北京大学

本科生毕业论文

题目： **LazySVD: 快速奇异值分解**

姓　　名：李翔

学　　号：1400010650

院　　系：数学科学学院

专　　业：统计学

研究方向：机器学习

导师姓名：张志华

二〇一八年六月

LazySVD: 快速奇异值分解

李翔　统计学

导师姓名：张志华

摘要

在这篇论文中，我们主要目的是求解数据协方差矩阵的 k-SVD. 为此，我们介绍了基于幂法求解矩阵最大特征向量的 Shift-and-Inverse 框架，然后 k 次使用该框架求解数据协方差矩阵的前 k 个特征向量，这种方法称为 LazySVD。

在 Shift-and-Inverse 框架中，原本需要求解一个矩阵的逆与向量的乘积被转化成求解一个凸函数的最小值，通过凸优化方法可以更快地求解这个问题。于是，我们介绍了确定性优化和随机优化两类的主要代表性算法，比较了不同算法求解该凸函数的近似最小值的计算复杂度，最后我们分析了 LazySVD 在不同算法下的复杂度。

关键词：奇异值分解，凸优化，随机优化，幂法

LazySVD: Fast Singular Value Decomposition

Xiang Li (Statistic)

Directed by Prof. Zhihua Zhang

Abstract

In this paper, we aim to solve the k-SVD of the covariance matrix of collected data. For that purpose, we introduce the Shift-and-Inverse framework which products the top eigenvector of the covariance matrix based on traditional power method and then conduct it repeatedly for k times to output its first k eigenvectors. This seemingly most-intuitive approach is called LazySVD.

In the process of Shift-and-Inverse, the originally existing matrix inversion is replaced by minimizing a specific convex function, which could be solved by various convex optimization at a faster calculating speed. For that sake, two group optimization methods have been introduced, namely the deterministic optimization and the stochastic optimization. In the end, we analyze the total computation complexity of different algorithms to approximate the minimizer of that specific convex function and then that to output k-eigenvectors in LazySVD.

Keywords: k-SVD, convex optimization, stochastic optimization, power method

目录

Introduction

Principal component analysis (PCA), invented by Pearson[1] and then developed by Hotelling[2], is a statistical procedures aiming to finding a linear combination of observed data which has the largest variance in all possible combinations. Usually an orthogonal transformation is used to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**. Principal components are widely utilized in feature generation, data visualization.

For data given as a set of n vectors in $\mathbb{R}^d$, $x_1, x_2, \cdots, x_n$, denote X as the matrix form data whose i^{th} row is the transpose of i^{th} data point $\frac{1}{\sqrt{n}}x_i$ and A as the normalized covariance matrix $A = X^T X = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$. The PCA find the k-dimensional subspace where the projected data has largest variance. Formally, denoting $W \in \mathbb{R}^{d\times k}$ as the orthogonal projection matrix, we can formalize PCA as the following optimization problem

$$\max_{W\in\mathbb{R}^{d\times k}, W^T W=I} \|AW\|_F^2 \tag{1}$$

where $\|\cdot\|_F$ is the Frobenius norm. This is a non-convex optimization even in the case where $k = 1$.

PCA can be solved explicitly by the singular value decomposition (SVD) of the data covariance matrix A. Generally speaking, for a rank-r matrix $A \in \mathbb{R}^{n\times d}$ has such decomposition $A = V\Sigma U^T$, where $V \in \mathbb{R}^{n\times r}$, $U \in \mathbb{R}^{r\times d}$ are two column orthonormal matrices and $\Sigma = diag\{\sigma_1, \cdots, \sigma_r\}$ is a diagonal matrix with non-negative entries decreasing listed on its diagonal. By Eckart–Young–Mirsky

theorem, the solution of problem (1) is

$$A_k^* = V_k V_k^T A = V_k \Sigma_k U_k^T$$

where V_k, U_k are the first k columns of V and U, $\Sigma_k = diag\{\sigma_1, \cdots, \sigma_k\}$.

Traditional algorithms to compute SVD essentially run in time $O(nd \min\{d, n\})$, which is a quite expensive under big data scenario. Allen-Zhu [3] summaries the performance among different recent methods solving k-SVD. We list them in Table 1. The first gap-free running-time result is obtained by Musco and Musco[4] by subspace PM and block Krylov. The first stochastic running-time result is achieved by Shamir [5]. But his method not only depends on eigenvalue gaps, but also requires a very accurate warm-start, which would take a long time to compute.

In this paper, we give other based on the algorithmic framework in[3] to solve k-SVD. It not only improves the aforementioned breakthroughs, but also relies only on simple convex analysis. The remainder of the paper is organized as follows. In Chapter 1, we introduce and analyze the Shift-and-Inverse framework for solving 1-SVD. In Chapter 2, we introduce two groups of optimization methods namely the deterministic optimization and the stochastic optimization. Specifically, we detail the recent accelerated stochastic momentum optimization method $Katyusha\ X^s$. In Chapter 3, we give a framework of $LazySVD$ and analyze the total complexity in cases where different optimization referred in Chapter 2 are applied in its optimization oracle.

Algorithms	Running Time	GF Running Time
subspace PM [4]	$\widetilde{O}(\frac{k\cdot nnz(A)}{gap}+\frac{k^2d}{gap})$	$\widetilde{O}(\frac{k\cdot nnz(A)}{\epsilon}+\frac{k^2d}{\epsilon})$
block Krylov [4]	$\widetilde{O}(\frac{k\cdot nnz(A)}{gap^{1/2}}+\frac{k^2d}{gap}+\frac{k^3}{gap^{3/2}})$	$\widetilde{O}(\frac{k\cdot nnz(A)}{\epsilon^{1/2}}+\frac{k^2d}{\epsilon}+\frac{k^3}{\epsilon^{3/2}})$
Shamir [5]	$\widetilde{O}(knd+\frac{k^4}{\sigma_k^4 gap^{1/2}})$	-
+ FGD	$\widetilde{O}\left(\frac{k\cdot nnz(A)}{gap}+\frac{k^2d}{gap}\right)$	$\widetilde{O}\left(\frac{k\cdot nnz(A)}{\epsilon}+\frac{k^2d}{\epsilon}\right)$
+ AGD [3]	$\widetilde{O}\left(\frac{k\cdot nnz(A)}{gap^{1/2}}+\frac{k^2d}{gap^{1/2}}\right)$	$\widetilde{O}\left(\frac{k\cdot nnz(A)}{\epsilon^{1/2}}+\frac{k^2d}{\epsilon^{1/2}}\right)$
+ SAG	$\widetilde{O}\left(\frac{kd}{gap}\right)$	$\widetilde{O}\left(\frac{kd}{\epsilon}\right)$
+ SVRG	$\widetilde{O}\left(k\cdot nnz(A)+k^2d+\frac{kd}{gap^2}\right)$	$\widetilde{O}\left(k\cdot nnz(A)+k^2d+\frac{kd}{\epsilon^2}\right)$
+ accSVRG [3]	$\widetilde{O}\left(knd+\frac{kn^{3/4}d}{\sigma_k^{1/4}gap^{1/2}}\right)$	$\widetilde{O}\left(knd+\frac{kn^{3/4}d}{\sigma_k^{1/4}\epsilon^{1/2}}\right)$
+ $KatyushaX^s$	$\widetilde{O}\left(k\cdot nnz(A)+k^2d+\frac{kn^{3/4}}{gap^{1/2}})\right)$	$\widetilde{O}\left(k\cdot nnz(A)+k^2d+\frac{kn^{3/4}}{\epsilon^{1/2}})\right)$

表 1: Performance comparison among direct methods. Define $gap = (\sigma_k - \sigma_{k+1})/\sigma_k \in [0,1]$. GF means the running time if free of gap. We call a algorithm means the gradient used for updating is a unbiased estimator of full gradient. Here $LazySVD + SAG, +SVRG, +accSVRG, +KatyushaX^s$ belongs to this stochastic group. Stochastic results in this table are assuming $\|x_i\|_2 \leq 1$. We call a algorithm accelerated if the dependence on gap or ϵ of its running time is $gap^{-1/2}$ or $\epsilon^{-1/2}$. Here block Krylov, $LazySVD + AGD, +accSVRG, +KatyushaX^s$ belong to the accelerated group. The three five items are from previous work and the last six items can be deduced from this work.

