
Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning

H. Brendan McMahan
Google, Inc.
Seattle, WA
mcmahan@google.com

Matthew Streeter
Duolingo, Inc.*
Pittsburgh, PA
matt@duolingo.com

Abstract

We analyze new online gradient descent algorithms for distributed systems with large delays between gradient computations and the corresponding updates. Using insights from adaptive gradient methods, we develop algorithms that adapt not only to the sequence of gradients, but also to the precise update delays that occur. We first give an impractical algorithm that achieves a regret bound that precisely quantifies the impact of the delays. We then analyze `AdaptiveRevision`, an algorithm that is efficiently implementable and achieves comparable guarantees. The key algorithmic technique is appropriately and efficiently revising the learning rate used for previous gradient steps. Experimental results show when the delays grow large (1000 updates or more), our new algorithms perform significantly better than standard adaptive gradient methods.

1 Introduction

Stochastic and online gradient descent methods have proved to be extremely useful for solving large-scale machine learning problems [1, 2, 3, 4]. Recently, there has been much work on extending these algorithms to parallel and distributed systems [5, 6, 7, 8, 9]. In particular, Recht et al. [10] and Duchi et al. [11] have shown that standard stochastic algorithms essentially “work” even when updates are applied asynchronously by many threads. Our experiments confirm this for moderate amounts of parallelism (say 100 threads), but show that for large amounts of parallelism (as in a distributed system, with say 1000 threads spread over many machines), performance can degrade significantly. To address this, we develop new algorithms that adapt to both the data and the amount of parallelism.

Adaptive gradient (AdaGrad) methods [12, 13] have proved remarkably effective for real-world problems, particularly on sparse data (for example, text classification with bag-of-words features). The key idea behind these algorithms is to prove a general regret bound in terms of an arbitrary sequence of non-increasing learning rates and the full sequence of gradients, and then to define an adaptive method for choosing the learning rates as a function of the gradients seen so far, so as to minimize the final bound when the learning rates are plugged in. We extend this idea to the parallel setting, by developing a general regret bound that depends on both the gradients and the exact update delays that occur (rather than say an upper bound on delays). We then present `AdaptiveRevision`, an algorithm for choosing learning rates and efficiently revising past learning-rate choices that strives to minimize this bound. In addition to providing an adaptive regret bound (which recovers the standard AdaGrad bound in the case of no delays), we demonstrate excellent empirical performance.

Problem Setting and Notation We consider a computation model where one or more computation units (a thread in a parallel implementation or a full machine in a distributed system) store and

*Work performed while at Google, Inc.

update the model $x \in \mathbb{R}^n$, and another larger set of computation units perform feature extraction and prediction. We call the first type the `Updaters` (since they apply the gradient updates) and the second type the `Readers` (since they read coefficients stored by the `Updaters`). Because the `Readers` and `Updaters` may reside on different machines, perhaps located in different parts of the world, communication between them is not instantaneous. Thus, when making a prediction, a `Reader` will generally be using a coefficient vector that is somewhat stale relative to the most recent version being served by the `Updaters`.

As one application of this model, consider the problem of predicting click-through rates for sponsored search ads using a generalized linear model [14, 15]. While the coefficient vector may be stored and updated centrally, predictions must be available in milliseconds in any part of the world. This leads naturally to an architecture in which a large number of `Readers` maintain local copies of the coefficient vector, sending updates to the `Updaters` and periodically requesting fresh coefficients from them. As another application, this model encompasses the `Parameter Server/ Model Replica` split of `Downpour SGD` [16].

Our bounds apply to general online convex optimization [4], which encompasses the problem of predicting with a generalized linear model (models where the prediction is a function of $a_t \cdot x_t$, where a_t is a feature vector and x_t are model coefficients). We analyze the algorithm on a sequence of $\tau = 1, \dots, T$ rounds; for the moment, we index rounds based on when each prediction is made. On each round, a convex loss function f_τ arrives at a `Reader`, the `Reader` predicts with $x_\tau \in \mathbb{R}^n$ and incurs loss $f_\tau(x_\tau)$. The `Reader` then computes a subgradient $g_\tau \in \partial f_\tau(x_\tau)$. For each coordinate i where $g_{\tau,i}$ is nonzero, the `Reader` sends an update to the `Updater(s)` for those coefficients. We are particularly concerned with sparse data, where n is very large, say $10^6 - 10^9$, but any particular training example has only a small fraction of the features $a_{t,i}$ that take non-zero values.

The regret against a comparator $x^* \in \mathbb{R}^n$ is

$$\text{Regret}(x^*) \equiv \sum_{\tau=1}^T f_\tau(x_\tau) - f_\tau(x^*). \quad (1)$$

Our primary theoretical contributions are upper bounds on the regret of our algorithms.

We assume a fully asynchronous model, where the delays in the read requests and update requests can be different for different coefficients even for the same training event. This leads to a combinatorial explosion in potential interleavings of these operations, making fine-grained adaptive analysis quite difficult. Our primary technique for addressing this will be the linearization of loss functions, a standard tool in online convex optimization which takes on increased importance in the parallel setting. An immediate consequence of convexity is that given a general convex loss function f_τ , with $g_\tau \in \partial f_\tau(x_\tau)$, for any x^* , we have $f_\tau(x_\tau) - f_\tau(x^*) \leq g_\tau \cdot (x_\tau - x^*)$. One of the key observations of Zinkevich [1] is that by plugging this inequality into (1), we see that if we can guarantee low regret against linear functions, we can provide the same guarantees against arbitrary convex functions. Further, expanding the dot products and re-arranging the sum, we can write

$$\text{Regret}(x^*) \equiv \sum_{i=1}^n \text{Regret}_i(x_i^*) \quad \text{where} \quad \text{Regret}_i(x_i^*) = \sum_{\tau=1}^T g_{\tau,i}(x_{\tau,i} - x_i^*). \quad (2)$$

If we consider algorithms where the updates are also coordinate decomposable (that is, the update to coordinate i can be applied independently of the update of coordinate j), then we can bound $\text{Regret}(x^*)$ by proving a per-coordinate bound for linear functions and then summing across coordinates. In fact, our computation architecture already assumes a coordinate decomposable algorithm since this lets us avoid synchronizing the `Updates`, and so in addition to leading to more efficient algorithms, this approach will greatly simplify the analysis. The proofs of Duchi et al. [11] take a similar approach.

Bounding per-coordinate regret Given the above, we will design and analyze asynchronous one-dimensional algorithms which can be run independently on each coordinate of the true learning problem. For each coordinate, each `Read` and `Update` is assumed to be an atomic operation. It will be critical to adopt an indexing scheme different than the prediction-based indexing τ used above. The net result will be bounding the sum of (2), but we will actually re-order the sum to make the analysis easier. Critically, this ordering could be *different* for different coordinates, and

so considering one coordinate at a time simplifies the analysis considerably.¹ We index time by the order of the `Updates`, so the index t is such that g_t is the gradient associated with the t th update applied and x_t is the value of the coefficient immediately *before* the update for g_t is applied. Then, the Online Gradient Descent (OGD) update consists of exactly the assumed-atomic operation

$$x_{t+1} = x_t - \eta_t g_t, \quad (3)$$

where η_t is a learning-rate. Let $r(t) \in \{1, \dots, t\}$ be the index such that $x_{r(t)}$ was the value of the coefficient used by the `Reader` to compute g_t (and to predict on the corresponding example). That is, update $r(t) - 1$ completed before the `Read` for g_t , but update $r(t)$ completed after. Thus, our loss (for coordinate i) is $g_t x_{r(t)}$, and we desire a bound on

$$\text{Regret}_i(x^*) = \sum_{t=1}^T g_t (x_{r(t)} - x^*).$$

Main result and related work We say an update s is outstanding at time t if the `Read` for Update s occurs before update t , but the Update occurs after: precisely, s is outstanding at t if $r(s) \leq t < s$. We let $\mathcal{F}_t \equiv \{s \mid r(s) \leq t < s\}$ be the set of updates outstanding at time t . We call the sum of these gradients the *forward gradient sum*, $g_t^{\text{fwd}} \equiv \sum_{s \in \mathcal{F}_t} g_s$. Then, ignoring constant factors and terms independent of T , we show that `AdaptiveRevision` has a per-coordinate bound of the form

$$\text{Regret} \leq \sqrt{\sum_{t=1}^T g_t^2 + g_t^{\text{fwd}}}. \quad (4)$$

Theorem 3 gives the precise result as well as the n -dimensional version. Observe that without any delays, $g_t^{\text{fwd}} = 0$, and we arrive at the standard AdaGrad-style bound. To prove the bound for `AdaptiveRevision`, we require an additional `InOrder` assumption on the delays, namely that for any indexes s_1 and s_2 , if $r(s_1) < r(s_2)$ then $s_1 < s_2$. This assumption should be approximately satisfied most of the time for realistic delay distributions, and even under a more pathological delay distributions (delays uniform on $\{0, \dots, m\}$ rather than more tightly grouped around a mean delay), our experiments show excellent performance for `AdaptiveRevision`.

