Elastic Consistency: A Practical Consistency Model for Distributed Stochastic Gradient Descent

Giorgi Nadiradze,[1] Ilia Markov, [1] Bapi Chatterjee, [1] Vyacheslav Kungurtsev, [2] Dan Alistarh [1]
[1] IST Austria
[2] Czech Technical University in Prague
giorgi.nadiradze@ist.ac.at, ilia.markov@ist.ac.at, bapi.chatterjee@ist.ac.at, vyacheslav.kungurtsev@fel.cvut.cz, dan.alistarh@ist.ac.at

Abstract

One key element behind the progress of machine learning in recent years has been the ability to train machine learning models in large-scale distributed shared-memory and message-passing environments. Most of these models are trained employing variants of stochastic gradient descent (SGD) based optimization. In this paper, we introduce a general consistency condition covering communication-reduced and asynchronous distributed SGD implementations. Our framework, called elastic consistency, decouples the system-specific aspects of the implementation from the SGD convergence requirements, giving a general way to obtain convergence bounds for a wide variety of distributed SGD methods used in practice. Elastic consistency can be used to re-derive or improve several previous convergence bounds in message-passing and shared-memory settings, but also to analyze new models and distribution schemes. In particular, we propose and analyze a new synchronization-avoiding scheme for distributed SGD, and show that it can be used to efficiently train deep convolutional models for image classification.

Introduction

Machine learning models can match or surpass humans on specialized tasks such as image classification (Krizhevsky, Sutskever, and Hinton 2012; He et al. 2016), speech recognition (Seide et al. 2014), or complex games (Silver et al. 2016). One key tool behind this progress is the *stochastic gradient descent (SGD)* family of methods (Robbins and Monro 1951), which are by and large the method of choice for training large-scale machine learning models. Essentially, SGD can serve to minimize a d-dimensional function $f : \mathbb{R}^d \to \mathbb{R}$, assumed to be differentiable. We commonly assume that we are given access to (possibly noisy) gradients of this function, denoted by $\widetilde{G}$. Sequential SGD will start at a randomly chosen point $\vec{x}_0$, say 0^d, and converge towards a minimum of the function by iterating the following procedure:

$$\vec{x}_{t+1} = \vec{x}_t - \alpha \widetilde{G}(\vec{x}_t) \tag{1}$$

where $\vec{x}_t$ is the current estimate of the optimum, also called the *parameter* and α is the *learning rate*. If the function is convex, this procedure is known to converge to the minimum of the function (Bubeck et al. 2015), whereas in the non-convex case, it will converge towards a point of zero gradient (Ghadimi and Lan 2013). In *supervised learning*, f is usually the total error of a given parameter $\vec{x}$ on a given dataset $\mathcal{D}$. For each sample s in $\mathcal{D}$, the classification error is encoded via the loss $\ell(s, \vec{x})$. Training minimizes the function $f(\vec{x}) = \frac{1}{m} \sum_{s \in \mathcal{D}} \ell(s, \vec{x})$, where m is the size of the dataset, and the gradient at a randomly chosen datapoint $\widetilde{G}$ is an unbiased estimator of ∇f.

Due to the size of datasets, it is common to *distribute* the optimization process across multiple processors. A standard way of parallelizing SGD is to process a *batch* of samples in parallel, dividing the computation of gradient updates among processors. Assume for simplicity that each processor is allotted one sample, whose corresponding gradient it computes with respect to the current parameter $\vec{x}_t$. Processors then *sum* their stochastic gradients, and update their local parameters by the resulting sum, leading to the following global iteration:

$$\vec{x}_{t+1} = \vec{x}_t - \frac{\alpha}{p} \sum_{i=1}^{p} \widetilde{G}^i(\vec{x}_t), \tag{2}$$

where $\widetilde{G}^i$ is the stochastic gradient obtained at the processor i at the given step, and p is the batch size, equal to the number of processors. Since this sum is the same at every processor, this procedure yields the same, perfectly consistent, parameter at each processor at the end of every parallel iteration. The average $(1/p) \sum_{i=1}^{p} \widetilde{G}^i(\vec{x}_t)$ is still a stochastic gradient, but with *lower variance* than gradients at single samples, which can lead to better convergence (Bubeck et al. 2015). Since samples are now processed in parallel, the number of samples processed per second should in theory be multiplied by p.

However, in practice, maintaining perfect consistency of the parameter $\vec{x}_t$ can negate the benefits of parallelization. Keeping the parameters perfectly consistent has a *communication cost*: since the size of gradient updates is *linear* in the size of the parameter, the resulting communication may easily become a system bottleneck for models with millions of parameters (Krizhevsky, Sutskever, and Hinton 2012; Alistarh et al. 2017). Consistency also induces a *synchronization cost*, since processors need to synchronize in a barrier-like fashion upon each iteration update, which can occur every few milliseconds. For this reason, there have been several proposals for relaxing the consistency requirements of SGD-like iterations, under various system constraints. These proposals can be broadly categorized as follows:

- **Asynchronous Methods:** Such implementations (Recht et al. 2011; Lian et al. 2015) allow processors to forgo the barrier-like synchronization step performed at each iteration, or even across parameter components, and move forward with computation without waiting for potentially slow straggler processors.
- **Communication Compression:** These methods aim to reduce the bandwidth cost of exchanging the gradients. This usually entails performing (possibly lossy) compression of the gradients before transmission, followed by efficient encoding and reduction/summation, and decoding on the receiver end. This can be either via bit-width reduction (quantization), e.g. (Seide et al. 2014; Alistarh et al. 2017), or via structured sparsification of the updates (Lin et al. 2018b; Aji and Heafield 2017; Alistarh et al. 2018; Stich and Karimireddy 2019).

