Analysis of Alternatives to Stochastic Gradient Descent

Omid Sadeghi Meibodi
Department of Electrical Engineering
University of Washington
Seattle, WA 98195
omids@uw.edu

Project Final Report

Introduction

Consider the following optimization problem:

$$\min_{\omega} P(\omega); \qquad P(\omega) = \frac{1}{n}\sum_{i=1}^{n} \psi_i(\omega) \tag{1}$$

A standard method for optimizing sums of convex functions is Gradient Descent (GD) which has the following update rule at each step:

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla P(\omega^{(t-1)}) = \omega^{(t-1)} - \frac{\eta_t}{n}\sum_{i=1}^{n} \nabla \psi_i(\omega^{(t-1)})$$

Even though Gradient Descent has a linear convergence rate, one major drawback of GD is that it requires evaluation of n derivatives at each step where n is the number of training samples. This is very computationally demanding and therefore, for problems with large datasets, it's almost infeasible to use Gradient Descent.
In order to overcome this problem, Stochastic Gradient Descent (SGD) can be used which has the following update rule:

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t \nabla \psi_{i_t}(\omega^{(t-1)})$$

where i_t is drawn uniformly random from $\{1, \ldots, n\}$.
Since $\mathbb{E}[\nabla \phi_{i_t}(\omega^{(t-1)})] = \frac{1}{n}\sum_{i=1}^{n} \nabla \phi_i(\omega^{(t-1)})$, SGD update rule is similar to GD step in expectation. Moreover, as it could be easily seen, SGD step just requires a single derivative evaluation which reduces the computational cost by a factor of n compared to GD. However, a downside of SGD is that despite the fact that it agrees with GD in expectation, it is not the same at each step and this in turn leads to introducing variance. SGD has a sublinear convergence rate.
There are a lot of algorithms that aim to reduce this inherent variance of SGD. For instance, as we saw in one of the homework problems, we can vary SGD by evaluating the gradient for a few examples instead of just one so that the gradient approximation has a lower variance.
In what follows, I will explore a few of alternatives to SGD which are introduced in [1], [2] and [3].

SDCA

As an alternative algorithm, we can use a primal-dual method called Stochastic Dual Coordinate Ascent (SDCA). In order to do so, we first need to derive the dual problem of 1.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Assume $\psi_i(\omega) = \phi_i(\omega^T x_i) + \frac{\lambda}{2}||\omega||^2$. Optimization problem 1 is equivalent to:

$$\min_{\omega} \frac{1}{n}\sum_{i=1}^{n}\phi_i(y_i) + \frac{\lambda}{2}||\omega||^2; \qquad \frac{1}{n}y_i = \frac{1}{n}\omega^T x_i \tag{2}$$

The Lagrangian dual of problem 2 could be derived as follows:

$$\begin{aligned}
D(\alpha) &= \inf_{\omega, y_i}\big(\frac{1}{n}\sum_{i=1}^{n}\phi_i(y_i) + \frac{\lambda}{2}||\omega||^2 + \sum_{i=1}^{n}\alpha_i(\frac{1}{n}y_i - \frac{1}{n}\omega^T x_i)\big) \\
&= \inf_{y_i}\big(\frac{1}{n}\sum_{i=1}^{n}(\phi_i(y_i) + \alpha_i y_i)\big) + \inf_{\omega}\big(\frac{\lambda}{2}||\omega||^2 - \frac{1}{n}\sum_{i=1}^{n}\alpha_i\omega^T x_i\big) \\
&= -\frac{1}{n}\sup_{y_i}\sum_{i=1}^{n}\big(-\alpha_i y_i - \phi_i(y_i)\big) + \inf_{\omega}\big(\frac{\lambda}{2}||\omega||^2 - \frac{1}{n}\sum_{i=1}^{n}\alpha_i\omega^T x_i\big) \\
&= -\frac{1}{n}\sum_{i=1}^{n}\sup_{y_i}\big(-\alpha_i y_i - \phi_i(y_i)\big) + \inf_{\omega}\big(\frac{\lambda}{2}||\omega||^2 - \frac{1}{n}\sum_{i=1}^{n}\alpha_i\omega^T x_i\big) \\
&= \frac{1}{n}\sum_{i=1}^{n}-\phi_i^*(-\alpha_i) + \inf_{\omega}\big(\frac{\lambda}{2}||\omega||^2 - \frac{1}{n}\sum_{i=1}^{n}\alpha_i\omega^T x_i\big)
\end{aligned}$$

