FAST LINEAR ALGORITHMS FOR MACHINE LEARNING

Yichao Lu

A DISSERTATION

in

Applied Mathematics and Computational Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2015

Supervisor of Dissertation

Dean P. Foster, Professor of Statistics

Graduate Group Chairperson

Charles L. Epstein, Thomas A. Scott Professor of Mathematics

Dissertation Committee:
Dean P. Foster, Professor of Statistics
Lyle H. Ungar, Professor of Computer and Information Science
Zongming Ma, Assistant Professor of Statistics

Acknowledgments

First and Foremost, I would like to express my sincere gratitude to my Advisor Dean Foster for his support during my PhD study. His broad knowledge, sharp intuition, patient guidance and passion about problems in statistics and machine learning providing me with an excellent atmosphere for doing research.

I am deeply grateful to Professor Lyle Ungar, Zonging Ma and Robert Stine for many stimulating ideas and valuable discussions at the StatNLP reading group and when I'm preparing for my thesis. I would like to thank Professor Charles Epstein for providing me the wonderful opportunity to study in the AMCS program and his kind support at the beginning of my PhD career.

I would like to thank my fellow PhD students Paramveer Dhillon, Jordan Rodu, Peichao Peng, Zhuang Ma, Joao Sedoc, Wei Han, Fan Yang, Anru Zhang, Fan Zhao, Pengfei Zheng, Shi Gu, Muzhi Yang, Yuanpei Cao and Ke Zeng. It's a great honor for me to spend my Phd years studying and having fun with a group of smart and interesting friends.

Last but not the least, I want to thank my family for their support, encouragement, and love.

ABSTRACT

FAST LINEAR ALGORITHMS FOR MACHINE LEARNING

Yichao Lu

Dean P. Foster, Advisor

Nowadays linear methods like Regression, Principal Component Analysis and Canonical Correlation Analysis are well understood and widely used by the machine learning community for predictive modeling and feature generation. Generally speaking, all these methods aim at capturing interesting subspaces in the original high dimensional feature space. Due to the simple linear structures, these methods all have a closed form solution which makes computation and theoretical analysis very easy for small datasets. However, in modern machine learning problems it's very common for a dataset to have millions or billions of features and samples. In these cases, pursuing the closed form solution for these linear methods can be extremely slow since it requires multiplying two huge matrices and computing inverse, inverse square root, QR decomposition or Singular Value Decomposition (SVD) of huge matrices. In this thesis, we consider three fast algorithms for computing Regression and Canonical Correlation Analysis approximate for huge datasets.

For linear regression, we consider a combination of two well known algorithms, Principal Component Regression (PCR) and Gradient Descent(GD). Since the feature matrices in our problems are huge, we use the fast randomized SVD algorithm proposed by Halko et al. for computing the top principal components. We show that a this combination will provide an approximate regression solution which is both fast and robust. Theoretical analysis and empirical results about the convergence speed and statistical accuracy of our algorithm are provided in Chapter 2.

For Canonical Correlation Analysis (CCA), we consider two different approaches. In the first approach, we reduce the CCA problem to a sequence of iterative regression problems. Plug in the fast regression algorithm into this framework generates our first fast CCA algorithm. A detailed analysis about the convergence speed and empirical performance of this algorithm is provide in Chapter 3. In the second approach, we regard CCA as a constrained optimization problem and solve it by a gradient style algorithm. The benefit of the second approach over the first approach is in the second approach, the gradient style updates allows the CCA subspace estimator to improve after every iteration while in the first approach the CCA subspace estimator can only be improved when a reasonably accurate regression is performed. Based on this observation a stochastic version of the second CCA approach is proposed which is very fast if aim at moderate accuracy. In Chapter 4 we discuss the second CCA approach in detail.

Contents

1 Introduction **1**

2 Fast Ridge Regression Algorithm **6**

2.1 Introduction 6

2.2 The Algorithm 8

2.2.1 Description of the Algorithm 8

2.2.2 Computational Cost 13

2.3 Theorems 15

2.3.1 The Fixed Design Model 16

2.4 Experiments 19

2.4.1 Simulated Data 20

2.4.2 Real Data 25

2.5 summary 29

3 Large Scale Canonical Correlation Analysis with Iterative Least Squares **30**

3.1 Introduction 30

3.2 Background: Canonical Correlation Analysis 33

3.2.1 Definition 33

3.2.2 CCA and SVD 33

3.3 Compute CCA by Iterative Least Squares 34

3.3.1 A Special Case 36

3.3.2 General Case 36

3.4 Algorithm 37

3.4.1 LING: a Gradient Based Least Square Algorithm 37

3.4.2 Fast Algorithm for CCA 40

3.4.3 Error Analysis of L-CCA 41

3.5 Experiments 42

3.5.1 Penn Tree Bank Word Co-ocurrence 46

3.5.2 URL Features 48

3.6 Conclusion and Future Work 50

4 Augmented Approximate Gradient Algorithm for Large Scale Canonical Correlation Analysis 51

4.1 Introduction 51

4.1.1 Background 51

4.1.2 Related Work 54

4.1.3 Main Contribution 55

4.2 Algorithm 56

4.2.1 An Optimization Perspective 56

4.2.2 *AppGrad* Scheme 60

4.2.3 General Rank-k Case 64

4.2.4 Kernelization 66

4.3 Stochastic *AppGrad* 67

4.4 Experiments 69

4.4.1 Details of Datasets 70

4.4.2 Implementations 71

4.4.3 Summary of Results 73

4.5 Conclusions and Future Work 76

Appendices 78

A Appendix for Chapter 3 79

A.1 Proof of Theorem 3.3.1 79

A.2 Randomized Algorithm for Finding Top Singular Vectors 80

A.3 Gradient Descent with Optimal Step Size 81

A.4 Error Analysis of LING 83

A.4.1 Proof of Theorem 3.4.2 86

A.5 Error Analysis of L-CCA 87

Chapter 1

Introduction

Linear methods like Regression, Principal Component Analysis and Canonical Correlation Analysis are well understood and widely used by the machine learning community for predictive modeling and feature generation. Generally speaking, all these methods aim at capturing interesting subspaces in the original high dimensional feature space. Due to the simple linear structures, these methods all have a closed form solution. For small datasets, one can implement these methods in R, Matlab, Python or other platforms as long as the computational tools can perform some basic linear algebra operations like matrix multiplication, matrix inversion, singular value decomposition, QR decomposition. However, in modern machine learning problems it's very common for a dataset to have millions or billions of features and samples. For example, in Natural Language Processing (NLP), using only unigram model will generate a feature space the dimension of which is the vocabulary size which can easily reach tens of thousands for corpora of moderate size and can get much larger for huge corpora. Moreover, it's very common to use bigram (or trigram) model for NLP tasks which make the feature dimension vocabulary size squared (or cubed). Another example is in collaborative filtering where the algorithms often need to handle an input matrix with size number of products times number of customers which can easily reach millions. In these cases, pursuing the closed form solution for these linear methods can be extremely slow since it requires multiplying two huge matrices and computing inverse, inverse square root, QR decomposition or Singular Value Decomposition (SVD) of huge matrices. In this thesis, we consider three fast algorithms for computing these linear methods approximately on huge datasets.

Fast Principal Component Analysis algorithms has been well studied in the past few years. One well known approach is the randomized SVD algorithm proposed by [28, 30] which is extremely fast if one is only interested in computing the top few principal components (singular vectors). In the randomized SVD algorithm, first a random projection is performed to generate an estimator of the subspace spanned by the top few left singular vectors. Then this subspace is used to project the original huge data matrix down to a much smaller size and it suffices to compute the SVD of the small matrix. Detailed theoretical analysis of this algorithm is available in [28] and a brief introduction of the algorithm will be provided in chapter 2 and appendix since we are going to use it as a building block of our fast Regression and CCA algorithms.

Fast Regression (or Least Squares) has also been well studied and a lot of algorithms have been proposed based on the idea of random projection or subsampling. When the number of observations is much larger than the number of features, [21, 6, 47] use different kinds of fast random projections that approximately preserve inner products in the Euclidian space to reduce the actual sample size of the problem and then solve the least squares problem on reduced dataset. Random projection with such properties are sometimes called fast Johnson-Lindenstrauss transforms. Different random projections with this property are introduced in [2, 48, 54] with concentration bounds on how well the inner product is preserved. These techniques are applied in a fast ridge regression algorithm by [39] when the number of features are much larger than the number of samples. Another slightly different idea is to subsample the observations from some non-uniform distribution determined by the statistical leverage as discussed in [20, 44]. The statistical accuracy of the above algorithms are discussed by [42, 16] in which they also proposed some improvements based on the statistical analysis.

However, the main draw back for the fast regression algorithms is that they are still very slow for problems with a huge amount of features. In fact, the acceleration of these algorithms comes from a fast approximation of the matrix multiplication $\mathbf{X}^\top\mathbf{X}$ where $\mathbf{X} \in n \times p$ is a data matrix with n samples and p features. On the other hand, these algorithms still need to invert a $p \times p$ square matrix which is very slow when p is large. To obtain a fast regression solution in this case, one can either compute the top few principal components by the randomized SVD algorithm and then regress only on the top principal components (i.e. run a principal component regression or PCR) or regard regression as a quadratic minimization problem and approximate the solution by gradient descent (GD). Both algorithms are trading accuracy for speed. For PCR, if only a few principal component are selected, then the algorithm will be extremely fast but will probably miss interesting signals on the bottom principal components. If a relatively large amount of principal components are selected (but not overfit), more signals will be captured but the algorithm will slow down. For GD, every gradient iteration is super fast and as the number of iterations increases, the solution gets more accurate but the algorithm takes more time. In our fast algorithm [58], we combine PCR and GD together to get a new fast regression algorithm which archives a better tradeoff between accuracy and speed. As shown in the experiments, to achieve a certain accuracy, the number of principal components and gradient iterations in our algorithm is significantly less than running PCR or GD alone.

Fast Canonical Correlation Analysis (CCA) algorithm is a relatively new topic. Following the same idea in regression, [5] applied fast Johnson-Lindenstrauss transform to reduce the sample size of the data matrices and then compute CCA on the reduced data. Same as regression, this algorithm only works for dataset with a large sample size but not with a large amount of features. In this thesis we propose two fast CCA algorithms (the second is an improvement of the first) which provide fast approximations of the

CCA subspace given to huge data matrices. Our algorithms can handle the case where the number of features is extremely large and works well with sparse data matrices. In the first approach [40], we reduce the CCA algorithm to several regression problems and apply fast regression algorithms to obtain a fast CCA algorithm. In the second approach, we view CCA as a constrained optimization problem and propose a gradient style iterative algorithm which will converge to the true CCA subspace. It's non trivial since the optimization problem for CCA is not convex. Moreover, our second algorithm can also be interpreted as a improvement of our first algorithm since it's essentially replacing a fast regression in the first algorithm with a simple gradient step. It's easy to generalize our second algorithm to an online (stochastic) setting due to its gradient nature. In fact, as shown by the experiments, a stochastic version of the second algorithm can be even faster than the batch version if we aim at moderate accuracy.

The thesis is organized as follows: in Chapter 2, we introduce the fast regression algorithm which is a combination of two classical algorithms PCR and GD. In Chapter 3, we discuss the regression formulation of CCA and apply the fast regression algorithm from Chapter 2 to get our first fast CCA algorithm. In Chapter 4, we introduce both the batch and stochastic version of our second fast CCA algorithm. The thesis is organized in a way that chapter 2,3,4 can be read separately as three independent papers. Each chapter will include some simple theorem proofs, but the long and complicated proofs will be deferred to the appendix.