Additional approaches exist, for instance to reduce the *frequency* of communication via large-batch methods or local steps, e.g. (Goyal et al. 2017; Lin et al. 2018a; Stich 2018). Another axis controls parameter maintenance: *centralized* methods such as the *parameter server* (Li et al. 2014) maintain the parameter at a single entity, whereas *decentralized* methods (Lian et al. 2017; Lu, Li, and Sa 2020) have each processor maintain their own version of the model.

The question of providing convergence bounds for distributed optimization goes back to the foundational work of Bertsekas and Tsitsiklis (1989), and has recently risen to prominence (Dean et al. 2012; Ho et al. 2013; Chilimbi et al. 2014; Recht et al. 2011; Ho et al. 2013; Sa et al. 2015; Lian et al. 2015; Chaturapruek, Duchi, and Ré 2015; Leblond, Pedregosa, and Lacoste-Julien 2017). However, many of these proofs are often specialized to the algorithm and models, and do not generalize to different settings. It is therefore natural to ask: are there generic conditions covering all natural consistency relaxations for SGD, under which one can prove convergence?

Contribution. In this paper, we introduce a convergence criterion for SGD-based optimization called *elastic consistency*, which is independent of the system model, but can be specialized to cover various model consistency relaxations.

In a nutshell, elastic consistency says that, for SGD to converge, it is sufficient that the distance between the *view* of the parameter perceived by a processor, with respect to which the gradient is taken, and the "true" view of the system, corresponding to all the updates to the parameter generated up to that point by all processors, be uniformly bounded across iterations, and decreasing proportionally to the learning rate. Intuitively, in this case, the perturbed iterates do not stray "too far" from eachother, and can still globally converge. To our knowledge, elastic consistency is satisfied in most settings where asynchronous or communication-reduced methods have been analyzed so far, although proving this property for some systems is not always immediate.

Elastic consistency provides a unified analysis framework for all the method types discussed above. Consequently, we are able to re-prove or improve convergence bounds for several methods, and to tackle new models and consistency relaxations. Our contributions are as follows:

1. Under standard smoothness assumptions on the loss, elastic consistency is sufficient to guarantee convergence rates for inconsistent SGD iterations for both *convex* and *non-convex* objectives. This condition is also *necessary* for SGD convergence: we provide simple worst-case instances where SGD convergence is linear in the elastic consistency parameter, showing that the iterations will diverge if elastic consistency is regularly broken.
2. We show that elastic consistency is satisfied by both asynchronous message-passing and shared-memory models, centralized or decentralized, with or without faults, and by communication-reduced methods. This implies new convergence bounds for SGD in the classic asynchronous and semi-synchronous message-passing models (Attiya and Welch 2004) and extends previous analyses for the shared-memory model (Sa et al. 2015; Alistarh, De Sa, and Konstantinov 2018).
3. This convergence condition inspires a new scheduling mechanism for parallel SGD, called *elastic scheduling*, which controls communication-compression and asynchrony guided by our consistency condition in order to reduce synchronization overhead, while maintaining accuracy. Its implementation provides non-trivial speedup upon BytePS (Peng et al. 2019), the state-of-the-art scheduler for training deep neural networks, while maintaining accuracy.

Elastic Consistency

Distributed Model and Adversarial Scheduling. We consider distributed systems consisting of p processors: $\{1, 2, \ldots, p\}$, some of which may be faulty, where communication happens either by message-passing, or via shared-memory. For simplicity, we will specify the system and fault models in the corresponding sections. We assume that the scheduling of steps (e.g. reads/writes in shared-memory, or message delivery in message-passing) is controlled by an *oblivious* adversarial entity. This means that scheduling decisions may be adversarial, but are *independent* of the randomness in the algorithm, and in particular of the data sampling. Practically, this implies that the conditioning on any random event involving previous SGD iterations $s < t$ does not impact choices made at iteration t.

Distributed Optimization. We assume that each of the p processors is given access to random samples coming from an unknown d-dimensional data distribution $\mathcal{D}$, and collaborates to jointly minimize $f : \mathcal{X} \to \mathbb{R}$ over the distribution $\mathcal{D}$, where $\mathcal{R}^d$ is a compact subset of $\mathbb{R}^d$. In practice, nodes optimize over a finite set of samples $S = \{S_1, S_2, \ldots, S_m\}$, and the function f is defined as

$$f(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m} \ell(S_i, \vec{x}) \tag{3}$$

where ℓ is the loss function at a sample s. The goal is to find

$\vec{x}^* \in \mathcal{R}^d$, which minimizes the expected loss over samples, defined as: $\vec{x}^* = \text{argmin}_{\vec{x}} f(\vec{x}) = \text{argmin}_{\vec{x}} \mathbb{E}_{s\sim\mathcal{D}}[\ell(s, \vec{x})]$.

Properties of Stochastic Gradients. Let $\tilde{G}$ be a stochastic gradient. We make the following standard assumptions (Sa et al. 2015):

1. **Unbiasedness.** The i.i.d. stochastic gradients are unbiased estimators of the true gradient of the function f:
$$\forall \vec{x} \in \mathbb{R}^d,\ \mathbb{E}\left[\tilde{G}(\vec{x})\right] = \nabla f(\vec{x}). \tag{4}$$
2. **Bounded Variance.** The stochastic gradients have bounded variance:
$$\forall \vec{x} \in \mathbb{R}^d,\ \mathbb{E}\left[\|\widetilde{G}(\vec{x}) - \nabla f(\vec{x})\|^2\right] \leq \sigma^2. \tag{5}$$
3. **Bounded Second Moment.** It is sometimes assumed that the second moment of the stochastic gradients over the sample space is bounded:
$$\forall \vec{x} \in \mathbb{R}^d,\ \mathbb{E}\left[\|\widetilde{G}(\vec{x})\|^2\right] \leq M^2. \tag{6}$$
Elastic consistency *does not* require the second moment bound to ensure convergence—the variance bound is sufficient. However, in e.g. asynchronous shared-memory (Sa et al. 2015), this stronger assumption is common, and we will use it to bound the elastic consistency constant.