The key challenge is that unlike in the AdaGrad case, conceptually we need to know gradients that have not yet been computed in order to calculate the optimal learning rate. We surmount this by using an algorithm that not only chooses learning rates adaptively, but also *revises* previous gradient steps. Critically, these revisions require only moderate additional storage and network cost: we store a sum of gradients along with each coefficient, and for each `Read`, we remember the value of this gradient sum at the time of the `Read` until the corresponding Update occurs. This later storage can essentially be implemented on the network, if the gradient sum is sent from the `Updater` to the `Reader` and back again, ensuring it is available exactly when needed. This is the approach taken in the pseudocode of Algorithm 1.

Against a true adversary and a maximum delay of m , in general we cannot do better than just training synchronously on a single machine using a $1/m$ fraction of the data. Our results surmount this issue by producing strongly data-dependent bounds: we do not expect fully adversarial gradients and delays in practice, and so on real data the bound we prove still gives interesting results. In fact, we can essentially recover the guarantees for `AsyncAdaGrad` from Duchi et al. [11], which rely on stochastic assumptions on the sparsity of the data, by applying the same assumptions to our bound. To simplify the comparison, WLOG we consider a 1-dimensional problem where $\|x^*\|_2 = 1$, $\|g_t\|_2 \leq 1$, and we have the stochastic assumption that each g_t is exactly 0 independently with probability p (implying $M_j = 1$, $M = 1$, and $M^2 = p$ in their notation). Then, simple calculations (given in Appendix B) show our bound for `AdaptiveRevision` implies a bound on expected regret of $\mathcal{O}(\sqrt{(1+mp)pT})$ without knowledge of p or m , ignoring terms independent of T .² `AsyncAdaGrad` achieves the same bound, but critically this requires knowledge of both p and

¹Our analysis could be extended to non-coordinate-decomposable algorithms, but then the full gradient update across all coordinates would need to be atomic. This case is less interesting due to the computational overhead.

²In the analysis, we choose the parameter G_0 based on an upper bound m on the delay, but this only impacts an additive term independent of T .

m in advance in order to tune the learning rate appropriately (in the general n -dimensional case, this would mean knowing not just one parameter p , but a separate sparsity parameter p_j for each coordinate, and then using an appropriate per-coordinate scaling of the learning rate depending on this); without such knowledge, AsyncAdaGrad only obtains the much worse bound $\mathcal{O}((1+mp)\sqrt{pT})$. AdaptiveRevision will also provide significantly better guarantees if most of the delays are much less than the maximum, or if the data is only approximately sparse (e.g., many $g_t = 10^{-6}$ rather than exactly 0). The above analysis also makes a worst-case assumption on the $g_t g_t^{\text{fwd}}$ terms, but in practice many gradients in g_t^{fwd} are likely to have opposite signs and cancel out, a fact our algorithm and bounds can exploit.

2 Algorithms and Analysis

We first introduce some additional definitions. Let $o(t) \equiv \max \mathcal{F}_t \cup \{t\}$, the index of the highest update outstanding at time t , or t itself if nothing is outstanding. The sets \mathcal{F}_t fully specify the delay pattern. In light of (4), we further define $G_t^{\text{fwd}} \equiv g_t^2 + 2g_t g_t^{\text{fwd}}$. We also define \mathcal{B}_t , the set of updates applied while update t was outstanding. Under our notation, this set is easily defined as $\mathcal{B}_t = \{r(t), \dots, t-1\}$ (or the empty set if $r(t) = t$, so in particular $\mathcal{B}_1 = \emptyset$). We will also frequently use the *backward gradient sum*, $g_t^{\text{bck}} \equiv \sum_{s=r(t)}^{t-1} g_s$. These vectors most often appear in the products $G_t^{\text{bck}} \equiv g_t^2 + 2g_t g_t^{\text{bck}}$. Figure 3 in Appendix A shows a variety of delay patterns and gives a visual representation of the sums G_t^{fwd} and G_t^{bck} . We say the delay is (upper) bounded by m if $t - r(t) \leq m$ for all t , which implies $|\mathcal{F}_t| \leq m$ and $|\mathcal{B}_t| \leq m$. Note that if $m = 0$ then $r(t) = t$. We use the compressed summation notation $c_{1:t} \equiv \sum_{s=1}^t c_s$ for vectors, scalars, and functions.

Our analysis builds on the following simple but fundamental result (Appendix C contains all proofs and lemmas omitted here).

Lemma 1. *Given any non-increasing learning-rate schedule η_t , define σ_t where $\sigma_1 = 1/\eta_1$ and $\sigma_t = 1/\eta_t - 1/\eta_{t-1}$ for $t > 1$, so $\eta_t = 1/\sigma_{1:t}$. Then, for any delay schedule, unprojected online gradient descent achieves, for any $x^* \in \mathbb{R}$,*

$$\text{Regret}(x^*) \leq \frac{(2R_T)^2}{2\eta_T} + \frac{1}{2} \sum_{t=1}^T \eta_t G_t^{\text{fwd}} \quad \text{where} \quad (2R_T)^2 \equiv \sum_{t=1}^T \frac{\sigma_t}{\sigma_{1:T}} |x^* - x_t|^2.$$

Proof. Given how we have indexed time, we can consider the regret of a hypothetical online gradient descent algorithm that plays x_t and then observes g_t , since this corresponds exactly to the update (3). We can then bound regret for this hypothetical setting using a simple modification to standard bound for OGD [1],

$$\sum_{t=1}^T g_t \cdot x_t - g_{1:T} \cdot x^* \leq \sum_{t=1}^T \frac{\sigma_t}{2} |x^* - x_t|^2 + \frac{1}{2} \sum_{t=1}^T \eta_t g_t^2.$$

The actual algorithm used $x_{r(t)}$ to predict on g_t , not x_t , so we can bound its Regret by

$$\text{Regret} \leq \frac{(2R_T)^2}{2\eta_T} + \frac{1}{2} \sum_{t=1}^T \eta_t g_t^2 + \sum_{t=1}^T g_t (x_{r(t)} - x_t). \quad (5)$$

Recalling $x_{t+1} = x_t - \eta_t g_t$, observe that $x_{r(t)} - x_t = \sum_{s=r(t)}^{t-1} \eta_s g_s = \sum_{s \in \mathcal{B}_t} \eta_s g_s$ and so

$$\sum_{t=1}^T g_t (x_{r(t)} - x_t) = \sum_{t=1}^T g_t \sum_{s \in \mathcal{B}_t} \eta_s g_s = \sum_{s=1}^T \eta_s g_s \sum_{t \in \mathcal{F}_s} g_t = \sum_{s=1}^T \eta_s g_s g_s^{\text{fwd}},$$

using Lemma 4(E) from the Appendix to re-order the sum. Plugging into (5) completes the proof. \square

For projected online gradient descent, by projecting onto a feasible set of radius R and assuming x^* is in this set, we immediately get $|x^* - x_t| \leq 2R$. Without projecting, we get a more adaptive bound which depends on the weighted quadratic mean $2R_T$. Though less standard, we choose to

analyze the unprojected variant of the algorithm for two reasons. First, our analysis rests heavily on the ability to represent points played by our algorithms exactly as weighted sums of past gradients, a property not preserved when projection is invoked. More importantly, we know of no experiments on real-world prediction problems (where any $x \in \mathbb{R}^n$ is a valid model) where the projected algorithm actually performs better. In our experience, once the learning-rate schedule is tuned appropriately, the resulting R_T values will not be more than a constant factor of $\|x^*\|$. This makes intuitive sense in the stochastic case, where it is known that averages of the x_t should in fact converge to x^* .³ For learning rate tuning we assume we know in advance a constant \tilde{R} such that $R_T \leq \tilde{R}$; again, in practice this is roughly equivalent to assuming we know $\|x^*\|$ in advance in order to choose the feasible set.