第一章　The Shift-and-Inverse Framework for 1-SVD

1.1 Introduction

Traditionally, the power method solves the top eigenvector of matrix A converge in $O(log(d/\epsilon)/gap)$ iterations, where $gap = (\lambda_1 - \lambda_2)/\lambda_1$ and λ_i denotes the i^{th} largest eigenvalue of A. We will use this notation till the end of the paper except specifically stated in certain sections. It is quite unsatisfactory when the gap is quite small.

In order to get free of eigenvalue gap, we aim to solve k-SVD by power method modified by Shift-and-Inverse framework. The framework is a combination of traditional ideas, namely the *shifted power method* and the *inverse iteration*[6]. The former applies power method to shifted covariance matrix $A + \sigma I$ and the later choose $(A - \sigma I)^{-1}$ as the counterpart. Unlike these two methods, here we choose $(\lambda I - A)^{-1}$. If $\lambda > \lambda_1$, we can see the top eigenvector of B is equal to that of A, but the new gap has become $\frac{\lambda_1 - \lambda_2}{\lambda - \lambda_2}$. As long as λ is sufficiently close to λ_1, there will be constant gap such that power iteration only needs $O(log(d/\epsilon))$ to converge, which is gap-free.

However, after we can get rid of gap dependency in the iteration, here comes the problem – matrix inversion. Dan Garber[7] proposes to solve the linear system $Mx = b$ via convex optimization, i.e. to find the minimizer of the convex function

$F(x) = \frac{1}{2}x^T Mx - b^T x$ instead of inversing matrix directly. Recent stochastic optimizers could be applied to solve it, here we denote such algorithm as $\mathcal{A}$. We will discuss what kind of $\mathcal{A}$ need to be chosen in next chapter. Therefore, in order to obtain the computation cost of a ϵ-tolerated solution, we only need to figure out how many times $\mathcal{A}$ has been called.

We list the pseudo code in Algorithm 1, which is referenced from [3].

Algorithm 1 APPXPCA($\mathcal{A}, A, \delta, \epsilon, p$)

Input: $\mathcal{A}$, an approximate matrix inversion method. $A \in \mathbb{R}^{d\times d}$, a covariance matrix satisfying $0 \prec A \prec I$; δ, a multiplicative error; ϵ, numerical accuracy parameter; $p \in (0, 1)$, failure probability parameters.
Setting parameters: $m_1 \leftarrow T^{PM}(8, \frac{1}{32}, p)$, $m_2 \leftarrow T^{PM}(2, \frac{\epsilon}{4}, p)$, $\tilde{\epsilon}_1 \leftarrow \frac{1}{64m_1}(\frac{\delta}{6})^{m_1}$, $\tilde{\epsilon}_2 \leftarrow \frac{1}{8m_2}(\frac{\delta}{6})^{m_2}$.
Initialization: $\hat{w}_0 \leftarrow$ a random unit vector; $s \leftarrow 0$; $\lambda^{(0)} \leftarrow 1 + \delta$.
repeat
    $s \leftarrow s + 1$
    for $t = 1...m_1$ **do**
        Apply $\mathcal{A}$ to find $\hat{w}_t$ s.t. $\|\hat{w}_t - (\lambda^{(s-1)}I - A)^{-1}\hat{w}_{t-1}\| \leq \tilde{\epsilon}_1$
    end for
    $w \leftarrow \frac{\hat{w}_{m_1}}{\|\hat{w}_{m_1}\|}$
    Apply $\mathcal{A}$ to find v s.t. $\|v - (\lambda^{(s-1)}I - A)^{-1}w\| \leq \tilde{\epsilon}_1$
    Update parameters: $\Delta^{(s)} \leftarrow \frac{1}{2} \cdot \frac{1}{w^T v - \tilde{\epsilon}_1}$; $\lambda^{(s)} \leftarrow \lambda^{(s-1)} - \frac{\Delta^{(s)}}{2}$.
until $\Delta^{(s)} \leq \frac{\delta\lambda^{(s)}}{3}$
$f \leftarrow s$
for $t = 1...m_2$ **do**
    Apply $\mathcal{A}$ to find $\hat{w}_t$ s.t. $\|\hat{w}_t - (\lambda^{(f)}I - A)^{-1}\hat{w}_{t-1}\| \leq \tilde{\epsilon}_2$
end for
return $w_f \leftarrow \frac{\hat{w}_{m_2}}{\|\hat{w}_{m_2}\|}$

1.2 Analysis

Denote $M_s = (\lambda^{(s-1)}I - A)^{-1}$, and when analyzed, the subscript s of M_s will be omitted for simplicity. The inner loop deals with finding the top eigenvector of M_s. The classic power method to do that thing will first find a random initial unit vector $\hat{w}_0$ and then applies $w_t \leftarrow \frac{Mw_{t-1}}{\|Mw_{t-1}\|}$ iteratively. Lemma (1.2.1) states that only $T^{PM}(\kappa, \epsilon, p)$ iterations are needed to obtain an ϵ-approximate solution with probability at least $1 - p$.

Lemma 1.2.1 (Exact Power Method)**.** *Assume M is a PSD matrix with non-increasing eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ and their corresponding eigenvectors $u_1, u_2, \cdots, u_d$. Given an error tolerance $\epsilon > 0$, approximation-control parameter $\kappa \geq 1$, and failure probability $p > 0$, let*

$$T^{PM}(\kappa, \epsilon, p) = \lceil \frac{\kappa}{2} log\left(\frac{9d}{p^2\epsilon}\right) \rceil$$

Then, with probability at least $1 - p$, it holds that as long as $t \geq T^{PM}(\kappa, \epsilon, p)$, we have

$$\sum_{i \in [d], \lambda_i \leq (1-\frac{1}{\kappa})\lambda_1} (w_t^T u_i) \leq \epsilon \quad and \quad w_t^T M w_t \geq (1 - \frac{1}{\kappa} - \epsilon)\lambda_1$$

However, in order to avoid matrix inversion, exact power method would be replaced by inexact power method, just like what we do in Algorithm 1. In each inner loop, we only calculate a $\tilde{\epsilon}_1$-approximate of the product of inverted matrix $(\lambda^{(s-1)}I - A)^{-1}$ and last result $\hat{w}_{s-1}$. After sufficient iterations, namely m_1, we normalize $\hat{w}_t$ and regard it as the approximate top eigenvector of $(\lambda^{(s-1)}I - A)^{-1}$. Lemma (1.2.2) states how the accumulated error grows during inner iterations, which we can utilize to control the ultimate error.