Properties of the Objective Function. We will make use of the following standard definitions regarding the objective function f: $\forall\, \vec{x}, \vec{y} \in \mathbb{R}^d$,

1. **Smoothness.** The function $f : \mathbb{R}^d \to \mathbb{R}$ is smooth iff:
$$\|\nabla f(\vec{x}) - \nabla f(\vec{y})\| \leq L\|\vec{x} - \vec{y}\| \text{ for } L > 0. \tag{7}$$
2. **Strong convexity.** Problems such as linear regression have a strongly convex objective:
$$(\vec{x} - \vec{y})^T(\nabla f(\vec{x}) - \nabla f(\vec{y})) \geq c\,\|x - y\|^2 \text{ for } c > 0. \tag{8}$$
For such functions, the bound over second moment of stochastic gradients does not hold $\forall\, \vec{x} \in \mathbb{R}^d$ (Nguyen et al. 2018). Therefore, we restrict $f : \mathcal{X} \to \mathbb{R}$ for a convex set $\mathcal{X} \subset \mathbb{R}^d$, such that $\forall x \in \mathcal{X}$, (6) is satisfied. For simplicity, we omit the projection step onto $\mathcal{X}$, in the case when $\vec{x_t}$ does not belong to $\mathcal{X}$ for some iteration t.
3. **Lower bound for non-strongly-convex functions.** In many settings, such as training of neural networks, the objective function is not necessarily strongly-convex. In such "non-strongly-convex" settings, it is necessary to assume that f is bounded from below:
$$\exists f^* \text{ finite s.t. } \forall\, \vec{x} \in \mathbb{R}^d,\ f(\vec{x}) \geq f^*. \tag{9}$$

Elastic Consistency Definition

An Abstract Consistency Model. We assume that we have p processors, which share a *parameter oracle* $\mathcal{O}$. In each iteration, each processor i invokes this oracle, and receives a *local view* at of the parameter at step t, which we denote $\vec{v}_t^i$. The processor then uses this view of the parameter to generate a new update (stochastic gradient) $\tilde{G}(\vec{v}_t^i)$, which it applies to the shared model.

The key question is how to express the consistency of the local views across processors. For this, we introduce an auxiliary variable $\vec{x}_t$, which we call the *global parameter*. Initially we have that $\vec{x}_0 = \vec{v}_0^1 = ... = \vec{v}_0^p$.

We consider two cases, depending on how data-parallel SGD is implemented. The first is the **single steps** case, where the gradient generated locally by each processor is *directly applied* to the model. This is done in asynchronous shared-memory (Sa et al. 2015) or some message-passing implementations (Lian et al. 2015), and is modelled as:
$$\vec{x}_{t+1} = \vec{x}_t - \alpha\tilde{G}(\vec{v}_t^i). \tag{10}$$

The second is the **parallel steps** case, where processors' gradients are aggregated before they are applied to the shared model, for instance via averaging. This is common in synchronous message-passing settings, e.g. (Li et al. 2014). Formally, we are given a set $I_t \subseteq \{1, 2..., p\}$, of gradients to be averaged, such that $p/2 \leq |I_t| \leq p$. Each processor $i \in I_t$ calculates a stochastic gradient based on its local view at step t, and the global model is updated by aggregating all the gradients in I_t:
$$\vec{x}_{t+1} = \vec{x}_t - \frac{\alpha}{p}\sum_{i\in I_t}\tilde{G}(\vec{v}_t^i). \tag{11}$$

Then, elastic consistency says the following:

Definition 1 (Elastic Consistency). *A distributed system provides* elastic consistency *for the SGD iteration defined in (10) or (11), if there exists a constant $B > 0$, which, for every time t, bounds the expected norm difference between the true parameter $\vec{x}_t^i$, and the view $\vec{v}_t^i$ returned by the oracle at processor i at iteration t. Importantly, B should be independent of the iteration count t, but possibly dependent on the system definition. Formally,*
$$\mathbb{E}\left[\|\vec{x}_t - \vec{v}_t^i\|^2\right] \leq \alpha^2 B^2, \tag{12}$$
where $B > 0$, α is the learning rate at iteration t, and the expectation is taken over the randomness in the algorithm. We call B the elastic consistency constant.

Discussion. First, please note that, in the above, time t counts each time step at which a stochastic gradient is generated at a node, in sequential order. Intuitively, the auxiliary parameter $\vec{x}_t$ is defined as the sum of generated gradients up to t, multiplied by the corresponding learning rate. In the analysis, we need to define $\vec{x}_t$ precisely for each application; please see Table 1 for example bouunds, or the full version of the paper (Nadiradze et al. 2020) for carefully worked-out examples, for different system models.

Generally, the process for deriving the elastic consistency bound B for a given system is as follows. We assume a *fixed, given* LR sequence, which ensures convergence w.r.t. the sequential iteration (1). We then examine the distributed system specification to bound $\|\vec{x}_t - \vec{v}_t\|^2$. Crucially, for all the systems and consistency relaxations we analyze, this bound separates cleanly into the *LR part* (α^2), and a *constant part* (B^2) which is *independent of time or LR*. (Please see (Nadiradze et al. 2020) for a step-by-step example.) This holds for all t. Finally, the learning rate sequence used by the algorithm may need to be adjusted (e.g. normalized) to satisfy certain convergence constraints.

