Practical Stochastic Optimization for PCA and PLS: Variance Reduction and Asynchronism

***Abstract*—Principal Component Analysis(PCA) is a dimensionality reduction technique which extract the representative components of the data. Partial Least Squares(PLS) models the covariance structure between a pair of data matrices. Both of them are widely used in many areas and can be represented as a similar objective function. Deterministic methods like the full singular value decomposition (SVD) for these problems suffers from prohibitive computation cost in large-scale applications, while the stochastic or incremental algorithms fail to achieve a high accuracy. 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. Moreover, we explore an asynchronous implement of stochastic optimization for the two problems. Extensive experiments illustrates the significant performance of our method.**

I. 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 $X = (\mathbf{x}_1, ..., \mathbf{x}_n)$. 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, thus 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 $O(\min(k^2 d, kd^2))$ 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 $\mathbf{x}_i {\mathbf{y}_i}^\top$. 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 guarantee. In contrast to the deterministic algorithm, these algorithm randomly sample some examples to update the parameters, which performs much cheaper iterations. The drawback of stochastic algorithms is that the noise caused by stochastic sample means they converge sublinearly and have difficulty in obtaining a high-accuracy solution. Recently, Ohad Shamir made a breakthrough in [12] by proposing VR-PCA, a novel algorithm for PCA problem, which is based on cheap stochastic iterations, yet converging linearly to obtain accuracy ϵ. This algorithm combines the stochastic power method with the stochastic variance reduced gradient(SVRG) proposed in [13], shows better performance than the primal stochastic power method. Inspired by VR-PCA, we similarly apply the variance reduced stochastic algorithm to PLS and propose VR-PLS. Moreover, as SVRG-based algorithms require passing through the entire data as a “Preparation Step” for generating the consequent variance reduced stochastic iterations at the beginning of each epoch, which is rather time-consuming. As a result, when using VR-PCA, the objective will not be optimized until at least $O(nd)$ runtime is consumed, thus it cannot be applied when computation resource is very limited. Inspired by another variance reduced stochastic gradient algorithm, i.e., SAGA[14], we design two novel stochastic algorithms, VR-PCA+ and VR-PLS+, to solve PCA and PLS respectively. The SAGA-based algorithms are superior to SVRG-based algorithms in our

Algorithm 1 SVRG

```
Require: the learning rate $\eta$, the epoch size $m$, initial $\tilde{\omega}_0$
 1: for $s = 0, 1, 2, ...$ do
 2:     $\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{\omega}_s)$
 3:     $\omega_0 = \tilde{\omega}_s$
 4:     for $t = 0, 1, ..., m-1$ do
 5:         Randomly pick $i_t \in \{1, 2, ..., n\}$
 6:         $\omega_{t+1} = \omega_t - \eta(\nabla f_{i_t}(\omega_t) - \nabla f_{i_t}(\tilde{\omega}_s) + \tilde{\mu})$
 7:     end for
 8:     Option I: $\tilde{\omega}_{s+1} = \omega_m$
 9:     Option II: $\tilde{\omega}_{s+1} = \omega_t$ for randomly chosen $t \in \{0, ..., m-1\}$
10: end for
```

Algorithm 2 SAGA

```
Require: the learning rate $\eta$, initial $\omega_0$
 1: for $i = 1, 2, ...n$ do
 2:     $\nabla f_i(\omega_{[i]}) = \nabla f_i(\omega_0)$
 3: end for
 4: for $k = 1, 2, 3, ...$ do
 5:     Randomly pick $j \in 1, 2, ..., n$
 6:     $g_k = \nabla f_j(\omega_k) - \nabla f_j(\omega_{[j]}) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\omega_{[i]})$
 7:     $\nabla f_j(\omega_{[j]}) = \nabla f_j(\omega_k)$
 8:     $\omega_{k+1} = \omega_k - \eta g_k$
 9: end for
```

experiments, and optimize the objective immediately without such "Preparation step". Furthermore, since stochastic power method is intrinsically suitable for asynchronism, we design an asynchronous stochastic power method to utilize the multi-core systems. In brief, our contributions are highlighted as follows:

- We extend the variance reduction technique to the stochastic optimization for partial least square, proposing the VR-PLS algorithm, which is capable to achieve high accuracy solutions.
- We introduce the SAGA algorithm into the optimization of PCA and PLS, resulting in two novel algorithms, VR-PCA+ and VR-PLS+.
- We implement an asynchronous stochastic power method for PCA and PLS, which accelerates the convergence through multi-threads.
- Extensive experiments demonstrate the effectiveness and efficiency of our proposed algorithms, which outperform their counterparts significantly on the convergence performance.