Lemma 1.2.2. *Denote the sequence of iterations in Exact Power Method as w_i^*, which satisfies $w_0^* = w_0, w_t^* = \frac{Mw_{t-1}^*}{\|Mw_{t-1}^*\|}$, and the sequence of iterations in Inexact*

Power Method as $w_t = \frac{\hat{w}_t}{\|\hat{w}_t\|}$, $\|\hat{w}_t - M\hat{w}_{t-1}\| \leq \tilde{\epsilon}$. *Then define*

$$\Gamma(M,t) = \frac{2}{\lambda_d^t} \begin{cases} t & if\ \lambda_1 = 1 \\ \dfrac{\lambda_1^t - 1}{\lambda_1 - 1} & if\ \lambda_1 \neq 1 \end{cases},$$

it satisfies that

$$\|w_t - w_t^*\| \leq \tilde{\epsilon} \cdot \Gamma(M,t)$$

Based on lemma (1.2.2), we can get a similar converge result in Inexact Power Method case, we list it in theorem 1.2.3. The proof of theorem 1.2.3 and lemma 1.2.1 see [3] and the proof of lemma 1.2.2 sees [7].

Theorem 1.2.3 (Inexact Power Method)**.** *Assume M is a PSD matrix with non-increasing eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ and their corresponding eigenvectors $u_1, u_2, \cdots, u_d$. Given an error tolerance $\epsilon > 0$, approximation-control parameter $\kappa \geq 1$, and failure probability $p > 0$, then, with probability at least $1 - p$, it holds that for every $\epsilon \in (0,1)$ and every $t \geq T^{PM}(\kappa, \frac{\epsilon}{4}, p)$, if w_t is generated by Inexact Power Method with per-iteration error $\tilde{\epsilon} = \frac{\epsilon}{4\Gamma(M,t)}$, then*

$$\sum_{i \in [d], \lambda_i \leq (1-\frac{1}{\kappa})\lambda_1} (w_t^T u_i) \leq \epsilon \quad and \quad w_t^T M w_t \geq (1 - \frac{1}{\kappa} - \epsilon)\lambda_1$$

Now we have figured out how error each inner loop will bring. Let's focus on the outer loop. In each outer loop, after calculating an approximate top eigenvector of $(\lambda^{(s-1)}I - A)^{-1}$, we then calculate $\Delta^{(s)}$ and shrink $\lambda^{(s)}$ based on it, since $\Delta^{(s)}$ measure the how far $\lambda^{(s-1)}$ is away from λ_1. By carefully choosing the parameters in Algorithm 1, $\tilde{\epsilon}_1 \leq \frac{1}{32\Gamma((\lambda^{(s-1)}-M)^{-1}, m_1)}$ for each s and $\tilde{\epsilon}_2 \leq \frac{1}{4\Gamma((\lambda^{(f)}-M)^{-1}, m_2)}$, which guarantees that w_t generated by Inexact Power Method is with per-iteration error ϵ. The way $\Delta^{(s)}$ computed satisfies $0 \leq \frac{\lambda^{(s-1)} - \lambda_1}{2} \leq \Delta^{(s)} \leq \lambda^{(s-1)} - \lambda_1$ for each s, which results $\lambda^{(f)} - \lambda_1$ couldn't lie out of $[\frac{\delta}{12}\lambda^{(f)}, \delta\lambda_1]$ when setting $s = f$. Putting all the results together, we can give analysis about how accurate the output of Algorithm 1 in theorem 1.2.4 is.

Theorem 1.2.4 (Approximation under Shift-and-Inverse framework)**.** *Let A is a PSD matrix with non-increasing eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ and their corresponding eigenvectors $u_1, u_2, \cdots, u_d$. Given an error tolerance $\epsilon > 0$, a multiplicative error δ, and failure probability $p > 0$, then with probability at least $1 - p$, the output w produced by Algorithm 1 satisfies*

$$\sum_{i \in [d], \lambda_i \leq (1-\delta)\lambda_1} (w^T u_i)^2 \leq \epsilon \quad \textit{and} \quad w^T A w \geq (1-\delta)(1-\epsilon)\lambda_1$$

Furthermore, the total number of oracle calls to $\mathcal{A}$ if $O\left(log(\frac{1}{\delta})m_1 + m_2\right)$ and each time we call $\mathcal{A}$, we have $\frac{\lambda^{(s)}}{\lambda_{min}(\lambda^{(s)}I - A)} \leq \frac{12}{\delta}$ and $\frac{1}{\lambda_{min}(\lambda^{(s)}I - A)} \leq \frac{12}{\delta\lambda_1}$.

In theorem 1.2.4, conclusions about A rather that M are given, since this theorem is produced after the whole framework is analyzed. If $0 < \delta < \frac{\lambda_1 - \lambda_2}{\lambda_1}$, then $\|w^T u_i\|^2 = 1 - \sum_{i \in [d], \lambda_i \leq (1-\delta)\lambda_1} (w^T u_i)^2 \geq 1 - \epsilon$, indicating w is a good approximator. Therefore, the result that $w^T A w$ is not far away from λ_1 is easily deduced.

In order to analyze the arithmetic complexity of Algorithm 1 using a specific implementation for the optimization oracle $\mathcal{A}$, it is not only important to bound the number of calls to $\mathcal{A}$ (as done in Theorem 1.2.4), but to also bound important parameters of the optimization problem that naturally arise when considering the arithmetic complexity of different implementations for $\mathcal{A}$. For this issue, we should next introduce the concept of convexity and smoothness and then various optimization methods.

第二章　The choice of optimization oracle

2.1　Definition of convexity

Often the complexity of $\mathcal{A}$ outputting $\hat{w}_t$ has some dependency on conditional number κ. To give the definition of conditional number, the smoothness and strong convexity of the loss function should first be defined[8].

Definition 2.1.1. *Function f(x) is said to* ***convex****, if for all* $x, y \in \mathbb{R}^d$ *and* $\forall \alpha \in [0, 1]$*, it holds that*

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + \alpha f(y)$$

Definition 2.1.2. *Function f(x) is said to be* ***L-smooth****, if f(x) is derivable and for for all* $x, y \in \mathbb{R}^d$*, it holds that*

$$|\nabla f(x) - \nabla f(y)| \leq L|x - y|$$

Definition 2.1.3. *Function f(x) is said to be* σ***-strong convex****, if for all* $x, y \in \mathbb{R}^d$*, it holds that*

$$f(x) \geq f(y) + \langle \nabla f(y), x - y \rangle + \frac{\sigma}{2}\|x - y\|^2$$

If a loss function f is both L-smooth and σ-strong-convex, then its conditional number is defined as $\kappa = \frac{L}{\sigma}$. Therefore, when we decide to choose the Algorithm $\mathcal{A}$, its running cost should have less dependency on κ.

Lemma 2.1.1. *Let λ, w be such that during the run of Algorithm 1, the optimization oracle $\mathcal{A}$ is applied to the minimization of the function*

$$F_{w,\lambda}(z) = \frac{1}{2} z^T (\lambda I - A) z - w^T z.$$

Then, under the conditions stated in Theorem 1.2.4 it holds that $F_{w,\lambda}(z)$ is $(\lambda - \lambda_1) = \Omega(\delta)$-strongly convex and for all $i \in [n]$ it holds that the function $f_i(z) = \frac{1}{2} z^T (\lambda I - x_i x_i^T) z - w^T z$ is $1 + \delta = O(1)$-smooth.

2.2 Optimization methods

In our case, inverting a matrix is equivalent to minimizing a convex function $F_{w,\lambda}(\cdot)$. Various stochastic optimization techniques can be applied to solve it. In this chapter, we want to approximate matrix inverse by convex optimization. We first give the definition of our problem, and then introduce some classic or famous algorithms. These algorithms could be categoried into two groups, one being the deterministic optimization and one being the stochastic optimization. In the former group, the full gradient descent (FGD) and accelerated gradient descent (AGD) will be introduced, and in the later group, stochastic gradient descent (SGD), stochastic variance reduced gradient (SVRG) and stochastic average gradient (SAG) will be detailed.

