Variance Reduced Stochastic Optimization for PCA and PLS

Erxue Min, Yawei Zhao, Jun Long, and Jianping Yin

College of Computer, National University of Defense Technology,
Changsha, China 410073

Abstract. Principal Component Analysis(PCA) is a dimensionality reduction technique which extracts the representative components of the data. Partial Least Squares(PLS) models the covariance structure between a pair of data matrices. The objective function of the two problems are similar and thus can often be solved by identical algorithms. Deterministic methods suffers from prohibitive computation cost in large-scale applications, while the stochastic algorithms fail to achieve high-accuracy solutions. In this paper, we propose stochastic optimization methods with variance reduction to solve PCA and PLS, which ensure to obtain high-accuracy solutions with enough computation cost and rapidly converge to an approximate optima with few iterations. Extensive experiments demonstrate the significant performance of our method.

Keywords: PCA, PLS, SAGA

1 Introduction

Principal component analysis (PCA) is a popular dimensionality reduction technique widely used in many areas, such as machine learning, statistics, computer vision [1]. The goal of PCA is to find the maximal (uncentered) variance k-dimensional subspace with respect to a $d \times n$ matrix $X = (\mathbf{x}_1, ..., \mathbf{x}_n)$, where $\mathbf{x}_i \in \mathbb{R}^d$ with $i \in \{1, 2, ..., n\}$. It is equivalent to solve the following optimization problem:

$$\begin{aligned} \underset{W}{\text{maximize}} \quad & \text{Trace}(W^\top (\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i{}^\top) W) \\ \text{subject to} \quad & W^\top W = I. \end{aligned} \tag{1}$$

The $d \times k$ matrix W is used to parameterize the subspace. Partial least squares (PLS) [2], a ubiquitous method for bilinear factor analysis, is widely used in information retrieval [3] and machine learning. It solves the following problem: given a dataset of n samples including two sets of features, the $d_x \times n$ matrix $X = (\mathbf{x}_1, ..., \mathbf{x}_n)$ and the $d_y \times n$ matrix $Y = (\mathbf{y}_1, ..., \mathbf{y}_n)$, what is the k-dimensional subspace that captures most of the covariance between the two views. The PLS

problem can be expressed as:

$$\begin{aligned} \underset{U,V}{\text{maximize}} \quad & \text{Trace}(U^\top(\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i{\mathbf{y}_i}^\top)V) \\ \text{subject to} \quad & U^\top U = I, V^\top V = I \end{aligned} \tag{2}$$

The pair of matrices $U \in \mathbb{R}^{d_x \times k}$ and $V \in \mathbb{R}^{d_y \times k}$ are the solution of PLS. It is obvious that the objective functions of PCA and PLS are quite similar and PCA is a special case of PLS, so they can be solved through the same algorithms.

It is well known that the subspaces can be given by applying the singular value decomposition to the $d_x \times d_x$ covariance matrix $XX^\top$ for PCA and the $d_x \times d_y$ cross-covariance matrix $XY^\top$ for PLS. However, the exact singular value decomposition is infeasible for big-data scenarios as its required runtime is $O(\min(k^2d, kd^2))$ for a $k \times d$ matrix. In recent years, studies on solving k-SVD have made many breakthroughs [4][5][6], becoming great options to solving such the covariance matrices. However, in this paper, we only focus on objective functions like problem (1) and (2), whose covariance matrix can be approximated by a random rank-1 update. Standard deterministic approaches based on power iterations are accurate but require a full pass over the entire dataset, which is also expensive in the "data laden" regime. Recently, stochastic optimization algorithms have developed a lot to deal with massive data. The simplest stochastic algorithm for PCA and PLS are the stochastic power methods such as Oja algorithm [7] and Hebbian algorithm [8]. Another straightforward approach is the incremental method [9], which can be implemented efficiently. Besides, the online algorithms such as Matrix Stochastic Gradient(MSE) and Matrix Exponentiated Gradient(MEG) [10][11] have been proposed, with great theoretical guarantees. In contrast to the deterministic algorithm, these algorithm randomly sample some examples to update the parameters. The drawback of stochastic algorithms is that the noise caused by stochastic sample means they converge sub-linearly and have difficulty in obtaining a high-accuracy solution. In recent years, variance reduced stochastic gradient algorithms such as SAG [12], SVRG[13], SAGA [14] has been proposed to solve unconstrained problems, using cheap stochastic iterations but converge linearly to a given accuracy. In this paper, we apply the SAGA algorithm to optimize PCA and PLS, proposing two novel stochastic algorithms, VR-PCA+ and VR-PLS+.