This paper is organized as follows: Section II introduces the SVRG algorithm and SAGA algorithm. Section III presents the SVRG-based algorithm, i.e., VR-PCA and VR-PLS. Section IV presents the SAGA-based algorithm, i.e. , VR-PCA+ and VR-PLS+. Section V explores the asynchronous implement of stochastic power method. Section VI demonstrates the numerical results of our algorithms. Section VII concludes the strengths and weaknesses of our work and presents the future works.

II. Preliminaries

In this section we review the SVRG algorithm and SAGA algorithm. As is shown in Algorithm 1, there are two loops in SVRG. At the beginning of each outer loop (which is called an *epoch*), the entire dataset is computed and a batch gradient $\tilde{\mu}$ is achieved. In each inner loop, a stochastic gradient with lower variance is generated to update the parameters. The update rule is $\omega_t = \omega_{t-1} - \eta g_t$, while

$$g_t = \nabla f_{i_t}(\omega_t) - \nabla f_{i_t}(\tilde{\omega}_s) + \tilde{\mu}$$

Note that the expectation of $\nabla f_{i_t}(\tilde{\omega}_s)$ over i_t is $\tilde{\mu}$, and the expectation of $\nabla f_{i_t}(\omega_t)$ over i_t is $\nabla F(\omega_t)$. We thus obtain

$$\mathbb{E}[\omega_{t+1}|\omega_t] = \omega_t - \eta \nabla F(\omega_t)$$

And the variance of g_t is expected to be smaller than simply choosing $g_t = \nabla f_{i_t}(\omega_{t-1})$. At the end of each epoch, $\tilde{\omega}_{s+1}$ is updated based on the outputs of inner loop. Note there are two options for the update. SVRG with both the options have been theoretically proved to converge linearly, while SVRG with Option I performs better in practice. Thus we adopt Option I in this paper.

Note that SVRG requires to compute a batch gradient at the start of each epoch, which is quite time-consuming. 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. 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 proved to also have a linear convergence rate, not requiring to compute batch gradients occasionally. 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. For example, if the form of the component functions are $f_j(\omega_k) = \tilde{f}(\mathbf{x}_j{}^\top \omega_k)$, one only need to store a scalar to construct the $\nabla f_j(\omega_k)$. In this paper, one important reason for applying SAGA to PCA and PLS is that both of them benefit from this character.

III. SVRG-based algorithm

In this section we present the SVRG-based algorithms for PCA and PLS. We first review the VR-PCA algorithm proposed in [15], and then analogically propose VR-PLS, a variance reduced stochastic algorithm for PLS.

Algorithm 3 VR-PCA

```
Require: learning rate $\eta$, epoch size $m$
Input: Data matrix X={$\mathbf{x}_1, ...\mathbf{x}_n$}; Initial $W_0$
1: for $s = 0, 1, ...$ do
2:     $\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{x}_i^\top \tilde{W}_s)$
3:     $W_0 = \tilde{W}_s$
4:     for $t = 1, 2, ..., m$ do
5:         Randomly pick $i_t \in \{1, 2, ..., n\}$
6:         $W_{t+1} = P_{orth}(W_t - \eta(\mathbf{x}_{i_t}(\mathbf{x}_{i_t}^\top W_t - \mathbf{x}_{i_t}^\top \tilde{W}_s) + \tilde{\mu}))$
7:     end for
8:     $\tilde{W_{s+1}} = W_t$
9: end for
```