Our first algorithm, `HypFwd` (for Hypothetical-Forward), assumes it has knowledge of all the gradients, so it can optimize its learning rates to minimize the above bound. If there are no delays, that is, $g_t^{\text{fwd}} = 0$ for all t , then this immediately gives rise to a standard AdaGrad-style online gradient descent method. If there are delays, the G_t^{fwd} terms could be large, implying the optimal learning rates should be smaller. Unfortunately, it is impossible for a real algorithm to know g_t^{fwd} when η_t is chosen. To work toward a practical algorithm, we introduce `HypBack`, which achieves similar guarantees (but is still impractical). Finally, we introduce `AdaptiveRevision`, which plays points very similar to `HypBack`, but can be implemented efficiently. Since we will need non-increasing learning rates, it will be useful to define $\tilde{G}_{1:t}^{\text{bck}} \equiv \max_{s \leq t} G_{1:s}^{\text{bck}}$ and $\tilde{G}_{1:t}^{\text{fwd}} \equiv \max_{s \leq t} G_{1:s}^{\text{fwd}}$. In practice, we expect $\tilde{G}_{1:T}^{\text{bck}}$ to be close to $G_{1:T}^{\text{bck}}$. We assume WLOG that $G_1^{\text{fwd}} > 0$, which at worst adds a negligible additive constant to our regret.

Algorithm HypFwd This algorithm “cheats” by using the forward sum g_t^{fwd} to choose η_t ,

$$\eta_t = \frac{\alpha}{\sqrt{\tilde{G}_{1:t}^{\text{fwd}}}} \quad (6)$$

for an appropriate scaling parameter $\alpha > 0$. Then, Lemma 1 combined with the technical inequality of Corollary 10 (given in Appendix D) gives

$$\text{Regret} \leq 2\sqrt{2}\tilde{R}\sqrt{\tilde{G}_{1:T}^{\text{fwd}}}. \quad (7)$$

when we take $\alpha = \sqrt{2}\tilde{R}$ (recalling $\tilde{R} \geq R_T$). If there are no delays, this bound reduces to the standard bound $2\sqrt{2}\tilde{R}\sqrt{\sum_{t=1}^T g_t^2}$. With delays, however, this is a hypothetical algorithm, because it is generally not possible to know g_t^{fwd} when update t is applied. However, we can implement this algorithm efficiently in a single-machine simulation, and it performs very well (see Section 3). Thus, our goal is to find an efficiently implementable algorithm that achieves comparable results in practice and also matches this regret bound.

Algorithm HypBack The next step in the analysis is to show that a second hypothetical algorithm, `HypBack`, approximates the regret bound of (7). This algorithm plays

$$\hat{x}_{t+1} = - \sum_{s=1}^t \hat{\eta}_s g_s \quad \text{where} \quad \hat{\eta}_t = \frac{\alpha}{\sqrt{\tilde{G}_{1:o(t)}^{\text{bck}} + G_0}} \quad (8)$$

is a learning rate with parameters α and G_0 . This is a hypothetical algorithm, since we also can't (efficiently) know $G_{1:o(t)}^{\text{bck}}$ on round t . We prove the following guarantee:

Lemma 2. *Suppose delays bounded by m and $|g_t| \leq L$. Then when the `InOrder` property holds, `HypBack` with $\alpha = \sqrt{2}\tilde{R}$ and $G_0 = m^2 L^2$ has*

$$\text{Regret} \leq 2\sqrt{2}\tilde{R}\sqrt{\tilde{G}_{1:T}^{\text{fwd}}} + 2\tilde{R}mL.$$

³For example, the arguments of Nemirovski et al. [17, Sec 2.2] hold for unprojected gradient descent.

Algorithm 1 Algorithm AdaptiveRevision

Procedure Read(loss function f):

Read (x_i, \bar{g}_i) from the Updaters for all necessary coordinates
 Calculate a subgradient $g \in \partial f(x)$
for each coordinate i with a non-zero gradient **do**
 Send an update tuple $(g \leftarrow g_i, \bar{g}^{\text{old}} \leftarrow \bar{g}_i)$ to the Updater for coordinate i

Procedure Update(g, \bar{g}^{old}): *The Updater initializes state $(\bar{g} \leftarrow 0, z \leftarrow 1, z' \leftarrow 1, x \leftarrow 0)$ per coordinate.*

Do the following atomically:

$g^{\text{bck}} \leftarrow \bar{g} - \bar{g}^{\text{old}}$ $\eta^{\text{old}} \leftarrow \frac{\alpha}{\sqrt{z'}}$ $z \leftarrow z + g^2 + 2g \cdot g^{\text{bck}}; z' \leftarrow \max(z, z')$ $\eta \leftarrow \frac{\alpha}{\sqrt{z'}}$ $x \leftarrow x - \eta g$ $x \leftarrow x + (\eta^{\text{old}} - \eta)g^{\text{bck}}$ $\bar{g} \leftarrow \bar{g} + g$	<i>For analysis, assign index t to the current update.</i> <i>Invariant: effective η for all g^{bck}.</i> <i>Maintain $z = G_{1:t}^{\text{bck}}$ and $z' = \tilde{G}_{1:t}^{\text{bck}}$, to enforce non-increasing η.</i> <i>New learning rate.</i> <i>The main gradient-descent update.</i> <i>Apply adaptive revision of some previous steps.</i> <i>Maintain $\bar{g} = g_{1:t}$.</i>
--	---

Algorithm AdaptiveRevision Now that we have shown that HypBack is effective, we can describe AdaptiveRevision, which efficiently approximates HypBack. We then analyze this new algorithm by showing its loss is close to the loss of HypBack. Pseudo-code for the algorithm as implemented for the experiments is given in Algorithm 1; we now give an equivalent expression for the algorithm under the InOrder assumption. Let β_t be the learning rate based on $\tilde{G}_{1:t}^{\text{bck}}$, $\beta_t = \alpha / \sqrt{\tilde{G}_{1:t}^{\text{bck}} + G_0}$. Then, AdaptiveRevision plays the points

$$x_{t+1} = \sum_{s=1}^t \eta_s^t g_s \quad \text{where} \quad \eta_s^t = \beta_{\min(t, o(s))}. \quad (9)$$

When $s \ll t$ then we will usually have $\min(t, o(s)) = o(s)$, and so we see that $\eta_s^t = \beta_{o(s)} = \hat{\eta}_s$, and so the effective learning rate applied to gradient g_s is the same one HypBack would have used (namely $\hat{\eta}_s$); thus, the only difference between AdaptiveRevision and HypBack is on the leading edge, where $o(s) > t$. See Figure 4 in Appendix A for an example. When InOrder holds, Lemma 6 (in Appendix C) shows Algorithm 1 plays the points specified by (9).