Later, we show that virtually all known models and consistency relaxations satisfy this condition (see Table 1). Elastic consistency gives wide latitude to the parameter oracle about which exact values to return to the processor: the returned view can contain random or even adversarial noise, or updates may be delayed or missing, as long as their relative weight is bounded and independent of time.

Elastic Consistency and SGD Convergence

We now show that this notion of consistency implies non-trivial convergence guarantees for SGD for different types of objective functions, and that this notion is in some sense necessary for convergence. The complete proofs are available in the full version of the paper (Nadiradze et al. 2020).

Elastic Consistency is *Sufficient* for Convergence

The Non-Convex Case. We begin with the more general case where the objective function is not necessarily convex. In this case, since convergence to a global minimum is not guaranteed for SGD, we will only require convergence to a point of vanishing gradients, as is standard, e.g. (Wangni et al. 2018). Specifically, assuming elastic consistency, we prove the following theorems:

Theorem 2. *Consider SGD iterations defined in (10) and satisfying elastic consistency bound (12). For a smooth non-convex objective function f, whose minimum x^* we are trying to find and the constant learning rate $\alpha = \frac{1}{\sqrt{T}}$, where $T \geq 36L^2$ is the number of iterations:*

$$\min_{t\in[T-1]} \mathbb{E}\|\nabla f(\vec{x}_t)\|^2 \leq \frac{4(f(\vec{x}_0) - f(x^*))}{\sqrt{T}} + \frac{2B^2L^2}{T} + \frac{6L\sigma^2}{\sqrt{T}} + \frac{6L^3B^2}{T\sqrt{T}}.$$

Assuming that L, B, σ and $f(\vec{x}_0) - f(x^*)$ are constant, we get a convergence rate of $O(1/\sqrt{T})$. This result can be specialized to parallel iterations (11), yielding the theorem which ensures $\sqrt{p}$ parallel speedup in the number of nodes p (Please see (Nadiradze et al. 2020)).

The Strongly Convex Case. We can provide improved guarantees under strong convexity: If the total number of SGD iterations is T, then for iterations defined in (10), $\mathbb{E}\|\vec{x}_T - x^*\|^2 \leq \tilde{O}(1/T)$ and for iterations defined in (11), $\mathbb{E}\|\vec{x}_T - x^*\|^2 \leq \tilde{O}(1/pT)$. Here, $\tilde{O}$ notation hides factors which are polynomial in $\log T$ and $\log p$, and all parameters except T and p are treated as constants. We refer the reader to (Nadiradze et al. 2020) for the details and proofs.

Discussion. The parameter B abstracts the distributed-system specific parameters to provide a clean derivation of the convergence theory. In turn, depending on the system setting, B might depend on the second-moment bound M^2 or variance bound σ^2, but also on system parameters such as the maximum delay τ, on the number of failures f, or on the characteristics of the compression scheme.

Later, we exhibit natural distribution schemes for which the bound on B does not require a second-moment bound on gradients.

Elastic Consistency is *Necessary* for Convergence

We can also show that elastic consistency can be directly linked to convergence; in particular, in the worst case, an adversarial parameter oracle can slow down convergence *linearly* in B. The intuition is that once the algorithm is close to the minimum, an adversarial oracle can force it to evaluate gradient at point which is distance B away and this will cause the algorithm to overshoot the minimum The argument is similar to that of (Alistarh, De Sa, and Konstantinov 2018).

Lemma 3 (Convergence Lower Bound)**.** *There exists a convex (quadratic) function f and an adversarial oracle $\mathcal{O}$ s.t. the algorithm with elastic consistency bound converges B times slower than the exact algorithm (B=0).*

Distributed System Models and their Elastic Consistency Bounds

Fault-Tolerant Message-Passing Systems

We consider a message-passing (MP) system of p nodes $\mathcal{P} = \{1, 2, \ldots, p\}$ executing SGD iterations, which are connected to each other by point-to-point links. To simplify the description, we will focus on the case where the system is *decentralized*: in this case, each node i acts both as a worker (generating gradients) and as a parameter server (PS) (maintaining a local parameter copy). (The only difference is that, in the *centralized* PS case, a single designated node would maintain a global parameter copy.) Without loss of generality, all nodes start with the same parameter $\vec{v}_0^i = \vec{0}$. The system proceeds in global iterations, indexed by t. In each iteration, workers generate a stochastic gradient based on its current model copy $\vec{v}_t^i$, and broadcasts it to all other nodes.

Consistency Relaxations: Consider node i, and recall that it acts as a parameter server, in addition to being a worker itself. Let $\mathcal{L}_t^i \subseteq \{1, \ldots, p\}$ denote the set of nodes from which i receives stochastic gradients at iteration t. By convention, $i \in \mathcal{L}_t^i$. In the *synchronous failure-free* case, all nodes would receive exactly the same set of messages, and the execution would be equivalent to a sequential batch-SGD execution. In a real system, not all nodes may have the same view of each round, due to failures or asynchrony. Specifically, we consider the following distinct consistency relaxations:

(a) **Crash faults.** A node $i \in \mathcal{P}$ may *crash* during computation or while sending messages. In this case, the node will remain inactive for the rest of the execution. Importantly, node i's crash during broadcasting may cause other nodes to have different views at the iteration, as some of them may receive i's message while others may not, resulting in inconsistent updates of parameter $\vec{x}$ across the nodes. We will assume that $f \leq p/2$ nodes may crash in the message-passing system.