Algorithm 4 VR-PLS

```
Require: learning rate $\eta$, epoch size $m$
Input: Data matrix X = {$\mathbf{x}_1, ..., \mathbf{x}_n$}, Y = {$\mathbf{y}_1, ..., \mathbf{y}_n$}; Initial $U_0$, $V_0$
1: for $s = 0, 1, ...$ do
2:     $\tilde{\mu}_U = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i^\top \tilde{V}_s)$
3:     $\tilde{\mu}_V = \frac{1}{n}\sum_{i=1}^{n} \mathbf{y}_i(\mathbf{x}_i^\top \tilde{U}_s)$
4:     $U_0 = \tilde{U}_s$
5:     $V_0 = \tilde{V}_s$
6:     for $t = 0, 1, ..., m-1$ do
7:         Randomly pick $i_t \in \{1, 2, ..., n\}$
8:         $U_{t+1} = P_{orth}(U_t - \eta(\mathbf{x}_{i_t}(\mathbf{y}_{i_t}{}^\top V_t - \mathbf{y}_{i_t}{}^\top \tilde{V}_s) + \tilde{\mu}_U))$
9:         $V_{t+1} = P_{orth}(V_t - \eta(\mathbf{y}_{i_t}(\mathbf{x}_{i_t}{}^\top U_t - \mathbf{x}_{i_t}{}^\top \tilde{U}_s) + \tilde{\mu}_V))$
10:    end for
11:    $\tilde{U}_{s+1} = U_t$
12:    $\tilde{V}_{s+1} = V_t$
13: end for
14: return $\tilde{U}_{s+1}$ and $\tilde{V}_{s+1}$
```

A. VR-PCA

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]. It takes the following update rule in each iteration:

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

Notice that 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. In our implements, we adopt a more stable transformation proposed in [12] as follows:

$$W_t = \hat{W}_t(\hat{W_t}^\top \hat{W}_t)^{-1/2} \tag{4}$$

The $\hat{W}_t$ is the output of iteration t without normalization. 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, the a fast stochastic algorithm called VR-PCA was proposed in [15], inspired by the SVRG algorithm. As is shown in Algorithm 3, VR-PCA consists of several epochs, and each epoch includes a batch gradient computation and m cheap stochastic iterations. The variance reduced stochastic iteration is similar to that of SVRG, can be express as

$$W_{t+1} = P_{orth}(W_t - \eta(\mathbf{x}_{i_t}\mathbf{x}_{i_t}^\top W_t - \mathbf{x}_{i_t}\mathbf{x}_{i_t}^\top \tilde{W}_s + \tilde{\mu})) \tag{5}$$

As computing the matrix-vector multiplication takes the main cost of one iteration, that Equation 5 can be reformed as

$$W_{t+1} = P_{orth}(W_t - \eta(\mathbf{x}_{i_t}\mathbf{x}_{i_t}^\top (W_t - \tilde{W}_s) + \tilde{\mu})) \tag{6}$$

Note that the computation cost of 6 is the half of that of 5. Both theory and experiments in [15] demonstrate the linear convergence of VR-PCA.

B. VR-PLS

It is natural to extend the variance reduced SGD algorithm of PCA to PLS as their objective functions have the analogous structure. As is shown in Algorithm 4, the difference is that PLS chooses a pair of samples $\mathbf{x}_t, \mathbf{y}_t$ in each iteration and we should compute gradients with regards to U and V respectively. Thus the stochastic gradient update consists two parts:

$$\begin{aligned} U_{t+1} &= P_{orth}(U_t - \eta(\mathbf{x}_{i_t}\mathbf{y}_{i_t}{}^\top (V_t - \tilde{V}_s) + \tilde{\mu}_U)) \\ V_{t+1} &= P_{orth}(V_t - \eta(\mathbf{y}_{i_t}\mathbf{x}_{i_t}{}^\top (U_t - \tilde{U}_s) + \tilde{\mu}_V)) \end{aligned} \tag{7}$$

Besides, at the beginning of each epoch, we have to compute two batch gradients $\tilde{\mu_U}$ and $\tilde{\mu_V}$, with respect to U and V.

IV. SAGA-based algorithm

Although SVRG-based algorithms prove to exponentially converge to a high precision solution, they have 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 does not optimize the objective immediately. In this section we proposed 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 analysis the VR-PCA+.