2 Related Works

Stochastic gradient descent (SGD) is a simple but efficient optimization method, which is commonly used in a variety of unconstrained optimization problems. Although the objective function of PCA is constraint, the stochastic gradient descent still works. The variant of SGD, called stochastic power method, was proposed in [9] to solve PCA. It takes the following update rule in each iteration:

$$W_{t+1} = P_{orth}(W_t - \eta_t \mathbf{x}_t \mathbf{x}_t^\top W_t). \tag{3}$$

Note that η_t is the learning rate, usually adopting a decaying strategy. The runtime of calculating $\mathbf{x}_t\mathbf{x}_t^\top W$ is only $O(kd)$, which is rather cheap. P_{orth} indicates the normalization step such as Gram-Schmidt procedure to ensure the orthogonal condition $W^\top W = I$. The orthogonalization procedure requires runtime $O(k^2 d)$, but may be performed infrequently without affecting the correctness of the algorithm, thus its computational cost can be ignored. For simplicity, we still use SGD to represent the stochastic power method in this paper. Despite the fact that the SGD algorithm for PCA converges rapidly to a neighbourhood of optima, it fluctuates near the optima and fails to converge. Hence, a fast stochastic algorithm called VR-PCA was proposed in [16], inspired by the SVRG algorithm. VR-PCA consists of several epochs, and at the beginning of each epoch, it stores the current parameters as $\tilde{W}$ and computes a batch gradient $\tilde{\mu}$. Then the variance reduced stochastic update is generated as follow:

$$W_{t+1} = P_{orth}(W_t - \eta(\mathbf{x}_{i_t}\mathbf{x}_{i_t}^\top W_t - \mathbf{x}_{i_t}\mathbf{x}_{i_t}^\top \tilde{W}_s + \tilde{\mu})). \tag{4}$$

Note that the learning rate η can be set as a constant. Both theory and experiments in [16] demonstrate the linear convergence of VR-PCA.

Although the SVRG-based algorithm proves to converge linearly, it has the following inherent drawbacks: (1) Requiring to pass over the entire dataset occasionally, which is time-consuming. (2) Failing to apply to time-limited conditions directly as it makes no update in the first pass through data. In order to overcome the two drawbacks, we proposed the novel SAGA-based algorithms, i.e., VR-PCA+ and VR-PLS+ for PCA and PLS respectively.

3 Preliminaries

In this section we review the SAGA algorithm. SAGA avoids the calculation of batch gradients, at the expense of some storage overhead. The algorithm has to store $\nabla f_i(\omega_{[i]})(i \in 1, ..., n)$, where $\omega_{[i]}$ denotes the latest iteration at which ∇f_i was computed. As shown in Algorithm 1, in each iteration, a random integer $j \in 1, ..., n$ is chosen and the following stochastic vector is executed:

$$g_k = \nabla f_j(\omega_k) - \nabla f_j(\omega_{[j]}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\omega_{[i]})$$

Taking the expectation of g_k over j, one again obtains that $\mathbb{E}[g_k|\omega_t] = \nabla F(\omega_t)$ Therefore, g_k is an unbiased gradient estimate, and has lower variance than stochastic gradients. SAGA proves to also have a linear convergence rate, not requiring to compute batch gradients periodically. Hence, SAGA often performs better than SVRG in practice. One important drawback of SAGA is the requirement of storing n stochastic gradient vectors, which is infeasible in large-scale applications. However, fortunately, in many cases, ∇f_j is not necessary to be stored explicitly. In this paper, one important reason for applying SAGA to PCA and PLS is that both of them benefit from this character.

