Stochastic Repeated Gradient Descent (SRGD)

Jeff Hajewski
College of Arts and Sciences
University of Iowa
Iowa City, Iowa
Email: jeffrey-hajewski@uiowa.edu

***Abstract*—Stochastic Repeated Gradient Descent (SRGD) is a derivative of mini-batched SGD that performs multiple parameter updates for a given mini-batch within a single training epoch. This approach results in dramatic speedups for training large datasets without loss to accuracy. Additionally, even faster results were achieved using an adaptive step-size suggested by [ref-here].**

1. Introduction

We introduce Stochastic Repeated Gradient Descent(SRGD), a modification to Stochastic Gradient Descent that takes advantage of CPU cache to improve compute time and allow the use of large mini-batches, resulting in dramatically faster time to convergence versus mini-batch SGD.

Stochastic Gradient Descent (SGD) has long been the standard optimization method in machine learning. Recent developments such as Adam and AdaGrad solve some of the shortcomings of SGD by using adaptive learning rates and scaled gradients. While these methods have seen substantial success, they can suffer from increased storage requirements, which is problematic for large problems. Additionally, they can be more challenging to implement, as the algorithms are more complex than standard SGD. Stochastic Repeated Gradient Descent achieves dramatic gains in compute time by taking advantage of CPU cache and allowing the use of larger mini-batch sizes without degrading convergence results. Further, for the mini-batch case of a single data point, SRGD reduces to standard SRGD, with a marginally slower compute time. If SRGD is given an adaptive learning rate, such as that proposed by Yang et. al. [ref-here], performance is consistently better across the board.

1.1. Background

Formally, we are solving the problem

$$\theta = \arg\min_{\theta} \sum_{i=1}^{n} ||f(x_i;\theta) - y_i||_2^2 \tag{1}$$

where we have n number of data points, $x_i \in \mathbb{R}^d$ and corresponding labels $y_i \in \mathbb{R}$, and $f : \mathbb{R}^d \to \mathbb{R}$. We assume $f(x)$ is a linear function given by 2

$$f(x;\theta) = \sum_{i=1}^{d} \theta_i x_i \tag{2}$$

1.1.1. Stochastic Gradient Descent. Stochastic Gradient Descent is a derivative of Gradient Descent where the gradient $\nabla f(x)$ is replaced by $\nabla f_i(x)$, an approximation of the gradient calculated by computing the gradient at a randomly selected point, determined by uniformly sampling (without replacement) over the data set. The parameter update is then performed via the stand update 3

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla f_i(x) \tag{3}$$

However, this approach has a couple of drawbacks. The main drawback is due to an increased variance in the approximate gradient from calculating the gradient at only a single point. This results in slowing the rate of convergence when compared to Gradient Descent. A number of approaches attempt to reduce this variance [refs-here], but the simplest and most common approach to reduce the variance is mini-batching.

1.1.2. Mini-Batch SGD. Mini-batch SGD randomly samples some pre-determined number of points (say b points, and without loss of generality let $m = \frac{n}{b}$, such that b is a multiple of the number of points) from a uniform distribution of the data, without replacement. This mini-batch is used to attain a better (more stable) approximation of the approximate gradient, which is then used in the weight update 1.

1.2. Stochastic Repeated Gradient Descent

SRGD improves upon mini-batch SGD by using recently cached data to perform several updates of the weights via 1 before sampling new data. By not sampling new data at each update, the mini-batch data remains in cache. While this increases the variance in the theoretical convergence of our method, the compute time is dramatially reduced. Additionally, we use a dynamic learning rate, which is geometrically

Algorithm 1

```
Require: $r \geq 1$ and $b \geq 1$
1: for $i = 0$ to $\frac{n}{b}$ do
2:     $x \leftarrow x_{I_i}$
3:     for $j = 0$ to $r$ do
4:         $\theta_{k+1} \leftarrow \theta_k - \alpha \nabla f(x)$
5:     end for
6:     if $||\theta_k - \theta_{k-1}|| < \epsilon$ then
7:         $\alpha \leftarrow \frac{\alpha}{2}$
8:     end if
9: end for
```

decreased by a factor of 0.5 when successive weight updates are within some threshold value ϵ of eachother.

2. Algorithm

Algorithm 1 details the SRGD algorithm. Additionally, we check for convergence and stop early if convergence as been achieved. The key step is repeating with weight update r times using the same mini-batch data. This results in r approximate gradient computations along with r weight updates. After the weights are updated, we check to see how close they are to the prior weights and if they are close enough (i.e., convergence is slowing down), we decrease the learning rate. This approach allows us to start out with a coarser learning rate while avoiding over-shooting the minimum. By default, SRGD doubles the input learning rate to make it even coarser. In the dynamic version, of course this is offset by dynamically updating the learning rate based on rate of convergence (measured by the distance between prior weight updates).

3. Theory

4. Experiment

Experiments were performed on a 3.4GHz Intel i7-6700K hexa-core workstation with 64GB of RAM. In addition to measuring the time to convergence, we also measure the number of steps taken to converge to the solution (where a step is considered a weight update).

4.1. Problem Definition

The labeled data is generated via the function shown in equation 4,

$$y(x; \omega) = \sum_{i=1}^{d} \omega_i x_i \tag{4}$$

where $\omega \in [0, 10]$ and $x \in [-10, 10]$. While this is a small problem, it is easy to control and measure approximation error (since we know the coefficients exactly. The coefficients, ω, are sample from the distribution $U(0, 10)$ while the x values are generated by sample x_i (the i-th component of a given x) from the distribution $U(-10, 10)$.