Chapter 2

Fast Ridge Regression Algorithm

2.1 Introduction

Ridge Regression (RR) is one of the most widely applied penalized regression algorithms in machine learning problems. Suppose $\mathbf{X}$ is the $n \times p$ design matrix and $\mathbf{Y}$ is the $n \times 1$ response vector, ridge regression tries to solve the problem

$$\hat{\beta} = \arg\min_{\beta \in \mathcal{R}^p} \|\mathbf{X}\beta - \mathbf{Y}\|^2 + n\lambda\|\beta\|^2 \tag{2.1.1}$$

which has an explicit solution

$$\hat{\beta} = (\mathbf{X}^\top\mathbf{X} + n\lambda)^{-1}\mathbf{X}^\top\mathbf{Y} \tag{2.1.2}$$

However, for modern problems with huge design matrix $\mathbf{X}$, computing (2.1.2) costs $O(np^2)$ FLOPS. When $p > n \gg 1$ one can consider the dual formulation of (2.1.1) which also has an explicit solution as mentioned in [39, 49] and the cost is $O(n^2p)$ FLOPS. In summary, trying to solve (2.1.1) exactly costs $O(np\min\{n,p\})$ FLOPS which can be very slow.

There are faster ways to approximate (2.1.2) when computational cost is the concern. One can view RR as an optimization problem and use Gradient Descent (GD) which takes $O(np)$ FLOPS in every iteration. However, the convergence speed for GD depends on the spectrum of $\mathbf{X}$ and the magnitude of λ. When $\mathbf{X}$ is ill conditioned and λ is small, GD requires a huge number of iterations to converge which makes it very slow. For huge datasets, one can also apply stochastic gradient descent (SGD) [59, 34, 11], a powerful tool for solving large scale optimization problems.

Another alternative for regression on huge datasets is Principal Component Regression (PCR) as mentioned in [4, 35], which runs regression only on the top k_1 principal components (PCs) of the $\mathbf{X}$ matrix. PCA for huge $\mathbf{X}$ can be computed efficiently by randomized algorithms like [29, 30]. The cost for computing top k_1 PCs of $\mathbf{X}$ is $O(npk_1)$ FLOPS. The connection between RR and PCR is well studied by [18]. The problem of PCR is that when a large proportion of signal sits on the bottom PCs, it has to regress on a lot of PCs which makes it both slow and inaccurate (see later sections for detailed explanations).

In this paper, we propose a two stage algorithm LING[1] which is a faster way of computing the RR solution (2.1.2). A detailed description of the algorithm is given in section 2.2. In section 2.3, we prove that LING has the same risk as RR under the fixed design setting. In section 2.4, we compare the performance of PCR, GD, SGD and LING in

[1] LING is the Chinese of ridge

terms of prediction accuracy and computational efficiency on both simulated and real data sets.

2.2 The Algorithm

2.2.1 Description of the Algorithm

LING is a two stage algorithm. The intuition of LING is quite straight forward. We start with the observation that regressing $\mathbf{Y}$ on $\mathbf{X}$ (OLS) is equivalent to projecting $\mathbf{Y}$ onto the span of $\mathbf{X}$. Let $\mathbf{U}_1$ denote the top k_2 PCs (left singular vectors) of $\mathbf{X}$ and let $\mathbf{U}_2$ denote the remaining PCs. The projection of $\mathbf{Y}$ onto the span of $\mathbf{X}$ can be decomposed into two orthogonal parts, the projection onto $\mathbf{U}_1$ and the projection onto $\mathbf{U}_2$. In the first stage, we pick a $k_2 \ll p$ and the projection onto $\mathbf{U}_1$ can be computed directly by $\hat{\mathbf{Y}}_1 = \mathbf{U}_1\mathbf{U}_1^\top\mathbf{Y}$ which is exactly the same as running a PCR on top k_2 PCs. For huge $\mathbf{X}$, computing the top k_2 PCs exactly is very slow, so we use a faster randomized SVD algorithm for computing $\mathbf{U}_1$ which is proposed by [28] and described below. In the second stage, we first compute $\mathbf{Y}_r = \mathbf{Y} - \hat{\mathbf{Y}}_1$ and $\mathbf{X}_r = \mathbf{X} - \mathbf{U}_1\mathbf{U}_1^\top\mathbf{X}$ which are the residuals of $\mathbf{Y}$ and $\mathbf{X}$ after projecting onto $\mathbf{U}_1$. Then we compute the projection of $\mathbf{Y}$ onto the span of $\mathbf{U}_2$ by solving the optimization problem $\min_{\hat{\gamma}_2 \in \mathcal{R}^p} \|\mathbf{X}_r\hat{\gamma}_2 - \mathbf{Y}_r\|^2 + n\lambda\|\hat{\gamma}_2\|$ with GD (Algorithm 3). Finally, since RR shrinks the projection of $\mathbf{Y}$ onto $\mathbf{X}$ (the OLS solution) via regularization, we also shrink the projections in both stages accordingly. Shrinkage in the first stage is performed directly on the estimated regression coefficients and shrinkage

```
Algorithm 1 LING
  Input : Data matrix $\mathbf{X}$ ,$\mathbf{Y}$. $\mathbf{U}_1$, an orthonormal matrix consists of top $k_2$ PCs of $\mathbf{X}$. $d_1, d_2, ...d_{k_2}$, top $k_2$ singular values of $\mathbf{X}$. Regularization parameter $\lambda$, an initial vector $\hat{\gamma}_{2,0}$ and number of iterations $n_2$ for GD .
  Output : $\hat{\gamma}_{1,s}$, $\hat{\gamma}_2$, the regression coefficients.
  1.Regress $\mathbf{Y}$ on $\mathbf{U}_1$, let $\hat{\gamma}_1 = \mathbf{U}_1^\top \mathbf{Y}$.
  2.Compute the residual of previous regression problem. Let $\mathbf{Y}_r = \mathbf{Y} - \mathbf{U}_1\hat{\gamma}_1$.
  3.Compute the residual of $\mathbf{X}$ regressing on $\mathbf{U}_1$. Use $\mathbf{X}_r = \mathbf{X} - \mathbf{U}_1\mathbf{U}_1^\top\mathbf{X}$ to denote the residual of $\mathbf{X}$.
  4.Use gradient descent with optimal step size with initial value $\hat{\gamma}_{2,0}$ (see algorithm 3) to solve the RR problem $\min_{\hat{\gamma}_2 \in \mathcal{R}^p} \|\mathbf{X}_r\hat{\gamma}_2 - \mathbf{Y}_r\|^2 + n\lambda\|\hat{\gamma}_2\|^2$.
  5. Compute a shrinkage version of $\hat{\gamma}_1$ by $(\hat{\gamma}_{1,s})_i = \frac{d_i^2}{d_i^2 + n\lambda}(\hat{\gamma}_1)_i$
  6.The final estimator is $\hat{\mathbf{Y}} = \mathbf{U}_1\hat{\gamma}_{1,s} + \mathbf{X}_r\hat{\gamma}_2$.
```

in the second stage is performed by adding a regularization term to the optimization problem mentioned above. Detailed description of LING is shown in Algorithm 1.

Remark 2.2.1. LING can be regarded as a combination of PCR and GD. The first stage of LING is a crude estimation of the projection of $\mathbf{Y}$ onto $\mathbf{X}$ and the second stage adds a correction to the first stage estimator. Since we do not need a very accurate estimator in the first stage it suffices to pick a very small k_2 in contrast with the k_1 PCs needed for PCR. In the second stage, the design matrix $\mathbf{X}_r$ is a much better conditioned matrix than the original $\mathbf{X}$ since the directions with largest singular values are removed. As

Algorithm 2 Random SVD

```
Input : design matrix $\mathbf{X}$, target dimension $k_2$, number of power iterations $i$.
Output : $\mathbf{U}_1 \in n \times k_2$, the matrix of top $k_2$ left singular vectors of $\mathbf{X}$, $d_1, d_2, ...d_{k_2}$, the top $k_2$ singular values of $\mathbf{X}$.
1.Generate random matrix $R_1 \in p \times k_2$ with i.i.d standard Gaussian entries.
2.Estimate the span of top $k_2$ left singular vectors of $\mathbf{X}$ by $A_1 = (\mathbf{X}\mathbf{X}^\top)^i \mathbf{X} R_1$.
3.Use QR decomposition to compute $Q_1$ which is an orthonormal basis of the column space of $A_1$.
4.Compute SVD of the reduced matrix $Q_1^\top \mathbf{X} = \mathbf{U}_0 \mathbf{D}_0 \mathbf{V}_0^\top$.
5.$\mathbf{U}_1 = Q_1 \mathbf{U}_0$ gives the top $k_2$ singular vectors of $\mathbf{X}$ and the diagonal elements of $\mathbf{D}_0$ gives the singular values.
```

introduced in section 2.2, Algorithm 3 converges much faster with a better conditioned matrix. Hence GD in the second stage of LING converges faster than directly applying GD for solving (2.1.1). The above property guarantees that LING is both fast and accurate compared with PCR and GD. More details about on the computational cost will be discussed in section 2.2.2.

Remark 2.2.2. Algorithm 2 is essentially approximating the subspace of top left singular vectors by random projection. It provides a fast approximation of the top singular values and vectors for large $\mathbf{X}$ when computing the exact SVD is very slow. Theoretical guarantees and more detailed explanations can be found in [28]. Empirically we find in the experiments, Algorithm 2 may occasionally generate a bad subspace estimator due to

Algorithm 3 Gradient Descent with Optimal Step Size (GD)

```
**Goal :** Solve the ridge problem $\min_{\hat\gamma\in\mathcal{R}^p} \|\mathbf{X}\hat\gamma - \mathbf{Y}\|^2 + n\lambda\|\hat\gamma\|^2$.
**Input :** Data matrix $\mathbf{X}$, $\mathbf{Y}$, regularization parameter $\lambda$, number of iterations $n_2$, an initial vector $\hat\gamma_0$
**Output :** $\hat\gamma$
**for** $t = 0$ **to** $n_2 - 1$ **do**
    $Q = 2\mathbf{X}^\top\mathbf{X} + 2n\lambda I$
    $w_t = 2\mathbf{X}^\top\mathbf{Y} - Q\hat\gamma_t$
    $s_t = \frac{w_t^\top w_t}{w_t^\top Q w_t}$. $s_t$ is the step size which makes the target function decrease the most in direction $w_t$.
    $\hat\gamma_{t+1} = \hat\gamma_t + s_t \cdot w_t$.
**end for**
```

randomness which makes PCR perform badly. On the other hand, LING is much more robust since in the second stage it compensates for the signal that was missed in the first stage. In all the experiments, we set $i = 1$.

The shrinkage step (step 5) in Algorithm 1 is only necessary for theoretical purposes since the goal is to approximate Ridge Regression which shrinks the Least Squares estimator over all directions. In practice shrinkage over the top k_2 PCs is not necessary. Usually the number of PCs selected (k_2) is very small. From the bias variance trade off perspective, the variance reduction gained from the shrinkage over top k_2 PCs is at most $O(\frac{k_2}{n})$ under the fixed design setting [18] which is a tiny number. Moreover, since the top singular values of $\mathbf{X}^\top\mathbf{X}$ are usually very large compared with $n\lambda$ in most real problems, the shrinkage factor $\frac{d_i^2}{d_i^2+n\lambda}$ will be pretty close to 1 for top singular values. We use shrinkage in Algorithm 1 because the risk of the shrinkage version of LING is exactly the same as RR as proved in section 2.3.

Algorithm 2 can be further simplified if we skip the shrinkage step mentioned in previous paragraph. Instead of computing the top k_2 PCs, the only thing we need to know is the subspace spanned by these PCs since the first stage is essentially projecting $\mathbf{Y}$ onto this subspace. In other words, we can replace $\mathbf{U}_1$ in step 1, 2, 3 of Algorithm 1 with Q_1 obtained in step 3 of Algorithm 2 and directly let $\hat{\mathbf{Y}} = Q_1\hat{\gamma}_1 + \mathbf{X}_r\hat{\gamma}_2$. In the experiments of section 4 we use this simplified version.

