A Theoretical Analysis of a Warm Start Technique

Martin A. Zinkevich
Yahoo! Labs
701 First Avenue
Sunnyvale, CA

Abstract

Batch gradient descent looks at every data point for every step, which is wasteful for early steps where the current position is nowhere near optimal. There has been a lot of interest in warm-start approaches to gradient descent techniques, but little analysis. In this paper, we formally analyze a method of warm-starting batch gradient descent using small batch sizes. We argue that this approach is fundamentally different than mini-batch, in that after an initial shuffle, it requires only sequential passes over the data, improving performance on datasets stored on a disk drive.

1 Support Vector Machine Problems

Suppose you have a set of examples $(x_1, y_1) \ldots (x_m, y_m)$, drawn from a distribution D, and a regularization parameter λ. For each example i, there is some convex loss $L_i : \mathbf{R}^d \to \mathbf{R}$ over the parameter space associated with example i, such that for any $S \subseteq \{1 \ldots m\}$, we have a cost function $f_S : \mathbf{R}^d \to \mathbf{R}$:

$$f_S(w) = \frac{\lambda}{2} w^2 + \frac{1}{|S|} \sum_{i \in S} L_i(w). \tag{1}$$

Also, we can have a distribution D^* over loss functions. Using this we can define:

$$f_D(w) = \frac{\lambda}{2} w^2 + \int L(w) D^*(dL). \tag{2}$$

Here, we will focus on finding a point w such that $f_{D^*}(w)$ is as small as possible. This is often done by minimizing $f_{1 \ldots m}$. As an intermediate step, we focus on Euclidean distance to $w^* = \operatorname{argmin}_w f_{D^*}(w)$, as many times this is a symmetric problem. Define G to be a bound on the magnitude of the gradient of functions in D^*. Define $w_{1 \ldots m} \in \mathbf{R}^d$ such that $w_{1 \ldots m} = \operatorname{argmin}_w f_{1 \ldots m}(w)$. We will first bound $\nabla f_S(w^*)$ with high probability, and then connect this to the distance between $w_{1 \ldots m}$ and w^*. A quick theorem, in the vein of statistical learning theory:

Lemma 1 *With probability of at least* $1 - \delta$, $\|w_{1 \ldots m} - w^*\| \leq \sqrt{\frac{(\ln d + \ln 2 - \ln \delta) 8 d G^2}{\lambda^2 m}}$.

The proof is in the appendix. In [1], they argue that with access to an infinite amount of data, it makes sense to balance the loss due to optimization error and the loss due to estimation error. Here, we use this balance not just to tune the algorithm but to alter its structure. We modify batch gradient by having multiple epochs, doubling the size of the batch each epoch.

One key practical element that we want to emphasize is that random access has a high cost, especially on hard disk drives. For example, a modern disk drive can read or write about 100 megabytes a second of data written in sequence in a file: however, if you want a random chunk off the disk, it can take about 10 milliseconds for the disk to spin to a specific location and for the head to move to access that one byte. Therefore, if your examples are 1 kilobyte in size, then you can either read 10,000 sequentially per second or 100 randomly per second. On more expensive solid state drives, the gap is smaller, but this can make storage prohibitively expensive. However, special tricks can be applied to shuffle data (see below).

Thus, stochastic gradient descent [2] and mini-batch [7, 5] can be significantly more expensive than one might expect by just looking at the number of iterations. In practical implementations (such as [6]), the stochastic aspect of the stochastic gradient descent) is ignored, and the examples are passed over sequentially. Although in practice this algorithm works fine and you can prove

Algorithm 1 Shuffled Stochastic Gradient Descent

```
Input: Cost functions $L_1 \dots L_m$, epochs $T$, step size $\eta$, initial $w \in \mathbf{R}^n$.
If not already shuffled, shuffle $L_1 \dots L_m$.
for $t = 1$ to $T$ do
    for $i = 1$ to $m$ do
        $w \leftarrow w - \eta \nabla f_i(w)$
    end for
end for
```

that it converges, getting practical convergence rates is incredibly difficult. This represents a huge discrepency between theory and practice in the field.