Problem Given a $d \times d$ matrix $M \succeq 0$ satisfying $\lambda I - M \succeq \mu I$ for some constant λ and $\mu > 0$, one can minimize the quadratic

$$F_{w,\lambda}(x) = x^T (\lambda I - M) x - w^T x, \tag{2.1}$$

in order to invert $(\lambda I - M)^{-1} b$, i.e. to find a solution x such that $\|x - (\lambda I - M)^{-1} b\| \leq \epsilon$ for some prescribed ϵ. Our problem is how to find a 'good' algorithm to give such x.

2.2.1 Accelerated Gradient Descent (AGD)

Full gradient decent (FGD) is a classic first-order optimization method. For the function $F_{w,\lambda}(x)$ to optimize, FGD updates the parameter by moving the parameter towards the steepest direction on the empirical loss surface, i.e.

$$x_t \leftarrow x_{t-1} - \eta \nabla F_{w,\lambda}(x_{t-1}) \tag{2.2}$$

where η is the learning rate. Classic convex optimization[8] shows that in our μ-strong convex case, FGD finds an ϵ-approxiated minimizer of (2.1) in $O(\frac{\lambda}{\mu} log \frac{\lambda}{\epsilon\mu})$ iteration, each requiring $O(d)$ time plus the time needed to multiply M with a vector. We could regard $\frac{\lambda}{\mu}$ as the condition number of function 2.1.

Lemma 2.2.1. *FGD produces such an output x in $O(\frac{\lambda}{\mu} log \frac{\lambda}{\epsilon\mu})$ iterations, each requiring $O(d)$ time plus the time needed to multiply M with a vector, i.e.*

$$O\left((d + nnz(M)) \frac{\lambda}{\mu} log \frac{\lambda}{\epsilon\mu} \right)$$

Nesterov accelerate scheme, which we call Accelerated Gradient Decent, could reduce the needed iteration number to $O(\frac{\lambda^{1/2}}{\mu^{1/2}} log \frac{\lambda}{\epsilon\mu})$, which has better dependency on condition number. AGD updates parameters in the next fashion

$$\begin{aligned} y_t &\leftarrow x_{t-1} - \frac{1}{\lambda} \nabla F_{w,\lambda}(x) \\ x_t &\leftarrow (1 - \gamma_{t-1}) y_t + \gamma_{t-1} y_{t-1} \end{aligned} \tag{2.3}$$

where $\{\gamma_t\}_{t=0}$ satisfies $\gamma_t = \frac{1-\theta_t}{\theta_{t+1}}$, and $\theta_0 = 0$, $\theta_t = \frac{1+\sqrt{1+4\theta_{t-1}^2}}{2}$ following [9]. Further, we have the following lemma.

Lemma 2.2.2. *AGD produces such an output x in $O(\frac{\lambda^{1/2}}{\mu^{1/2}} log \frac{\lambda}{\epsilon\mu})$ iterations, each requiring $O(d)$ time plus the time needed to multiply M with a vector, i.e.*

$$O\left((d + nnz(M)) \frac{\lambda^{1/2}}{\mu^{1/2}} log \frac{\lambda}{\epsilon\mu} \right)$$

2.2.2 Stochastic Gradient Descent (SGD)

If $\nabla F_{w,\lambda}(x_{t-1})$ is replaced by some subfunction $\nabla f_{i_t}(x_{t-1})$ at iteration t, where i_t is uniformly and independently drawn from $[n] := \{1, 2, \cdots, n\}$, we obtain the most practical optimization method, stochastic gradient descent (SGD). Often gradients estimated by only one subfunction will have large variance. One can take the place of $\nabla f_{i_t}(x_{t-1})$ by $\frac{1}{B_t}\sum_{i\in B_t}\nabla f_i(x_{t-1})$ to lower the variance, where B_t is a random set of b different number uniformly drawn from $[n]$. We call this kind of SGD as minibatch SGD, which has been proved to enjoy linear convergence rate just like FGD[10]. Specifically, for our problem, SGD will give an ϵ-approxiated minimizer of (2.1) in $\widetilde{O}\left(\frac{\lambda}{\mu}log\frac{\lambda}{\mu\epsilon} + \frac{\sigma^2 d}{\epsilon}\right)$, where σ^2 is the variance of the gradient estimator.

Recently, there outbursts plenty of SGD variants aiming to adjust learning rate η automatically. We list some famous here. AdaGrad[11] pointwisely normalizes learning rate by the square root of accumulative second moments, while RMSProp[12] dose the same thing by the moving average of the magnitudes of recent gradients and accumulative ones. Adam[13] bases on adaptive estimates of first and second order moments to obtain an unbiased estimator of gradient.

Another famous question is whether SGD could be accelerated in the way FGD modified to AGD. Actually, existing results show AGD not robust to deterministic noise ([14], [15]), but is robust to random additive noise ([16], [17]). Stochastic approximation falls between the above two cases. [18] introduces an accelerated stochastic gradient method that provably achieves the minimax optimal statistical risk faster than SGD, which gives an ϵ-approxiated minimizer of (2.1) in $\widetilde{O}\left(\sqrt{\frac{\tilde{\kappa}\lambda}{\mu}}log\frac{\lambda}{\mu\epsilon} + \frac{\sigma^2 d}{\epsilon}\right)$, where $\tilde{\kappa}$ is the statistical condition number. As we could see, as long as $\tilde{\kappa} \ll \kappa = \frac{\lambda}{\mu}$, acceleration is possible.

2.2.3 Stochastic Variance Reduced Gradient (SVRG)

SVRG maintains a full gradient each outer loop and computes a single random gradient each inner loop to reduce the large noise and further reduce the variance of gradients. Theorem 1 in [19] proves that SVRG enjoys a linear convergence rate and theorem $B.1$ in [7] further proves that when $F_{w,\lambda}(\cdot)$ is σ-strong convex and $f_i(\cdot)$ is L-smooth, then when setting $\eta = O(\frac{\sigma}{L^2})$, then only $m = \widetilde{O}(\frac{1}{\eta^2 L^2})$ inner iterations need to obtain a ϵ-approximate solution. To say it formally, we have

Lemma 2.2.3. *Given $\epsilon, p \in (0,1)$, there exists a choice of η, m, such that Algorithm 2 finds with probability at least $1-p$, an ϵ-approxiated minimizer of (2.1) in overall time*

$$O\left((nnz(M) + \frac{d\lambda^2}{\mu^2})log\frac{\lambda}{\mu\epsilon}\right)$$

Algorithm 2 SVRG(A, $\tilde{x}_0, \eta, m$)

Input: $\tilde{x}_0$ initial x; η, learning rate; m, iteration numbers.
for $s = 1, 2, ...$ **do**
  Initialization: $\tilde{x} \leftarrow \tilde{x}_{s-1}, \tilde{\mu} \leftarrow \nabla F_{w,\lambda}(\tilde{x}), x_0 \leftarrow \tilde{x}$
  for $t = 1, 2, ..., m$ **do**
    Randomly pick $i_t \in [n]$
    $x_t \leftarrow x_{t-1} - \eta\left(\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{\mu}\right)$
  end for
  $\tilde{x}_s \leftarrow \frac{1}{m}\sum_{t=0}^{m-1} x_t$
end for

Here we assume different f_i has the same smoothness coefficient L which could be attributed to the uniform sampling of index i in each inner loop. If β could be varied, non-uniform sampling could be applied just as what [20] did. In [20], index i is drawn with probability $p_i = \frac{\|x_i\|_2^2}{\sum_{k=1}^n \|x_k\|_2^2}$, and the corresponding results have some modification of the dependency on $\{L_i\}_{i=1}^n$.