TABLE 1. BATCH-SIZE OF 1 IN 50 DIMENSIONS

Optimizer	$r = 1$		$r = 5$		$r = 10$	
	Error	Time	Error	Time	Error	Time
SGD	0.0020	14.1	0.0020	14.1	0.0020	14.1
ASSGD	0.4435	29.0	0.4435	29.0	0.4435	29.0
SRGD	0.0001	9.8	0.0001	6.3	0.0000	6.3
ASSRGD	0.0090	25.1	0.0009	7.9	0.0002	6.4

TABLE 2. BATCH-SIZE OF 50 IN 50 DIMENSIONS

Optimizer	$r = 1$		$r = 5$		$r = 10$	
	Error	Time	Error	Time	Error	Time
SGD	1.1893	4.9	1.1893	4.9	1.1893	4.9
ASSGD	1.3576	5.0	1.3576	5.0	1.3576	5.0
SRGD	0.084	5.0	0.0000	2.8	0.0000	1.7
ASSRGD	0.0245	4.8	0.0001	3.3	0.0000	2.5

4.2. Methodology

We look at severl metrics in assessing the validity and performance of SRGD. The first metric is run time. This is simply the time it takes for comparably tuned algorithms to converge to the solution. Our experiments consider four algorithms: SGD, ASSGD, SRGD, ASSRGD. The ASSGD algorithm proposed by Yang et al. [ref-here] is the adaptive learning rate algorithm mentioned in the preceding section. The SRGD runs do not use the adpative learning rate while the ASSRGD runs do use an adaptive learning rate.

Table [REF HERE] shows the parameters of the runs we performed. Most importantly, we tested our methodology for parameters spaces of dimension 10, 50, and 1000.

4.3. Results

Based on our experiments there appear to be two domains in which SRGD is optimal. In one domain, typically as the mini-batch size approaches the problem size, SRGD takes more compute time achieves materially better accuracy. In the other domain, SRGD runs at a fraction of the time of SGD and achieves better accuracy (frequently an order of magnitude better).

5. Future Work

There are two key next steps in assessing this mehtod. The first is to increase the compute time of the gradient. The reasoning behind this approach is to perform a cost-benefit analysis of cache and additional information versus additional compute time. In other words, there is likely a tipping point where the additional information gained by the repeated weight updates is offset by the time required to compute the gradient. Whereas standard SGD extracts less information from each data point per weight update than SRGD in a shorter period of time, SRGD extracts more information but requires more time to do this. For the

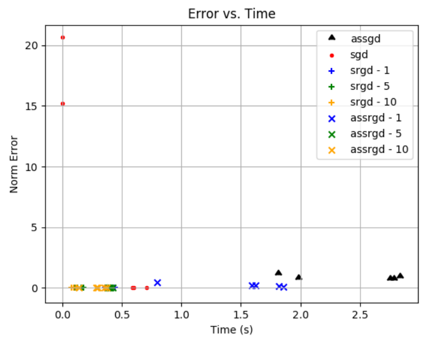

Figure 1. Comparison of error as a function of compute time for a single data point and a 10 dimensional parameter space.

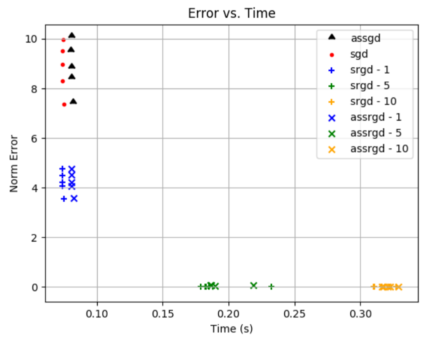

Figure 2. Comparison of error as a function of compute time for a mini-batch size of 50 points and a 10 dimensional parameter space.

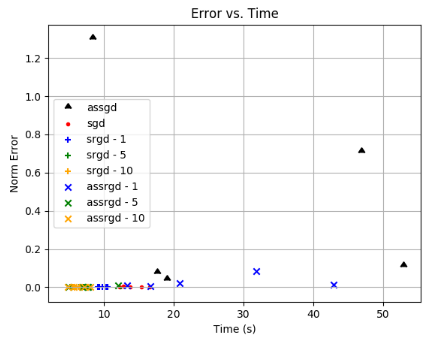

Figure 3. Comparison of error as a function of compute time for a single data point and a 50 dimensional parameter space.

results seen in this paper, that time tradeoff is offset by the additional information and the help of CPU cache for data access. However, one would expect there is a point where the additional time ultimately increases the total compute time of SRGD such that SGD is actually faster.

Acknowledgments

The authors would like to thank...

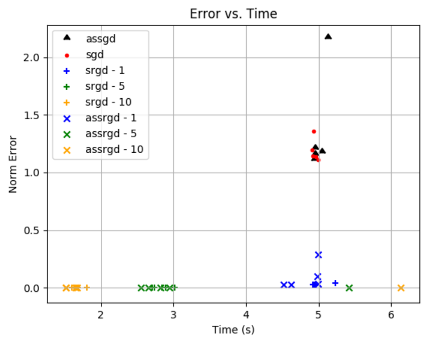

Figure 4. Comparison of error as a function of compute time for a mini-batch size of 50 points and a 50 dimensional parameter space.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.