2.2.2 Computational Cost

We claim that the cost of LING is $O\big(np(k_2 + n_2)\big)$ where k_2 is the number of PCs used in the first stage and n_2 is the number of iterations of GD in the second stage. According to [28], the dominating step in Algorithm 2 is computing $(\mathbf{X}\mathbf{X}^\top)^i\mathbf{X}R_1$ and $Q_1^\top\mathbf{X}$ which costs $O(npk_2)$ FLOPS. Computing $\hat{\gamma}_1$ and $\mathbf{Y}_r$ costs less than $O(npk_2)$. Computing $\mathbf{X}_r$ costs $O(npk_2)$. So the computational cost before the GD step is $O(npk_2)$. For the GD stage, note that in every iteration Q never needs to be constructed explicitly. While computing w_t and s_t, always multiplying matrix and vector first gives a cost of $O(np)$ for every iteration. So the cost for GD stage is $O(n_2np)$. Add all pieces together the cost of LING is $O\big(np(k_2 + n_2)\big)$ FLOPS.

Let n_1 be the number of iterations required for solving (2.1.1) directly by GD and k_1 be the number of PCs used for PCR. It's easy to check that the cost for GD is $O(n_1np)$ FLOPS and the cost for PCR is $O(npk_1)$. As mentioned in remark 2.2.1, the advantage of LING over GD and PCR is that k_1 and n_1 might have to be really large to achieve high accuracy but much smaller values of the pair (k_2, n_2) will work for LING.

In the remaining part of the chapter we use "signal on certain PCs" to refer to the projection of $\mathbf{Y}$ onto certain principal components of $\mathbf{X}$. Consider the case when the signal is widely spread among all PCs (i.e. the projection of $\mathbf{Y}$ onto the bottom PCs of $\mathbf{X}$ is not very small) instead of concentrating on the top ones, k_1 needs to be large to make PCR perform well since the signal on bottom PCs are discarded by PCR. But LING does not need to include all the signal in the first stage regression since the signal left over will be estimated in the second GD stage. Therefore LING is able to recover most of the signal even with a small k_2.

In order to understand the connection between accuracy and number of iterations in Algorithm 3 , we state the following theorem in [1]:

Theorem 2.2.3. *Let $g(z) = \frac{1}{2}z^\top Mz + q^\top z$ be a quadratic function where M is a PSD matrix. Suppose $g(z)$ achieves minimum at z^*. Apply Algorithm 3 to solve the minimization problem. Let z_t be the z value after t iterations, then the gap between $g(z_t)$ and $g(z^*)$, the minimum of the objective function satisfies*

$$\frac{g(z_{t+1}) - g(z^*)}{g(z_t) - g(z^*)} \leq C = \left(\frac{A-a}{A+a}\right)^2 \tag{2.2.1}$$

where A, a are the largest and smallest eigenvalue of the M matrix.

Theorem 2.2.3 shows that the sub optimality of the target function decays exponentially as the number of iterations increases and the speed of decay depends on the largest and smallest singular value of the PSD matrix that defines the quadratic objective function. If we directly apply GD to solve (2.1.1), Let $f_1(\beta) = \|\mathbf{X}\beta - \mathbf{Y}\|^2 + n\lambda\|\beta\|^2$. Assume f_1 reaches its minimum at $\hat{\beta}$. Let $\hat{\beta}_t$ be the coefficient after t iterations and let d_i denote the i^{th} singular value of $\mathbf{X}$. Applying theorem 2.2.3, we have

$$\frac{f_1(\hat{\beta}_{t+1}) - f_1(\hat{\beta})}{f_1(\hat{\beta}_t) - f_1(\hat{\beta})} \leq C = \left(\frac{d_1^2 - d_p^2}{d_1^2 + d_p^2 + 2n\lambda}\right)^2 \tag{2.2.2}$$

Similarly for the second stage of LING, Let $f_2(\gamma_2) = \|\mathbf{X}_r\gamma_2 - \mathbf{Y}_r\|^2 + n\lambda\|\gamma_2\|^2$. Assume f_2 reaches its minimal at $\hat{\gamma}_2$. We have

$$\frac{f_2(\hat{\gamma}_{2,t+1}) - f_2(\hat{\gamma}_2)}{f_2(\hat{\gamma}_{2,t}) - f_2(\hat{\gamma}_2)} \leq C = \left(\frac{d_{k_2+1}^2}{d_{k_2+1}^2 + 2n\lambda}\right)^2 \tag{2.2.3}$$

In most real problems, the top few singular values of $\mathbf{X}^\top \mathbf{X}$ are much larger than the other singular values and $n\lambda$. Therefore the constant C obtained in (2.2.2) can be very close to 1 which implies that the GD algorithm converges very slowly. On the other hand, removing the top few PCs will make C in (2.2.3) significantly smaller than 1. In other words, GD may take a lot of iterations to converge when solving (2.1.1) directly while the second stage of LING takes much less iterations to converge. This can also be seen in the experiments of section 2.4.

2.3 Theorems

In this section we compute the risk of LING estimator (explained below) under the fixed design setting. For simplicity, assume $\mathbf{U}_1, \mathbf{D}_0$ generated by Algorithm 2 give exactly the top k_2 left singular vectors and singular values of $\mathbf{X}$ and GD in step 4 of Algorithm 1 converges to the optimal solution. Let $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ be the SVD of $\mathbf{X}$ where $\mathbf{U} = (\mathbf{U}_1, \mathbf{U}_2)$ and $\mathbf{V} = (\mathbf{V}_1, \mathbf{V}_2)$. Here $\mathbf{U}_1, \mathbf{V}_1$ are top k_2 singular vectors and $\mathbf{U}_2, \mathbf{V}_2$ are bottom $p - k_2$ singular vectors. Let $\mathbf{D} = \text{diag}(\mathbf{D}_1, \mathbf{D}_2)$ where $\mathbf{D}_1 \in k_2 \times k_2$ contains top k_2 singular values denoted by $d_1 \geq d_2 \geq ... \geq d_{k_2}$ and $\mathbf{D}_2 \in p - k_2 \times p - k_2$ contains bottom $p - k_2$ singular values. Let $\mathbf{D}_3 = \text{diag}(\mathbf{0}, \mathbf{D}_2)$ (replace $\mathbf{D}_1$ in $\mathbf{D}$ by a zero matrix of the same size).

2.3.1 The Fixed Design Model

Assume $\mathbf{X}$, $\mathbf{Y}$ comes from the fixed design model $\mathbf{Y} = \mathbf{X}\beta + \epsilon$ where $\epsilon \in n \times 1$ are i.i.d noise with mean 0 and variance σ^2. Here $\mathbf{X}$ is fixed and the randomness of $\mathbf{Y}$ only comes from ϵ. Note that $\mathbf{X} = \mathbf{U}_1\mathbf{D}_1\mathbf{V}_1^\top + \mathbf{X}_r$, the fixed design model can also be written as

$$\mathbf{Y} = (\mathbf{U}_1\mathbf{D}_1\mathbf{V}_1^\top + \mathbf{X}_r)\beta + \epsilon = \mathbf{U}_1\gamma_1 + \mathbf{X}_r\gamma_2 + \epsilon$$

where $\gamma_1 = \mathbf{D}_1\mathbf{V}_1^\top\beta$ and $\gamma_2 = \beta$. We use the l_2 distance between $\mathbb{E}(\mathbf{Y}|\mathbf{X})$ (the best possible prediction given $\mathbf{X}$ under l_2 loss) and $\hat{\mathbf{Y}} = \mathbf{U}_1\hat{\gamma}_{1,s} + \mathbf{X}_r\hat{\gamma}_2$ (the prediction by LING) as the loss function, which is called risk in the following discussions. Actually $\mathbb{E}(\mathbf{Y}|\mathbf{X}) = \mathbf{X}\beta$ is linear in $\mathbf{X}$ under fixed design model. The risk of LING can be written as

$$\begin{aligned} & \frac{1}{n}\mathbb{E}\|\mathbb{E}(\mathbf{Y}|\mathbf{X}) - \mathbf{U}_1\hat{\gamma}_{1,s} - \mathbf{X}_r\hat{\gamma}_2\|^2 \\ = \quad & \frac{1}{n}\mathbb{E}\|\mathbf{U}_1\gamma_1 + \mathbf{X}_r\gamma_2 - \mathbf{U}_1\hat{\gamma}_{1,s} - \mathbf{X}_r\hat{\gamma}_2\|^2 \end{aligned}$$

We can further decompose the risk into two terms:

$$\begin{aligned} & \frac{1}{n}\mathbb{E}\|\mathbf{U}_1\gamma_1 + \mathbf{X}_r\gamma_2 - \mathbf{U}_1\hat{\gamma}_{1,s} - \mathbf{X}_r\hat{\gamma}_2\|^2 = \\ & \frac{1}{n}\mathbb{E}\|\mathbf{U}_1\gamma_1 - \mathbf{U}_1\hat{\gamma}_{1,s}\|^2 + \frac{1}{n}\mathbb{E}\|\mathbf{X}_r\gamma_2 - \mathbf{X}_r\hat{\gamma}_2\|^2 \end{aligned} \tag{2.3.1}$$

because $\mathbf{U}_1^\top\mathbf{X}_r = 0$. Note that here the expectation is taken with respect to ϵ.

Let's calculate the two terms in (2.3.1) separately. For the first term we have:

Lemma 2.3.1.

$$\frac{1}{n}\mathbb{E}\|\mathbf{U}_1\gamma_1 - \mathbf{U}_1\hat{\gamma}_{1,s}\|^2 = \frac{1}{n}\sum_{j=1}^{k_2}\frac{d_j^4\sigma^2 + \gamma_{1,j}^2 n^2\lambda^2}{(d_j^2 + n\lambda)^2} \tag{2.3.2}$$

Here $\gamma_{1,j}$ is the j^{th} element of γ_1.

Proof. Let $S \in k_2 \times k_2$ be the diagonal matrix with $S_{j,j} = \frac{d_j^2}{d_j^2+n\lambda}$. So we have $\hat{\gamma}_{1,s} = S\mathbf{U}_1^\top \mathbf{Y} = S\gamma_1 + S\mathbf{U}_1^\top \epsilon$, $\mathbb{E}(\hat{\gamma}_{1,s}) = S\gamma_1$.

$$\begin{aligned}
& \frac{1}{n}\mathbb{E}\|\mathbf{U}_1\gamma_1 - \mathbf{U}_1\hat{\gamma}_{1,s}\|^2 \\
= & \frac{1}{n}\mathbb{E}\|\mathbf{U}_1\mathbb{E}(\hat{\gamma}_{1,s}) - \mathbf{U}_1\hat{\gamma}_{1,s}\|^2 \\
& + \frac{1}{n}\|\mathbf{U}_1\gamma_1 - \mathbf{U}_1\mathbb{E}(\hat{\gamma}_{1,s})\|^2 \\
= & \frac{1}{n}\mathbb{E}\|\mathbf{U}_1 S\mathbf{U}_1^\top \epsilon\|^2 + \frac{1}{n}\|\gamma_1 - S\gamma_1\|^2 \\
= & \frac{1}{n}\mathbb{E}\text{Tr}(\mathbf{U}_1 S^2\mathbf{U}_1^\top \epsilon\epsilon^\top) + \frac{1}{n}\|\gamma_1 - S\gamma_1\|^2 \\
= & \frac{1}{n}\mathbb{E}\text{Tr}(S^2)\sigma^2 + \frac{1}{n}\|\gamma_1 - S\gamma_1\|^2 \\
= & \frac{1}{n}\sum_{j=1}^{k_2} \frac{d_j^4\sigma^2 + \gamma_{1,j}^2 n^2\lambda^2}{(d_j^2+n\lambda)^2}
\end{aligned}$$

□

Now consider the second term in (2.3.1).