Algorithm 5 VR-PCA+

```
Require: learning rate $\eta$, epoch size $m$
Input: Data matrix X={$\mathbf{x}_1, ...\mathbf{x}_n$}; Initial $W_0$; $\tilde{\mu}$ = $\mathbf{0}$
 1: for $i = 1, 2, ...n$ do
 2:     $\Phi[i] = \mathbf{0}$
 3: end for
 4: for $k = 0, 1, 2...$ do
 5:     if $k < n$ then
 6:         Sample $j_k \in \{1, 2, ..., n\}$ without replacement
 7:     else
 8:         Sample $j_k \in \{1, 2, ..., n\}$ with replacement
 9:     end if
10:     $W_{k+1} = P_{orth}(W_k - \eta(\mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k]) + \tilde{\mu}))$
11:     $\Phi[j_k] = \mathbf{x}_{j_k}{}^\top W_k$
12:     if $k < n$ then
13:         $\tilde{\mu} = \frac{1}{k+1} \sum_{t=0}^{k} \mathbf{x}_{j_t} \Phi[j_t]$
14:     else
15:         $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \Phi[i]$
16:     end if
17: end for
```

Algorithm 6 VR-PLS+

```
Require: learning rate $\eta$, the tables $\Phi_U$ and $\Phi_V$.
Input: Data matrix X = $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, Y = $\{\mathbf{y}_1, ..., \mathbf{y}_n\}$; Initial $U_0$, $V_0$
 1: for $i = 1, 2, ...n$ do
 2:     $\Phi_U[i] = \Phi_V[i]$ = $\mathbf{0}$
 3: end for
 4: for $k = 0, 1, 2, ...$ do
 5:     if $k < n$ then
 6:         Sample $j_k \in \{1, 2, ..., n\}$ without replacement
 7:     else
 8:         Sample $j_k \in \{1, 2, ..., n\}$ with replacement
 9:     end if
10:     $U_{k+1} = P_{orth}(U_k - \eta(\mathbf{x}_{j_k}(\mathbf{y}_{j_k}^\top V_k - \Phi_U[j_k]) + \tilde{\mu_U}))$
11:     $V_{k+1} = P_{orth}(V_k - \eta(\mathbf{y}_{j_k}(\mathbf{x}_{j_k}^\top U_k - \Phi_V[j_k]) + \tilde{\mu_V}))$
12:     $\Phi_U[j_k] = \mathbf{y}_{j_k}^\top V_k$
13:     $\Phi_V[j_k] = \mathbf{x}_{j_k}^\top U_k$
14:     if $k < n$ then
15:         $\tilde{\mu} = \frac{1}{k+1} \sum_{t=0}^{k} \mathbf{x}_{j_t} \Phi_U[j_t]$
16:         $\tilde{\mu} = \frac{1}{k+1} \sum_{t=0}^{k} \mathbf{y}_{j_t} \Phi_V[j_t]$
17:     else
18:         $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \Phi_U[i]$
19:         $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}_i \Phi_V[i]$
20:     end if
21: end for
22: return $U_k, V_k$
```

A. Dimension-free storage overhead

As is mentioned in II, 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 viable 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

$$W_{k+1} = P_{orth}(W_k - \eta(\mathbf{x}_{j_k}\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k] + \tilde{\mu})) \tag{8}$$

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 8 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 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 5, the parameter update rule can be reformed as

$$W_{k+1} = P_{orth}(W_k - \eta(\mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k]) + \tilde{\mu})) \tag{9}$$

The main computational cost of Equation 9 is still two matrix-vector multiplication while the storage cost decreases to $O(nk)$, making a remarkable improvement on Equation 8. 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 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.

B. 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. One solution is running SGD for some iterations at first, then switching to the SVRG-based algorithm, but it is hard to choose the optimal switching time. As is shown in Algorithm 2, SAGA also requires passing the dataset once as a preparation step before updating the parameters. The reason is that at the beginning, all the $\nabla f_j(\omega_{[j]})(j \in 1, 2, ..., n)$

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. Take VR-PCA+ as an example, as is shown in Algorithm 5, 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 μ 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 $k >= n$, the computational rule of $\tilde{\mu}$ is expressed as:

$$\tilde{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \Phi[i] \tag{10}$$

When it comes to $k < n$ and 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

$$\tilde{\mu} = \frac{1}{k+1}\sum_{t=0}^{k} \mathbf{x}_{j_t}\mathbf{x}_{j_t}{}^\top W_t = \frac{1}{k+1}\sum_{t=0}^{k} \mathbf{x}_{j_t}\Phi[j_t] \tag{11}$$

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:

$$\tilde{\mu} = \begin{cases} (k\tilde{\mu} + \mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k]))/(k+1) & k < n \\ \tilde{\mu} + \mathbf{x}_{j_k}(\mathbf{x}_{j_k}{}^\top W_k - \Phi[j_k])/n & k >= n \end{cases} \tag{12}$$

It is equal to Equation 11 and 12, and we just use 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.