One critical issue is that you must shuffle the data: if the data was generated independently and identically (each example drawn at random from the same distribution), then no shuffling is necessary. On the other hand, if it was then reordered (for example, the data was ordered by date, or label), then it must be shuffled. However, oddly, shuffling the data does not require much random access. For example, we can take a dataset, and add a random key to each entry. Then, using fast methods for large scale sorting (such as mergesort), we can sort by the random key, effecting a shuffle using $n \log n$ time. Therefore, the question we ask in this paper is:

> Given a dataset that is shuffled once and accessed sequentially thereafter, what theoretical guarantees can be obtained?

Although we do not have a tight analysis of Shuffled Stochastic Gradient Descent, in this paper we present another practical algorithm which is easier to analyze.

Algorithm 2 Doubling Batch Gradient Descent

```
Input: Cost functions $L_1 \dots L_m$, initial batch size $m_{initial}$, iterations $T$ per size, step size $\eta$,
initial $w \in \mathbf{R}^n$, iterations $T_0$ in the first phase.
If not already shuffled, shuffle $L_1 \dots L_m$.
$m' \leftarrow m_{initial}$.
{The first epoch}
for $t = 1$ to $T_0$ do
    $w \leftarrow w - \eta \nabla f_{1 \dots m'}(w)$
end for
while $m' < m$ do
    {Each iteration of the while loop is another epoch}
    $m' \leftarrow \min(2m', m)$.
    for $t = 1$ to $T$ do
        $w \leftarrow w - \eta \nabla f_{1 \dots m'}(w)$
    end for
end while
```

2 Theoretical Analysis of Doubling Batch Gradient Descent

In this section we show how to set the parameters to prove a significant improvement over traditional analysis. Define $C = \frac{\sqrt{(\ln d + \ln 2 - \ln \delta + \ln \log_2 m) 8dG}}{\lambda}$ (see Lemma 3 for why).

Theorem 2 *Running Algorithm 3 with $m' = 1$, $T_0 = T = \lceil(\kappa+1)\ln(1+2\sqrt{2})\rceil$, $\eta = \frac{1}{H+\lambda}$ achieves an error of $\frac{(H+\lambda)4C^2}{m}$.*

Before we prove this, we need two lemmas.

The parameters T_0 and T are selected such that at the end of each epoch, the bound on $\|w - w_{1\dots m'}\|$ is roughly equal to the bound on $\|w_{1\dots m'} - w^*\|$.

Lemma 3 *With probability at least $1-\delta$, for all m' used, $\|w_{1\dots m'} - w^*\| \leq \frac{C}{\sqrt{m'}}$.*

Proof: This follows from union bounds over the $\log_2 m$ bounds, using Lemma 1, substituting $\delta/\log_2 m$ for δ. ■

Suppose that there is a upper bound H on the Lipschitz constant on the gradient of the functions $L_1 \dots L_m$. Then, for all $S \subseteq \{1\dots m\}$, we have a bound on the Lipschitz constant of the gradient of f_S. Define $d(x,y) = \|x-y\|$. Therefore, we can use a slight tweak on Lemma 16 from the appendix of [8], similar to the exponential convergence in cost described in [3], chapter 9:

Lemma 4 *Given a convex function L where ∇L is Lipschitz continuous with constant H, define $c(x) = \frac{\lambda}{2}x^2 + L(x)$. If $\eta \leq (\lambda+H)^{-1}$, and $\eta < 1$, then for all x:*

$$d(w - \eta\nabla c(w), w^*) \leq (1-\eta\lambda)d(w,w^*) \tag{3}$$

The proof is mostly in [8], but can be slightly generalized (making strict inequalities conditions simply inequalities).

Proof (of Theorem 2): Thus, if we set $\eta = \frac{1}{H+\lambda}$, then we know that we move exponentially fast. Define $\kappa = \frac{H}{\lambda}$ to be the conditioning number of the system. Define w_0 to be the initial position at the beginning of a phase. After T steps:

$$d(w, w_{1\dots m'}) \leq \left(1 - \frac{\lambda}{H+\lambda}\right)^T d(w, w_0) \tag{4}$$

$$d(w, w_{1\dots m'}) \leq \exp\left(-\frac{\lambda}{H+\lambda}T\right) d(w, w_0) \tag{5}$$

