The Chinese University of HK

Summer Research Program

Multi-core implementation of stochastic variance reduced algorithms

Student:
LI Zhirong
SID:
1155092195

Supervisor:
Prof. James Cheng
Advisor:
Phd Yan Xiao

August 18, 2017

1 Introduction

Large-scale data, such as web pages, users and social connections, plays an essential role in data analysis, including user behavior analysis, ad targeting, recommendations and so on. Large-scale machine learning is a series of machine learning algorithms that are performed to extract information from large-scale data. Machine-learning practitioners use the data as a training set, to train an algorithm of one of the many types used by machine-learning practitioners, such as Bayes nets, support-vector machines, decision trees, hidden Markov models, and many others. Nowadays, the general way to process large-scale data is using MapReduce-like parallel-processing framework, which is a powerful tool developed at Google for extracting information from huge data. In addition, Stochastic Gradient Descent (SGD) is a popular algorithm, which is proved to be well suited to data-intensive machine learning tasks[1][2].

Parallel computing is gaining increasingly popularity due to necessity of high efficiency in large-scale machine learning and the emergence of inexpensive multicore processors. Some researchers proposed schemes to parallelize SGD, requiring memory locking and synchronization[3][4][5]. Some researchers proposed a better approach called HOGWILD!, without memory locking and synchronization[6]. In addition, stochastic dual coordinate ascent(SDCA) and Stochastic Average Gradient(SAG) are developed to accelerate SGD[8][7]. For smooth and strongly convex functions, Stochastic Variance Reduced Gradient(SVRG) is another better method to accelerate stochastic gradient descent using predictive variance reduction[9].

Logistic regression is the appropriate regression analysis to explain the relationship between one dependent binary variable and one or more nominal variables. Support Vector Machines(SVMs) are among the best off-the-shelf supervised learning algorithm in machine learning, which is to find the best hyperplane that represents the largest separation of data. To prove that those schemes are accurate and make a greater improve in time memory complexity, in this project, we will apply Gradient Descent(GD), Stochastic Gradient Descent(SGD), Stochastic Variance Reduced Gradient(SVRG) with and without lazy update on logistic regression and Support Vector Machine on sparse data set RCV1 and discuss the result on experiment.

2 Methods

2.1 GD & SGD

In machine learning, we ofen deal with the following optimization problem. Let ψ_i be the loss function and w be the weight vevtor that needs learning. Our goal is to minimize the total average loss of samples

$$\min \frac{1}{n}\sum_{i=1}^{n}\psi_i(w)$$

A standard method is gradient descent, described by the following update rule for t=1,2,....

$$w^{(t)} = w^{(t-1)} - \frac{\eta}{n}\sum_{i=1}^{n}\nabla\psi_i(w^{(t-1)})$$

In general, η is learning rate that will not change in training. A modification is stochastic gradient descent(SGD) for each iteration t=1,2,..., we draw i_t randomly from 1,...n, and update:

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

where learning rate η will decrease with the increase of iteration.

2.1.1 Logistic Regression[10]

In two-classes(1 and 0) logistic regression problem, we assume that the log likelihood ratio of the class-conditional densities $p(\mathbf{x}|C_i)$ is linear:

$$\log\frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} = w^T\mathbf{x} + w_0^0$$

Using Bayes' rule, we have:

$$\begin{aligned}\text{logit}(P(C_1|\mathbf{x})) &= \log\frac{P(C_1|\mathbf{x})}{1-P(C_1|\mathbf{x})}\\ &= \log\frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} + \log\frac{P(C_1)}{P(C_2)}\\ &= w^T\mathbf{x} + w_0\end{aligned}$$

where $w_0 = w_0^0 + \log\frac{P(C_1)}{P(C_2)}$.
Rearranging terms, we get the sigmoid function as our estimator of $P(C_1|\mathbf{x})$:

$$y = \hat{P}(C_1|\mathbf{x}) = \frac{1}{1+\exp[-(w^T\mathbf{x}+w_0)]}$$

.
The sample likelihood is:

$$\mathbf{I} = \prod_t (y^t)^{(r^t)}(1-y^t)^{(1-r^t)}$$

The loss function is:

$$\mathbf{E} = -\log\mathbf{I} = -\sum_t r^t \log y^t + (1 - r^t)\log(1 - y^t)$$

Full Gradient is:

$$\Delta w_j = \eta \sum_t (r^t - y^t) x_j^t, j = 1, ..., d$$

Stochastic Gradient is:

$$\Delta w_j = \eta (r^t - y^t) x_j^t, j = 1, ..., d$$

2.1.2 SVM

In two-class(1 and -1) classification problem, SVM model in general form is:

$$f(w) = \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{N} \max(0, 1 - y_i(w^T X_i + b))$$

where λ is a penalty factor to limit the length of weights.

Full gradient is:

$$\nabla f(w) = \lambda w + \sum_{i=1}^{N} \Phi(1 - y_i(w^T X_i + b) > 0)(-y_i x_i)$$

Stochastic gradient is:

$$\nabla f_i(w) = \lambda w + \Phi(1 - y_i(w^T X_i + b) > 0)(-y_i x_i)$$

where function Φ is defined by:

$$\Phi(a) = \begin{cases} 1 & \text{if a is true} \\ 0 & \text{if a is false} \end{cases}$$

2.2 Stochastic Variance Reduced Gradient[9]

In recent research, researchers show that SVRG does not require the storage of full gradient resulting in the fast convergence by explicitly connecting the idea to variance reduction in SGD. In addition, the relatively intuitive variance reduction explanation also applies to non-convex optimization problems, leading to a widely use for complex problems such as training deep neural networks.

The practical issue for SGD is slow convergence. So in SVRG, at each time, we keep a version of estimated w as $\tilde{w}$ that is close to the optimal w. Then maintain the average gradient after every specific SGD iterations:

$$\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \psi_i(\tilde{w}) \tag{1}$$

Generalized update rule for SVRG is follow: randomly draw i_t from 1,...n:

$$w^{(t)} = w^{(t-1)} - \eta_t(\nabla\psi_{i_t}(w^{(t-1)}) - \nabla\psi_{i_t}(\tilde{w}) + \tilde{\mu}) \tag{2}$$

In this method, the learning rate η_t does not have to decay, which leads to faster convergence as one can use a relatively large learning rate.

Procedure SVRG

Parameters update frequency m and learning rate η
Initialize $\tilde{w}_0$
Iterate: for $s = 1, 2, \ldots$
  $\tilde{w} = \tilde{w}_{s-1}$
  $\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n}\nabla\psi_i(\tilde{w})$
  $w_0 = \tilde{w}$
  Iterate: for $t = 1, 2, \ldots, m$
    Randomly pick $i_t \in \{1, \ldots, n\}$ and update weight
      $w_t = w_{t-1} - \eta(\nabla\psi_{i_t}(w_{t-1}) - \nabla\psi_{i_t}(\tilde{w}) + \tilde{\mu})$
  end
  option I: set $\tilde{w}_s = w_m$
  option II: set $\tilde{w}_s = w_t$ for randomly chosen $t \in \{0, \ldots, m-1\}$
end

Figure 1: Pseudo code for general SVRG[9]

2.3 Lazy Update

Recently, researchers present an update scheme called HOGWILD!, allowing processors access to shared memory, achieves a optimal rate of convergence when performing on sparse optimization problem.[6]. From equation (2), on sparse data, $\nabla\psi_{i_t}(w^{(t-1)})$ and $\nabla\psi_{i_t}(\tilde{w})$ are sparse, however, the full gradient $\tilde{\mu}$ in equation(1) is dense. This leads to uselessness of paralleling computing on weight update.

Fortunately, the idea of lazy update fix this issue[11]. Let S_i be a set in which of coordinate in instance X_i is not zero. From (2), each stochastic update reads from the set S_i but updates to every coordinate of weight. However, updates can be performed lazily only when they are required. Let $\rho(j), j = 0, 1, ..., d$ record the last time that μ is updated to w_j. Let τ_j be the times that μ needs to be updated in w_j. This allows the stochastic updates to only write to coordinates in S_i and defer writes to other coordinates. Procedure is described following.