Algorithm 1 SAGA

Require: the learning rate η, initial ω_0

for $i = 1, 2, ...n$ **do**
  $\nabla f_i(\omega_{[i]}) = \nabla f_i(\omega_0)$
end for
for $k = 1, 2, 3, ...$ **do**
  Randomly pick $j \in 1, 2, ..., n$
  $g_k = \nabla f_j(\omega_k) - \nabla f_j(\omega_{[j]}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\omega_{[i]})$
  $\nabla f_j(\omega_{[j]}) = \nabla f_j(\omega_k)$
  $\omega_{k+1} = \omega_k - \eta g_k$
end for

4 SAGA-based algorithm for PCA and PLS

In this section we describe the novel SAGA-based algorithms, i.e., VR-PCA+ and VR-PLS+. As the optimization objectives of PCA and PLS are special, when applying SAGA, both the computational overhead and storage overhead can be relieved a lot. According to the discussion above, we argue that it is not trivial to apply SAGA into PCA and PLS. For simplicity, we mainly analyze the VR-PCA+.

4.1 Dimension-free storage overhead

As is mentioned in Section 3, SAGA requires to store n stochastic gradient vectors throughout the optimizing procedure, which leads to an $O(nd)$ memory cost. In large-scale applications, both the sample number n and the feature size d are fairly large, thus the storage requirement is unsatisfiable. However, fortunately, it is possible for PCA and PLS to consume only $O(nk)$ memory cost without additional computational cost, which is free from the adverse effect of high dimension. Note that k represents the number of principal components to extract, conventionally a small number. Specifically, when the goal is finding the most important component, it just needs to store n scalars. Now we describe our novel storage methods. Take PCA as an example, if we directly apply SAGA to PCA, for iteration k, we compute the $\mathbf{x}_{j_k}\mathbf{x}_{j_k}{}^{\top}W_k$ to generate a variance reduced stochastic gradient, then use it to replace the table item $\Phi[j_k]$. The parameter update rule can be expressed as

$$W_{k+1} = P_{orth}(W_k - \eta(\mathbf{x}_{j_k}\mathbf{x}_{j_k}{}^{\top}W_k - \Phi[j_k] + \tilde{\mu})). \tag{5}$$

Notice that $\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\Phi[i]$, which can be updated by $\tilde{\mu}+\mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^{\top}W_k-\Phi[j_k])/n$. Neglecting the normalization step P_{orth}, the main computational cost of Equation (5) is two matrix-vector multiplication of runtime $O(kd)$, and the storage cost is $O(nkd)$(n parameter matrices of $d \times k$), which is unbearable for large n and d. It is worth noting that the size of matrix $\mathbf{x}_{j_k}{}^{\top}W_k$ is $1 \times k$, thus we can

Algorithm 2 VR-PCA+

Require: learning rate η, epoch size m
Input: Data matrix X=$\{\mathbf{x}_1, ...\mathbf{x}_n\}$; Initial W_0; $\tilde{\mu} = \mathbf{0}$
for $i = 1, 2, ...n$ **do**
  $\Phi[i] = \mathbf{0}$
end for
for $k = 0, 1, 2...$ **do**
  if $k < n$ **then**
    Sample $j_k \in \{1, 2, ..., n\}$ without replacement
  else
    Sample $j_k \in \{1, 2, ..., n\}$ with replacement
  end if
  $W_{k+1} = P_{orth}(W_k - \eta(\mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k]) + \tilde{\mu}))$
  $\Phi[j_k] = \mathbf{x}_{j_k}{}^\top W_k$
  if $k < n$ **then**
    $\tilde{\mu} = \frac{1}{k+1} \sum_{t=0}^{k} \mathbf{x}_{j_t} \Phi[j_t]$
  else
    $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \Phi[i]$
  end if
end for

store $\mathbf{x}_{j_k}{}^\top W_k$ in $\Phi[j_k]$ instead of $\mathbf{x}_{j_k}\mathbf{x}_{j_k}{}^\top W_k$. As is shown in Algorithm 2, the parameter update rule can be reformed as

$$W_{k+1} = P_{orth}(W_k - \eta(\mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k]) + \tilde{\mu})). \tag{6}$$