(b) **Message-omission failures.** In practical systems, each node could implement iteration t by waiting until an interval Υ^t_{max} in terms of *clock time* has elapsed: if the message from peer j is not received in time, node i moves on, updating its local parameter $\vec{x}^i_t$ only w.r.t. received messages. However, node i will include j's message into its view if received in a later iteration, although some messages may be permanently delayed. We assume a parameter f which upper bounds the number of messages omitted at any point during the execution. We note that this model is stronger than the Crash-fault model considered above, as we can simulate a node's failure by discarding all its messages after the crash.

(c) **Asynchrony.** The two above models assume that nodes proceed synchronously, in *lock-step*, although they may have inconsistent views due to node or message failures. An alternative relaxation (Lian et al. 2015) is if nodes proceed *asynchronously*, i.e. may be in different iterations at the same point in time. Specifically, in this case, we assume that there exists a maximum delay $\tau_{\max}$ such that each message/gradient can be delayed by at most $\tau_{\max}$ iterations from the iteration when it was generated.

(d) **Communication-Compression.** Another way of reducing the distribution cost of SGD has been to compress the stochastic gradients communicated at each round. In this context, sparsification with memory (Strom 2015; Seide et al. 2014; Aji and Heafield 2017; Alistarh et al. 2018; Karimireddy et al. 2019) has proven to be particularly effective. This process can be modelled as follows. We assume that each node maintains a local version of the parameter $\vec{v}^i_t$, and an *error/memory* vector $\vec{\epsilon}^i_t$, initially $\vec{0}$. In each iteration, each node computes a new gradient $\widetilde{G}\left(\vec{v}^i_t\right)$ based on its local parameter. It then adds the current error vector $\vec{\epsilon}^i_t$ to the gradient, to obtain its full proposed update $\vec{\nabla}^i_t$. However, before transmitting this update, it compresses it by using a (lossy) compression function Q. The compressed update $Q(\vec{\nabla}^i_t)$ is then transmitted, and the error vector is updated to $\vec{\epsilon}^i_{t+1} \leftarrow \vec{\nabla}^i_t - Q(\vec{\nabla}^i_t)$. Our analysis will only require that Q satisfies $\|Q(\vec{\nabla}) - \vec{\nabla}\|^2 \leq \gamma\|\vec{\nabla}\|^2,\ \forall\vec{\nabla} \in \mathbb{R}^d$, for some $1 > \gamma \geq 0$. All memory-based techniques satisfy this, for various definitions of Q and γ. (We provide examples in the additional material.)

We note that the above discussion considered these methods independently. However, we do note that our method does allow for these relaxations to be combined—for instance, one can analyze an asynchronous, fault-tolerant method with communication compression.

Asynchronous Shared-Memory Systems

We consider a system with p processors (or threads) $\mathcal{P} = \{1, 2, \ldots, p\}$, which can communicate through shared memory. Specifically, we assume that the parameter vector $\vec{w} \in \mathbb{R}^d$ is shared by the processors, and is split into d components, one per dimension. Processors can atomically read a component via a `read` operation, and update it via the atomic `fetch&add` (`faa`) operation, which reads the current value of the component and updates it in place, in a single atomic step. In each iteration t, each processor first obtains a local view $\vec{v}^i_t$ of the parameter by scanning through the shared parameter $\vec{w}$ component-wise. It then generates a stochastic gradient $\widetilde{G}\left(\vec{v}^i_t\right)$ based on this view, and proceeds to update $\vec{x}$ via `faa` on each component, in order. (We refer the reader to (Nadiradze et al. 2020) for a full description, including pseudocode.)

Consistency Relaxation. Ideally, threads would proceed in lock step, first obtaining perfect, identical snapshots of $\vec{w}$, calculating gradients in terms of this identical parameter, and then summing the gradients before proceeding to the next iteration. However, in practice, threads are *asynchronous*, and proceed at arbitrary speeds. This causes their snapshots to be inconsistent, as they might contain some partial concurrent updates, but not others. The challenge is to prove SGD convergence in this case. It is common (Recht et al. 2011; Sa et al. 2015) to assume a bound $\tau_{\max}$ on the maximum delay between the time (iteration) when an individual update was generated, and the iteration when it has been applied, and becomes visible to all processors. In this case, the auxiliary variable $\vec{x}_t$ used by elastic consistency will correspond to the sum of first t stochastic gradients, ordered by the time when the atomic `faa` over the first index of $\vec{w}$ was performed.

Elastic Consistency Bounds for Specific Systems

Given these definitions, we can now state the elastic consistency bounds for the different types of distributed systems and consistency relaxations. Please see Table 1, and (Nadiradze et al. 2020) for the detailed derivations.

Implications. Plugging the asynchronous shared-memory bound into Theorem 2 (and its version with multiple steps) implies convergence bounds in the smooth non-convex case, extending (Sa et al. 2015; Alistarh, De Sa, and Konstantinov 2018), which focus on the convex case, whereas the asynchronous message-passing bound implies similar bounds to the best known in the non-convex case for this model (Lian et al. 2015). For synchronous message-passing with communication-compression, our framework implies the first general bounds for the *parallel, multi-node* case: references (Stich, Cordonnier, and Jaggi 2018; Karimireddy et al. 2019) derive tight rates for such methods, but in the *sequential* case, where there is a single node which applies the compressed gradient onto its model, whereas (Alistarh et al. 2018) considers the multi-node case, but requires an additional analytic assumption. Please see Section for additional discussion on related work. In the crash-prone case, elastic consistency implies new convergence bounds for crash or message-omission faults. Note that, although the elastic bound is the same for both f crash and message-omissions

