both after a fix number of iterations (26.4 for SGP vs 25.8 for AR-SGD), and for fix runtime budget (27.5 for SGP vs 26.9 for AR-SGD).

7. Conclusion

We propose SGP and OSGP for accelerating distributed training of DNNs. We provide theoretical convergence guarantees in the smooth non-convex setting, matching known convergence rates for parallel SGD. We also empirically study the methods over several computing infrastructures, and provide assessments on image classification (ImageNet, ResNet-50) and neural machine translation (Transformer, WMT16 EN-DE) tasks. We find that SGP and OSGP can run significantly faster than parallel SGD in communication-bound settings, and can train better models in less time.

ACKNOWLEDGMENTS

We thank Shubho Sengupta and Teng Li for useful discussions and for maintaining the computing infrastructure used to conduct these experiments.

References

Akiba, T., Suzuki, S., and Fukuda, K. Extremely large minibatch sgd: training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

Alistarh, D., Grubic, D., Li, J. Z., Tomioka, R., and Vojnovic, M. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2007.

Assran, M. and Rabbat, M. Asynchronous subgradient-push. *arXiv preprint arXiv:1803.08950*, 2018.

Bernstein, J., Zhao, J., Azizzadenesheli, K., and Anandkumar, A. signSGD with majority vote is communication efficient and fault tolerant. In *International Conference on Learning Representations*, 2019.

Blot, M., Picard, D., Cord, M., and Thome, N. Gossip training for deep learning. In *NIPS Workshop on Optimization for Machine Learning*, 2016.

Charalambous, T., Yuan, Y., Yang, T., Pan, W., Hadjicostis, C. N., and Johansson, M. Distributed finite-time average consensus in digraphs in the presence of time delays. *IEEE Transactions on Control of Network Systems*, 2(4): 370–381, 2015.

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

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1243–1252, 2017.

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

Hadjicostis, C. N. and Charalambous, T. Average consensus in the presence of delays in directed graph topologies. *IEEE Transactions on Automatic Control*, 59(3):763–768, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Jakovetić, D., Xavier, J., and Moura, J. M. Fast distributed gradient methods. *IEEE Transactions on Automatic Control*, 59(5):1131–1146, 2014.

Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

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

Jin, P. H., Yuan, Q., Iandola, F., and Keutzer, K. How to scale distributed deep learning? In *NIPS ML Systems Workshop*, 2016.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

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

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 5330–5340, 2017.

Lian, X., Zhang, W., Zhang, C., and Liu, J. Asynchronous decentralized parallel stochastic gradient descent. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 3049–3058, 2018.

Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 181–196, 2018.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and Agüera y Arcas, B. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

Nedić, A. and Olshevsky, A. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. *IEEE Trans. Automatic Control*, 61(12):3936–3947, 2016.

Nedić, A., Olshevsky, A., and Rabbat, M. G. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, 106 (5):953–976, 2018.

Nesterov, Y. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

Ott, M., Edunov, Sergey, G. D., and Auli, M. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018.

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

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

Seneta, E. *Non-negative Matrices and Markov Chains*. Springer, 1981.

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

Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. TernGrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*, pp. 1509–1519, 2007.

Wolfowitz, J. Products of indecomposible, aperiodic, stochastic matrices. *Proceedings of the American Mathematical Society*, 14(5):733–737, 1963.

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

Zhang, S., Choromanska, A., and LeCun, Y. Deep learning with elastic averaged SGD. In *Advances in Neural Information Processing Systems*, pp. 685–693, 2015.