The main computational cost of Equation (6) is still two matrix-vector multiplication while the storage cost decreases to $O(nk)$, making a remarkable improvement on Equation (5). Moreover, the output of $\mathbf{x}_{j_k}{}^\top W_k$ and $\mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k])$ can be stored to avoid redundant computation for updating the $\Phi[j_k]$ and $\tilde{\mu}$. In conclusion, the improved variant of SAGA algorithm for PCA has almost the same computational cost as SGD algorithm (both of them require two matrix-vector multiplications at each iteration), with additional cheaper matrix additions (subtractions) and acceptable memory overhead. However, it has a superior performance, benefiting from the variance reduction virtue of SAGA.

4.2 Applicability with limited computational overhead

SVRG-based algorithm work well when the computational resources or the computational time is sufficient. However, the computational overhead is limited and high precision is not required in many scenes, which is common for PCA and PLS problems. Then SVRG-based algorithms are not suitable as they do not begin optimizing the objective until passing all the samples once. As is shown in Algorithm 1, SAGA also requires to pass the dataset once as a preparation step before updating the parameters. The reason is that at the beginning, all

the $\nabla f_j(\omega_{[j]})$ with $j \in \{1, 2, ..., n\}$ are not stochastic gradients vectors, and if we cancel such preparation step and update parameters directly, we obtain $\mathbb{E}[g_k|\omega_t] \neq \nabla F(\omega_t)$, which deteriorates the performance. Note that in each iteration, one sample is picked uniformly in random, which means that after n iterations, there are at least $n/3$ samples have never been picked. Hence the adverse effects will last longer. Therefore, we conduct an improvement for the first n iterations of SAGA as follows. Taking VR-PCA+ as an example, as shown in Algorithm 2, we cancel the preparation step, and initialize all items in table Φ as zero-vector and begin iterations directly. In the first n iterations, we sample j_k without replacement, i.e., each iteration has different j_k. After n iterations, we sample j_k with replacement, i.e. each j_k is independent. This modification ensures after n iterations, each sample has been picked once.

Another important modification is the update rule of $\tilde{\mu}$. We initialize $\tilde{\mu}$ as a zero matrix and compute the $\tilde{\mu}$ for next iteration at the end of each iteration. Notice that $\tilde{\mu}$ represents the average of $\nabla f_i(\omega_{[i]})$ in the primal SAGA, and $\nabla f_i(\omega_{[i]})$ denotes the latest stochastic gradient computed by sample $\mathbf{x}_i$. Since VR-PCA+ store ${\mathbf{x}_{j_k}}^\top W_k$ in $\Phi[j_k]$ instead of exact stochastic gradient, for $k \geqslant n$, the computational rule of $\tilde{\mu}$ is expressed as:

$$\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \Phi[i]. \tag{7}$$

When it comes to $k < n$ with the preparation step cancelled (n stochastic gradients are not computed and all values in table Φ are zero before iterating), we have only $k + 1$ items to compute the $\tilde{\mu}$. As a result, the computational rule of $\tilde{\mu}$ is expressed as

$$\tilde{\mu} = \frac{1}{k+1} \sum_{t=0}^{k} \mathbf{x}_{j_t} {\mathbf{x}_{j_t}}^\top W_t = \frac{1}{k+1} \sum_{t=0}^{k} \mathbf{x}_{j_t} \Phi[j_t]. \tag{8}$$