Given Lemma 2, it is sufficient to show that the difference between the loss of HypBack and the loss of AdaptiveRevision is small. Lemma 8 (in the appendix) accomplishes this, showing that under the InOrder assumption and with $G_0 = m^2 L^2$ the difference in loss is at most $2\alpha Lm$ (a quantity independent of T). Our main theorem is then a direct consequence of Lemma 2 and Lemma 8:

Theorem 3. *Under an InOrder delay pattern with a maximum delay of at most m , the AdaptiveRevision algorithm guarantees $\text{Regret} \leq 2\sqrt{2}\tilde{R}\sqrt{\tilde{G}_{1:T}^{\text{fwd}}} + (2\sqrt{2} + 2)\tilde{R}mL$ when we take $G_0 = m^2 L^2$ and $\alpha = \sqrt{2}\tilde{R}$. Applied on a per-coordinate basis to an n -dimensional problem, we have*

$$\text{Regret} \leq 2\sqrt{2}\tilde{R} \sum_{i=1}^n \sqrt{\sum_{t=1}^T \left(g_{t,i}^2 + 2 \sum_{s \in \mathcal{F}_{t,i}} g_{s,i} g_{s,i} \right)} + n(2\sqrt{2} + 2)\tilde{R}mL.$$

We note the n -dimensional guarantee is at most $\mathcal{O}(n\tilde{R}L\sqrt{Tm})$, which matches the lower bound for the feasible set $[-R, R]^n$ and $g_t \in [-L, L]^n$ up to the difference between \tilde{R} and R (see, for example, Langford et al. [18]).⁴ Our point, of course, is that for real data our bound will often be much much better.

⁴To compare to regret bounds stated in terms of L_2 bounds on the feasible set and the gradients, note for $g_t \in [-L, L]^n$ we have $\|g_t\|_2 \leq \sqrt{n}L$, and similarly for $x \in [-R, R]^n$ we have $\|x\|_2 \leq \sqrt{n}R$, so the dependence on n is a necessary consequence of using these norms, which are quite natural for sparse problems.

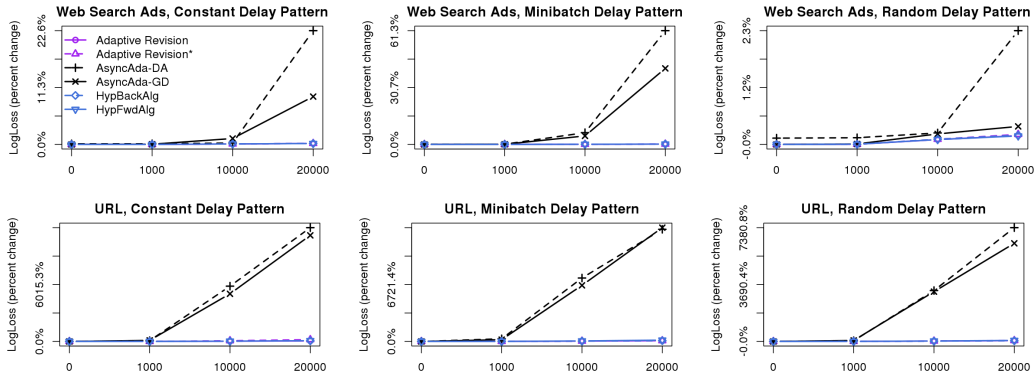


Figure 1: Accuracy as a function of update delays, with learning rate scale factors optimized for each algorithm and dataset for the zero delay case. The x -axis is non-linear. The results are qualitatively similar across the plots, but note the differences in the y -axis ranges. In particular, the random delay pattern appears to hurt performance significantly less than either the minibatch or constant delay patterns.

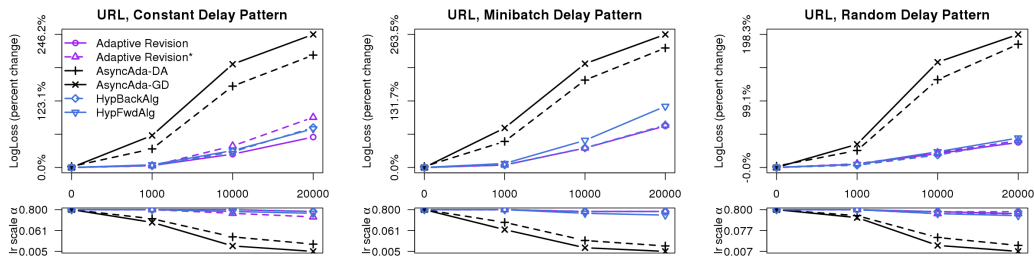


Figure 2: Accuracy as a function of update delays, with learning rate scale factors optimized as a function of the delay. The lower plot in each group shows the best learning rate scale α on a log-scale.

3 Experiments

We study the performance of both hypothetical algorithms and AdaptiveRevision on two real-world medium-sized datasets. We simulate the update delays using an update queue, which allows us to implement the hypothetical algorithms and also lets us precisely control both the exact delays as well as the delay pattern. We compare to the dual-averaging AsyncAdaGrad algorithm of Duchi et al. [11] (AsyncAda-DA in the figures), as well as asynchronous AdaGrad gradient descent (AsyncAda-GD), which can be thought of as AdaptiveRevision with all g^{bck} set to zero and no revision step. As analyzed, AdaptiveRevision stores an extra variable (z') in order to enforce a non-increasing learning rate. In practice, we found this had a negligible impact; in the plots above, AdaptiveRevision* denotes the algorithm without this check. With this improvement AdaptiveRevision stores three numbers per coefficient, versus the two stored by AsyncAda-grad DA or GD.

We consider three different delay patterns, which we parameterize by D , the *average* delay; this yields a more fair comparison across the delay patterns than using the the maximum delay m . We consider: 1) constant delays, where all updates (except at the beginning and the end of the dataset) have a delay of exactly D (e.g., rows (B) and (C) in Figure 3 in the Appendix); 2) A minibatch delay pattern⁵, where $2D + 1$ Reads occur, followed by $2D + 1$ Updates; and 3) a random delay pattern, where the delays are chosen uniformly from the set $\{0, \dots, 2D\}$, so again the mean delay is D . The first two patterns satisfy InOrder, but the third does not.

⁵It is straightforward to show that under this delay pattern, when we do not enforcing non-increasing learning rates, AdaptiveRevision and HypBack are in fact equivalent to standard AdaGrad run on the minibatches (that is, with one update per minibatch using the combined minibatch gradient sum).

We evaluate on two datasets. The first is a web search advertising dataset from a large search engine. The dataset consists of about 3.1×10^6 training examples with a large number of sparse anonymized features based on the ad and query text. Each example is labeled $\{-1, 1\}$ based on whether or not the person doing the query clicked on the ad. The second is a shuffled version of the malicious URL dataset as described by Ma et al. [19] (2.4×10^6 examples, 3.2×10^6 features).⁶ For each of these datasets we trained a logistic regression model, and evaluated using the logistic loss (LogLoss). That is, for an example with feature vector $a \in \mathbb{R}^n$ and label $y \in \{-1, 1\}$, the loss is given by $\ell(x, (a, y)) = \log(1 + \exp(-ya \cdot x))$. Following the spirit of our regret bounds, we evaluate the models online, making a single pass over the data and computing accuracy metrics on the predictions made by the model immediately before it trained on each example (i.e., progressive validation). To avoid possible transient behavior, we only report metrics for the predictions on the second half of each dataset, though this choice does not change the results significantly.

The exact parametrization of the learning rate schedule is particularly important with delayed updates. We follow the common practice of taking learning rates of the form $\eta_t = \alpha / \sqrt{S_t + 1}$, where S_t is the appropriate learning rate statistic for the given algorithm, e.g., $\bar{G}_{1:o(t)}^{\text{bck}}$ for HypBack or $\sum_{s=1}^t g_s^2$ for vanilla AdaGrad. In the analysis, we use $G_0 = m^2 L^2$ rather than $G_0 = 1$; we believe $G_0 = 1$ will generally be a better choice in practice, though we did not optimize this choice.⁷ When we optimize α , we choose the best setting from a grid $\{\alpha_0(1.25)^i \mid i \in \mathbb{N}\}$, where α_0 is an initial guess for each dataset.

All figures give the average delay D on the x -axis. For Figure 1, for each dataset and algorithm, we optimized α in the zero delay ($D = m = 0$) case, and fixed this parameter as the average delay D increases. This leads to very bad performance for standard AdaGrad DA and GD as D gets large. In Figure 2, we optimized α individually for each delay level; we plot the accuracy as before, with the lower plot showing the optimal learning rate scaling α on a log-scale. The optimal learning rate scaling for GD and DA decrease by *two orders of magnitude* as the delays increase. However, even with this tuning they do not obtain the performance of AdaptiveRevision. The performance of AdaptiveRevision (and HypBack and HypFwd) is slightly improved by lowering the learning rate as delays increase, but the effect is comparatively very minor. As anticipated, the performance for AdaptiveRevision, HypBack, and HypFwd are closely grouped.