By taking the gradient of the second term and setting it to zero, we would have:

$$\nabla_\omega\big(\frac{\lambda}{2}||\omega||^2 - \frac{1}{n}\sum_{i=1}^{n}\alpha_i\omega^T x_i\big) = \lambda\omega - \frac{1}{n}\sum_{i=1}^{n}\alpha_i x_i = 0 \Rightarrow \omega = \frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i \tag{3}$$

$$\begin{aligned}
D(\alpha) &= \frac{1}{n}\sum_{i=1}^{n}-\phi_i^*(-\alpha_i) + \inf_{\omega}\big(\frac{\lambda}{2}||\omega||^2 - \frac{1}{n}\sum_{i=1}^{n}\alpha_i\omega^T x_i\big) \\
&= \frac{1}{n}\sum_{i=1}^{n}-\phi_i^*(-\alpha_i) + \big(\frac{\lambda}{2}||\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i||^2 - \frac{1}{n}\sum_{i=1}^{n}\alpha_i(\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i)^T x_i\big) \\
&= \frac{1}{n}\sum_{i=1}^{n}-\phi_i^*(-\alpha_i) - \frac{\lambda}{2}||\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i||^2
\end{aligned}$$

Thus, the dual problem for the optimization problem 1 is:

$$\max_{\alpha\in\mathbb{R}^n}\frac{1}{n}\sum_{i=1}^{n}-\phi_i^*(-\alpha_i) - \frac{\lambda}{2}||\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i||^2 \tag{4}$$

If ϕ_i's are convex, SDCA uses Coordinate Ascent method (randomly choosing one coordinate at each iteration and optimizing the dual function with respect to that) to maximize the dual function. At each iteration, only one entry of α is updated and ω is also computed using 3. The pseudocode for SDCA algorithm is as follows:

- Randomly initialize $\alpha(0)$
  Let $\omega(0) = \omega(\alpha(0))$
- Iterate: for t=1,. . . ,T:
  Randomly pick i
  Find $\Delta\alpha_i$ to maximize $-\phi_i^*\big(-(\alpha_i^{(t-1)} + \Delta\alpha_i)\big) - \frac{\lambda n}{2}||\omega^{(t-1)} + \frac{1}{\lambda n}\Delta\alpha_i x_i||^2$
  $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \Delta\alpha_i e_i$
  $\omega^{(t)} \leftarrow \omega^{(t-1)} + \frac{1}{\lambda n}\Delta\alpha_i x_i$
- Output:
  Let $\bar{\alpha} = \alpha^{(t)}$ and $\bar{\omega} = \omega^{(t)}$ for some random $t \in T_0 + 1, \ldots, T$

- Return $\bar{\omega}$

The goal is to minimize the duality gap defined as $\mathbb{E}[P(\bar{\omega}) - D(\bar{\alpha})]$ where $\bar{\omega}$ and $\bar{\alpha}$ are the outputs of the algorithm.
If ϕ_i's are convex, it is known that the duality gap is zero and therefore, the optimal ω^* could be computed in the following way:

$$\omega^* = \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i^* x_i$$

where α^* is the optimal point of the optimization problem 4.
It has been proved that in order to achieve a duality gap less than ϵ, the SDCA algorithm converges with the rate of $O(n + \frac{1}{\lambda\epsilon})$ for Lipschitz functions and $O\big((n + \frac{1}{\lambda})log(\frac{1}{\epsilon})\big)$ for smooth functions, whereas SGD has a convergence rate of $(\frac{1}{\lambda\epsilon})$ in all cases. Therefore, SDCA has a superior theoretical guarantee.