What's more, [21] proposed a universal framework to accelerate arbitrary gradient optimization in an almost black-box fashion, known as the Accelerated Proximal-point algorithm. This method will use convex optimization to find an approximated global minimizer of the modified function

$$\widetilde{F}_{w,\lambda}(x) = F_{w,\lambda}(x) + \frac{\theta}{2}\|x - \widetilde{x}\|^2 \tag{2.4}$$

where θ is the regulation parameter and $\widetilde{x}$ is the iterating result in previous loop. By Theorem 3.1(rephrased by our needs) in [20], when fixing θ, there exist an acceleration scheme for Algorithm 2 that find an ϵ-approximated minimizer of $F_{w,\lambda}(x)$ after approximately minimizing $\widetilde{O}\left(\sqrt{\frac{\sigma+\theta}{\sigma}}\right)$ instances of 2.4. Therefore, we could obtain an accelerated result

Lemma 2.2.4. *Given $\epsilon, p \in (0,1)$, if $\mu = o(\sqrt{\frac{d}{n}})$, there exist an accelerated Algorithm 2 finds with probability at least $1-p$, an ϵ-approxiated minimizer of (2.1) in overall time*

$$O\left(\frac{nnz(M)^{3/4}d^{1/4}\lambda^{1/4}}{\sqrt{\mu}} log\frac{1}{\epsilon}\right)$$

Sometimes $\mu = o(\sqrt{\frac{d}{n}})$ is barely met. [22] gives an improved SVRG by replacing the gradient descent with a proximal gradient. (line 6 in Algorithm 2). Under mild assumptions, it can achieve comparable results in the accelerated case.

Lemma 2.2.5. *If $M = A = \frac{1}{n}\sum_{i=1}^{n} x_i^T x_i$ and $\|a_i\|_2 \leq 1$, then in expectation there exists an accelerated version of SVRG (see for instance [22]) producing an ϵ-approxiated minimizer of (2.1) in overall time*

$$O\left(max\{nd, \frac{n^{3/4}d\lambda^{1/4}}{\mu^{1/2}}\} log\frac{\lambda}{\epsilon\mu}\right)$$

2.2.4 Stochastic Average Gradient (SAG)

SAG[23] keeps maintaining a full gradient each iteration instead of computing a new full gradient each outer loop like SVRG, which will reduce the computation

complexity. By incorporating a memory of previous gradient values, SAG method also achieves a linear convergence rate[24]. However, the advantage of less computation is at the price of $O(nd)$ space complexity, since for each data point should maintain its current gradient.

Algorithm 3 SAG(A, x_0, η, m)

1: Input: x_0 initial x; η, learning rate; m, iteration numbers.
2: Initialization: $\tilde{\mu} \leftarrow 0, g_i = 0$ for $\forall i \in [n]$.
3: **for** $t = 1, 2, ..., m$ **do**
4: Randomly pick $i_t \in [n]$
5: $\tilde{\mu} \leftarrow \tilde{\mu} - g_{i_t} + \nabla f_{i_t}(x_{t-1})$
6: $g_{i_t} \leftarrow \nabla f_{i_t}(x_{t-1})$
7: $x_t \leftarrow x_{t-1} - \frac{\eta}{n}\tilde{\mu}$
8: **end for**

Applying the linear convergence rate in our problem, we have the following lemma. In term of condition number, SAG fails to outperform AGD and SVRG, where the power of $\kappa = \frac{\lambda}{\mu}$ in the total complexity is no more than $\frac{1}{2}$. However, the case where the covariance matrix A is dense, i.e. $nnz(A)$ is pretty large, would favor SAG rather than others, since SAG is free of $nnz(A)$.

Lemma 2.2.6. *SAG produces such an output x in $O(\frac{\lambda}{\mu} log \frac{\lambda}{\epsilon\mu})$ iterations, each requiring $O(d)$, i.e.*

$$O\left(\frac{d\lambda}{\mu} log \frac{\lambda}{\epsilon\mu}\right)$$

2.3 Katyusha X

From previous sections, we have seen that it seems hard to accelerate stochastic algorithms, such as SGD. Though SVRG can outperform SGD in both convex and strong convex case, how to accelerate it may not an easy task. Recently, Allen-Zhu[25] proposes a new acclerated and stochastic method for minimizing (2.1) by

introducing a carefully-designed interpolation of gradient descent and mirror descent. This method called **Katyusha X**. Actually this method can solve more general problem, i.e. sum-of-nonconvex problem. Here we call a function sum-of-nonconvex, if the function is actually a sum of nonconvex subfunctions but itself still convex function. For example, function (2.1) belongs to that group, since each $f_i(x)$ is smooth and non-convex, but their average $\frac{1}{n}\sum_{i=1}^n f_i(x)$ is μ-strong convex.

Algorithm 4 $SVRG^{1ep}(F_{w,\lambda}, x_0, b, \eta)$

Input: $F_{w,\lambda} = \frac{1}{n}\sum_{i=1}^n f_i(x)$; starting vector x_0; mini-batch size $b \in [n]$; learning rate $\eta > 0$.
Output: x^+
for $s = 1, 2, ...$ **do**
  Initialization: $m \leftarrow min\{\lceil \frac{n}{b} \rceil, 2\}; M \sim Geom(\frac{1}{m}); \mu \leftarrow \nabla F_{w,\lambda}(x_0)$
  for $t = 1, 2, ..., M$ **do**
    Let S_t be b i.i.d uniform random indices from $[n]$ with replacement
    $\widetilde{\nabla}_t \leftarrow \mu + \frac{1}{b}\sum_{i\in S_t}(\nabla f_i(x_t) - \nabla f_i(x_0))$
    $x_{t+1} \leftarrow \arg\min_{y\in\mathbb{R}^d}\{\|y - x_t\|^2 + 2\eta\langle\widetilde{\nabla}_t, y\rangle\}$
  end for
  $x^+ \leftarrow x_{M+1}$
end for

To keep consistent with the notation in [25], we denote each $f_i(x)$ is L-smooth (obviously here $L = 1 + \delta$) and $F_{w,\lambda}(x)$ is still μ-strong convex. This method first modifies proximal SVRG to $SVRG^{1ep}$ (Algorithm 4, with some modification from original version in [25]), and finds that up to a constant factor 2, the output of $SVRG^{1ep}$ can be viewed as a fuull gradient descent with a virtual step length $m\eta$.