AdaptiveRevision’s delay tolerance can lead to enormous speedups in practice. For example, the leftmost plot of Figure 2 shows that AdaptiveRevision achieves better accuracy with an update delay of 10,000 than AsyncAda-DA achieves with a delay of 1000. Because update delays are proportional to the number of Readers, this means that AdaptiveRevision can be used to train a model an order of magnitude faster than AsyncAda-DA, with no reduction in accuracy. This allows for much faster iteration when data sets are large and parallelism is cheap, which is the case in important real-world problems such as ad click-through rate prediction [14].

4 Conclusions and Future Work

We have demonstrated that adaptive tuning and revision of per-coordinate learning rates for distributed gradient descent can significantly improve accuracy as the update delays become large. The key algorithmic technique is maintaining a sum of gradients, which allows the adjustment of all learning rates for gradient updates that occurred between the current Update and its Read. The analysis method is novel, but is also somewhat indirect; an interesting open question is finding a general analysis framework for algorithms of this style. Ideally such an analysis would also remove the technical need for the InOrder assumption, and also allow for the analysis of AdaptiveRevision variants of OGD with Projection and Dual Averaging.

⁶We also ran experiments on the `rcv1.binary` training dataset (0.6×10^6 examples, 0.05×10^6 features) from Chang and Lin [20]; results were qualitatively very similar to those for the URL dataset.

⁷The main purpose of choosing a larger G_0 in the theorems was to make the performance of HypBack and AdaptiveRevision provably close to that of HypFwd, even in the worst case. On real data, the performance of the algorithms will typically be close even with $G_0 = 1$.

References

- [1] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.
- [2] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML 2004*, 2004.
- [3] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*. 2008.
- [4] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 2012.
- [5] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *J. Mach. Learn. Res.*, 13(1), January 2012.
- [6] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *arXiv:1212.0873 [math.OC]*, 2012. URL <http://arxiv.org/abs/1212.0873>.
- [7] Martin Takáč, Avleen Bijral, Peter Richtárik, and Nati Srebro. Mini-batch primal and dual methods for SVMs. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.
- [8] Daniel Hsu, Nikos Karampatziakis, John Langford, and Alexander J. Smola. *Scaling Up Machine Learning*, chapter Parallel Online Learning. Cambridge University Press, 2011.
- [9] John C. Duchi, Alekh Agarwal, and Martin J. Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Trans. Automat. Contr.*, 57(3):592–606, 2012.
- [10] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: a lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.
- [11] John C. Duchi, Michael I. Jordan, and H. Brendan McMahan. Estimation, optimization, and parallelism when data is sparse. In *NIPS*, 2013.
- [12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *COLT*, 2010.
- [13] H. Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. In *COLT*, 2010.
- [14] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. In *KDD*, 2013.
- [15] Thore Graepel, Joaquin Quiñero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft’s bing search engine. In *ICML*, 2010.
- [16] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc’Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.
- [17] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM J. on Optimization*, 19(4):1574–1609, January 2009. ISSN 1052-6234. doi: 10.1137/070704277.
- [18] John Langford, Alex Smola, and Martin Zinkevich. Slow Learners are Fast. In *Advances in Neural Information Processing Systems 22*. 2009.
- [19] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML ’09, 2009.
- [20] Chih-Chung Chang and Chih-Jen Lin. LIBSVM data sets. <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>, 2010.
- [21] Peter Auer, Nicolò Cesa-Bianchi, and Claudio Gentile. Adaptive and self-confident on-line learning algorithms. *Journal of Computer and System Sciences*, 2002.

A Delay Pattern Examples

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1	2	3			
g_2	2	2	3			
g_3	3	3	3			
g_4				4	5	6
g_5				5	5	6
g_6				6	6	6

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1	1	1			
g_2	1	2	2			
g_3	1	2	3			
g_4				4	4	4
g_5				4	5	5
g_6				4	5	6

Read 1,2,3	Update 1	$\mathcal{F}_1 = \{2, 3\}$
	Update 2	$\mathcal{F}_2 = \{3\}$
	Update 3	$\mathcal{F}_3 = \{\}$
Read 4,5,6	Update 4	$\mathcal{F}_4 = \{5, 6\}$
	Update 5	$\mathcal{F}_5 = \{6\}$
	Update 6	$\mathcal{F}_6 = \{\}$

(A) Minibatches of size 3

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1	2				
g_2	2	2	3			
g_3		3	3	4		
g_4			4	4	5	
g_5				5	5	6
g_6					6	6

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1	1				
g_2	1	2	2			
g_3		2	3	3		
g_4			3	4	4	
g_5				4	5	5
g_6					5	6

Read 1, 2	Update 1	$\mathcal{F}_1 = \{2\}$
Read 3	Update 2	$\mathcal{F}_2 = \{3\}$
Read 4	Update 3	$\mathcal{F}_3 = \{4\}$
Read 5	Update 4	$\mathcal{F}_4 = \{5\}$
Read 6	Update 5	$\mathcal{F}_5 = \{6\}$
	Update 6	$\mathcal{F}_6 = \{\}$

(B) Fixed delay of 1

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1	2	3			
g_2	2	2	3	4		
g_3	3	3	3	4	5	
g_4		4	4	4	5	6
g_5			5	5	5	6
g_6				6	6	6

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1	1	1			
g_2	1	2	2	2		
g_3	1	2	3	3	3	
g_4		2	3	4	4	4
g_5			3	4	5	5
g_6				4	5	6

Read 1,2,3	Update 1	$\mathcal{F}_1 = \{2, 3\}$
Read 4	Update 2	$\mathcal{F}_2 = \{3, 4\}$
Read 5	Update 3	$\mathcal{F}_3 = \{4, 5\}$
Read 6	Update 4	$\mathcal{F}_4 = \{5, 6\}$
	Update 5	$\mathcal{F}_5 = \{6\}$
	Update 6	$\mathcal{F}_6 = \{\}$

(C) Fixed delay of 2

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1		3			
g_2		2	3		5	
g_3	3	3	3		5	
g_4				4	5	
g_5		5	5	5	5	6
g_6					6	6

	g_1	g_2	g_3	g_4	g_5	g_6
g_1	1		1			
g_2		2	2		2	
g_3	1	2	3		3	
g_4				4	4	
g_5		2	3	4	5	5
g_6					5	6

Read 1, 3	Update 1	$\mathcal{F}_1 = \{3\}$
Read 2, 5	Update 2	$\mathcal{F}_2 = \{3, 5\}$
	Update 3	$\mathcal{F}_3 = \{5\}$
Read 4	Update 4	$\mathcal{F}_4 = \{5\}$
Read 6	Update 5	$\mathcal{F}_5 = \{6\}$
	Update 6	$\mathcal{F}_6 = \{\}$

(D) Arbitrary order

Figure 3: Each row corresponds to a different delay pattern: batches of size 3, a constant delay of 2, a constant delay of 3, and an arbitrary delay pattern. Each pattern is shown as symmetric matrix, where cell (i, j) with $i > j$ is gray if update j is outstanding when update i is applied. The left column emphasizes the backward gradient sums associated with each update: In particular, letting B_t be the set of cells (i, j) labeled t , we have $g_t(g_t + 2g_t^{\text{bck}}) = \sum_{(i,j) \in B_t} g_i g_j$. Similarly, for the right-hand matrices, letting F_t be the set of cells (i, j) labeled t , we have $g_t(g_t + 2g_t^{\text{fwd}}) = \sum_{(i,j) \in F_t} g_i g_j$. These quantities play a pivotal role in our algorithms and analysis.