Algorithm 1 Lazy Stochastic Updates for SVRG

Input: x; μ; T
Initialize $\rho(j) = 0, j = 0, ..., d$
for t=1:T **do**
    Randomly sample i $\in \{1, ..., n\}$
    x_{S_i}=read coordinates S_i from x
    for j $\in S_I$ **do**
        τ_j=t-$\rho(j)$-1
        $w_j \leftarrow w_j - \eta\tau_j\mu_j$
        $w_j \leftarrow w_j - \eta(\nabla\psi_{i_t}(w^{(t-1)})$ - $\nabla\psi_{i_t}(\tilde{w}))$
        $\rho(j) \leftarrow$t.

3 Experiment and Results

Figure 2: With cores increasing it converges faster

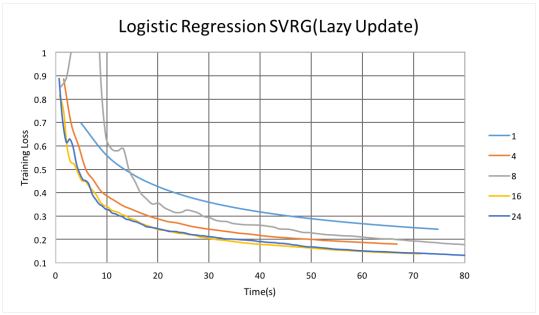

Figure 3: With cores increasing it converges faster but is a little unstable

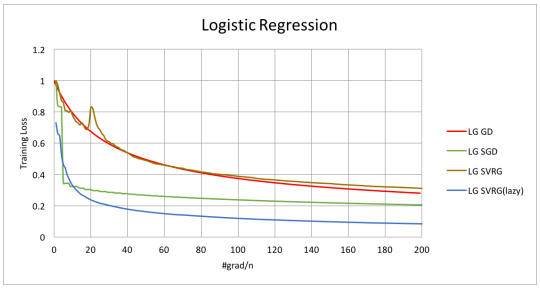

Figure 4: With lazy update, SVRG is more stable and converging faster

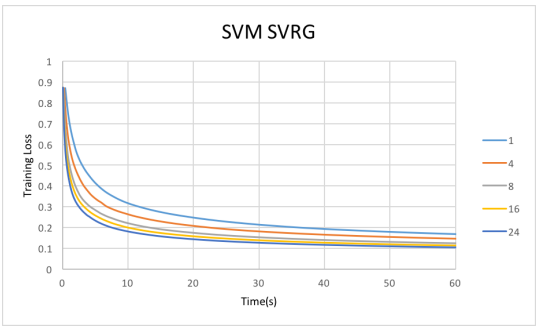

Figure 5: With cores increasing it converges faster

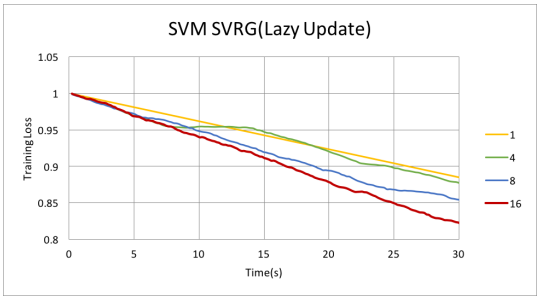

Figure 6: With cores increasing it converges faster but is a little unstable

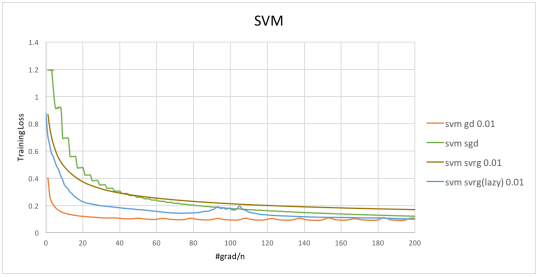

Figure 7: SVRG and lazy updated SVRG worked better than original SGD

Firstly, we performed multi-core binary class logistic regression on sparse dataset RCV1 with rate η=0.01(Fig2,3). When number of core increase, convergence was faster in SVRG(Fig.2). When lazy update was applied, convergence was also faster but it was not so smooth. When the number of cpu core is 4, lazy updated SVRG performed better than others(Fig.5). In stochastic gradient descent, the learning rate decay in such a decrease scheduling: $\frac{\alpha}{iteration*\beta+0.5} + 0.01$, where α and β was adjusted according to experiment(In this experiment, α=8 and β=1/50).