Note that

$$\mathbf{X}_r = \mathbf{U}\mathbf{D}_3\mathbf{V}^\top$$

The residual $\mathbf{Y}_r$ after the first stage can be represented by

$$\mathbf{Y}_r = \mathbf{Y} - \mathbf{U}_1\hat{\gamma}_1 = (I - \mathbf{U}_1\mathbf{U}_1^\top)\mathbf{Y} = \mathbf{X}_r\gamma_2 + (I - \mathbf{U}_1\mathbf{U}_1^\top)\epsilon$$

and the optimal coefficient obtained in the second GD stage is

$$\hat{\gamma}_2 = (\mathbf{X}_r^\top \mathbf{X}_r + n\lambda I)^{-1}\mathbf{X}_r^\top \mathbf{Y}_r$$

For simplicity, let $\epsilon_2 = (I - \mathbf{U}_1\mathbf{U}_1^\top)\epsilon$.

Lemma 2.3.2.

$$\mathbb{E}\|\mathbf{X}_r\gamma_2 - \mathbf{X}_r\hat{\gamma}_2\|^2 = \sum_{i=k_2+1}^{p} \frac{1}{(d_i^2+n\lambda)^2}(d_i^4\sigma^2 + n\lambda^2 d_i^2\alpha_i^2) \tag{2.3.3}$$

where α_i is the i^{th} element of $\alpha = \mathbf{V}^\top\gamma_2$

Proof. First define

$$\begin{aligned}\mathbf{X}_\lambda &= \mathbf{X}_r^\top\mathbf{X}_r + n\lambda I\\ \mathbf{D}_\lambda &= \mathbf{D}_3^2 + n\lambda I\end{aligned}$$

$$\begin{aligned}\mathbb{E}\|\mathbf{X}_r\gamma_2 - \mathbf{X}_r\hat{\gamma}_2\|^2 &= \|\mathbf{X}_r\gamma_2 - \mathbf{X}_r\mathbb{E}(\hat{\gamma}_2)\|^2 &(2.3.4)\\ &+ \mathbb{E}\|\mathbf{X}_r\mathbb{E}(\hat{\gamma}_2) - \mathbf{X}_r\hat{\gamma}_2\|^2 &(2.3.5)\end{aligned}$$

Consider (2.3.4) and (2.3.5) separately.

$$\begin{aligned}(2.3.4) &= \|\mathbf{X}_r\mathbf{X}_\lambda^{-1}\mathbf{X}_r^\top\mathbf{X}_r\gamma_2 - \mathbf{X}_r\gamma_2\|^2\\ &= \|\mathbf{U}\mathbf{D}_3\mathbf{D}_\lambda^{-1}\mathbf{D}_3^2\mathbf{V}^\top\gamma_2 - \mathbf{U}\mathbf{D}_3\mathbf{V}^\top\gamma_2\|^2\\ &= \|\mathbf{D}_3\mathbf{D}_\lambda^{-1}\mathbf{D}_3^2\alpha - \mathbf{D}_3\alpha\|^2\\ &= \sum_{i=k_2+1}^{p}\alpha_i^2 d_i^2(\frac{n\lambda}{d_i^2+n\lambda})^2\end{aligned}$$

$$\begin{aligned}(2.3.5) = \mathbb{E}_{\epsilon_2}&\|\mathbf{X}_r\mathbf{X}_\lambda^{-1}\mathbf{X}_r^\top\epsilon_2\|^2\\ &= \mathbb{E}_{\epsilon_2}\mathrm{Tr}\big(\mathbf{X}_r\mathbf{X}_\lambda^{-1}\mathbf{X}_r^\top\mathbf{X}_r\mathbf{X}_\lambda^{-1}\mathbf{X}_r^\top\epsilon_2\epsilon_2^\top\big)\\ &= \mathbb{E}_{\epsilon_2}\mathrm{Tr}\big(\mathbf{D}_3\mathbf{D}_\lambda^{-1}\mathbf{D}_3^2\mathbf{D}_\lambda^{-1}\mathbf{D}_3\mathbf{U}^\top\epsilon_2\epsilon_2^\top\mathbf{U}\big)\\ &= \mathrm{Tr}\big(\mathbf{D}_3\mathbf{D}_\lambda^{-1}\mathbf{D}_3^2\mathbf{D}_\lambda^{-1}\mathbf{D}_3\mathbb{E}_{\epsilon_2}[\mathbf{U}^\top\epsilon_2\epsilon_2^\top\mathbf{U}]\big)\end{aligned}$$

Note that

$$\mathbb{E}_{\epsilon_2}[\mathbf{U}^\top \epsilon_2 \epsilon_2^\top \mathbf{U}] = \text{diag}(0, I_{p-k_2})\sigma^2$$

(diag$(0, I_{p-k_2})$replace the top $k_2 \times k_2$ block of the identity matrix with 0),

$$(2.3.5) = \sum_{i=k_2+1}^{p} \frac{d_i^4}{(d_i^2 + n\lambda)^2}\sigma^2 \tag{2.3.6}$$

Add the two terms together finishes the proof. □

Plug (2.3.2) (2.3.3) into (2.3.1) we have

Theorem 2.3.3. *The risk of LING algorithm under fixed design setting is*

$$\frac{1}{n}\sum_{j=1}^{k_2} \frac{d_j^4\sigma^2 + \gamma_{1,j}^2 n^2\lambda^2}{(d_j^2 + n\lambda)^2} + \frac{1}{n}\sum_{i=k_2+1}^{p} \frac{d_i^4\sigma^2 + n^2\lambda^2 d_i^2\alpha_i^2}{(d_i^2 + n\lambda)^2} \tag{2.3.7}$$

Remark 2.3.4. This risk is the same as the risk of ridge regression provided by Lemma 1 in [18]. Actually, LING gets exactly the same prediction as RR on the training dataset. This is very intuitive since on the training set LING is essentially decomposing the RR solution into the first stage shrinkage PCR predictor on top k_2 PCs and the second stage GD predictor over the residual spaces as explained in section 2.2.

2.4 Experiments

In this section we compare the accuracy and computational cost (evaluated in terms of FLOPS) of 3 different algorithms for solving Ridge Regression: Gradient Descent with Optimal step size (GD), Stochastic Variance Reduction Gradient (SVRG) [34] and LING.

Here SVRG is an improved version of stochastic gradient descent which achieves exponential convergence with constant step size. We also consider Principal Component Regression (PCR) [4, 35] which is another common way for running large scale regression. Experiments are performed on 3 simulated models and 2 real datasets. In general, LING performs well on all 3 simulated datasets while GD, SVRG and PCR fails in some cases. For two real datasets, all algorithms give reasonable performance while SVRG and LING are the best. Moreover, both stages of LING require only a moderate amount of matrix multiplications each cost $O(np)$, much faster to run on matlab compared with SVRG which contains a lot of loops.

2.4.1 Simulated Data

Three different datasets are constructed based on the fixed design model $\mathbf{Y} = \mathbf{X}\beta + \epsilon$ where $\mathbf{X}$ is of size 2000×1500. In the three experiments $\mathbf{X}$ and β are generated randomly in different ways (more details in following sections) and i.i.d Gaussian noise is added to $\mathbf{X}\beta$ to get $\mathbf{Y}$. Then GD, SVRG, PCR and LING are performed on the dataset. For GD, we try different number of iterations n_1. For SVRG, we vary the number of passes through data denoted by n_{SVRG}. The numbers of iterations SVRG takes equals the number of passes through data times sample size and each iteration takes $O(p)$ FLOPS. The step size of SVRG is chosen by cross validation but this cost is not considered when evaluating the total computational cost. Note that one advantage of GD and LING is that due to the simple quadratic form of the target function, their step size can be computed directly

Table 2.1: parameter setup for simulated data

	Model 1	Model 2	Model 3
k_1	21,22,23,26 30,50,100	20,30,50 100,150,400	20,30,50,100 150,400
n_1	10,20,30 50,80,100 150,200	2,4,6,8,10 15,20,30	6,10,15,20 30,50,80 120,180,250
k_2	20	20	20
n_2	1,2,3,5 8,13,20	2,4,6,8,10 15,20,30	2,4,6,8,10 15,30
n_{SVRG}	30,50,80 120,150	5,10,20 30,50	5,10,15,25 40,60,90

from the data without cross validation which introduces extra cost. For PCR we pick different number of PCs (k_1). For LING we pick top k_2 PCs in the first stage and try different number of iterations n_2 in the second stage. The computational cost and the risk of the four algorithms are computed. The above procedure is repeated over 20 random generations of $\mathbf{X}$, β and $\mathbf{Y}$. The risk and computational cost of the traditional RR solution (2.1.2) for every dataset is also computed as a benchmark.

The parameter set up for the three datasets are listed in table 2.1.

Model 1

In this model the design matrix $\mathbf{X}$ has a steep spectrum. The top 30 singular values of $\mathbf{X}$ decay exponentially as 1.3^i where $i = 40, 39, 38..., 11$. The spectrum of $\mathbf{X}$ is shown in figure 2.4. To generate $\mathbf{X}$, we fix the diagonal matrix $\mathbf{D}_e$ with the designed spectrum and construct $\mathbf{X}$ by $\mathbf{X} = \mathbf{U}_e \mathbf{D}_e \mathbf{V}_e^\top$ where $\mathbf{U}_e$, $\mathbf{V}_e$ are two random orthonormal matrices. The elements of β are sampled uniformly from interval $[-2.5, 2.5]$. Under this set up, most of the energy of the $\mathbf{X}$ matrix lies in top PCs since the top singular values are much larger than the remaining ones so PCR works well. But as indicated by (2.2.2), the convergence of GD is very slow.

The computational cost and average risk of the four algorithms and also the RR solution (2.1.2) over 20 repeats are shown in figure 2.1. As shown in figure 2.1 both PCR and LING work well by achieving risk close to RR at less computational cost. SVRG is worse than PCR and LING but much better than GD.

Model 2

In this model the design matrix $\mathbf{X}$ has a flat spectrum. The singular values of $\mathbf{X}$ are sampled uniformly from $[\frac{\sqrt{2000}}{2}, \sqrt{2000}]$. The spectrum of $\mathbf{X}$ is shown in figure 2.5. To generate $\mathbf{X}$, we fix the diagonal matrix $\mathbf{D}_e$ with the designed spectrum and construct $\mathbf{X}$ by $\mathbf{X} = \mathbf{U}_e \mathbf{D}_e \mathbf{V}_e^\top$ where $\mathbf{U}_e$, $\mathbf{V}_e$ are two random orthonormal matrices. The elements of β are sampled uniformly from interval $[-2.5, 2.5]$. Under this set up, the signal is widely spread among all PCs since the spectrum of $\mathbf{X}$ is relatively flat. PCR breaks down

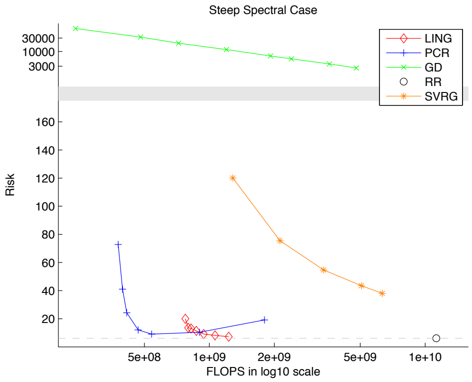

Figure 2.1: Model 1, Risk VS. Computational Cost plot. PCR and LING approaches the RR risk very fast. SVRG also approaches RR risk but cost more than the previous two. GD is very slow and inaccurate.

because it fails to catch the signal on bottom PCs. As indicated by (2.2.2), GD converges relatively fast due to the flat spectrum of $\mathbf{X}$.