B Comparing Regret Bounds for Sparse Learning with Delays

In this section, we compare regret bounds for a variety of algorithms under both a fully-adversarial assumption, and a stochastic sparsity assumption like the one used by Duchi et al. [11]. For simplicity, consider a single dimension, T rounds, and gradients $g_t \in \{-1, 0, 1\}$. When appropriate, we consider a feasible set $[-R, R]$; we neglect the potential difference between R and \tilde{R} when comparing to our regret bounds, i.e., we assume $R = \tilde{R}$. In the fully adversarial model we assume exactly

	g_1	g_2	g_3	g_4	g_5	g_6					
g_1	1	1	1				Read 1,2,3	$r(1) = r(2) = r(3) = 1$	Update 1	$\mathcal{F}_1 = \{2, 3\}$	
g_2	1	2	2	2			Read 4	$r(4) = 2$	Update 2	$\mathcal{F}_2 = \{3, 4\}$	
g_3	1	2	3	3	3		Read 5	$r(5) = 3$	Update 3	$\mathcal{F}_3 = \{4, 5\}$	
g_4		2	3	4	4	4	Read 6	$r(6) = 4$	Update 4	$\mathcal{F}_4 = \{5, 6\}$	
g_5			3	4	5	5			Update 5	$\mathcal{F}_5 = \{6\}$	
g_6				4	5	6			Update 6	$\mathcal{F}_6 = \{\}$	
							Consider the difference $x_5 - \hat{x}_5$, where we have:				
							$x_5 = -\beta_3 g_1 - \beta_4 g_2 - \beta_4 g_3 - \beta_4 g_4$		AdaptiveRevision		
							$\hat{x}_5 = -\beta_3 g_1 - \beta_4 g_2 - \beta_5 g_3 - \beta_6 g_4$		HypBack		

Figure 4: An example the difference between HypBack (which plays \hat{x}_5) and AdaptiveRevision (which plays x_5), in terms of the common learning rates β_t .

$(1 - p)T$ of the gradients are 0, but the adversary chooses the gradients arbitrarily subject to this constraint. Under the stochastic sparsity assumption, each g_t is exactly 0 with probability $1 - p$ (chosen independently for each t); if g_t is not zero, the adversary chooses it arbitrarily (WLOG from $\{-1, 1\}$).

Synchronized minibatches We consider a mini-batch delay pattern with batches of size m . Of course, enforcing such a delay pattern requires synchronization: one of the key questions addressed here is whether similar bounds are possible with arbitrary delay patterns (with a maximum delay of m , say). We index batches by j , and let b_j be the sum of the m gradients in batch j . We assume m divides T so $J \equiv T/m$ is the total number of batches.

Without delays and given dense gradients ($p = 1$), standard online gradient descent with an appropriate adaptive learning rate can achieve a bound of $R\sqrt{\sum_{t=1}^T g_t^2}$ (we generally ignore constants in this section).

Thus, in the adversarial delayed case, but with a minibatch delay pattern, we can run J steps of online gradient descent on the combined gradients b_j , for a bound of $R\sqrt{\sum_{j=1}^J b_j^2} \leq R\sqrt{m^2 J} = \sqrt{mT}$ (when $p = 1$). This is the best we can do in the worst case. If only a p fraction of the g_t are non-zero, then we get a bound of \sqrt{mpT} , as the adversary can simply put all the non-zero gradients in the first pT rounds.

If, instead, the non-zero rounds are chosen randomly with probability p , but the adversary still controls what the gradient is (given that it is nonzero), he can still ensure all non-zero gradients in the same batch are in the same direction. Then b_j has binomial distribution (with an adversary-controlled sign), and so

$$\mathbb{E}[b_j^2] = \text{Var}[b_j] + \text{Mean}[b_j]^2 = mp(1 - p) + (mp)^2 = mp(1 + mp - p).$$

Again, starting from the bound $R\sqrt{\sum_j b_j^2}$, taking expectations and applying Jensen's inequality, we have

$$\mathbb{E} \left[\sqrt{\sum_{j=1}^J b_j^2} \right] \leq \sqrt{\mathbb{E} \left[\sum_{j=1}^J b_j^2 \right]} \leq \sqrt{Jmp(1 + mp - p)} = \sqrt{(1 + p(m - 1))pT}.$$

Note that $1 + p(m - 1) \leq m$, and so we have a strictly sharper bound than the \sqrt{mpT} result when $p < 1$. Replacing $m - 1$ with m does not weaken the bound in practice, and so we have

$$\text{Regret} \leq \sqrt{(1 + pm)pT}. \quad (10)$$

To see the improvement over the fully adversarial case, suppose $p = 1/100$ and $m = 100$. Then against an adversary we have regret \sqrt{T} , but with stochastic sparsity we have regret less than $\sqrt{T/50}$.

Subsampling data Suppose we have m machines. We should be able to just train on a $1/m$ fraction of the data on a single machine sequentially, and our regret will simply be about m times the regret this machine sees. This gives us $m\sqrt{T/m} = \sqrt{mT}$ in the dense case. So in the worst case we get the same bound by subsampling that we got by doing minibatches of size m (where we looked at m times as much data and did m times as much work!). Intuitively, this is because we don't account for any reduction in the variance of the gradient estimate due to increasing the minibatch size (because in a worst-case world that may not happen). Similarly, we get the same \sqrt{mpT} bound when data is sparse but the adversary controls the sparsity.

However, in the stochastic sparsity model, the subsampling approach still gets a bound of only of \sqrt{mpT} , so now minibatching has an advantage, obtaining the better bound of (10).

AdaptiveRevision The AdaptiveRevision algorithm achieves a bound like \sqrt{mpT} in the fully-adversarial case, and a bound like (10) if we run the algorithm on a problem with stochastic sparsity, without knowing the delay pattern or sparsity in advance. Recall we have bounds of the form

$$\sqrt{\sum_{t=1}^T g_t(g_t + g_t^{\text{fwd}})} = \sqrt{G_{1:t}^{\text{fwd}}}.$$

First, in the fully-adversarial setting, again we can assume all the non-zeros occur in the first pT rounds. On these rounds each g_t is bounded by 1 and g_t^{fwd} is bounded by m , and so we immediately have a bound of \sqrt{mpT} .

Now we consider a stochastic sparsity pattern. Under the assumptions of this section, note $\mathbb{E}[g_t^2] = p$ and $\mathbb{E}[|g_t^{\text{fwd}}|] \leq mp$, and so, taking advantage of independence,

$$\mathbb{E}[g_t(g_t + g_t^{\text{fwd}})] \leq \mathbb{E}[g_t^2] + \mathbb{E}[|g_t|] \mathbb{E}[|g_t^{\text{fwd}}|] = p + mp^2.$$

Thus, taking expectations, we have

$$\mathbb{E} \left[\sqrt{\sum_t G_{1:t}^{\text{fwd}}} \right] \leq \sqrt{\mathbb{E} \left[\sum_t G_{1:t}^{\text{fwd}} \right]} \leq \sqrt{(1 + mp)pT},$$

matching the performance of the synchronized mini-batch algorithm, (10). Note, however, we achieve this bound asynchronously, for arbitrary delay patterns, and without needing to know m or p in advance.

AsyncDA and AsyncAdaGrad We now compare these results to those of Duchi et al. [11]. Under our assumptions, we have (using their notation for the moment) $\|x^*\|_2 = 1$, $M_j = 1$, and $M = 1$, and consider a single coordinate j . Then, with $m = 1$, their lower bound matches the \sqrt{pT} result above, and this is achieved by (synchronous) OGD or Dual Averaging (their Eq. 5) and by AdaGrad (their Eq. 6). Their results for AsyncDA and AsyncAdaGrad apply in the stochastic sparsity model.

For AsyncDA, note that in our simple case $M^2 = \mathbb{E}[g_t^2] \leq p \cdot 1 + (1 - p) \cdot 0 = p$ and $M_j = 1$. Then, their Theorem 3 becomes

$$\mathbb{E}[\text{Regret}] \leq \frac{1}{2\eta} + \frac{\eta}{2}Tp + \eta Tmp^2.$$

With an optimal learning rate $\eta = 1/\sqrt{T + 2mp^2T}$ chosen with knowledge of both m and p , this gives regret $\sqrt{(1 + 2pm)pT}$, which matches (10). On the other hand, generally p cannot be known in advance in an online setting, so using $\eta = 1/\sqrt{T + 2mT}$ gives only $\mathcal{O}((1 + p^2)\sqrt{mT})$. This bound is significantly worse as $p \rightarrow 0$.