Specifically, if $b \in [n]$ is the mini-batch size and $m = \min\{\lceil \frac{n}{b} \rceil, 2\}$ is the epoch length of $SVRG^{1ep}$, when $\eta \leq \min\{\frac{1}{L}, \frac{\sqrt{b}}{2L\sqrt{m}}\}$, let $x^+ = SVRG^{1ep}(F_{w,\lambda}, x_0, b, \eta)$, then

for $\forall u \in \mathbb{R}^d$, we have

$$\mathbb{E}\left[F_{w,\lambda}(x^+) - F_{w,\lambda}(u)\right] \leq \left[-\frac{1}{4m\eta}\|x^+ - x_0\|^2 + \langle\frac{x_0 - x^+}{m\eta}, x_0 - u\rangle - \frac{\mu}{4}\|x^+ - u\|^2\right]$$

Denoting $\mathcal{G} = \frac{x_0 - x^+}{m\eta}$, then it satisfies that

$$\mathbb{E}\left[F_{w,\lambda}(x^+) - F_{w,\lambda}(u)\right] \leq \left[-\frac{m\eta}{4}\|\mathcal{G}\|^2 + \langle\mathcal{G}, x_0 - u\rangle - \frac{\mu}{4}\|x^+ - u\|^2\right] \tag{2.5}$$

In comparison, if a full proximal gradient descent with step length $\frac{1}{L}$ is applied $y^+ \leftarrow \arg\min_{z\in\mathbb{R}^d}\{\frac{L}{2}\|y - z\|^2 + \langle\nabla F_{w,\lambda}(y), z\rangle\}$ and denote by $\mathcal{G} = L(y - y^+)$ the so-called gradient mapping, the classical theory[26] essentially tells us

$$\mathbb{E}\left[F_{w,\lambda}(y^+) - F_{w,\lambda}(u)\right] \leq \left[-\frac{1}{2L}\|\mathcal{G}\|^2 + \langle\mathcal{G}, y - u\rangle - \frac{\mu}{2}\|y^+ - u\|^2\right] \tag{2.6}$$

By comparing (2.5) and (2.6), up to a constant factor 2, the output of $SVRG^{1ep}$ can be viewed as a fuull gradient descent with a virtual step length $m\eta$. We shall later use $SVRG^{1ep}$ in a black-box way to obtain a gradient.

Then the Nesterov-kind interpolation could be applied to accelerate $SVRG^{1ep}$. Specifically we want to apply

$$x_{k+1} = \frac{\frac{3}{2}y_k + \frac{1}{2}x_k - (1-\tau)y_{k-1}}{1+\tau} \tag{2.7}$$

as the new choice of momentum which is motivated by the linear-coupling analysis of accelerated methods[27]. If one replaces the x_k term on the right hand side of (2.7) with y_k, the original Nesterov's momentum will be got. Such new momentum is actually a special case of a general framework(2.3) of accelerated methods.

General Framework

Starting from $z_0 = y_0 = x_0$, then in each iteration $k = 1, 2, \cdots, K-1$,

- $x_{k+1} \leftarrow \tau_k z_k + (1-\tau_k)y_k$ for some $\tau_k \in [0, 1]$;
- $y_{k+1} = SVRG^{1ep}(F_{w,\lambda}, x_{k+1}, b, \eta)$ and let $\mathcal{G}_{k+1} = \frac{x_{k+1}y_{k+1}}{m\eta}$ be the gradient mapping;
- $z_{k+1} \leftarrow \arg\min_{z\in\mathcal{R}^d}\{\frac{1}{\alpha_{k+1}}\|z - z_{z_k}\|^2 + \langle\mathcal{G}_{k+1}, z\rangle + \frac{\mu}{4}\|z - y_{k+1}\|^2\}$ for some $\alpha_{k+1} > 0$.

In the general framework, the first line is to interpolate two kinds of gradients, one from gradient descent y_k and another from mirror descent z_k. The updating for y_{k+1} is to implement $SVRG^{1ep}$ to x_{k+1} as a virtual gradient descent, and the gradient mapping $\mathcal{G}_{k+1}$ is the byproduct. z_{k+1} can be viewed as mirror descent product with quadratic function as the Bregman Divergence.

If we choose $\tau_k = \tau := \frac{\sqrt{mn\mu}}{2}$ and $\alpha_{k+1} = \frac{m\eta}{2\tau} = \frac{2\tau}{\mu}$, the general framework is turned into **Katyusha X**. Specifically, after plugging parameters and eliminating the sequence $\{z_k\}_{k=0}^K$, the updating rule becomes

$$x_{k+1} \leftarrow \frac{\frac{3}{2}y_k + \frac{1}{2}x_k - (1-\tau)y_{k-1}}{1+\tau}$$
$$y_{k+1} = SVRG^{1ep}(F_{w,\lambda}, x_{k+1}, b, \eta)$$

Theorem 2.3.1. *To find the minimizer of the function (2.1), which is μ-strong convex and each $f_i(x)$ is L-smooth, then $KatyushaX^s$ with $\eta = \min\{\frac{1}{2L}, \frac{\sqrt{b}}{2L\sqrt{m}}\}$ and $\tau = min\{\frac{1}{2}, \frac{\sqrt{\mu m \eta}}{2}\}$ outputs a point x with $\mathbb{E}\left[F_{w,\lambda}(x) - F_{w,\lambda}(x^*)\right] \leq \epsilon$ with*

$$O\left(\left(nnz(M) + \frac{\sqrt{Lbn}}{\sqrt{\mu}} + \frac{n^{3/4}\sqrt{L}}{\sqrt{\mu}}\right) log\frac{1}{\epsilon}\right)$$

In our case, setting $b = 1$, then $\eta = \Theta\left(\frac{1}{\sqrt{n}L}\right)$ and $\tau = \{\frac{1}{2}, \Theta\left(\frac{n^{1/4}\sqrt{\mu}}{\sqrt{L}}\right)\}$, then $KatyushaX^s$ will output a ϵ-approximate minimizer in the sense of expectation in total complexity of (since $L = 1 + \delta \leq \lambda$)

$$O\left(\left(nnz(M) + \frac{n^{3/4}\sqrt{\lambda}}{\sqrt{\mu}}\right) log\frac{\lambda}{\epsilon\mu}\right)$$

第三章　The LazySVD Framework for k-SVD

3.1 High-level ideas about LazySVD

LazySVD(Algorithm 5) performs 1-SVD repeatedly, k times in total. Set $A_0 = A$. Specifically, at s^{th} round, LazySVD will first compute the leading eigenvector of current data covariance matrix A_{s-1}, then project it into the complement of the subspace spanned by computed $s-1$ eigenvectors, and last normalized it denoted by v_s. After updating A_{s-1} by left-projecting and the right-projecting $I - v_s v_s^T$, repeat such loop until s reaches k.

3.2 Analysis of LazySVD

We state the approximation and running time core theorems of LazySVD below, and then provide corollaries to translate them into gap-dependent and gap-free theorems on k-SVD.

Theorem 3.2.1 (Approximation of k-top eigenvectors)**.** *Let $A \in \mathcal{A}^{d\times d}$ be a symmetric matrix with non-decreasing eigenvalues $1 \geq \lambda_1 \geq \cdots \geq \lambda_d \geq 0$ and their corresponding eigenvectors $u_1, u_2, \cdots, u_d$. Let $k \in [d], \delta, p \in (0,1)$. Then with probability at least $1-p$, LazySVD outputs a column orthonormal matrix $V_k = (v_1, v_2, \cdots, v_k) \in \mathbb{R}^{d\times k}$ satisfying all of the following properties, as long as ϵ_{pca}*

Algorithm 5 LazySVD$(\mathcal{A}, A, k, \delta, \epsilon_{pca}, p)$

1: Input: $\mathcal{A}$, an approximate matrix inversion method. $A \in \mathbb{R}^{d\times d}$, a covariance matrix satisfying $0 \prec A \prec I$; $k \in [d]$, the desired rank; δ, a multiplicative error; ϵ_{pca}, numerical accuracy parameter; $p \in (0, 1)$, failure probability parameters.
2: Initialization: $A_0 \leftarrow A, V_0 \leftarrow []$
3: **for** $t = 1$ to k **do**
4: $\quad v_s^{'} \leftarrow AppxPCA(\mathcal{A}, A_{s-1}, \frac{\delta}{2}, \epsilon_{pca}, \frac{p}{k})$
5: $\quad v_s \leftarrow \left((I - V_{s-1}V_{s-1}^T)v_s^{'}\right) / \| \left((I - V_{s-1}V_{s-1}^T)v_s^{'}\right) \|$
6: $\quad V_s \leftarrow [V_{s-1}, v_s]$
7: $\quad A_s \leftarrow \left(I - v_s^T v_s\right) A_{s-1} \left(I - v_s^T v_s\right)$
8: **end for**
9: **return** V_k

satisfies corresponding conditions. Denote by $A_k = (I - V_kV_k^T)A(I - V_kV_k^T)$.