The computational cost and average risk of the four algorithms and also the RR solution (2.1.2) over 20 repeats are shown in figure 2.2. As shown by the figure GD works best since it approaches the risk of RR at the the lowest computational cost. LING and SVRG also work by achieving reasonably low risk with less computational cost. PCR works poorly as explained before.

Model 3

This model presented a special case where both PCR and GD will break down. The singular values of $\mathbf{X} \in 2000 \times 1500$ are constructed by first uniformly sample from $[\frac{\sqrt{2000}}{2}, \sqrt{2000}]$. The top 15 sampled values are then multiplied by 10. The top 100

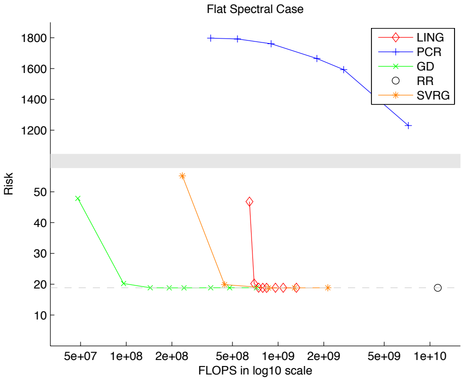

Figure 2.2: Model 2, Risk VS. Computational Cost plot. GD approaches the RR risk very fast. SVRG and LING are slower than GD but still achieves risk close to RR at less cost. PCR is slow and has huge risk.

singular values of $\mathbf{X}$ are shown in figure 2.6. To generate $\mathbf{X}$, we fix the diagonal matrix $\mathbf{D}_e$ with the designed spectrum and construct $\mathbf{X}$ by $\mathbf{X} = \mathbf{U}_e\mathbf{D}_e$ where $\mathbf{U}_e$ is a random orthonormal matrix. The first 15 and last 1000 elements of the coefficient vector $\beta \in 1500 \times 1$ are sampled uniformly from interval $[-2.5, 2.5]$ and other elements of β remains 0. In this set up, $\mathbf{X}$ has orthogonal columns which are the PCs, and the signal lies only on the top 15 and bottom 1000 PCs. PCR won't work since a large proportion of signal lies on the bottom PCs. On the other hand, GD won't work as well since the top few singular values are too large compared with other singular values, which makes GD converges very slowly.

The computational cost and risk of the four algorithms and also the RR solution (2.1.2) over 20 repeats are shown in figure 2.3. As shown by the figure LING works best in this set up. SVRG is slightly worse than LING but still approaching RR with less cost. In this case, GD converges slowly and PCR is completely off target as explained before.

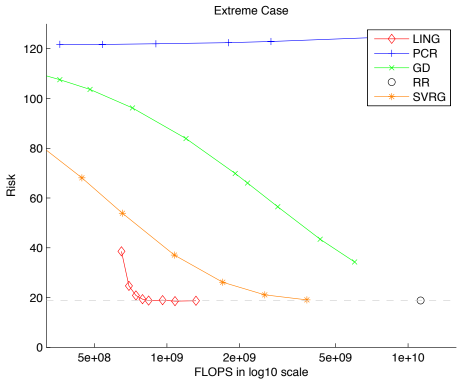

Figure 2.3: Model 3, Risk VS. Computational Cost plot. LING approaches RR risk the fastest. SVRG is slightly slower than LING. GD also approaches RR risk but cost more than LING. PCR has a huge risk no matter how many PCs are selected.

2.4.2 Real Data

In this section we compare PCR, GD, SVRG and LING with the RR solution (2.1.2) on two real datasets.

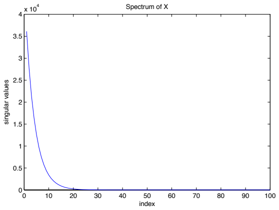

Figure 2.4: Top 100 singular values of $\mathbf{X}$ in Model 1

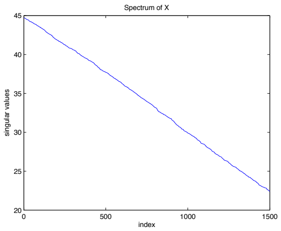

Figure 2.5: Singular values of $\mathbf{X}$ in Model 2

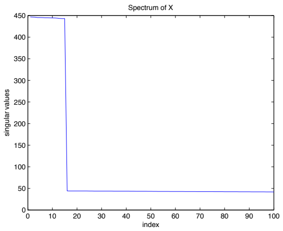

Figure 2.6: Top 100 singular values of $\mathbf{X}$ in Model 3

Gisette Dataset

The first is the gisette data set [27] from the UCI repository which is a bi-class classification task. Every row of the design matrix $\mathbf{X} \in 6000 \times 5000$ consists of pixel features of a single digit ”4” or ”9” and $\mathbf{Y}$ gives the class label. Among the 6000 samples, we use 5000 for training and 1000 for testing. The classification error rate for RR solution (2.1.2) is 0.019. Since the goal is to compare different algorithms for regression, we don’t care about achieving the state of the art accuracy for this dataset as long as regression works reasonably well. When running PCR, we pick top $k_1 = 10, 20, 40, 80, 150, 300, 400$ PCs and in GD we iterate $n_1 = 2, 5, 10, 15, 20, 30, 50, 100, 150$ times. For SVRG we try $n_{\text{SVRG}} = 1, 2, 3, 5, 10, 20, 40, 80$ passes through the data. For LING we pick $k_2 = 5, 15$ PCs in the first stage and try $n_2 = 1, 2, 4, 8, 10, 15, 20, 30, 50$ iterations in the second stage. The computational cost and average classification error of the four algorithms and also the RR solution (2.1.2) on test set over 6 different train test splits are shown in figure 2.7. The top 150 singular values of $\mathbf{X}$ are shown in figure 2.9. As shown in the figure, SVRG gets close to the RR error very fast. The two curves of LING with $k_2 = 5, 15$ are slower than SVRG since some initial FLOPS are spent on computing top PCs but after that they approach RR error very fast. GD also converges to RR but cost more than the previous two algorithms. PCR performs worst in terms of error and computational cost.

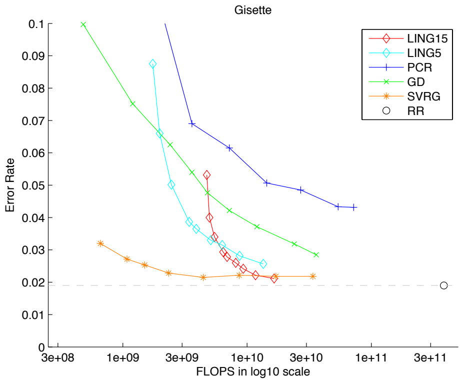

Figure 2.7: Gisette, Error Rate VS. Computational Cost plot. SVRG achieves small error rate fastest. Two LING lines with different n_2 spent some FLOPS on computing top PCs first, but then converges very fast to a lower error rate. GD and PCR also provide reasonably small error rate and are faster than RR, but suboptimal compared with SVRG and LING.

Buzz in Social Media

The second dataset is the UCI buzz in social media dataset which is a regression task. The goal is to predict popularity (evaluated by the mean number of active discussions) of a certain topic on Twitter over a period. The original feature matrix contains some statistics about this topic over that period like number of discussions created and new authors interacting on the topic. The original feature dimension is 77. We add quadratic interactions to make it 3080. To save time, we only used a subset of 8000 samples. The samples are split into 6000 train and 2000 test. We use MSE on the test data set as the error measure. For PCR we pick $k_1 = 10, 20, 30, 50, 100, 150$ PCs and in GD we iterate $n_1 = 1, 2, 4, 6, 8, 10, 15, 20, 30, 40, 60, 100$ times. For SVRG we try $n_{\text{SVRG}} = 1, 2, 3, 5, 10, 15, 20, 40, 80$ passes through the dataset and for LING we pick $k_2 = 5, 15$ in the first stage and $n_2 = 0, 1, 2, 4, 6, 8, 10, 15, 20, 25$ iterations in the second stage. The computational cost and average MSE on test set over 5 different train test splits are shown in figure 2.8. The top 150 singular values of $\mathbf{X}$ are shown in figure 2.10. As shown in the figure, SVRG approaches MSE of RR very fast. LING spent some initial FLOPS for computing top PCs but after that converges fast. GD and PCR also achieves reasonable performance but suboptimal compared with SVRG and LING. The MSE of PCR first decays when we add more PCs into regression but finally goes up due to overfit.

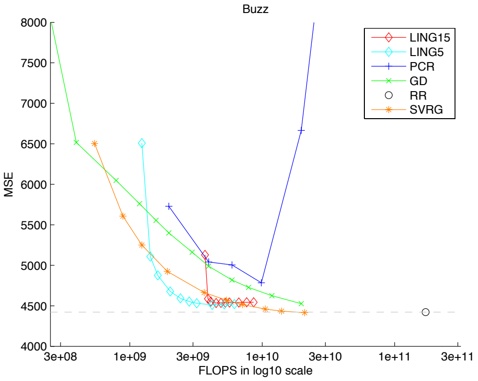

Figure 2.8: Buzz, MSE VS. Computational Cost plot. SVRG and two LING lines with different n_2 achieves small MSE fast. GD is slower than LING and SVRG. PCR reaches its smallest MSE at $k_1 = 50$ then overfits.

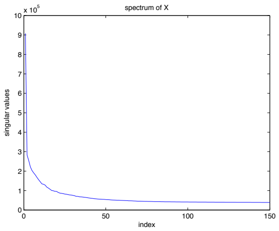

Figure 2.9: Top 150 singular values of $\mathbf{X}$ in Gisette Dataset

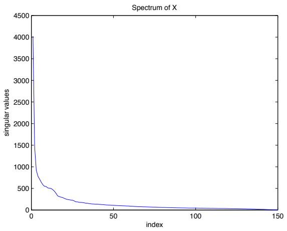

Figure 2.10: Top 150 singular values of $\mathbf{X}$ in Social Media Buzz Dataset

2.5 Summary

In this paper we present a two stage algorithm LING for computing large scale Ridge Regression which is both fast and robust in contrast to the well known approaches GD and PCR. We show that under the fixed design setting LING actually has the same risk as Ridge Regression assuming convergence. In the experiments, LING achieves good performances on all datasets when compare with three other large scale regression algorithms.

We conjecture that same strategy can be also used to accelerate the convergence of stochastic gradient descent when solving Ridge Regression since the first stage in LING essentially removes the high variance directions of $\mathbf{X}$, which will lead to variance reduction for the random gradient direction generated by SGD.

Chapter 3

Large Scale Canonical Correlation Analysis with Iterative Least Squares

3.1 Introduction

Canonical Correlation Analysis (CCA) is a widely used spectrum method for finding correlation structures in multi-view datasets introduced by [33]. Recently, [7, 22, 36] proved that CCA is able to find the right latent structure under certain hidden state model. For modern machine learning problems, CCA has already been successfully used as a dimensionality reduction technique for the multi-view setting. For example, A CCA between the text description and image of the same object will find common structures between the two different views, which generates a natural vector representation of the object. In [22], CCA is performed on a large unlabeled dataset in order to generate low dimensional features to a regression problem where the size of labeled dataset is small. In [17, 19] a CCA between words and its context is implemented on several large corpora to generate low dimensional vector representations of words which captures useful semantic features.