C. Extend VR-PCA+ to VR-PLS+

It is natural to extend VR-PCA+ to VR-PLS+. As illustrated in Algorithm 6, the structure is the same with that of VR-PCA+, and the main difference is that the parameter matrices U_k and V_k require to be updated respectively. 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.

V. Asynchronous stochastic power method

In this section, we explore the asynchronous feasibility of stochastic optimization for constrained objective like PCA and PLS. For simplicity, we only discuss the asynchronous SGD for PCA algorithm, which is convenient to extend to PLS. We first review the implements of asynchronous SGD for general unconstrained finite-sum problems. The straightforward method is several threads compute the stochastic gradient in an asynchronous manner. Its requires a lock(writie-lock) when each thread is going to update(write) the parameter vector, in other words, only one thread can assess the parameter vector at the same time. Another meaningful method is Hogwild! [16], running SGD in parallel without locks, eliminating the overhead associated with locking. Experimental results in [16] have shown the superior performance of the lock-free method.

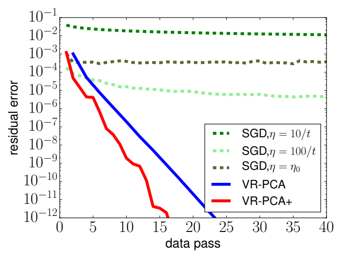
Under the condition of $W^\top W = I$ in PCA, directly applying the methods mentioned above is not feasible. It is because SGD for PCA occasionally requires an additional orthogonalization operation, which includes many steps and is time-consuming compared with single SGD process. Thus when one thread is performing the orthogonalization operation, the stochastic gradient update performed by other threads is completely useless and may affect the correctness of the orthogonalization operation. Hence, we propose a novel asynchronous SGD framework for the PCA-like algorithm. The key components are briefly described below.

1) **Read:** Read example $\mathbf{x}_i$ and compute the gradient ∇f_i with a randomly chosen i.
2) **Update:** if the ***Ready-signal*** is $true$, update the parameter vector in a lock-free manner.
3) **Normalization:** If the ***thread-id*** is 0, do the following operations at an interval of m iterations: Set the ***Ready-signal*** to be $false$ and suspend all threads, perform the orthogonalization operation, then recover the ***Ready-signal*** as $true$ and wake up all blocking threads.

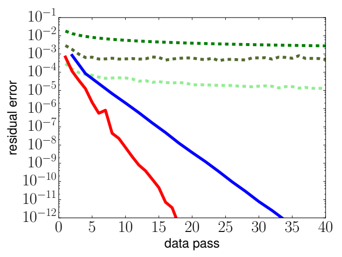
Each thread repeatedly run these procedures concurrently. This algorithm means that each thread performs the read and update steps fully asynchronously, while only one thread perform the normalization procedure. Note that not only this framework is suitable for PCA, but also it can be adapted to some PCA-like algorithm, i.e. PLS, whose objective function has several simple constrains. Experiments illustrate the obvious speedup benefiting from multi-core systems.

VI. Numerical Experiments

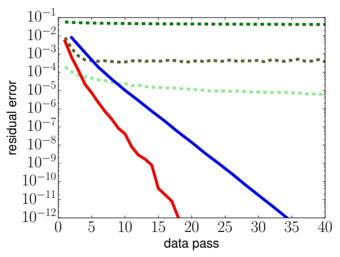
In this section, we present extensive experiments to demonstrate the performance of our proposed algorithms, i.e. VR-PLS and VR-PLS+. Our experiments include three parts, In the first part, we run algorithms for several data passes to verify the performances of our proposed algorithms when high-precision is required. In the second part, we run algorithms for only one data pass to show the superior performances of VR-PCA+ and VR-PLS+ when computational overhead is limited. In the third part, we test asynchronous SGD for PCA