SAG

Assume $\psi_i(\omega) = \phi_i(\omega^T x_i) + \frac{\lambda}{2}||\omega||^2$. Each iteration of the Stochastic Average Gradient (SAG) is as follows:

$$\omega^{(t+1)} = \omega^{(t)} - \frac{\eta_k}{n} \sum_i y_i^{(t)} \tag{5}$$

where $y_i^{(t)} = \nabla\psi_i(\omega^{(t-1)})$ if $i = i_t$, $y_i^{(t)} = y_i^{(t-1)}$ otherwise and i_t is randomly drawn from $\{1, \ldots, n\}$.
As it could be seen, each SAG iteration is the mixture of SGD and GD updates. Similar to GD, each step incorporates a gradient with respect to each of the functions. But, just as the SGD algorithm, each iteration involves just a single gradient evaluation. So, this algorithm aims to incorporate the low variance property of GD and low cost per iteration feature of SGD. The only additional cost here is that a memory of the most recent gradient evaluation for each function needs to be maintained. It can be proved that SAG achieves a linear convergence rate for strongly convex P.

SVRG

One major drawback of SGD is that in order to ensure convergence, the step length has to decay to zero. This requirement is due to the high variance of SGD. One way to overcome this problem is through maintaining $\tilde{\omega}$ which is a version of estimated ω that is close to the optimal point. For instance, we can keep the output of SGD update after every m iterations as $\tilde{\omega}$. Another additional cost is to keep the average gradient $\tilde{\mu} = \nabla P(\tilde{\omega}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{\omega})$.
The SVRG update rule is as follows:

$$\omega^{(t)} = \omega^{(t-1)} - \eta_t(\nabla f_{i_t}(\omega^{(t-1)}) - \nabla f_{i_t}(\tilde{\omega}) + \tilde{\mu}) \tag{6}$$

where i_t is randomly drawn from $\{1, \ldots, n\}$.

The SVRG algorithm performs in the following way:

- **Parameters** update frequency m and learning rate η
- **Initialize** $\tilde{\omega}_0$
- **Iterate**: for $s = 1, 2, \ldots$
  $\tilde{\omega}_0 = \tilde{\omega}_{s-1}$
  $\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \nabla\psi_i(\tilde{\omega})$
  $\omega_0 = \tilde{\omega}$
  Iterate: for $t = 1, \ldots, m$
  Randomly pick $i_t \in \{1, \ldots, n\}$ and update weight
  $\omega^{(t)} = \omega^{(t-1)} - \eta(\nabla\psi_{i_t}(\omega^{(t-1)}) - \nabla\psi_{i_t}(\tilde{\omega}) + \tilde{\mu})$
  end
  Set $\tilde{\omega}_s = \omega_t$ for randomly chosen $t \in \{0, \ldots, m-1\}$
  end

As it could be seen by the pseudo-code, $\tilde{\omega}$ is updated every m iterations and this in turn leads to lower variance for this algorithm. In fact, we have:

$$\begin{aligned}\mathbb{E}[\nabla\psi_{i_t}(\omega^{(t-1)}) - \nabla\psi_{i_t}(\tilde{\omega}) + \tilde{\mu}] &= \frac{1}{n}\sum_{i=1}^{n}\big(\nabla\psi_i(\omega^{(t-1)}) - \nabla\psi_i(\tilde{\omega}) + \nabla\psi_i(\tilde{\omega})\big) \\ &= \frac{1}{n}\sum_{i=1}^{n}\nabla\psi_i(\omega^{(t-1)}) \\ &= \nabla P(\omega^{(t-1)})\end{aligned}$$

So, like SGD, the update rule of SVRG is similar to GD in expectation. However, using this method, we have also reduced the variance compared to SGD, because if both $\tilde{\omega}$ and $\omega^{(t)}$ converge to the optimal point ω^*, then we would have:

$$\nabla\psi_{i_t}(\omega^{(t-1)}) - \nabla\psi_{i_t}(\tilde{\omega}) + \tilde{\mu} = \nabla\psi_{i_t}(\omega^{(t-1)}) - \nabla\psi_{i_t}(\tilde{\omega}) + \nabla P(\tilde{\omega}) \to \nabla\psi_{i_t}(\omega^*) - \nabla\psi_{i_t}(\omega^*) = 0$$