In fact, there is no need to compute $k + 1$ or n items in each iteration, instead, we can update $\tilde{\mu}$ before updating $\Phi[j_k]$ with the following rule:

$$\tilde{\mu} = \begin{cases} (k\tilde{\mu} + \mathbf{x}_{j_k}({\mathbf{x}_{j_k}}^\top W_k - \Phi[j_k]))/(k+1) & k < n \\ \tilde{\mu} + \mathbf{x}_{j_k}({\mathbf{x}_{j_k}}^\top W_k - \Phi[j_k])/n & k \geqslant n. \end{cases} \tag{9}$$

It is equal to Equation (8) and (9), and just using the new computed ${\mathbf{x}_{j_k}}^\top W_k$ to update $\tilde{\mu}$. It is worth noting that the variance reduced stochastic gradient $\mathbf{x}_{j_k}({\mathbf{x}_{j_k}}^\top W_k - \Phi[j_k]) + \tilde{\mu}$ is a bias estimate of $\nabla f(\omega_k)$ when in the first n iterations, but it performs obviously better than naive SGD. The main reason is that the $\tilde{\mu}$ plays a role of a stale mini-batch gradient, which reduce the variance of stochastic gradient.

4.3 Extend VR-PCA+ to VR-PLS+

It is natural to extend VR-PCA+ to VR-PLS+. We do not need to modify the algorithm structure at all, and we just require to update the parameter matrices

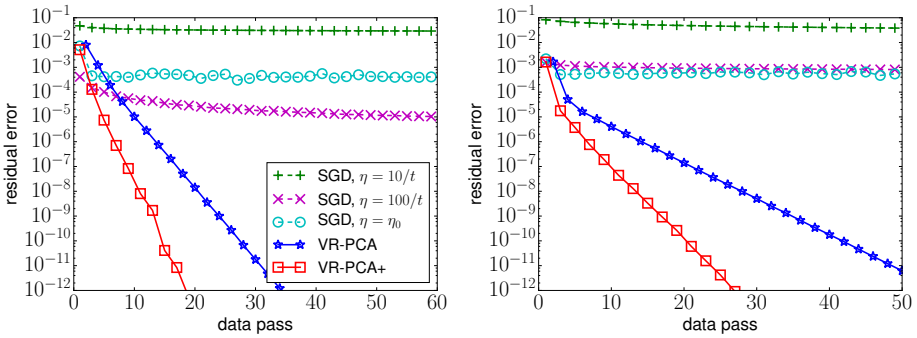
U_k and V_k respectively. In other words, we should replace line 10 in Algorithm 2 as follow:

$$\begin{aligned} U_{k+1} &= P_{orth}(U_k - \eta(\mathbf{x}_{j_k}(\mathbf{y}_{j_k}^\top V_k - \Phi_U[j_k]) + \tilde{\mu_U})) \\ V_{k+1} &= P_{orth}(V_k - \eta(\mathbf{y}_{j_k}(\mathbf{x}_{j_k}^\top U_k - \Phi_V[j_k]) + \tilde{\mu_V})). \end{aligned} \tag{10}$$

Meanwhile, it has two tables Φ_U and Φ_V to store the components for stale stochastic gradients and two matrices $\tilde{\mu}_U$ and $\tilde{\mu}_V$ to store the stale batch gradient.