System	Consistency Relaxation	Bound B	Novelty
Shared-Memory	τ_{max}-Bounded Asynchrony	$\sqrt{d}\tau_{max}M$	Extends (Sa et al. 2015; Alistarh, De Sa, and Konstantinov 2018)
Message-Passing	τ_{max}-Bounded Asynchrony	$\frac{(p-1)\tau_{max}M}{p}$	Reproves (Lian et al. 2015)
Message-Passing	τ_{max}-Bounded Asynchrony	$O(\frac{(p-1)\tau_{max}\sigma}{p})$	New
Message-Passing	*Distributed* Communication-Compression with Error Feedback	$\sqrt{\frac{(2-\gamma)\gamma}{(1-\gamma)^3}}M$	Improves (Alistarh et al. 2018; Stich, Cordonnier, and Jaggi 2018; Karimireddy et al. 2019)
Message-Passing	Synchronous, f Crash or Message-drop Faults	Mf/p	New
Message-Passing	Synchronous, f Crash or Message-drop Faults	$O(\sigma f/p)$	New
Message-Passing	Variance bounded *Elastic Scheduler*	$O(\sigma)$	New

Table 1: Summary of elastic consistency bounds.

(Mf/p), the derivations are slightly different. The framework also allows us to combine consistency relaxations, i.e. consider communication-compression with crashes.

One relative weakness of the above results is that the bounds depend on the gradient second-moment bound. This is not due to elastic consistency itself, but due to the fact that we needed a bound on M to bound the elastic consistency constant for these systems, which is consistent with previous work, e.g. (Sa et al. 2015; Alistarh, De Sa, and Konstantinov 2018; Lian et al. 2015). Next, we show that this limitation, which is common in the literature, can be removed by slightly altering the algorithms.

Practical Application: Elastic Scheduling

So far, we have used elastic consistency to derive bounds for existing models and methods. Next, we ask whether it can inspire *new* distribution schemes. Our target application will be communication scheduling in the context of training deep neural networks (DNNs). More precisely, when performing distributed training of DNNs via back-propagation, it is common to schedule parts of the communication in parallel with the computation. For instance, assuming we train a three-layer network $A \to B \to C$, the gradient of the last layer C will be "ready" to sync before layers A, B, and can be transmitted earlier. Several recent papers, e.g. (Jayarajan et al. 2019; Peng et al. 2019) investigate intricate scheduling mechanisms for leveraging this type of communication-computation overlap. A common feature of these schedulers (Jayarajan et al. 2019; Peng et al. 2019) is ensuring *perfect consistency* at each processor: communication can be reordered w.r.t. computation only if it doing so does not deviate from the sequential execution.

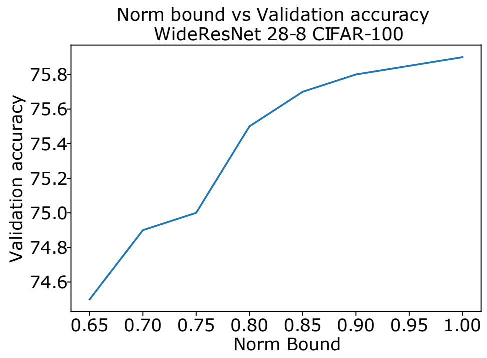
Our analysis suggests that consistency can be relaxed as long as the consistency bound is *small*. Specifically, we will allow processes to start their next forward pass *before* all layers are synchronized, as long as enough gradient norm has been received to ensure a small elastic consistency constant.

The Elastic Scheduler. We will present two relaxed schedulers, a *norm-bounded* and a *variance-bounded* one. Given $0 \leq \beta \leq 1$, the $\beta-$norm-bounded algorithm at a fixed processor i is as follows. Assume that the processor is at iteration t, and has completed its backward pass over the network, obtaining a local gradient $\widetilde{G}(\vec{v}_t^i)$ (computed over the local view $\vec{v}_i^t$). Normally, the processor would need to wait for all parameters to be synchronized, i.e. to receive all the other processors' gradients. However, the *elastic scheduling rule* will allow the processor to start its next forward pass *before* this point, on the inconsistent view, as long as the norm of the received update is at least a β-fraction of *its own gradient* at the step. In this case, the processor speculatively goes forward with its forward-backward step. For both versions, the processor cannot speculate ahead by more than 1 step. In this case, we can easily show that the elastic consistency constant B is upper bounded by $O(M)$, since a processor cannot miss more than one gradient.

The *variance-bounded* version is slightly cleverer: if a processor finishes its forward-backward pass "early" and does not receive all the other processors' gradients within a small timeout, it will proceed to replace the missing gradients *with its own*, and speculatively performs the forward-backward pass based on this inconsistent view. Critically, the processor will "correct" the gradient step retroactively, once it has received the full gradient in the next step. The consistency bound in this case becomes 3σ. The proof of this statement is provided in (Nadiradze et al. 2020).

More generally, variance-bounded scheduler also inspires a way of improving the elastic consistency bounds for crash and message-drop faults and asynchrony with delay τ_{max} : instead of proceeding without the dropped messages, each node can *replace the corresponding missing gradient with its own*. This will allow us to replace the second moment bound M with variance bound $O(\sigma)$.