Suppose that we choose T_0 and the initial sample size $m_{initial}$ such that $d(w, w_{1\dots m'}) = \frac{C}{\sqrt{m'}}$ after the first epoch. Then, since $d(w^*, w_{1\dots m'}) \leq \frac{C}{\sqrt{m'}}$ and $d(w^*, w_{1\dots 2m'}) \leq \frac{C}{\sqrt{2m'}}$, then:

$$d(w, w_{1\dots 2m'}) \leq d(w, w_{1\dots m'}) + d(w^*, w_{1\dots m'}) + d(w^*, w_{1\dots 2m'}) \tag{6}$$

$$d(w, w_{1\dots 2m'})\frac{C}{\sqrt{m'}} + \frac{C}{\sqrt{m'}} + \frac{C}{\sqrt{2m'}} \tag{7}$$

$$d(w, w_{1\dots 2m'})\left(1+2\sqrt{2}\right)\frac{C}{\sqrt{2m'}} \tag{8}$$

Therefore, if $T = \lceil(\kappa+1)\ln(1+2\sqrt{2})\langle 1 + (\kappa+1)\ln(1+2\sqrt{2}) < 2.68\kappa + 3.68$, then at the end of the epoch, $d(w, w_{1\dots 2m'}) \leq \frac{C}{\sqrt{2m'}}$.

Here is the key element: note that T only has a linear dependence on κ, and is independent of m', or even the final error of the algorithm. Moreover, a single iteration has a cost of $m'd$, and so a single epoch (except the first) has a cost of $\ln(1+2\sqrt{2})(\kappa+1)m'd$.

Note that $C > \frac{G}{\lambda}$, so if $m' = 1$, and $w_0 = 0$, then $\|w_0 - w_{1\dots m'}\| \leq C/\sqrt{m'}$. Therefore, we can choose $T_0 < T$, and push $m'_{initial}$ to be significantly higher than 1. So, the first epoch will not take longer than others.

Assume that $m/m_{initial}$ is a power of 2. The computational cost of the algorithm will be:

$$\textbf{Cost} \leq (m_{initial} + 2m_{initial} + 4m_{initial} + \ldots + m/2 + m)(2.68\kappa + 3.68)d \tag{9}$$

$$\textbf{Cost} \leq \left(\sum_{t=0}^{\infty} 2^{-t}\right) m(2.68\kappa + 3.68)d \tag{10}$$

$$\textbf{Cost} \leq (2.68\kappa + 3.68)md. \tag{11}$$

This algorithm achieves a parameter $\frac{2C}{\sqrt{m}}$ distance from the optimal. Using the bounds on the Lipschitz constant on the second derivative, this achieves a bound in cost[1] of $\frac{(H+\lambda)4C^2}{m}$.

■

Note that we did not change m' and T_0 from the natural defaults of 1 and T. One other reason for increasing m' is to affect the conditioning number of the resulting problem. As the loss functions from many examples are averaged, the problem can become more well-conditioned. Moreover, in certain cases, one can empirically bound the value of H by one pass over the data where you average the examples, allowing you to adjust the number of iterations per epoch accordingly. See Appendix B. Running batch gradient method to obtain this optimization error, referring to [1], or using the analysis here, requires at least on the order $\kappa md \log \frac{m}{4C^2(H+\lambda)}$ (barring varying definitions of the conditioning number).[2] Therefore, by scaling up the data set size, we have eliminated this last term from our bounds.

3 Conclusion

Most discussions of large-scale learning focus on parallelization across different machines. However, another direction is to push learning deeper into the memory hierarchy. This paper has argued that deeper in the memory hierarchy, the difference between sequential access time and random access time is very significant, making stochastic gradient and mini-batch approaches less attractive than they would otherwise be. One very powerful argument against batch is the overkill involved in looking at the gradient of every example at the initial point: this paper has to some degree addressed this issue. However, barring the issues of random access, stochastic gradient descent and mini-batch have very different properties, not to mention second order methods. Overall, the larger question is:

> Given a dataset that is shuffled once and accessed sequentially thereafter, what theoretical guarantees can be obtained?

In other words, what guarantees can be obtained with pseudo-stochastic gradient descent, where the data is shuffled once, but then passed over multiple times? What if mini-batch is used? Theoretically speaking, Newton's method basically takes six steps once it has reached its asymptotic stage [3]. However, it can take a while to reach that phase: can this analysis apply in that phase?