When the data matrices are small, the classical algorithm for computing CCA involves first a QR decomposition of the data matrices which pre whitens the data and then a Singular Value Decomposition (SVD) of the whitened covariance matrix as introduced in [25]. This is exactly how Matlab computes CCA. But for huge datasets this procedure becomes extremely slow. For data matrices with huge sample size [5] proposed a fast CCA approach based on a fast inner product preserving random projection called Subsampled Randomized Hadamard Transform but it's still slow for datasets with a huge number of features. In this paper we introduce a fast algorithm for finding the top k_{cca} canonical variables from huge sparse data matrices (a single multiplication with these sparse matrices is very fast) $\mathbf{X} \in n \times p_1$ and $\mathbf{Y} \in n \times p_2$ the rows of which are i.i.d samples from a pair of random vectors. Here $n \gg p_1, p_2 \gg 1$ and k_{cca} is relatively small number like 50 since the primary goal of CCA is to generate low dimensional features. Under this set up, QR decomposition of a $n \times p$ matrix cost $O(np^2)$ which is extremely slow even if the matrix is sparse. On the other hand since the data matrices are sparse, $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{Y}^\top\mathbf{Y}$ can be computed very fast. So another whitening strategy is to compute $(\mathbf{X}^\top\mathbf{X})^{-\frac{1}{2}}, (\mathbf{Y}^\top\mathbf{Y})^{-\frac{1}{2}}$. But when p_1, p_2 are large this takes $O(\max\{p_1^3, p_2^3\})$ which is both slow and numerically unstable.

The main contribution of this paper is a fast iterative algorithm L-CCA consists of only QR decomposition of relatively small matrices and a couple of matrix multiplications which only involves huge sparse matrices or small dense matrices. This is achieved by reducing the computation of CCA to a sequence of fast Least Square iterations. It is proved that L-CCA asymptotically converges to the exact CCA solution and error analysis for finite iterations is also provided. As shown by the experiments, L-CCA also has favorable performance on real datasets when compared with other CCA approximations given a fixed CPU time.

It's worth pointing out that approximating CCA is much more challenging than SVD (or PCA). As suggested by [28, 30], to approximate the top singular vectors of $\mathbf{X}$, it suffices to randomly sample a small subspace in the span of $\mathbf{X}$ and some power iteration with this small subspace will automatically converge to the directions with top singular values. On the other hand CCA has to search through the whole $\mathbf{X}$ $\mathbf{Y}$ span in order to capture directions with large correlation. For example, when the most correlated directions happen to live in the bottom singular vectors of the data matrices, the random sample scheme will miss them completely. On the other hand, what L-CCA algorithm doing intuitively is running an exact search of correlation structures on the top singular vectors and an fast gradient based approximation on the remaining directions.

3.2 Background: Canonical Correlation Analysis

3.2.1 Definition

Canonical Correlation Analysis (CCA) can be defined in many different ways. Here we use the definition in [22, 36] since this version naturally connects CCA with the Singular Value Decomposition (SVD) of the whitened covariance matrix, which is the key to understanding our algorithm.

Definition 3.2.1. Let $\mathbf{X} \in n \times p_1$ and $\mathbf{Y} \in n \times p_2$ where the rows are i.i.d samples from a pair of random vectors. Let $\mathbf{\Phi_x} \in p_1 \times p_1, \mathbf{\Phi_y} \in p_2 \times p_2$ and use $\phi_{x,i}, \phi_{y,j}$ to denote the columns of $\mathbf{\Phi_x}, \mathbf{\Phi_y}$ respectively. $\mathbf{X}\phi_{x,i}, \mathbf{Y}\phi_{y,j}$ are called canonical variables if

$$\phi_{x,i}^\top \mathbf{X}^\top \mathbf{Y} \phi_{y,j} = \begin{cases} d_i & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases}$$

$$\phi_{x,i}^\top \mathbf{X}^\top \mathbf{X} \phi_{x,j} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases} \qquad \phi_{y,i}^\top \mathbf{Y}^\top \mathbf{Y} \phi_{y,j} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases}$$

$\mathbf{X}\phi_{x,i}, \mathbf{Y}\phi_{y,i}$ is the i^{th} pair of canonical variables and d_i is the i^{th} canonical correlation.

3.2.2 CCA and SVD

First introduce some notation. Let

$$\mathbf{C}_{xx} = \mathbf{X}^\top \mathbf{X} \quad \mathbf{C}_{yy} = \mathbf{Y}^\top \mathbf{Y} \quad \mathbf{C}_{xy} = \mathbf{X}^\top \mathbf{Y}$$

For simplicity assume $\mathbf{C}_{xx}$ and $\mathbf{C}_{yy}$ are full rank and Let

$$\tilde{\mathbf{C}}_{xy} = \mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{C}_{xy}\mathbf{C}_{yy}^{-\frac{1}{2}}$$

The following lemma provides a way to compute the canonical variables by SVD.

Lemma 3.2.2. *Let $\tilde{\mathbf{C}}_{xy} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ be the SVD of $\tilde{\mathbf{C}}_{xy}$ where u_i, v_j denote the left, right singular vectors and d_i denotes the singular values. Then $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}u_i$, $\mathbf{Y}\mathbf{C}_{yy}^{-\frac{1}{2}}v_j$ are the canonical variables of the $\mathbf{X}$, $\mathbf{Y}$ space respectively.*

Proof. Plug $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}u_i$, $\mathbf{Y}\mathbf{C}_{yy}^{-\frac{1}{2}}v_j$ into the equations in Definition 3.2.1 directly proves lemma 3.2.2 □

As mentioned before, we are interested in computing the top k_{cca} canonical variables where $k_{\text{cca}} \ll p_1, p_2$. Use $\mathbf{U}_1, \mathbf{V}_1$ to denote the first k_{cca} columns of $\mathbf{U}$, $\mathbf{V}$ respectively and use $\mathbf{U}_2, \mathbf{V}_2$ for the remaining columns. By lemma 3.2.2, the top k_{cca} canonical variables can be represented by $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1$ and $\mathbf{Y}\mathbf{C}_{yy}^{-\frac{1}{2}}\mathbf{V}_1$.

3.3 Compute CCA by Iterative Least Squares

Since the top canonical variables are connected with the top singular vectors of $\tilde{\mathbf{C}}_{xy}$ which can be compute with orthogonal iteration [24] (it's called simultaneous iteration in [53]), we can also compute CCA iteratively. A detailed algorithm is presented in Algorithm4:

The convergence result of Algorithm 4 is stated in the following theorem:

Theorem 3.3.1. *Assume $|d_1| > |d_2| > |d_3| ... > |d_{k_{cca}+1}|$ and $\mathbf{U}_1^\top \mathbf{C}_{xx}^{\frac{1}{2}}\mathbf{G}$ is non singular (this will hold with probability 1 if the elements of $\mathbf{G}$ are i.i.d Gaussian). The columns of*

Algorithm 4 CCA via Iterative LS

```
Input : Data matrix $\mathbf{X} \in n \times p_1$ ,$\mathbf{Y} \in n \times p_2$. A target dimension $k_{\text{cca}}$. Number of orthogonal iterations $t_1$
Output : $\mathbf{X}_{k_{\text{cca}}} \in n \times k_{\text{cca}}$, $\mathbf{Y}_{k_{\text{cca}}} \in n \times k_{\text{cca}}$ consist of top $k_{\text{cca}}$ canonical variables of $\mathbf{X}$ and $\mathbf{Y}$.
1.Generate a $p_1 \times k_{\text{cca}}$ dimensional random matrix $\mathbf{G}$ with i.i.d standard normal entries.
2.Let $\mathbf{X}_0 = \mathbf{XG}$
3.
for $t = 1$ to $t_1$ do
    $\mathbf{Y}_t = \mathbf{H_Y X}_{t-1}$ where $\mathbf{H_Y} = \mathbf{Y}(\mathbf{Y}^\top \mathbf{Y})^{-1}\mathbf{Y}^\top$
    $\mathbf{X}_t = \mathbf{H_X Y}_t$ where $\mathbf{H_X} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$
end for
4.$\mathbf{X}_{k_{\text{cca}}} = \text{QR}(\mathbf{X}_{t_1}), \mathbf{Y}_{k_{\text{cca}}} = \text{QR}(\mathbf{Y}_{t_1})$
Function QR($\mathbf{X}_t$) extract an orthonormal basis of the column space of $\mathbf{X}_t$ with QR decomposition
```

$\mathbf{X}_{k_{cca}}$ *and* $\mathbf{Y}_{k_{cca}}$ *will converge to the top* k_{cca} *canonical variables of* $\mathbf{X}$ *and* $\mathbf{Y}$ *respectively if* $t_1 \to \infty$.

Theorem 3.3.1 is proved by showing it's essentially an orthogonal iteration [24, 53] for computing the top k_{cca} eigenvectors of $\mathbf{A} = \tilde{\mathbf{C}}_{xy}\tilde{\mathbf{C}}_{xy}^\top$. A detailed proof is provided in the appendix.

3.3.1 A Special Case

When $\mathbf{X}$ $\mathbf{Y}$ are sparse and $\mathbf{C}_{xx}, \mathbf{C}_{yy}$ are diagonal (like the Penn Tree Bank dataset in the experiments), Algorithm 4 can be implemented extremely fast since we only need to multiply with sparse matrices or inverting huge but diagonal matrices in every iteration. QR decomposition is performed not only in the end but after every iteration for numerical stability issues (here we only need to QR with matrices much smaller than $\mathbf{X}, \mathbf{Y}$). We call this fast version D-CCA in the following discussions.

When $\mathbf{C}_{xx}, \mathbf{C}_{yy}$ aren't diagonal, computing matrix inverse becomes very slow. But we can still run D-CCA by approximating $(\mathbf{X}^\top \mathbf{X})^{-1}$, $(\mathbf{Y}^\top \mathbf{Y})^{-1}$ with $(\text{diag}(\mathbf{X}^\top \mathbf{X}))^{-1}$, $(\text{diag}(\mathbf{Y}^\top \mathbf{Y}))^{-1}$ in algorithm 4 when speed is a concern. But this leads to poor performance when $\mathbf{C}_{xx}, \mathbf{C}_{yy}$ are far from diagonal as shown by the URL dataset in the experiments.

3.3.2 General Case

Algorithm 4 reduces the problem of CCA to a sequence of iterative least square problems. When $\mathbf{X}, \mathbf{Y}$ are huge, solving LS exactly is still slow since it consists inverting a huge matrix but fast LS methods are relatively well studied. There are many ways to approximate the LS solution by optimization based methods like Gradient Descent [1, 58], Stochastic Gradient Descent [34, 12] or by random projection and subsampling based methods like [21, 16]. A fast approximation to the top k_{cca} canonical variables can be obtained by replacing the exact LS solution in every iteration of Algorithm 4 with a fast approximation. Here we choose LING [58] which works well for large sparse design matrices for solving the LS problem in every CCA iteration.

The connection between CCA and LS has been developed under different setups for different purposes. [52] shows that CCA in multi label classification setting can be formulated as an LS problem. [55] also formulates CCA as a recursive LS problem and builds an online version based on this observation. The benefit we take from this iterative LS formulation is that running a fast LS approximation in every iteration will give us a fast CCA approximation with both provable theoretical guarantees and favorable experimental performance.

3.4 Algorithm

In this section we introduce L-CCA which is a fast CCA algorithm based on Algorithm 4.

3.4.1 LING: a Gradient Based Least Square Algorithm

First we need to introduce the fast LS algorithm LING as mentioned in section 3.3.2 which is used in every orthogonal iteration of L-CCA .

Remark 3.4.1. The version of LING introduced here is very similar to LING in Chapter 2. We modify some implementation details to make it work better with the dataset we run CCA on, but the mathematical content of LING algorithm in Chapter 2 and here are exactly the same. The introduction of LING here is self explanatory and the readers will be able to get a complete idea of the LING without referring to Chapter 2.