Implementation. We implement the elastic scheduler on top of the Horovod distributed training framework (Sergeev and Del Balso 2018), which allows us to interface with both Pytorch (Ketkar 2017) and Tensorflow (Abadi et al. 2016). Our implementation is decentralized—as opposed to BytePS (Peng et al. 2019), which has a Parameter Server



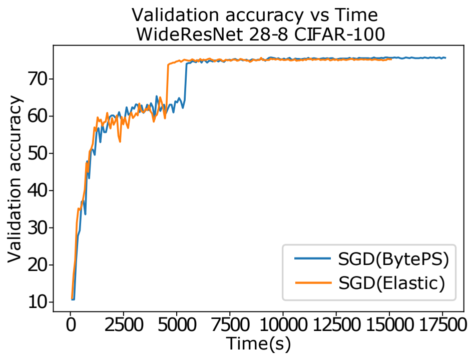


Figure 1: Elastic bound-v-accuracy (left) and accuracy-v-time (center) for WRN28x8 on CIFAR-100.

architecture—but the performance of our baseline implementation is identical to the original BytePS. We conduct our experiments in Pytorch, testing convergence and speedup for residual networks (He et al. 2016) applied to image classification tasks on the CIFAR-10/100 datasets (Krizhevsky and Hinton 2009). Hyperparameter values are standard, and are given in the full version (Nadiradze et al. 2020). Experiments are performed on two AWS EC2 P3.2xlarge instances, each with a V100 GPU, and averaged over 3 trials.

Experiments. We first examine the impact of the elastic consistency bound on accuracy. For this, we execute the norm-bounded variant with different values of $\beta \in [0, 1]$, and examine the top validation accuracy. Figure 1 (left) shows the strong correlation between these two measures for WideResNet28x8 (Zagoruyko and Komodakis 2016) on CIFAR-100, confirming our analysis. Next, we examine the potential for speedup of the elastic scheduler. As frequent re-computation of the L2 norm is expensive, we implement a relaxed variant which tracks the ratio of parameters received (L0 norm). The results for $\beta = 0.8$ are given in Figure 1 (right), showing a $\sim 20\%$ speedup versus the (highly performant) baseline implementation of BytePS, without accuracy loss. The full version of the paper (Nadiradze et al. 2020) contains additional experiments and a full report.

Related Work and Discussion

Distributed machine learning has recently gained significant practical adoption, e.g. (Dean et al. 2012; Ho et al. 2013; Chilimbi et al. 2014; Zhang, Choromanska, and LeCun 2015; Xing et al. 2015; Jayarajan et al. 2019; Peng et al. 2019). Consequently, there has been significant work on introducing and analyzing distributed relaxations of SGD (Recht et al. 2011; Ho et al. 2013; Sa et al. 2015; Lian et al. 2015; Chaturapruek, Duchi, and Ré 2015; Leblond, Pedregosa, and Lacoste-Julien 2017; Alistarh, De Sa, and Konstantinov 2018; Wang and Joshi 2018; Woodworth et al. 2018; Karimireddy et al. 2019; Stich and Karimireddy 2019; Lu, Nash, and De Sa 2020). Due to space constraints, we cover in detail only work that is technically close to ours.

Specifically, De Sa et al. (2015) were the first to consider a unified analysis framework for asynchonous and communication-compressed iterations. Relative to it, our framework improves in three respects: (i) it does not require stringent gradient sparsity assumptions; (ii) it is also able to analyze the case where the updates are not unbiased estimators of the gradient, which allows extensions to error-feedback communication-reduction; and (3) it also tackles convergence for general non-convex objectives. Reference (Lian et al. 2015) presented the first general analysis of asynchronous non-convex SGD , without communication-reduction. Qiao et al. (2019) model asynchrony and communication reduction as perturbations of the SGD iteration, and introduce a metric called "rework cost," which can be subsumed into the elastic consistency bound.

Karimireddy et al. (2019) analyze communication-compression with error feedback, and present a general notion of δ-compressor to model communication-reduced consistency relaxations; later, the framework was extended to include *asynchronous* iterations (Stich and Karimireddy 2019). Every method satisfying the δ-compressor property is elastically-consistent, although the converse is not true. Relative to this work, our framework generalizes in one important practical aspect, as it allows the analysis in *distributed* settings: (Karimireddy et al. 2019; Stich and Karimireddy 2019) assume that the iterations are performed at a single processor, which may compress gradients or view inconsistent information only with respect to *its own* earlier iterations. This extension is non-trivial; tackling this more realistic setting previously required additional analytic assumptions (Alistarh et al. 2018). Sparsified methods would not be competitive with our elastic scheduler, since they only impose sparsity to reduce communication, which would not necessarily improve scheduling.

We have proposed a new and fairly general framework for analyzing inconsistent SGD iterations. Its main advantages are *generality* and *simplicity*. Inspired by this technical condition, we introduce two new, efficient scheduling mechanisms. More generally, we believe that elastic consistency could inspire new distributed algorithms, and be used to derive convergence in a streamlined manner. One key line of extension which we plan to pursue is to study whether elastic consistency can be extended to other first-order distributed optimization methods, or zeroth- or second-order methods.

Acknowledgements

We would like to thank Christopher De Sa for his feedback on an earlier draft of this paper, as well as the anonymous AAAI reviewers for their useful comments. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 805223 ScaleML). Bapi Chatterjee was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 754411 (ISTPlus).

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, volume 16, 265–283.

Aji, A. F.; and Heafield, K. 2017. Sparse Communication for Distributed Gradient Descent. In *EMNLP*, 440–445.

Alistarh, D.; De Sa, C.; and Konstantinov, N. 2018. The convergence of stochastic gradient descent in asynchronous shared memory. In *PODC*, 169–178.

Alistarh, D.; Grubic, D.; Li, J.; Tomioka, R.; and Vojnovic, M. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *NIPS*, 1709–1720.

Alistarh, D.; Hoefler, T.; Johansson, M.; Konstantinov, N.; Khirirat, S.; and Renggli, C. 2018. The convergence of sparsified gradient methods. In *NIPS*, 5977–5987.

Attiya, H.; and Welch, J. 2004. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. J. W. & Sons.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1989. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ.

Bubeck, S.; et al. 2015. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning* 8(3-4): 231–357.

Chaturapruek, S.; Duchi, J. C.; and Ré, C. 2015. Asynchronous stochastic convex optimization: the noise is in the noise and SGD don't care. In *NIPS*, 1531–1539.