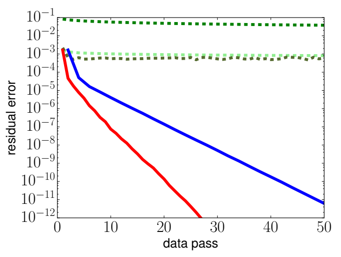
(a) mnist, $k = 4$



(b) cifar10, $k = 4$

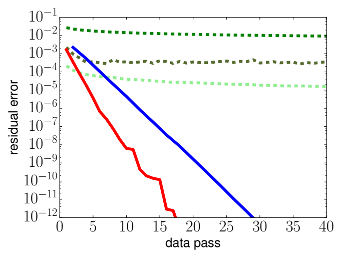


(c) mnist $k = 6$

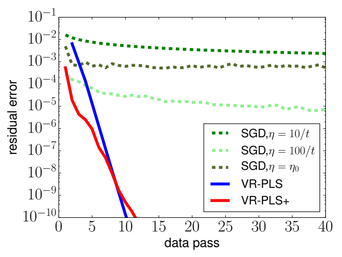


(d) cifar10 $k = 6$

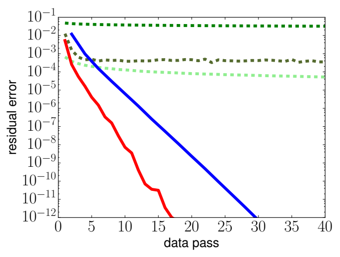
Fig. 1. Generally, both VR-PCA and VR-PCA+ converge to a high precision and VR-PCA+ has a better performance.



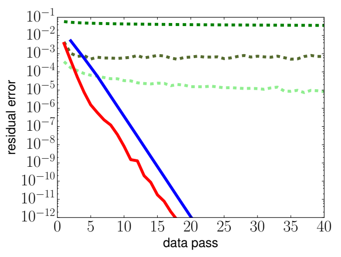
(a) mnist, $k = 4$



(b) cifar10, $k = 4$

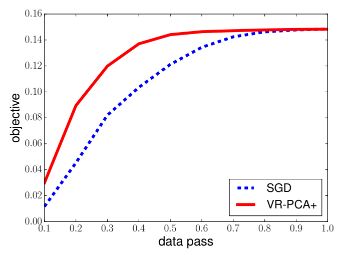


(c) mnist $k = 6$

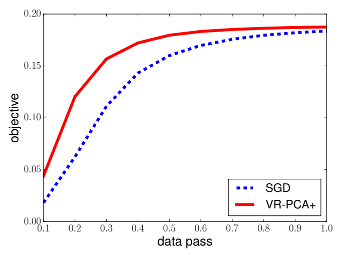


(d) cifar10 $k = 6$

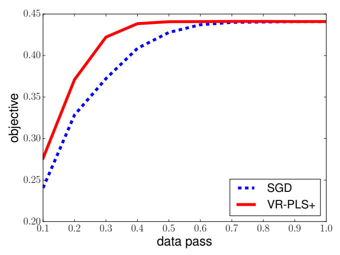
Fig. 2. Generally, both VR-PLS and VR-PLS+ converge to a high precision and VR-PLS+ has a better performance in most cases.