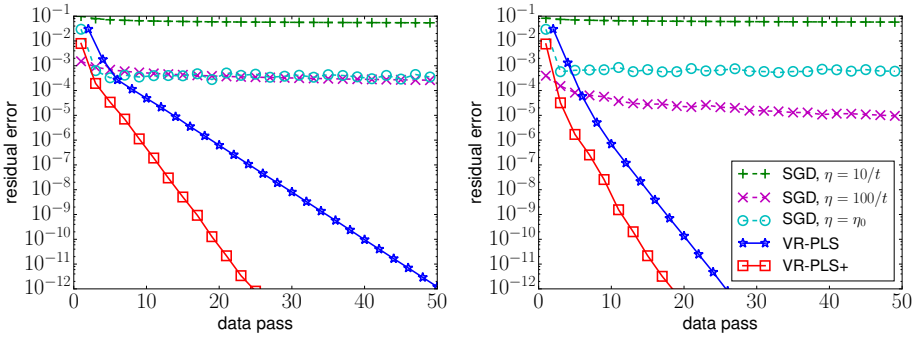
5 Numerical Experiments

In this section, we present extensive experiments to demonstrate the performance of our proposed algorithms. Our experiments include two parts, In the first part, we run algorithms for many data passes to verify the performances of our proposed algorithms when high-precision is required. In the second part, we run algorithms for only one data pass to show the superior performances of our algorithms when computational overhead is limited. Instead of tuning parameters, we set the learning rate η as recommended in [16]: $\eta_0 = \frac{1}{\gamma\sqrt{n}}$, where $\gamma = \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{x}_i\|^2$. Although this choice of η_0 is theoretically proved only for VR-PCA, we also find it suitable for our algorithms in practice. We conducted experiments on four famous datasets, MNIST, CIFAR-10, SVHN and CIFAR-100, which are widely used in many researches. For PCA experiments we directly use the datasets while for PLS experiments we divide each example in half to generate the dataset X and dataset Y. All experiments include the preprocessing step, consisting of centering the data and dividing each coordinate by its standard deviation times the squared root of the dimension.



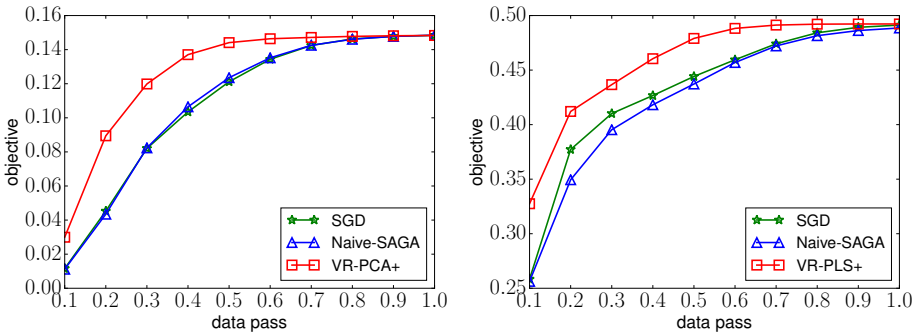
(a) mnist $k = 6$ (b) cifar10 $k = 6$

Fig. 1. Generally, both VR-PCA and VR-PCA+ converge to a high precision and VR-PCA+ has a better performance.



(a) SVHN, $k = 8$ (b) cifar100 $k = 8$

Fig. 2. Generally, both VR-PLS and VR-PLS+ converge to a high precision and VR-PLS+ has a better performance in most cases.



(a) mnist, $k = 4$, pca (b) cifar10 $k = 6$, pls

Fig. 3. VR-PCA(PLS)+ performs better than SGD and Naive-SAGA in the first data pass.

5.1 Experiments for high-precision requirement

In this section we demonstrate the effectiveness and efficiency of our proposed algorithms. The evaluation criterion is the progress made on the objective as a function of iteration number, which is suitable for evaluating gradient optimization algorithms. For comparison, we also implemented the SGD algorithm and SVRG-based algorithms, i.e., VR-PCA and VR-PLS. Note that VR-PLS is naturally extended from VR-PCA, just replacing the update rule. We did not implement other sophisticated iterative algorithms mentioned in Section 1 for the same reason with [16], as they are not directly comparable to the stochastic gradient algorithms due to inherent complexity per iteration and considerable memory requirement. The algorithms compared were initialized from the same random matrices. The results are displayed in Figure 1 and Figure 2. In all figures, the x-axis represents the number of effective data passes, and the y-