Since $\nabla\psi_{i_t}(\omega^{(t-1)}) - \nabla\psi_{i_t}(\tilde{\omega}) + \tilde{\mu}$ goes to zero asymptotically, the step length is no longer required to decay to zero and therefore, we can take larger steps which leads to faster convergence rate.
It can be shown that in case each ψ_i is convex and P is both smooth and strongly convex, SVRG enjoys a linear convergence rate as follows:

$$\mathbb{E}P(\tilde{\omega}_s) \leq \mathbb{E}P(\omega_*) + \alpha^s[P(\tilde{\omega}_0) - P(\omega_*)]$$

where $\alpha < 1$.

Experiments

As it was mentioned in the first part of the report, in case P is smooth and strongly convex, all three aforementioned algorithms achieve linear convergence rate on the training set. However, in Machine Learning applications, performance on the test set matters the most and the linear convergence rate for training set may not translate into the same rate for the test set. In order to evaluate these algorithms in practice, I implemented SGD, SAG and SVRG on MNIST dataset to find out which one performs the best (It is noteworthy that since it was shown in papers that SDCA and SAG have a quite similar performance, I decided to implement SAG which has been less investigated compared to SDCA that is already available in Python packages). Experiments were conducted in the following way:

- All 3 algorithm codes were written from scratch and without using any of existing solvers available in Python.
- The performance of algorithms were evaluated on MNIST dataset. This dataset consists of a large number of handwritten digits and the goal is to classify them into 10 classes of digits $0, 1, \ldots, 9$.
- All 3 algorithms were trained on the same training dataset and their test accuracy rate were measured on a subset of test set in the MNIST dataset.
- The objective loss function in all cases were the ℓ_2-regularized multiclass logistic regression with regularization parameter of $\lambda = 1e-4$.
- In multiclass logistic regression with a training set $\{(x_i, y_i) : i = 1, \ldots, n\}$ and $y_i \in \{0, \ldots, 9\}$, the following is true:

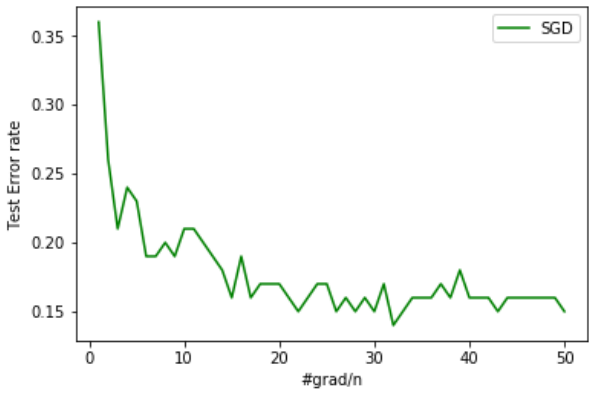
  $$P(y = k|x) = \frac{e^{\omega_k^T x}}{\sum_i e^{\omega_i^T x}} \quad i = 0, \ldots, 9$$

  Using the one-hot encoding of the labels, we would have:

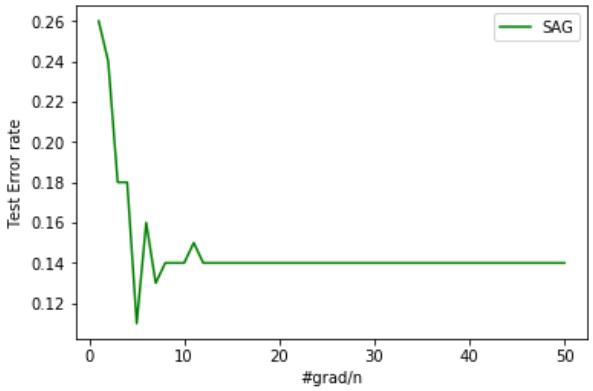
  $$\psi_i(\omega_0, \ldots, \omega_9) = \sum_{k=0}^{9} y_i[k] log P(k|x_i) \quad i \in \{1, \ldots, n\}$$
  $$\nabla_{\omega_k} \psi_i = (y_i[k] - P(k|x_i))x_i$$

- Each of the algorithms were simulated up to the point that the number of gradient evaluations divided by n (number of training samples) were equal to 50 (number of gradient evaluations could be considered as a measure of computational cost)
- $2\frac{\#of\ gradient\ computations}{n}$ is equivalent to 2n iterations of SGD, 2n iterations of SAG and 1 iteration of SVRG.