On reasonable argument for ignoring some of the practical issues with stochastic methods is the arrival of solid state drives, which have radically different performance characteristics than disk drives. But for the foreseeable future, a machine focused on processing data streaming from a disk will be a very cheap option. Secondly, the memory hierarchy itself presents interesting opportunities and challenges: it is conceivable that data could be accessed multiple times in memory while new data is being drawn from disk. These types of approaches are being enabled with tools like Torch [4], but there is little theoretical understanding.

We make no claims that this is the first example of shuffling once and running many times. In fact, this represents a standard trick (e.g., it can be applied using Vowpal Wabbit [6]), but one with little theory supporting it. If the theory behind shuffling algorithms and pseudo-stochastic algorithms can be expanded, it will better connect the theory of machine learning optimization and its practice.

[1] Note that this is a slightly different bound than in [1]: note that C has a $1/\lambda$ term, so that this bound is neither strictly better or worse than the bounds in that paper.

[2] In this work, we define the conditioning number to be the ratio of the Lipschitz constant of the gradient to the strong convexity: in [1], they focus on a similar ratio in the vicinity of the optimal point, whereas [3] use a measure similar to the one in this paper.

References

[1] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, 2008.

[2] Lon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, pages 146–168, 2004.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, 2004.

[4] R. Collobert, S. Bengio, L. Bottou, J. Weston, and I .Melvin. Torch 5, 2011. `http://torch5.sourceforge.net`.

[5] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning*, 2011.

[6] J. Langford. Vowpal wabbit, 2011. `https://github.com/JohnLangford/vowpal_wabbit/wiki`.

[7] N. Schraudolph and T. Graepel. Combining conjugate direction methods with stochastic approximation of gradients. In C. Bishop and B. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

[8] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, 2011.

A Non-Asymptotic Estimation Error

Lemma 5 ***Hoeffding's Inequality*** *for all i in $1 \dots n$, $\Pr(X_i \in [a_i, b_i]) = 1$, and $\overline{X} = (X_1 + \ldots + X_n)/n$, then*

$$\Pr(\left|\overline{X} - E[\overline{X}]\right| \geq t) \leq 2\exp\left(-\frac{2t^2n^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{12}$$

We will also use a basic result from strong convexity:

Lemma 6 ***[3], Equation 9.11****If x^* is the minimum of f, and f is h-strongly convex, then $\|x - x^*\| \leq \frac{2}{h}\|\nabla f(x)\|$.*

An interesting issue is establishing how accurate we are. Suppose that we know that regardless of the dataset sampled, we have a lower bound on the curvature. Define $f_t : \mathbf{R}^d \to \mathbf{R}$ to be a function with m samples. For any point x, if x^*_m is the optimal point of f_m, then:

Lemma 1 *With probability of at least $1 - \delta$, $\|w_{1\ldots m} - w^*\| \leq \sqrt{\frac{(\ln d + \ln 2 - \ln\delta)8dG^2}{\lambda^2 m}}$.*

Proof: Since w^* minimizes f_{D^*}, $\nabla f_{D^*}(w^*) = 0$. Therefore, if L is drawn from D^*, then the expected gradient of $\frac{\lambda}{2}w^2 + L(w)$ at w^* is zero. Thus, the ith component of the gradient $\lambda w^*_i + (\nabla L_(w^*))_i$ has expected value zero, and is between $G + \lambda w^*_i$ and $-G + \lambda w^*_i$. Thus, we can use Hoeffding bounds to show that the average for any dimension i:

$$\Pr(|\nabla f_{1\ldots m}(w^*)_i| \geq t) \leq 2\exp\left(-\frac{2t^2m^2}{\sum_{i=1}^{m}(2G)^2}\right) \tag{13}$$

$$\Pr(|\nabla f_{1\ldots m}(w^*)_i| \geq t) \leq 2\exp\left(-\frac{t^2m}{2G^2}\right). \tag{14}$$

Then, using union bounds:

$$\Pr(\forall i, |\nabla f_{1\ldots m}(w^*)_i| \geq t) \leq 2d\exp\left(-\frac{t^2m}{2G^2}\right) \tag{15}$$

$$\Pr(\|\nabla f_{1\ldots m}(w^*)\| \geq t\sqrt{d}) \leq 2d\exp\left(-\frac{t^2m}{2G^2}\right) \tag{16}$$