1. ***Approximate orthogonality guarantee****: If* $\epsilon_{pca} \leq \frac{\epsilon^4\delta^2}{2^{12}k^4(\frac{\lambda_1}{\lambda_k})^2}$*, then* $\|V_k^TU\| \leq \epsilon$*, where* $U = (u_j, \cdots, u_d)$ *is the column orthonormal matrix and* j *is the smallest index satisfying* $\lambda_j \leq (1 - \delta)\|M_{k-1}\|_2$.

2. ***Spectral norm guarantee****: If* $\epsilon_{pca} \leq \frac{\delta^6}{2^28k^4(\frac{\lambda_1}{\lambda_{k+1}})^6}$*, then* $\lambda_{k+1} \leq \|M_k\|_2 \leq \frac{\lambda_{k+1}}{1-\delta}$.

3. ***Rayleigh quotient guarantee****: If* $\epsilon_{pca} \leq \frac{\delta^6}{2^28k^4(\frac{\lambda_1}{\lambda_{k+1}})^6}$*, then* $(1 - \delta)\lambda_k \leq v_k^TM_kv_k \leq \frac{\lambda_k}{1-\delta}$.

4. ***Schatten-q norm guarantee****: For every* $q \geq 1$*, if* $\epsilon_{pca} \leq \frac{\delta^6}{2^28k^4(\frac{\lambda_1}{\lambda_{k+1}})^6}$*, then* $\|M_k\|_{S_q} \leq \left(\frac{1+\delta}{1-\delta}\right)^2 (\sum_{i=k+1}^d \lambda_i^q)^{1/q}$.

Theorem 3.2.1 gives theoretical guarantees of convergence of LazySVD from four aspects, among which is of importance the first guarantee since it makes sure that outputs produced by algorithm 5 approximately lie in the top-k eigenvector space and other guarantees could be deduced from it. Detail proof sees [3].

Remarks The Schatten-q norm of arbitrary symmetric matrix $B \in \mathbb{R}^{d\times d}$ is defined as $\|B\|_{S_q} = (\sum_{i=1}^{d} \lambda_i^q)^{1/q}$, where λ_i is the i^{th} largest eigenvalue of B. The Schatten-q norm is reduced to the Frobenius norm when $q = 2$ and reduced to spectral norm when $q = \infty$.

Below we state the running time of LazySVD.

Theorem 3.2.2 (Running time or computation complexity)**.** *Following the notation in theorem (3.2.1) and setting* $A = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$*,* LazySVD *can be implemented to run in time*

- $O\left(\frac{k\cdot nnz(A)+k^2d}{\delta} log\frac{1}{\delta\epsilon}\right)$ *if* $\mathcal{A}$ *is FGD;*
- $O\left(\frac{k\cdot nnz(A)+k^2d}{\delta^{1/2}} log\frac{1}{\delta\epsilon}\right)$ *if* $\mathcal{A}$ *is AGD;*
- $O\left(\frac{kd}{\delta} log\frac{1}{\delta\epsilon}\right)$ *if* $\mathcal{A}$ *is SAG ;*
- $O\left(\left(k\cdot nnz(A) + k^2d + \frac{kd}{\delta^2}\right) log\frac{1}{\delta\epsilon}\right)$ *if* $\mathcal{A}$ *is SVRG ;*
- $O\left(\left(k\cdot nnz(A) + k^2d + \frac{kn^{3/4}}{\delta^{1/2}})\right) log\frac{1}{\delta\epsilon}\right)$ *if* $\mathcal{A}$ *is* $KatyushaX^s$*;*
- $O\left(\left(knd + \frac{kn^{3/4}d}{\lambda_k^{1/4}\delta^{1/2}})\right) log\frac{1}{\delta\epsilon}\right)$ *if* $\mathcal{A}$ *is accelerated SVRG and* $\|x_i\|_2 \le 1$ *for* $\forall i \in [n]$.

Proof. We call k times AppxPCA, $M_{s-1} = (I - V_{s-1}V_{s-1}^T)M(I - V_{s-1}V_{s-1}^T)$ can be fed implicitly into AppxPCA each time thus the time needed to multiply M_{s-1} with a d-dimensional vector is $O(dk + nnz(M))$ or $O(dk + nnz(A))$. Here, the $O(dk)$ overhead is due to the projection of a vector into $V_{s-1}^{\perp}$. This proves the first five running times using Lemma (2.2.1), (2.2.2), (2.2.6), (2.2.3) and (2.3.1) respectively.

To obtain the last running time, when we compute M_s from M_{s-1}, we explicitly project $x_i^{'} \leftarrow (I - v_s v_s^T)x_i$ for each vector x_i, and feed the new $x_1^{'}, \cdots, x_n^{'}$ into AppxPCA. Now the running time follows from Lemma (2.2.5) together with the fact that $\|M_{s-1}\|_2 \ge \|M_{k-1}\|_2 \ge \lambda_k$. □

3.3 Main Results for k-SVD

The combination of Theorem (3.2.1) and Theorem (3.2.2) implies the following corollaries.

Corollary 3.3.1 (Gap-dependent k-SVD)**.** *Let $X \in \mathbb{R}^{n\times d}$ be a data matrix with singular values $1 \geq \sigma_1 \geq \cdots \geq \sigma_d \geq 0$ and the corresponding left singular vectors $u_1, \cdots, u_d \in \mathbb{R}^d$. Let $gap = \frac{\sigma_k - \sigma_{k+1}}{\sigma_k}$ be the relative gap. For fixed $\epsilon, p > 0$, consider the output*

$$V_k \leftarrow LazySVD\left(\mathcal{A}, X^T X, k, gap, O\left(\frac{\epsilon^4 gap^2}{k^4(\sigma_1/\sigma_k)^4}\right), p\right)$$

Then, defining $W = (u_{k+1}, \cdots, u_d)$, we have with probability at least $1 - p$:

$$V_k \ is \ a \ rank-k \ (column) orthonormal matrix with \|V_k^T W\|_2 \leq \epsilon$$

The running time is $\widetilde{O}\left(\frac{k\cdot nnz(A)+k^2 d}{gap^{1/2}}\right)$ for AGD and $\widetilde{O}\left(knd + \frac{kn^{3/4}d}{\sigma_k^{1/4} gap^{1/2}}\right)$ for accelerated SVRG. More running times of algorithms see Theorem (3.2.2).