Chilimbi, T. M.; Suzue, Y.; Apacible, J.; and Kalyanaraman, K. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *OSDI*, volume 14, 571–582.

Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q. V.; et al. 2012. Large scale distributed deep networks. In *NIPS*, 1223–1231.

Ghadimi, S.; and Lan, G. 2013. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization* 23(4): 2341–2368.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Ho, Q.; Cipar, J.; Cui, H.; Lee, S.; Kim, J. K.; Gibbons, P. B.; Gibson, G. A.; Ganger, G.; and Xing, E. P. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *NIPS*, 1223–1231.

Jayarajan, A.; Wei, J.; Gibson, G.; Fedorova, A.; and Pekhimenko, G. 2019. Priority-based parameter propagation for distributed DNN training. *arXiv preprint arXiv:1905.03960* .

Karimireddy, S. P.; Rebjock, Q.; Stich, S. U.; and Jaggi, M. 2019. Error Feedback Fixes SignSGD and other Gradient Compression Schemes. In *ICML*, 3252–3261.

Ketkar, N. 2017. Introduction to PyTorch. In *Deep Learning with Python*, 195–208. Springer.

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Technical report, University of Toronto.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1097–1105.

Leblond, R.; Pedregosa, F.; and Lacoste-Julien, S. 2017. ASAGA: Asynchronous Parallel SAGA. In *AISTATS*, 46–54.

Li, M.; Andersen, D. G.; Park, J. W.; Smola, A. J.; Ahmed, A.; Josifovski, V.; Long, J.; Shekita, E. J.; and Su, B.-Y. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*, volume 1, 3.

Lian, X.; Huang, Y.; Li, Y.; and Liu, J. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *NIPS*, 2737–2745.

Lian, X.; Zhang, C.; Zhang, H.; Hsieh, C.; Zhang, W.; and Liu, J. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. In *NIPS*, 5330–5340.

Lin, T.; Stich, S. U.; Patel, K. K.; and Jaggi, M. 2018a. Don't Use Large Mini-Batches, Use Local SGD. *arXiv preprint arXiv:1808.07217* .

Lin, Y.; Han, S.; Mao, H.; Wang, Y.; and Dally, B. 2018b. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *ICLR, Poster*.

Lu, Y.; Li, Z.; and Sa, C. D. 2020. Towards Optimal Convergence Rate in Decentralized Stochastic Training. *ArXiv* abs/2006.08085.

Lu, Y.; Nash, J.; and De Sa, C. 2020. MixML: A Unified Analysis of Weakly Consistent Parallel Learning. *arXiv preprint arXiv:2005.06706* .

Nadiradze, G.; Markov, I.; Chatterjee, B.; Kungurtsev, V.; and Alistarh, D. 2020. Elastic Consistency: A General Consistency Model for Distributed Stochastic Gradient Descent. *arXiv preprint arXiv:2001.05918* .

Nguyen, L. M.; Nguyen, P. H.; van Dijk, M.; Richtárik, P.; Scheinberg, K.; and Takác, M. 2018. SGD and Hogwild! Convergence Without the Bounded Gradients Assumption. In *ICML*, 3747–3755.

Peng, Y.; Zhu, Y.; Chen, Y.; Bao, Y.; Yi, B.; Lan, C.; Wu, C.; and Guo, C. 2019. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 16–29.

Qiao, A.; Aragam, B.; Zhang, B.; and Xing, E. P. 2019. Fault Tolerance in Iterative-Convergent Machine Learning. In *ICML*, 5220–5230.

Recht, B.; Re, C.; Wright, S.; and Niu, F. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 693–701.

Robbins, H.; and Monro, S. 1951. A stochastic approximation method. *The annals of mathematical statistics* 400–407.

Sa, C. D.; Zhang, C.; Olukotun, K.; and Ré, C. 2015. Taming the Wild: A Unified Analysis of Hogwild-Style Algorithms. In *NIPS*, 2674–2682.

Seide, F.; Fu, H.; Droppo, J.; Li, G.; and Yu, D. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *INTERSPEECH*, 1058–1062.

Sergeev, A.; and Del Balso, M. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* .

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529(7587): 484–489.

Stich, S. U. 2018. Local SGD converges fast and communicates little. *arXiv preprint arXiv:1805.09767* .

Stich, S. U.; Cordonnier, J.; and Jaggi, M. 2018. Sparsified SGD with Memory. In *NIPS*, 4452–4463.

Stich, S. U.; and Karimireddy, S. P. 2019. The error-feedback framework: Better rates for SGD with delayed gradients and compressed communication. *arXiv preprint arXiv:1909.05350* .

Strom, N. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Wang, J.; and Joshi, G. 2018. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *preprint arXiv:1808.07576* .

Wangni, J.; Wang, J.; Liu, J.; and Zhang, T. 2018. Gradient sparsification for communication-efficient distributed optimization. In *NIPS*, 1306–1316.

Woodworth, B. E.; Wang, J.; Smith, A.; McMahan, B.; and Srebro, N. 2018. Graph oracle models, lower bounds, and gaps for parallel stochastic optimization. In *Advances in neural information processing systems*, 8496–8506.

Xing, E. P.; Ho, Q.; Dai, W.; Kim, J. K.; Wei, J.; Lee, S.; Zheng, X.; Xie, P.; Kumar, A.; and Yu, Y. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* 1(2): 49–67.

Zagoruyko, S.; and Komodakis, N. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146* .

Zhang, S.; Choromanska, A. E.; and LeCun, Y. 2015. Deep learning with elastic averaging SGD. In *Advances in neural information processing systems*, 685–693.