Now, from Lemma 6, since $f_{1\ldots m}$ is λ-strongly convex, we can use $\nabla f_{1\ldots m}(w^*)$ to bound the distance between w^* and the optimal solution of $f_{1\ldots m}$. Formally,

$$\|w^* - w_{1\ldots m}\| \leq \frac{2}{\lambda}\|\nabla f_{1\ldots m}(w^*)\| \tag{17}$$

Therefore, we push this into our previous results:

$$\Pr\left(\frac{2}{\lambda}\|\nabla f_{1\ldots m}(w^*)\| \geq \frac{2}{\lambda}t\sqrt{d}\right) \leq 2d\exp\left(-\frac{t^2m}{2G^2}\right) \tag{18}$$

$$\Pr\left(\|w^* - w_{1\ldots m}\| \geq \frac{2}{\lambda}t\sqrt{d}\right) \leq 2d\exp\left(-\frac{t^2m}{2G^2}\right) \tag{19}$$

Setting $\epsilon = \frac{2t\sqrt{d}}{\lambda}$, and substituting $\frac{\epsilon\lambda}{2\sqrt{d}}$ for t:

$$\Pr(\|\nabla f_{1\ldots m}(w^*)\| \geq \epsilon) \leq 2d\exp\left(-\frac{\epsilon^2\lambda^2m}{8dG^2}\right) \tag{20}$$

Therefore, setting

$$\delta = 2d\exp\left(-\frac{\epsilon^2\lambda^2m}{8dG^2}\right). \tag{21}$$

We now have established:

$$\Pr(\|\nabla f_{1\ldots m}(w^*)\| \geq \epsilon) \leq \delta \tag{22}$$

Therefore, solving for ϵ:

$$\ln\delta = \ln 2 + \ln d + \left(-\frac{\epsilon^2\lambda^2 m}{8dG^2}\right) \tag{23}$$

$$\frac{(\ln\delta - \ln 2 - \ln d)8dG^2}{\lambda^2 m} = -\epsilon^2 \tag{24}$$

$$\sqrt{\frac{(\ln d + \ln 2 - \ln\delta)8dG^2}{\lambda^2 m}} = \epsilon \tag{25}$$

$$\sqrt{\frac{(\ln d + \ln 2 - \ln\delta)8dG^2}{\lambda^2 m}} = \epsilon \tag{26}$$

■

B Bounding H

One drawback of this approach is that it requires deep knowledge of the parameters of the system, such as the maximum gradient and the Lipschitz constant of the derivative of the average of the losses.

It turns out, it is fairly easy to bound H for the whole dataset in a single pass.

Suppose that you consider the maximum curvature of the average of a bunch of loss functions. Suppose that these loss functions are of the form:

$$L_i(w) = l(w \cdot x^i, y^i). \tag{27}$$

We assume that the Lipschitz constant of the derivative of the loss function l in the first term is bounded above by H^*, and for all i, $\left\|x^i\right\| = 1$. First, we are trying to bound the curvature of:

$$L_{1\ldots m}(w) = \frac{1}{m}\sum_{i=1}^{m} L_i(w) \tag{28}$$

Define $\overline{x} = \frac{1}{m}\sum_{i=1}^{m} x^i$.

Lemma 7 *If for all i, for all j, $x^i_j \geq 0$, then the Lipschitz constant of $\nabla L_{1\ldots m}(w)$ is bounded above by $H^* \|\overline{x}\|$.*

Proof: We know that:

$$\nabla L_i(w) = \left.\frac{\delta l(\hat{y}, y^i)}{\delta y^i}\right|_{\hat{y}=w\cdot x^i} x^i \tag{29}$$

$$\nabla L_i(w') - \nabla L_i(w) = \left.\frac{\delta l(\hat{y}, y^i)}{\delta y^i}\right|_{\hat{y}=w'\cdot x^i} x^i - \left.\frac{\delta l(\hat{y}, y^i)}{\delta y^i}\right|_{\hat{y}=w\cdot x^i} x^i \tag{30}$$

$$\nabla L_i(w') - \nabla L_i(w) = \left(\left.\frac{\delta l(\hat{y}, y^i)}{\delta y^i}\right|_{\hat{y}=w'\cdot x^i} - \left.\frac{\delta l(\hat{y}, y^i)}{\delta y^i}\right|_{\hat{y}=w\cdot x^i}\right) x^i \tag{31}$$

$$\left\|\nabla L_i(w') - \nabla L_i(w)\right\| = \left|\left.\frac{\delta l(\hat{y}, y^i)}{\delta y^i}\right|_{\hat{y}=w'\cdot x^i} - \left.\frac{\delta l(\hat{y}, y^i)}{\delta y^i}\right|_{\hat{y}=w\cdot x^i}\right| \left\|x^i\right\| \tag{32}$$