Corollary 3.3.2 (Gap-free k-SVD)**.** *Let $X \in \mathbb{R}^{n\times d}$ be a data matrix with singular values $1 \geq \sigma_1 \geq \cdots \geq \sigma_d \geq 0$ and the corresponding left singular vectors $u_1, \cdots, u_d \in \mathbb{R}^d$. Let $gap = \frac{\sigma_k - \sigma_{k+1}}{\sigma_k}$ be the relative gap. For fixed $\epsilon, p > 0$, consider the output*

$$(v_1, \cdots, v_k) = V_k \leftarrow LazySVD\left(\mathcal{A}, X^T X, k, \frac{\epsilon}{3}, O\left(\frac{\epsilon^6}{k^4 d^4 (\sigma_1/\sigma_k)^{12}}\right), p\right)$$

Then, defining $X_k = V_k V_k^T X$, we have with probability at least $1 - p$:

- *Spectral norm guarantee: $\|X - X_k\|_2 \leq (1+\epsilon)\|X - X_k^*\|_2$;*
- *Frobenius norm guarantee: $\|X - X_k\|_F \leq (1+\epsilon)\|X - X_k^*\|_F$;*
- *Rayleigh quotient guarantee: $\forall i \in [k], |v_i^T X^T X v_i - \sigma_i^2| \leq \epsilon\sigma_i^2$.*

The running time is $\widetilde{O}\left(\frac{k\cdot nnz(A)+k^2 d}{\epsilon^{1/2}}\right)$ for AGD and $\widetilde{O}\left(knd + \frac{kn^{3/4}d}{\sigma_k^{1/4}\epsilon^{1/2}}\right)$ for accelerated SVRG. More running times of algorithms see Theorem (3.2.2).

Conclusion

In this paper, we introduce a framework of $LazySVD$ and analyze the total complexity in different cases where stochastic or non-stochastic optimization are applied in its optimization oracle. When the optimization oracle uses AGD, accSVRG and $Katyusha\ X^s$ for 1-SVD, the complexity matches the optimal dependence on the gap or ϵ, i.e. $gap^{-1/2}$ or $\epsilon^{-1/2}$, faster than block Krylov [4]. What's more, when a variance-reduction stochastic method is used for 1-SVD, the stochastic optimization oracle doesn't need an accurate initial warm-start, outperforming its counterpart in [5].

Besides the running time advantages mentioned above, the analysis involved is completely based on convex optimization, since 1-SVD is solvable using convex techniques. LazySVD also works when k is not known to the algorithm, as opposed to block methods which need to know k in advance.

参考文献

[1] K. Person. On Lines and Planes of Closest Fit to Systems of Points in Space[J]. Philosophical Magazine. **2 (11): 559–572**

[2] Hotelling. Analysis of a complex of statistical variables into principal components[J]. Journal of Educational Psychology. **24, 417 – 441, and 498 – 520**

[3] Zeyuan Allen Zhu, Yuanzhi Li. Even Faster SVD Decomposition Yet Without Agonizing Pain[J]. CoRR. 2016, **abs/1607.03463**

[4] Christopher Musco Cameron Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition[C]. Advances in Neural Information Processing Systems. 2015, 1396–1404

[5] Ohad Shamir. Fast stochastic algorithms for SVD and PCA: Convergence properties and convexity[C]. International Conference on Machine Learning. 2016, 248–256

[6] Yousef Saad. Numerical methods for large eigenvalue problems[M], second . Addison-Wesley, 2011

[7] Dan Garber, Elad Hazan. Fast and Simple PCA via Convex Optimization[J]. 2015, **abs/1509.05647**

[8] Yurii Nesterov. Introductory lectures on convex optimization: A basic course[M], vol. 87. Springer Science & Business Media, 2013

[9] Yurii Nesterov. Introductory lectures on convex programming volume i: Basic course[J]. Lecture notes. 1998

[10] Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, Aaron Sidford. Parallelizing stochastic approximation through mini-batching and tail-averaging[J]. arXiv preprint arXiv:161003774. 2016

[11] John Duchi, Elad Hazan, Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization[J]. Journal of Machine Learning Research. 2011, **12**(Jul):2121–2159

[12] T Tieleman, G Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012[J]. Google Scholar

[13] Diederik P Kingma, Jimmy Ba. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:14126980. 2014

[14] Alexandre d'Aspremont. Smooth optimization with approximate gradient[J]. SIAM Journal on Optimization. 2008, **19**(3):1171–1183

[15] Olivier Devolder, François Glineur, Yurii Nesterov. First-order methods of smooth convex optimization with inexact oracle[J]. Mathematical Programming. 2014, **146**(1-2):37–75

[16] Saeed Ghadimi, Guanghui Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization I: A generic algorithmic framework[J]. SIAM Journal on Optimization. 2012, **22**(4):1469–1492

[17] Aymeric Dieuleveut, Francis Bach, et al. Nonparametric stochastic approximation with large step-sizes[J]. The Annals of Statistics. 2016, **44**(4):1363–1399

[18] Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, Aaron Sidford. Accelerating Stochastic Gradient Descent[J]. arXiv preprint arXiv:170408227. 2017

[19] Rie Johnson, Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction[C]. Advances in neural information processing systems. 2013, 315–323

[20] Dan Garber, Elad Hazan, Chi Jin, Cameron Musco, Praneeth Netrapalli, Aaron Sidford, et al. Faster eigenvector computation via shift-and-invert preconditioning[C]. International Conference on Machine Learning. 2016, 2626–2634

[21] Hongzhou Lin, Julien Mairal, Zaid Harchaoui. A universal catalyst for first-order optimization[C]. Advances in Neural Information Processing Systems. 2015, 3384–3392

[22] Zeyuan Allen-Zhu, Yang Yuan. Improved SVRG for non-strongly-convex or sum-of-non-convex objectives[C]. International conference on machine learning. 2016, 1080–1089

[23] Mark Schmidt, Nicolas Le Roux, Francis Bach. Minimizing finite sums with the stochastic average gradient[J]. Mathematical Programming. 2017, **162**(1-2):83–112

[24] Nicolas L Roux, Mark Schmidt, Francis R Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets[C]. Advances in Neural Information Processing Systems. 2012, 2663–2671

[25] Zeyuan Allen-Zhu. Katyusha X: Practical Momentum Method for Stochastic Sum-of-Nonconvex Optimization[J]. arXiv preprint arXiv:180203866. 2018

[26] Lin Xiao, Tong Zhang. A proximal stochastic gradient method with progressive variance reduction[J]. SIAM Journal on Optimization. 2014, **24**(4):2057–2075

[27] Zeyuan Allen-Zhu, Lorenzo Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent[J]. arXiv preprint arXiv:14071537. 2014

Acknowledgement

I could not have finished my undergraduate thesis without a lot of persons' help. First the deepest gratitude goes first and foremost to my director Prof. Zhihua Zhuang for his constant encouragement and guidance. He provides me a platform to meet other genius guys and helps me find out where my interest lies. Second, my thanks would go to my beloved family for their loving considerations and great confidence in me all through these years. Third, thanks to my senior fellow, Haishan Ye, who grants me this problem discussed in the paper and offers some inspiring insights. Last but not least, I also owe my sincere gratitudes to my colleagues in the Deep Learning laboratory in BIBDR and friends in SMS. They gave me their help and time in listening to me and helping me work out my problems during the difficult course of the thesis. They includes Deqing Jiang, Dachao Lin, Wenhao Yang, Guangzeng Xie, Long Chen, Chengzhuo Ni, Yitan Wang and Naixin Guo.

At the end of the undergraduate study, I appreciate the unforgettable four-year college experience in SMS but I also look forward to my future further studying and research. Chuangtse has well said, "Alas, my life is limited, while knowledge is limitless" (吾生也有涯，而知也无涯). I will step into my academic journey with heart-full humility, quiet determination and steely resolve in the pursuit of truth to diligently practice that philosophy.

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校 □ 一年 /□ 两年 /□ 三年以后在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：　　日期：　　年　　月　　日