Secondly, we performed SVM on RCV1 with learning rate η=0.01 and regularization parameter λ=1e-4(Fig5,6). As a result, when number of cpu core increase, convergence was a little faster but it is not apparent. In addition, the lazy update

also helped in convergence. In stochastic gradient descent, the decrease scheduling was the same as logistic regression, where α=50 and β=200. α. When the number of core is 4, We found that SVRG and lazy updated SVRG worked better than original SGD in the beginning, especially lazy updated SVRG(Fig.7).

4 Conclusion

In this research, we learned about stochastic variance reduced algorithms, such as HOGWILD!, SVRG, lazy update. We also implemented the multi-core version of logistic regression and SVM. From the result we can see that increasing the cores will reduce the time for convergence.

References

[1] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 2nd edition, 1999.

[2] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, 2008.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, Belmont, MA, 1997.

[4] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. *Optimal distributed online prediction using mini-batches.* Technical report, Microsoft Research, 2011.

[5] J. Duchi, A. Agarwal, and M. J. Wainwright. Distributed dual averaging in networks. In *Advances in Neural Information Processing Systems,* 2010.

[6] F. Niu, B. Recht, C. Re and S. J. Wright. *Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent.* Computer Sciences Department, University of Wisconsin-Madison, 2011

[7] Nicolas Le Roux, Mark Schmidt, and Francis Bach. A Stochastic Gradient Method with an Ex- ponential Convergence Rate for Strongly-Convex Optimization with Finite Training Sets. *arXiv preprint arXiv:1202.6258,* 2012.

[8] C.J. Hsieh, K.W. Chang, C.J. Lin, S.S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, pages 408415, 2008.

[9] R. Johnson, T. Zhang. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *Advances in Neural Information Processing Systems 26*, 2013.

[10] E. Alpaydn.*Introduction to Machine Learning.*The MIT Press Cambridge, Massachusetts London, England, Second Edition.

[11] X. Pan, M. Lam, S. Tu, D. Papailiopoulos, C. Zhang, M.I. Jordan, K. Ramchandran, C. Re, B. Recht. CYCLADES:Conflict-free Asynchronous Machine Learning. *arXiv:1605.09721v1*, 2016.

Appendices

A Pseudo-code for SVRG Logistic Regression

Parameters: update frequency m and learning rate η
$w \leftarrow rand(-0.01, 0.01)$
$\tilde{w} \leftarrow rand(-0.01, 0.01)$
For s=1,2,...
  $\mu \leftarrow 0.0$
  For i=1,...n
    o $\leftarrow \tilde{w}^T x_i$
    y $\leftarrow$ sigmoid(o)
    $\mu \leftarrow \mu + (r^i - \text{y}) x_i$
  $\mu \leftarrow \mu/\text{n}$
  $w = \tilde{w}$
  For t=1,2,...,m
    Randomly pick $i_t \in$ {1,...n}
    o1 $\leftarrow w^T x_{i_t}$
    o2 $\leftarrow \tilde{w}^T x_{i_t}$
    y1 $\leftarrow$ sigmoid(o1)
    y2 $\leftarrow$ sigmoid(o2)
    gra1 $\leftarrow (r^{i_t} - \text{y1}) x_{i_t}$
    gra2 $\leftarrow (r^{i_t} - \text{y2}) x_{i_t}$
    $w_t \leftarrow w_t + \eta(\text{gra1} - \text{gra2} + \mu)$
  $\tilde{w} \leftarrow w$