Using the Lipschitz bound on the derivative of l yields and the bound of $\left\|x^i\right\|$:

$$\left\|\nabla L_i(w') - \nabla L_i(w)\right\| \leq \left|w' \cdot x^i - w \cdot x^i\right| H^* \tag{33}$$

$$\left\|\nabla L_i(w') - \nabla L_i(w)\right\| \leq \left|(w' - w) \cdot x^i\right| H^* \tag{34}$$

Summing over all i:

$$\left\|L_{1\ldots m}(w') - L_{1\ldots m}(w)\right\| \leq \frac{1}{m}\sum_{i=1}^{m}\left\|\nabla L_i(w') - \nabla L_i(w)\right\| \tag{35}$$

$$\leq \frac{1}{m}\sum_{i=1}^{m}\left|(w'-w)\cdot x^i\right| H^* \tag{36}$$

$$\frac{\left\|\nabla L_i(w') - \nabla L_i(w)\right\|}{\|w'-w\|} \leq \frac{1}{m}\sum_{i=1}^{m}\frac{\left|(w'-w)\cdot x^i\right|}{\|w'-w\|}H^* \tag{37}$$

$$H = \max_{w\neq w'}\frac{\left\|\nabla L_i(w') - \nabla L_i(w)\right\|}{\|w'-w\|} \tag{38}$$

$$H \leq \max_{w\neq w'}\frac{1}{m}\sum_{i=1}^{m}\frac{\left|(w'-w)\cdot x^i\right|}{\|w'-w\|}H^* \tag{39}$$

Suppose now that for all i, $x^i_j > 0$. Thus, for all j, if we wish to upper bound the left hand side, we can assume without loss of generality, $w'_j - w_j \geq 0$, and remove the absolute values.

$$H \leq \max_{w\neq w'}\frac{1}{m}\sum_{i=1}^{m}\frac{(w'-w)\cdot x^i}{\|w'-w\|}H^* \tag{40}$$

$$H \leq \max_{w\neq w'}\frac{H^*}{m\,\|w'-w\|}(w'-w)\cdot\sum_{i=1}^{m}x^i \tag{41}$$

$$\tag{42}$$

Replace $w - w'$ with a vector v:

$$H \leq \max_{v\neq 0}\frac{H^*}{m\,\|v\|}v\cdot\sum_{i=1}^{m}x^i \tag{43}$$

$$H \leq \max_{v\neq 0}\frac{H^*}{\|v\|}v\cdot\overline{x} \tag{44}$$

Using Cauchy-Schwarz:

$$H \leq \max_{v\neq 0}\frac{H^*}{\|v\|}\|v\|\,\|\overline{x}\| \tag{45}$$

$$H \leq \max_{v\neq 0}H^*\,\|\overline{x}\| \tag{46}$$

$$H \leq H^*\,\|\overline{x}\| \tag{47}$$

■

C One Shuffle Mini-Batch

The impractical aspect of mini-batch to apply to a disk drive is to grab a bunch of independent elements. If one first divides the data into mini-batches, and these mini-batches are sufficiently large such that seek time does not dominate the time to read the batch data.

Algorithm 3 One Shuffle Mini-Batch

```
**Input:** Cost functions $L_1 \ldots L_m$, mini-batch size $m'$, iterations $T$, initial $w \in \mathbf{R}^n$.
If not already shuffled, shuffle $L_1 \ldots L_m$.
**for** $t = 1$ **to** $T$ **do**
    Draw $i^*$ from $0 \ldots (m/m' - 1)$.
    $w \leftarrow w - \eta\nabla f_{(i^*m'+1)\ldots(i^*m'+m')}(w)$
**end for**
```

The average of the gradients of the mini-batches are still correct, so running stochastic gradient descent on them for a sufficiently small gradient will work well. However, what is difficult is understanding what kind of win is obtained in terms of the variances of the gradients: moreover, it may be necessary to bound this over the entire domain of the parameters, which might prove difficult.