Consider the LS problem:

$$\beta^* = \arg\min_{\beta \in \mathbb{R}^p} \{\|X\beta - Y\|^2\}$$

for $X \in n \times p$ and $Y \in n \times 1$. For simplicity assume X is full rank. $X\beta^* = X(X^\top X)^{-1}X^\top Y$ is the projection of Y onto the column space of X. In this section we introduce a fast algorithm LING to approximately compute $X\beta^*$ without formulating $(X^\top X)^{-1}$ explicitly which is slow for large p. The intuition of LING is as follows. Let $U_1 \in n \times k_{\text{pc}}$ ($k_{\text{pc}} \ll p$) be the top k_{pc} left singular vectors of X and $U_2 \in n \times (p - k_{\text{pc}})$ be the remaining singular vectors. In LING we decompose $X\beta^*$ into two orthogonal components,

$$X\beta^* = U_1U_1^\top Y + U_2U_2^\top Y$$

the projection of Y onto the span of U_1 and the projection onto the span of U_2. The first term can be computed fast given U_1 since k_{pc} is small. U_1 can also be computed fast approximately with the randomized SVD algorithm introduced in [28] which only requires a few fast matrix multiplication and a QR decomposition of $n \times k_{\text{pc}}$ matrix. The details for finding U_1 are illustrated in the appendix. Let $Y_r = Y - U_1U_1^\top Y$ be the residual of Y after projecting onto U_1. For the second term, we compute it by solving the optimization problem

$$\min_{\beta_r \in \mathbb{R}^p} \{\|X\beta_r - Y_r\|^2\}$$

Algorithm 5 LING

```
Input : $X \in n \times p$ ,$Y \in n \times 1$. $k_{\text{pc}}$, number of top left singular vectors selected. $t_2$, number of iterations in Gradient Descent.
Output : $\hat{Y} \in n \times 1$, which is an approximation to $X(X^\top X)^{-1}X^\top Y$
1. Compute $U_1 \in n \times k_{\text{pc}}$, top $k_{\text{pc}}$ left singular vector of $X$ by randomized SVD (See appendix for detailed description).
2. $Y_1 = U_1 U_1^\top X$.
3.Compute the residual. $Y_r = Y - Y_1$
4.Use gradient descent initial at the $0$ vector (see appendix for detailed description) to approximately solve the LS problem $\min_{\beta_r \in \mathcal{R}^p} \|X\beta_r - Y_r\|^2$. Use $\beta_{r,t_2}$ to denote the solution after $t_2$ gradient iterations.
5. $\hat{Y} = Y_1 + X\beta_{r,t_2}$.
```

with Gradient Descent (GD) which is also described in detail in the appendix. A detailed description of LING are presented in Algorithm 5.

In the above discussion Y is a column vector. It is straightforward to generalize LING to fit into Algorithm 4 where Y have multiple columns by applying Algorithm 5 to every column of Y.

In the following discussions, we use LING $(Y, X, k_{\text{pc}}, t_2)$ to denote the LING output with corresponding inputs which is an approximation to $X(X^\top X)^{-1}X^\top Y$.

The following theorem gives error bound of LING .

Theorem 3.4.2. *Use λ_i to denote the i^{th} singular value of X. Consider the LS problem*

$$\min_{\beta \in \mathbb{R}^p} \{\|X\beta - Y\|^2\}$$

for $X \in n \times p$ and $Y \in n \times 1$. Let $Y^* = X(X^\top X)^{-1}X^\top Y$ be the projection of Y onto the column space of X and $\hat{Y}_{t_2} =$ LING (Y, X, k_{pc}, t_2). Then

$$\|Y^* - \hat{Y}_{t_2}\|^2 \leq Cr^{2t_2} \tag{3.4.1}$$

for some constant $C > 0$ and $r = \frac{\lambda^2_{k_{pc}+1} - \lambda^2_p}{\lambda^2_{k_{pc}+1} + \lambda^2_p} < 1$

The proof is in the appendix.

Remark 3.4.3. Theorem 3.4.2 gives some intuition of why LING decompose the projection into two components. In an extreme case if we set $k_{\text{pc}} = 0$ (i.e. don't remove projection on the top principle components and directly apply GD to the LS problem), r in equation 3.4.1 becomes $\frac{\lambda_1^2 - \lambda_p^2}{\lambda_1^2 + \lambda_p^2}$. Usually λ_1 is much larger than λ_p, so r is very close to 1 which makes the error decays slowly. Removing projections on k_{pc} top singular vector will accelerate error decay by making r smaller. The benefit of this trick is easily seen in the experiment section.

3.4.2 Fast Algorithm for CCA

Our fast CCA algorithm L-CCA are summarized in Algorithm 6:

There are two main differences between Algorithm 4 and 6. We use LING to solve Least squares approximately for the sake of speed. We also apply QR decomposition on every LING output for numerical stability issues mentioned in [53].

Algorithm 6 L-CCA

```
Input : $\mathbf{X} \in n \times p_1$ ,$\mathbf{Y} \in n \times p_2$: Data matrices.
$k_{\text{cca}}$: Number of top canonical variables we want to extract.
$t_1$: Number of orthogonal iterations.
$k_{\text{pc}}$: Number of top singular vectors for LING
$t_2$: Number of GD iterations for LING
Output : $\mathbf{X}_{k_{\text{cca}}} \in n \times k_{\text{cca}}$, $\mathbf{Y}_{k_{\text{cca}}} \in n \times k_{\text{cca}}$: Top $k_{\text{cca}}$ canonical variables of $\mathbf{X}$ and $\mathbf{Y}$.
1.Generate a $p_1 \times k_{\text{cca}}$ dimensional random matrix $\mathbf{G}$ with i.i.d standard normal entries.
2.Let $\mathbf{X}_0 = \mathbf{XG}$, $\hat{\mathbf{X}}_0 = \text{QR}(\mathbf{X}_0)$
3.
for $t = 1$ to $t_1$ do
    $\mathbf{Y}_t = \textbf{LING}(\hat{\mathbf{X}}_{t-1}, \mathbf{Y}, k_{\text{pc}}, t_2)$, $\hat{\mathbf{Y}}_t = \text{QR}(\mathbf{Y}_t)$
    $\mathbf{X}_t = \textbf{LING}(\hat{\mathbf{Y}}_t, \mathbf{X}, k_{\text{pc}}, t_2)$, $\hat{\mathbf{X}}_t = \text{QR}(\mathbf{X}_t)$
end for
4.$\mathbf{X}_{k_{\text{cca}}} = \hat{\mathbf{X}}_{t_1}, \mathbf{Y}_{k_{\text{cca}}} = \hat{\mathbf{Y}}_{t_1}$
```

3.4.3 Error Analysis of L-CCA

This section provides mathematical results on how well the output of L-CCA algorithm approximates the subspace spanned by the top k_{cca} true canonical variables for finite t_1 and t_2. Note that the asymptotic convergence property of L-CCA when $t_1, t_2 \to \infty$ has already been stated by theorem 3.3.1. First we need to define the distances between subspaces as introduced in section 2.6.3 of [24]:

Definition 3.4.4. Assume the matrices are full rank. The distance between the column space of matrix $\mathbf{W}_1 \in n \times k$ and $\mathbf{Z}_1 \in n \times k$ is defined by

$$\text{dist}(\mathbf{W}_1, \mathbf{Z}_1) = \|\mathbf{H}_{\mathbf{W}_1} - \mathbf{H}_{\mathbf{Z}_1}\|_2$$

where $\mathbf{H}_{\mathbf{W}_1} = \mathbf{W}_1(\mathbf{W}_1^\top \mathbf{W}_1)^{-1}\mathbf{W}_1^\top$, $\mathbf{H}_{\mathbf{Z}_1} = \mathbf{Z}_1(\mathbf{Z}_1^\top \mathbf{Z}_1)^{-1}\mathbf{Z}_1^\top$ are projection matrices. Here the matrix norm is the spectrum norm. Easy to see $\text{dist}(\mathbf{W}_1, \mathbf{Z}_1) = \text{dist}(\mathbf{W}_1\mathbf{R}_1, \mathbf{Z}_1\mathbf{R}_2)$ for any invertible $k \times k$ matrix $\mathbf{R}_1, \mathbf{R}_2$.

We continue to use the notation defined in section 3.2. Recall that $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1$ gives the top k_{cca} canonical variables from $\mathbf{X}$. The following theorem bounds the distance between the truth $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1$ and $\hat{\mathbf{X}}_{t_1}$, the L-CCA output after finite iterations.

Theorem 3.4.5. *The distance between subspaces spanned top k_{cca} canonical variables of $\mathbf{X}$ and the subspace returned by L-CCA is bounded by*

$$\textit{dist}(\hat{\mathbf{X}}_{t_1}, \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1) \leq C_1\left(\frac{d_{k_{cca}+1}}{d_{k_{cca}}}\right)^{2t_1} + C_2\frac{d_{k_{cca}}^2}{d_{k_{cca}}^2 - d_{k_{cca}+1}^2}r^{2t_2}$$

where C_1, C_2 are constants. $0 < r < 1$ is introduced in theorem 3.4.2. t_1 is the number of power iterations in L-CCA and t_2 is the number of gradient iterations for solving every LS problem.

The proof of theorem 3.4.5 is deferred to the appendix.

3.5 Experiments

In this section we compare several fast algorithms for computing CCA on large datasets. First let's introduce the algorithms we compared in the experiments.

- RPCCA : Instead of running CCA directly on the high dimensional $\mathbf{X}\,\mathbf{Y}$, RPCCA computes CCA only between the top k_{rpcca} principle components (left singular vector) of $\mathbf{X}$ and $\mathbf{Y}$ where $k_{\text{rpcca}} \ll p_1, p_2$. For large n, p_1, p_2, we use randomized algorithm introduced in [28] for computing the top principle components of $\mathbf{X}$ and $\mathbf{Y}$ (see appendix for details). The tuning parameter that controls the tradeoff between computational cost and accuracy is k_{rpcca}. When k_{rpcca} is small RPCCA is fast but fails to capture the correlation structure on the bottom principle components of $\mathbf{X}$ and $\mathbf{Y}$. When k_{rpcca} grows larger the principle components captures more structure in $\mathbf{X}\,\mathbf{Y}$ space but it takes longer to compute the top principle components. In the experiments we vary k_{rpcca}.

- D-CCA : See section 3.3.1 for detailed descriptions. The advantage of D-CCA is it's extremely fast. In the experiments we iterate 30 times ($t_1 = 30$) to make sure D-CCA achieves convergence. As mentioned earlier, when $\mathbf{C}_{xx}$ and $\mathbf{C}_{yy}$ are far from diagonal D-CCA becomes inaccurate.

- L-CCA : See Algorithm 6 for detailed description. We find that the accuracy of LING in every orthogonal iteration is crucial to finding directions with large correlation while a small t_1 suffices. So in the experiments we fix $t_1 = 5$ and vary t_2. In both experiments we fix $k_{\text{pc}} = 100$ so the top k_{pc} singular vectors of $\mathbf{X}, \mathbf{Y}$ and every LING iteration can be computed relatively fast.

- G-CCA : A special case of Algorithm 6 where k_{pc} is set to 0. I.e. the LS projection in every iteration is computed directly by GD. G-CCA does not need to compute top singular vectors of $\mathbf{X}$ and $\mathbf{Y}$ as L-CCA . But by equation 3.4.1 and remark 3.4.3 GD takes more iterations to converge compared with LING . Comparing G-CCA and L-CCA in the experiments illustrates the benefit of removing the top singular vectors in LING and how this can affect the performance of the CCA algorithm. Same as L-CCA we fix the number of orthogonal iterations t_1 to be 5 and vary t_2, the number of gradient iterations for solving LS.

RPCCA , L-CCA , G-CCA are all "asymptotically correct" algorithms in the sense that if we spend infinite CPU time all three algorithms will provide the exact CCA solution while D-CCA is extremely fast but relies on the assumption that $\mathbf{X}$ $\mathbf{Y}$ both have orthogonal columns. Intuitively, given a fixed CPU time, RPCCA dose an exact search on k_{rpcca} top principle components of $\mathbf{X}$ and $\mathbf{Y}$. L-CCA does an exact search on the top k_{pc} principle components ($k_{\text{pc}} < k_{\text{rpcca}}$) and an crude search over the other directions. G-CCA dose a crude search over all the directions. The comparison is in fact testing which strategy is the most effective in finding large correlations over huge datasets.