For AsyncAdaGrad, their Theorem 5 becomes

$$\mathbb{E}[\text{Regret}] \leq \frac{1}{\eta} \sqrt{m + Tp} + \eta \sqrt{Tp}(1 + pm).$$

With a learning rate scale factor of $\eta = \frac{1}{\sqrt{1+mp}}$ (again, dependent on both m and p), this gives a bound that is $\mathcal{O}(\sqrt{(1 + pm)(m + pT)})$, which matches (10) when we ignore terms independent of

T (noting $\sqrt{m+pT} \leq \sqrt{m} + \sqrt{pT}$). Without knowledge of p (say, taking $\eta = \frac{1}{\sqrt{1+m}}$), we arrive at bound like $\mathcal{O}((1+p)\sqrt{mpT})$; without knowledge of m or p , we arrive at a bound no better than $\mathcal{O}((1+pm)\sqrt{Tp})$ (e.g., taking $\eta = 1$).

C Complete Analysis and Proofs

Several results will depend on the following basic result:

Lemma 4. *Under the above definitions, we have*

- | | |
|--|---|
| A. $t \in \mathcal{F}_{r(t)} \iff t \neq r(t)$ | E. $t \in \mathcal{B}_s \iff s \in \mathcal{F}_t$ |
| B. $o(t) \geq t$ | F. <i>InOrder</i> implies $s_1 \leq s_2 \Rightarrow r(s_1) \leq r(s_2)$ |
| C. $s \leq t \Rightarrow o(s) \leq o(t)$ | G. <i>If delay is bounded by m, then $o(t) \leq t+m$.</i> |
| D. $o(r(t)) \geq t$ | |

It is worth remarking that our choice of indices ensures g_t^{bck} is a sum of consecutively-indexed updates, while this need not be the case for g_t^{fwd} . However, the *InOrder* property in fact implies G^{fwd} is sum of consecutively indexed gradients.

Proof. Most of these are immediate consequences of the definitions. For claim (C), first note if $s = t$, we are done. Suppose $s < t$, and consider two cases. First, suppose $o(s) \leq t$, then $o(s) \leq o(t)$ using (B), and we are done. For the second case, suppose $o(s) > t$. Then since $s < t$, we have $o(s) \in \mathcal{F}_t$, implying $o(t) \geq o(s)$. For (D), if $r(t) = t$, we are done by (B). If $r(t) < t$, then $t \in \mathcal{F}_{r(t)}$ (A), and so $o(r(t)) \geq t$. For (E), suppose $t \in \mathcal{B}_s = \{r(s), \dots, s-1\}$, so $r(s) \leq t < s$, and so $t \in \mathcal{F}_s$. For the other direction, if $s \in \mathcal{F}_t$, we have $r(s) \leq t < s$, which implies $t \in \{r(s), \dots, s-1\} = \mathcal{B}_s$. Claim (F) is the contrapositive of the definition of *InOrder*. For (G), if $o(t) = t$, we are done. Otherwise, let $s = o(t)$ with $s \in \mathcal{F}_t$, and so $r(s) \leq t < s$. Then, $o(t) - t = s - t \leq s - r(s) \leq m$. \square

C.1 Proof of Lemma 2

The analysis will use the following result:

Lemma 5. *Assume delays are bounded by m and $|g_t| \leq L$. Then, given *InOrder* delays, for all t ,*

$$G_{1:t}^{\text{fwd}} - m^2 L^2 \leq G_{1:o(t)}^{\text{bck}} \leq G_{1:t}^{\text{fwd}} + m^2 L^2.$$

Proof. Note

$$G_{1:o(t)}^{\text{bck}} = \sum_{u=1}^{o(t)} g_u^2 + 2 \sum_{u=1}^{o(t)} g_u \sum_{s \in \mathcal{B}_u} g_s.$$

Considering the last term,

$$\begin{aligned} \sum_{u=1}^{o(t)} g_u \sum_{s \in \mathcal{B}_u} g_s &= \sum_{u=1}^{o(t)} \sum_{s=1}^{o(t)} I(s \in \mathcal{B}_u) g_u g_s \\ &= \sum_{s=1}^{o(t)} \sum_{u=1}^{o(t)} I(u \in \mathcal{F}_s) g_u g_s && \text{Lemma 4(E).} \\ &= \sum_{s=1}^t \sum_{u \in \mathcal{F}_s} g_u g_s + \sum_{s=t+1}^{o(t)} \sum_{u=s+1}^{o(t)} I(u \in \mathcal{F}_s) g_u g_s. \end{aligned}$$

For the first part of the sum, observe that since $s \leq t$ we have $u \in \mathcal{F}_s \Rightarrow 1 \leq u \leq o(t)$; in the second part of the sum, we can start indexing at $u = s + 1$ since $u \in \mathcal{F}_s \Rightarrow u > s$. Plugging back

in, and dividing the sum over g_u^2 between the two terms,

$$G_{1:o(t)}^{\text{bck}} = G_{1:t}^{\text{fwd}} + \sum_{s=t+1}^{o(t)} \left(g_s^2 + 2 \sum_{u=s+1}^{o(t)} I(u \in \mathcal{F}_s) g_u g_s \right).$$

The result follows by observing there are at most m^2 terms of the form g_s^2 or $g_u g_s$ in the right-hand sum, and each of these is bounded by L^2 . \square

Proof of Lemma 2. Applying Lemma 1, we have

$$\text{Regret} \leq \frac{2R_T^2}{\hat{\eta}_T} + \frac{1}{2} \sum_{t=1}^T \hat{\eta}_t G_t^{\text{fwd}}$$

where the learning rates $\hat{\eta}_t$ are given by (8). Lemma 5 implies $\tilde{G}_{1:o(t)}^{\text{bck}} + m^2 L^2 \geq \tilde{G}_{1:t}^{\text{fwd}}$, which in turn implies $\hat{\eta}_t \leq \eta_t$. Thus,

$$\frac{1}{2} \sum_{t=1}^T \hat{\eta}_t G_t^{\text{fwd}} \leq \frac{1}{2} \sum_{t=1}^T \eta_t G_t^{\text{fwd}} \leq \alpha \sqrt{\tilde{G}_{1:T}^{\text{fwd}}},$$

where we have again used Corollary 10. However, we could have

$$\tilde{G}_{1:T}^{\text{bck}} > \tilde{G}_{1:T}^{\text{fwd}}$$

(even though $G_{1:T}^{\text{bck}} = G_{1:T}^{\text{fwd}}$), but we can still bound the second term as

$$\frac{2R_T^2}{\hat{\eta}_T} = \frac{2R_T^2}{\alpha} \sqrt{\tilde{G}_{1:T}^{\text{bck}} + G_0} \leq \sqrt{2}\tilde{R} \sqrt{\tilde{G}_{1:T}^{\text{bck}} + m^2 L^2} \leq \sqrt{2}\tilde{R} \sqrt{\tilde{G}_{1:T}^{\text{fwd}} + 2m^2 L^2},$$

using Lemma 5, $\alpha = \sqrt{2}\tilde{R}$, and $G_0 = m^2 L^2$. Recalling $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for $a, b \geq 0$ and then combining these results completes the proof. \square

C.2 Lemma 6

Lemma 6. *Under the InOrder assumption, Algorithm 1 plays the points specified by (9).*

The proof is a straightforward induction making use of the following simpler expression for the points played by AdaptiveRevision:

Lemma 7. *Under the InOrder assumption, an equivalent expression for (9) (for $t \geq 2$) is*

$$x_t = - \sum_{s=1}^{r(t)-1} \beta_{o(s)} g_s - \sum_{s=r(t)}^{t-1} \beta_{t-1} g_s.$$

Proof. Starting from (9), it is sufficient to show the η_s^t take on the claimed values. First, consider an $s < r(t)$. Note $r(o(s)) \leq s$ since $o(s) \in \mathcal{F}_s \cup \{s\}$. Thus $r(o(s)) \leq s < r(t)$, and so under InOrder, we have $o(s) < t$, which implies $o(s) \leq t-1$, and so $\eta_s^t = \beta_{o(s)}$.