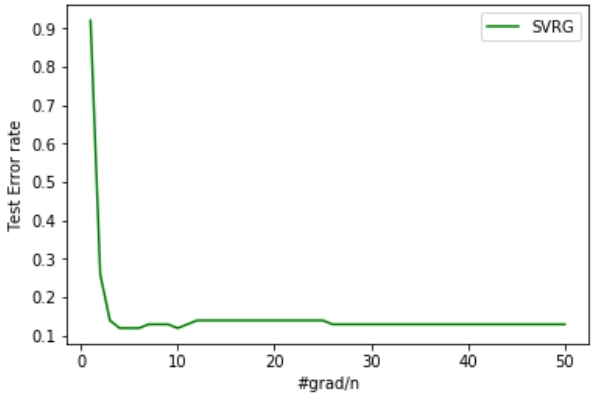
SGD: As it could be seen in the plot, the test error rate for the best tuned SGD (through cross validation) tends to fluctuate till the end and it never achieves an optimum test accuracy. Each weight update for SGD took $0.0036s$.



SAG: In case of SAG, the test error rate converges to it's optimum value pretty fast. Each weight update step of SAG took $0.064s$.



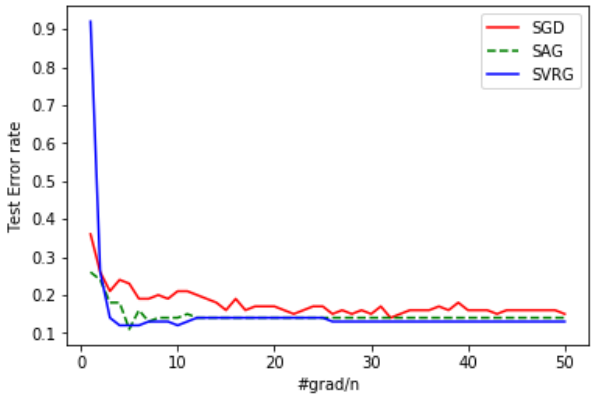
SVRG: For SVRG, after an initial phase in which the test error rate is very high, the algorithm reaches it's final value very fast. It took $3.965s$ for each weight update of SVRG to be completed. Note that $m = n$ where n is the number of training examples.



Comparison and Conclusion

By comparing the performance of the algorithms, we can conclude the following results:

- SAG and SVRG achieve lower test error rates compared to SGD.



- Each single update of the weights in SVRG took a lot longer than the updates of SGD and SAG, because it involves m+n gradient evaluations per update compared to single gradient computation in each update for SGD and SAG.
- Although each single update of SVRG took longer to be completed, the test error rate decreased the most after each update for SVRG. Therefore, SVRG was able to reach it's final test accuracy way faster than SAG.
- For SAG, it is needed to store the most recent gradient evaluation for each function. Therefore, for big datasets with a large number of training examples, SAG requires a huge amount of memory and therefore, it's not practical for those applications.

To wrap it up, it could be said that SVRG is the optimal algorithm among the evaluated algorithms because of the following reasons:

1. Unlike SAG and SDCA, SVRG does not require the storage of full gradients and therefore, it is more desirable for problems with big datasets and also instances where there is memory limitations.
2. SVRG converges faster to it's optimal test error rate compared to other algorithms.
3. The variance reduction idea used in SDCA is intuitive and it could be applied to more complex problems such as Neural Networks.

References

[1] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *J. Mach. Learn. Res.*, 14(1):567–599, February 2013.

[2] Nicolas Le Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2663–2671, USA, 2012. Curran Associates Inc.

[3] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pages 315–323, USA, 2013. Curran Associates Inc.