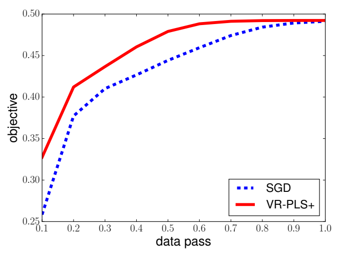
(a) mnist, $k = 4$, pca



(b) mnist $k = 6$, pca



(c) cifar10, $k = 4$, pls



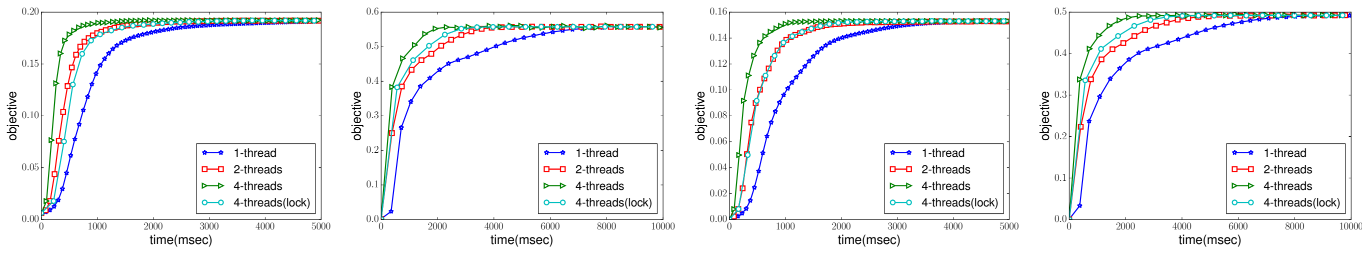
(d) cifar10 $k = 6$, pls

Fig. 3. Generally, VR-PCA(PLS)+ performs better than SGD in the first data pass.

and PLS with multi-threads. Instead of parameters tuning, we heuristically set the epoch size m of VR-PCA and VR-PLS as n (the size of the dataset), and set the learning rate η as recommended in [Shamir 2015]: $\eta_0 = \frac{1}{\gamma\sqrt{n}}$, 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 other three algorithms in practice. We conducted experiments on the two famous datasets, MNIST and CIFAR-10. MINST dataset(LeCun et al., 1998) consists of $60,000$ handwritten digits from 0 to 9 and each instance is a 28×28 image. The CIFAR-10 dataset consists of $60,000$ color images of 32×32 in 10 classes and we just select the $50,000$ training examples. For PCA experiments we directly use the datasets. For PLS experiments we split each pictures into left half part and right half part and generate a pair of dataset X and dataset Y. Therefore, for MNIST, $d_x = d_y = 392$ and for CIFAR-10, $d_x = d_y = 512$. 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. *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 implemented the SGD algorithm, using several different initial step sizes with decaying strategy. We did not implement other sophisticated iterative algorithms mentioned in Section I for the same reason with [15], 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 (assuming 2 per epoch for VR-PCA and VR-PLS, 1 per epoch for VR-PCA+ and VR-PLS+), 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



(a) mnist, pca (b) cifar10, pca (c) mnist pls (d) cifar10 pls

Fig. 4. Generally, asynchronous SGD for PCA(PLS) performs better with more threads. And the lock-free variant converges faster than lock variant.

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 also find the VR-PCA(PLS)+ have a better performance than VR-PCA(PLS) in most cases. It is not surprising since the SAGA-based algorithms do not require to compute the batch-gradient occasionally, which is time-consuming.

B. *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, setting 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 perform better than SGD algorithm, 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 than naive SGD.

C. *Asynchronous SGD for PCA and PLS*

In this experiment, we compare asynchronous SGD for PCA and PLS with varying number of threads. We run the lock-free asynchronous SGD for 1, 2 and 4 threads. Besides, we also implement a lock variant and run it with 4 threads. Note the the evaluation criterion is the progress made on the objective as a function of time pass, thus the x-axis represents the time pass and y-axis represents the value of objective function. As we can see in Figure 4, the objective function converges faster with more threads, which demonstrates the effectiveness of asynchronism. Moreover, the lock-free variant performs better than lock variant, which is consistent with the primal SGD.

VII. 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 review the SVRG-based algorithm for PCA, i.e., VR-PCA, and extend it to PLS. Besides, we propose the novel SAGA-based algorithms, which performs better than SVRG-based algorithms and apply to time limited conditions. Moreover, since stochastic gradient algorithms are inherently suitable for asynchronism, we explore an implement of asynchronous SGD for PCA and PLS. 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(PLS) and VR-PCA(PLS)+. (3) Explore other constrained objective functions which can be optimized through stochastic power methods and its variance reduced variants.

Acknowledgment

References

[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] Ohad Shamir. Fast stochastic algorithms for svd and pca: Convergence properties and convexity. *Mathematics*, 2015.

[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] Ohad Shamir. A stochastic pca and svd algorithm with an exponential convergence rate. *Mathematics*, 2015.

[16] 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.