Now suppose $s \geq r(t)$. Then $o(s) \geq o(r(t))$ and then $o(r(t)) \geq t$, using Lemma 4, parts (C) and (D), and so $\min(t-1, o(s)) = t-1$, and so $\eta_s^t = \beta_{t-1}$. \square

C.3 Lemma 8

Let \hat{x}_t be points played by HypBack, as in (8), and let x_t be the points played by AdaptiveRevision. Then, we need to bound $\sum_{t=1}^T g_t(x_{r(t)} - \hat{x}_{r(t)})$, the difference in the loss incurred by AdaptiveRevision and HypBack. Figure 4 gives an example. The following lemma provides the needed guarantee. Note the gap is *independent* of the number of rounds T :

Lemma 8. *When InOrder holds, the maximum delay is m , and we take $G_0 = m^2 L^2$, we have $\sum_{t=1}^T g_t(x_{r(t)} - \hat{x}_{r(t)}) \leq 2\alpha L m$.*

Proof. We begin by bounding

$$x_t - \hat{x}_t = \sum_{s=r(t)}^{t-1} -g_s(\eta_s^t - \hat{\eta}_s) = \sum_{s \in \mathcal{B}_t} -g_s(\beta_{t-1} - \beta_{o(s)}),$$

where we have used Lemma 7 and (8). For $1 \leq s \leq t$ and $d \geq 0$ define

$$\delta(s, t) \equiv \beta_s - \beta_t \quad \text{and} \quad \delta'(t, d) \equiv \beta_t - \beta_{t+d}.$$

Note δ and δ' are both decreasing in the first argument, and increasing in the second argument. When $r(t) = 1$ (for example, when $t = 1$), we have $\mathcal{B}_{r(t)} = \emptyset$, and so $x_{r(t)} - \hat{x}_{r(t)} = 0$; to handle this notationally, we let $\delta(0, t') = 0$ and $\delta'(0, d) = 0$ for any t' and d . Then, we have

$$\begin{aligned} & \sum_{t=1}^T g_t(x_{r(t)} - \hat{x}_{r(t)}) \\ &= - \sum_{t=1}^T g_t \sum_{s \in \mathcal{B}_{r(t)}} g_s \delta(r(t) - 1, o(s)) \\ &\leq L^2 \sum_{t=1}^T \sum_{s \in \mathcal{B}_{r(t)}} \delta(r(t) - 1, o(s)) \\ &\leq L^2 \sum_{t=1}^T m \delta(r(t) - 1, o(r(t) - 1)) \end{aligned}$$

where the last inequality uses Lemma 4(C) to show $\max_{s \in \mathcal{B}_{r(t)}} o(s) \leq o(r(t) - 1)$, since $\max(\mathcal{B}_{r(t)}) = r(t) - 1$, and then notes $|\mathcal{B}_{r(t)}| \leq m$. Continuing the inequality,

$$\begin{aligned} &\leq mL^2 \sum_{t=1}^T \delta(r(t) - 1, r(t) - 1 + m) && \text{Lemma 4(G)} \\ &= mL^2 \sum_{t=2}^T \delta'(r(t) - 1, m) && \text{Defn., } \delta'(0, m) = 0 \\ &\leq mL^2 \sum_{t=2}^T \delta'(\max(1, t - m - 1), m) && \text{Since } r(t) \geq t - m. \end{aligned}$$

We can bound the first m terms by $m(mL^2) \frac{\alpha}{mL} = \alpha Lm$ since $\delta'(1, m) \leq \beta_1 \leq \frac{\alpha}{mL}$. Now, re-indexing the remaining terms, we have

$$mL^2 \sum_{t=1}^{T-m-1} \delta'(t, m) = mL^2 \sum_{t=1}^{T-m} (\beta_t - \beta_{t+m}) \leq mL^2 \sum_{t=1}^m \beta_t \leq \alpha mL,$$

where we have used the fact that this sum telescopes with an offset of m terms, and we have again used $\beta_t \leq \beta_1 \leq \frac{\alpha}{mL}$. \square

D Technical Lemmas

We have the following slightly stronger version of the standard lemma (e.g., Auer et al. [21, Lemma 3.5]) used to analyze AdaGrad-style algorithms:

Lemma 9. *For any real numbers x_1, x_2, \dots, x_T such that $x_{1:t} > 0$ for $t \in \{1, \dots, T\}$,*

$$\sum_{t=1}^T \frac{x_t}{\sqrt{x_{1:t}}} \leq 2\sqrt{x_{1:T}}.$$

Proof. For $y \geq 0$, \sqrt{y} is concave with derivative $\frac{1}{2\sqrt{y}}$, so by concavity for $z \geq 0$,

$$\sqrt{z} \leq \sqrt{y} + \frac{1}{2\sqrt{y}}(z - y).$$

For a, b with $a \geq 0$ and $a + b \geq 0$, we can take $y = a + b$ and $z = a$, and so

$$2\sqrt{a} + \frac{b}{\sqrt{a+b}} \leq 2\sqrt{a+b}. \quad (11)$$

The proof proceeds by induction; the base case of $T = 1$ holds trivially, since $\sqrt{x_1} \leq 2\sqrt{x_1}$. Now, suppose the theorem holds for some $t \geq 1$. Then,

$$\begin{aligned} \sum_{s=1}^{t+1} \frac{x_s}{\sqrt{x_{1:s}}} &= \sum_{s=1}^t \frac{x_s}{\sqrt{x_{1:s}}} + \frac{x_{t+1}}{\sqrt{x_{1:t+1}}} \\ &\leq 2\sqrt{x_{1:t}} + \frac{x_{t+1}}{\sqrt{x_{1:t+1}}} && \text{By the IH} \\ &\leq 2\sqrt{x_{1:t+1}} && \text{Using (11).} \end{aligned}$$

□

Using this, we can prove:

Corollary 10. For any $x_1, x_2, \dots, x_T \in \mathbb{R}$, with $x_1 > 0$, we have

$$\sum_{t=1}^T \frac{x_t}{\sqrt{\max_{s \leq t} x_{1:s}}} \leq 2\sqrt{\max_{t \leq T} x_{1:t}}.$$

Proof. Define z_t inductively with $z_1 = x_1 > 0$ such that $z_{1:t} = \max_{s \leq t} x_{1:t}$. Thus, $z_{1:t}$ is non-decreasing in t so each $z_t \geq 0$. Thus, we can apply Lemma 9 to the sequence of z_t 's, which gives

$$\sum_{t=1}^T \frac{z_t}{\sqrt{z_{1:t}}} \leq 2\sqrt{z_{1:T}}.$$

To complete the proof, we argue by induction with the induction hypothesis

$$\sum_{t=1}^T \frac{z_t}{\sqrt{z_{1:t}}} - \sum_{t=1}^T \frac{x_t}{\sqrt{z_{1:t}}} \geq \frac{z_{1:T} - x_{1:T}}{\sqrt{z_{1:T}}}.$$

Observe the right-hand-side is non-negative by definition, so showing this is sufficient. The base case is trivial since $x_1 = z_1$. Suppose the IH holds for T . Then, we are adding

$$D = \frac{z_{T+1}}{\sqrt{z_{1:(T+1)}}} - \frac{x_{T+1}}{\sqrt{z_{1:(T+1)}}}$$

to the left-hand side. Further, since $z_{1:T} \leq z_{1:T+1}$, we have

$$\frac{z_{1:T} - x_{1:T}}{\sqrt{z_{1:T}}} \geq \frac{z_{1:T} - x_{1:T}}{\sqrt{z_{1:(T+1)}}}. \quad (12)$$

Thus,

$$\begin{aligned} \sum_{t=1}^{T+1} \frac{z_t}{\sqrt{z_{1:t}}} - \sum_{t=1}^{T+1} \frac{x_t}{\sqrt{z_{1:t}}} &\geq \frac{z_{1:T} - x_{1:T}}{\sqrt{z_{1:T}}} + \frac{z_{T+1}}{\sqrt{z_{1:(T+1)}}} - \frac{x_{T+1}}{\sqrt{z_{1:(T+1)}}} && \text{(IH)} \\ &\geq \frac{z_{1:T+1} - x_{1:T+1}}{\sqrt{z_{1:T+1}}}. && \text{Using (12)} \end{aligned}$$

□