axis represent the residual, i.e., $tr(\hat{W}^\top X^\top X\hat{W}) - tr(W^\top X^\top XW)$ for PCA and $tr(\hat{U}^\top X^\top Y\hat{V}) - tr(U^\top X^\top YV)$ for PLS. Note the X,Y are the data matrices, $\hat{W}$, $\hat{U}$, $\hat{V}$ are the optimal parameter matrices obtained by SVD, W, U, V are the matrices obtained so far. From the figures we see that for different datasets and k, SGD algorithms appears to have a sub-linear convergence rate and fail to achieve a high-precision result, no matter which step strategy is applied. In contrast, both VR-PCA(PLS) and VR-PCA(PLS)+ rapidly converge to an accuracy higher than 10^{-10}, the main reason is that they have much lower variance than naive SGD algorithm. Besides, we find the VR-PCA(PLS)+ have a better performance than VR-PCA(PLS). It is not surprising since the SAGA-based algorithms do not require to compute the batch-gradient occasionally, which is time-consuming.

5.2 Experiments with limited time overhead

In this part we limit the computational cost within one data pass. Since VR-PCA(PLS) do not optimize the objective function in the first data pass, they were not experimented. Hence we only compare the VR-PCA(PLS)+ with SGD algorithm. In particular, apart from our proposed VR-PCA(PLS)+, we also implement another variant called Naive-SAGA, which abandons the preparation step directly without modification. The goal is to demonstrate the superiority of our proposed sample strategy and new update rule of $\tilde{\mu}$. We set the learning rate as η_0 for all experiments. The results are illustrated in figure 3. Note that the x-axis still represents the number of data passes while the y-axis denotes the value of objective function. As we can see in figure 3, VR-PCA(PLS)+ always has the best performance, the main reason is that the variance reduced stochastic gradient of VR-PCA(PLS)+ is the combination of a stochastic gradient and a stale mini-batch gradient, which has a smaller variance.

6 Conclusion and Future Works

Deterministic optimization algorithms for PCA and PLS suffer from unbearable computation cost in large-scale application, while stochastic algorithm is prohibitive from a high-accuracy solution. Thus it is natural to apply the variance reduced stochastic method to solve PCA and PLS. In this paper, we propose the novel SAGA-based algorithms, which perform better than SVRG-based algorithms and apply to time limited conditions. Although we have not rigorously conducted convergence analysis on the proposed algorithms, all of them prove to be experimentally efficient. We leave the following future works: (1) Prove the convergence of proposed algorithms theoretically. (2) Research the asynchronous variants of VR-PCA+ and VR-PLS+. (3) Explore other constrained objective functions which can be optimized through stochastic power methods and its variance reduced variants.

Acknowledgment

Bibliography

1. Stockman and C George. *Computer vision*. Prentice Hall,, 2001.
2. Leo H. Chiang, Evan L. Russell, and Richard D. Braatz. *Partial Least Squares*. Springer London, 2001.
3. Gerard Salton and Donna Harman. Information retrieval. page 777, 2003.
4. Srinadh Bhojanapalli, Prateek Jain, and Sujay Sanghavi. Tighter low-rank approximation via sampling the leveraged element. *Eprint Arxiv*, 2014.
5. Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Computer Science*, 2015.
6. N Halko, P. G Martinsson, and J. A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *Siam Review*, 53(2):217–288, 2010.
7. Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.
8. Terence D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. 2(6):459–473, 1989.
9. R Arora, A Cotter, K Livescu, and N Srebro. Stochastic optimization for pca and pls. In *Allerton*, pages 861–868, 2014.
10. Raman Arora, Poorya Mianjy, and Teodor Marinov. Stochastic optimization for multiview representation learning using partial least squares. 2016.
11. Raman Arora, Andrew Cotter, and Nathan Srebro. Stochastic optimization of pca with capped msg. *Advances in Neural Information Processing Systems*, pages 1815–1823, 2013.
12. Mark Schmidt, Nicholas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 26(5):1–30, 2013.
13. IEEE. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems*, pages 315–323, 2013.
14. Aaron Defazio, Francis Bach, and Simon Lacostejulien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems*, 2:1646–1654, 2014.
15. Niu Feng, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, 24:693–701, 2011.
16. Ohad Shamir. A stochastic pca and svd algorithm with an exponential convergence rate. *Mathematics*, 2015.