Remark 3.5.1. Both RPCCA and G-CCA can be regarded as special cases of L-CCA . When t_1 is large and t_2 is 0, L-CCA becomes RPCCA and when k_{pc} is 0 L-CCA becomes G-CCA .

In the following experiments we aims at extracting 20 most correlated directions from huge data matrices $\mathbf{X}$ and $\mathbf{Y}$. The output of the above four algorithms are two $n \times 20$ matrices $\mathbf{X}_{k_{\text{cca}}}$ and $\mathbf{Y}_{k_{\text{cca}}}$ the columns of which contains the most correlated directions.

Then a CCA is performed between $\mathbf{X}_{k_{\text{cca}}}$ and $\mathbf{Y}_{k_{\text{cca}}}$ with matlab built-in CCA function. The canonical correlations between $\mathbf{X}_{k_{\text{cca}}}$ and $\mathbf{Y}_{k_{\text{cca}}}$ indicates the amount of correlations captured from the the huge $\mathbf{X}$ $\mathbf{Y}$ spaces by above four algorithms. In all the experiments, we vary k_{rpcca} for RPCCA and t_2 for L-CCA and G-CCA to make sure these three algorithms spends almost the same CPU time (D-CCA is alway fastest). The 20 canonical correlations between the subspaces returned by the four algorithms are plotted (larger means better).

We want to make to additional comments here based on the reviewer's feedback. First, for the two datasets considered in the experiments, classical CCA algorithms like the matlab built in function takes more than an hour while our algorithm is able to get an approximate answer in less than 10 minutes. Second, in the experiments we've been focusing on getting a good fit on the training datasets and the performance is evaluated by the magnitude of correlation captured in sample. To achieve better generalization performance a common trick is to perform regularized CCA [31] which easily fits into our frame work since it's equivalent to running iterative ridge regression instead of OLS in Algorithm 4. Since our goal is to compute a fast and accurate fit, we don't pursue the generalization performance here which is another statistical issue.

3.5.1 Penn Tree Bank Word Co-ocurrence

CCA has already been successfully applied to building a low dimensional word embedding in [17, 19]. So the first task is a CCA between words and their context. The dataset used is the full Wall Street Journal Part of Penn Tree Bank which consists of 1.17 million tokens and a vocabulary size of $43k$ [37]. The rows of $\mathbf{X}$ matrix consists the indicator vectors of the current word and the rows of $\mathbf{Y}$ consists of indicators of the word after. To avoid sample sparsity for $\mathbf{Y}$ we only consider 3000 most frequent words, i.e. we only consider the tokens followed by 3000 most frequent words which is about 1 million. So $\mathbf{X}$ is of size $1000k \times 43k$ and $\mathbf{Y}$ is of size $1000k \times 3k$ where both $\mathbf{X}$ and $\mathbf{Y}$ are very sparse. Note that every row of $\mathbf{X}$ and $\mathbf{Y}$ only has a single 1 since they are indicators of words. So in this case $\mathbf{C}_{xx}, \mathbf{C}_{yy}$ are diagonal and D-CCA can compute a very accurate CCA in less than a minute as mentioned in section 3.3.1. On the other hand, even though this dataset can be solved efficiently by D-CCA , it is interesting to look at the behavior of other three algorithms which do not make use of the special structure of this problem and compare them with D-CCA which can be regarded as the truth in this particular case. For RPCCA L-CCA G-CCA we try three different parameter set ups shown in table 3.1 and the 20 correlations are shown in figure 3.1. Among the three algorithms L-CCA performs best and gets pretty close to D-CCA as CPU time increases. RPCCA doesn't perform well since a lot correlation structure of word concurrence exist in low frequency words which can't be captured in the top principle components of $\mathbf{X}$ $\mathbf{Y}$. Since the most frequent word occurs $60k$ times and the least frequent words occurs only once, the spectral of $\mathbf{X}$ drops quickly which makes GD converges very slowly. So G-CCA doesn't perform well either.

Table 3.1: Parameter Setup for Two Real Datasets

PTB word co-occurrence					URL features				
id	k_{rpcca}	t_2	t_2	CPU	id	k_{rpcca}	t_2	t_2	CPU
	RPCCA	L-CCA	G-CCA	time		RPCCA	L-CCA	G-CCA	time
1	300	7	17	170	1	600	4	7	220
2	500	38	51	460	2	600	11	16	175
3	800	115	127	1180	3	600	13	17	130

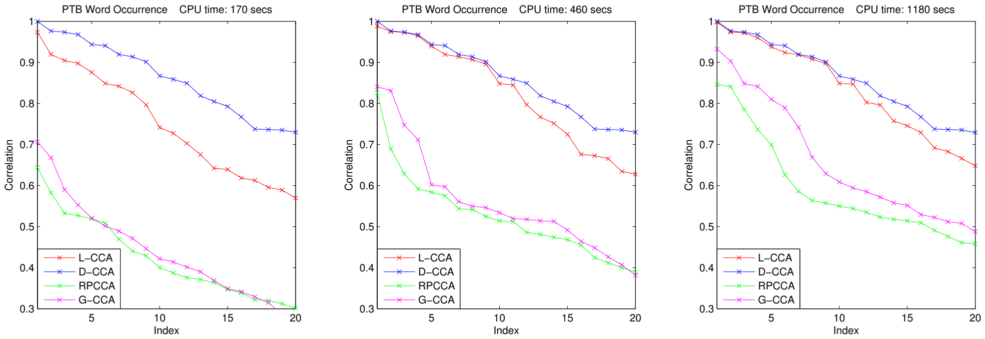

Figure 3.1: PTB word co-ocurrence: Canonical correlations of the 20 directions returned by four algorithms. x axis are the indices and y axis are the correlations.

3.5.2 URL Features

The second dataset is the URL Reputation dataset from UCI machine learning repository. The dataset contains 2.4 million URLs each represented by 3.2 million features. For simplicity we only use first $400k$ URLs. 38% of the features are host based features like WHOIS info, IP prefix and 62% are lexical based features like Hostname and Primary domain. See [41] for detailed information about this dataset. Unfortunately the features are anonymous so we pick the first 35% features as our $\mathbf{X}$ and last 35% features as our $\mathbf{Y}$. We remove the 64 continuous features and only use the Boolean features. We sort the features according to their frequency (each feature is a column of 0s and 1s, the column with most 1s are the most frequent feature). We run CCA on three different subsets of $\mathbf{X}$ and $\mathbf{Y}$. In the first experiment we select the $20k$ most frequent features of $\mathbf{X}$ and $\mathbf{Y}$ respectively. In the second experiment we select $20k$ most frequent features from $\mathbf{X}$ $\mathbf{Y}$ after removing the top 100 most frequent features of $\mathbf{X}$ and 200 most frequent features of $\mathbf{Y}$. In the third experiment we remove top 200 most frequent features from $\mathbf{X}$ and top 400 most frequent features of $\mathbf{Y}$. So we are doing CCA between two $400k * 20k$ data matrices in these experiments. In this dataset the features within $\mathbf{X}$ and $\mathbf{Y}$ has huge correlations, so $\mathbf{C}_{xx}$ and $\mathbf{C}_{yy}$ aren't diagonal anymore. But we still run D-CCA since it's extremely fast. The parameter set ups for the three subsets are shown in table 3.1 and the 20 correlations are shown in figure 3.2.

For this dataset the fast D-CCA doesn't capture largest correlation since the correlation within $\mathbf{X}$ and $\mathbf{Y}$ make $\mathbf{C}_{xx}, \mathbf{C}_{yy}$ not diagonal. RPCCA has best performance in experiment 1 but not as good in 2, 3. On the other hand G-CCA has good performance in experiment 3 but performs poorly in 1, 2. The reason is as follows: In experiment 1 the data matrices are relatively dense since they includes some frequent features. So every gradient iteration in L-CCA and G-CCA is slow. Moreover, since there are some high frequency features and most features has very low frequency, the spectrum of the data matrices in experiment 1 are very steep which makes GD in every iteration of G-CCA converges very slowly. These lead to poor performance of G-CCA . In experiment 3 since the frequent features are removed data matrices becomes more sparse and has a flat spectrum which is in favor of G-CCA . L-CCA has stable and close to best performance despite those variations in the datasets.

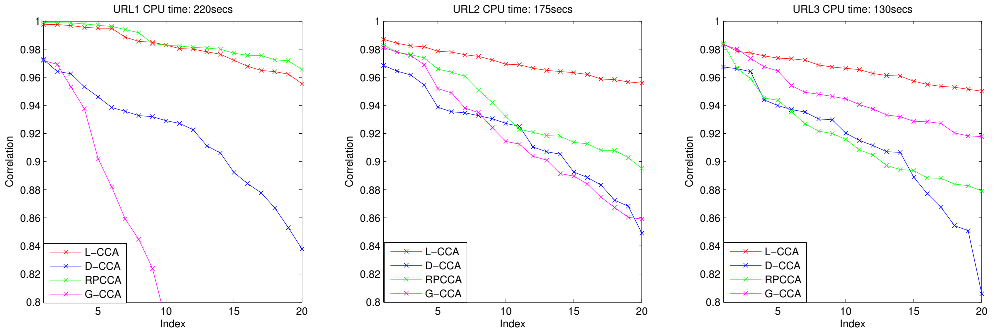

Figure 3.2: URL: Canonical correlations of the 20 directions returned by four algorithms. x axis are the indices and y axis are the correlations.

3.6 Conclusion and Future Work

In this paper we introduce L-CCA , a fast CCA algorithm for huge sparse data matrices. We construct theoretical bound for the approximation error of L-CCA comparing with the true CCA solution and implement experiments on two real datasets in which L-CCA has favorable performance. On the other hand, there are many interesting fast LS algorithms with provable guarantees which can be plugged into the iterative LS formulation of CCA. Moreover, in the experiments we focus on how much correlation is captured by L-CCA for simplicity. It's also interesting to use L-CCA for feature generation and evaluate it's performance on specific learning tasks.

Chapter 4

Augmented Approximate Gradient Algorithm for Large Scale Canonical Correlation Analysis

4.1 Introduction

4.1.1 Background

Canonical Correlation Analysis (CCA), first introduced in 1936 by [33], is a foundamental statistical tool to characterize the relationship between two multidimensional variables, which finds a wide range of applications. For example, CCA naturally fits into multi-view learning tasks and tailored to generate low dimensional feature representations using abandunt and inexpensive unlabeled datasets to supplement or refine the expensive labeled data in a semi-supervised fashion. Improved generalization accuracy has been witnessed or proved in areas such as regression [36], clustering [14, 10], dimension reduction [22, 43], word embeddings [17, 19], etc. Besides, CCA has also been successfully applied to genome-wide association study (GWAS) and has been shown powerful for understanding the relationship between genetic variations and phenotypes [56, 15].

There are various equivalent ways to define CCA and here we use the linear algebraic formulation of [26], which captures the very essense of the procedure, pursuing the directions of maximal correlations between two data matrices.

Definition 4.1.1. Let $\mathbf{X} \in \mathbb{R}^{n\times p_1}, \mathbf{Y} \in \mathbb{R}^{n\times p_2}$ and $p = \min\{p_1, p_2\}$. Canonical correlations $\lambda_1, \cdots, \lambda_p$ and corresponding canonical vectors $\mathbf{\Phi} = (\phi_1, \cdots, \phi_p), \mathbf{\Psi} = (\psi_1, \cdots, \psi_p)$ of the matrix pair $(\mathbf{X}, \mathbf{Y})$ are defined recursively by

$$(\phi_j, \psi_j) = \arg \max_{\substack{\|\mathbf{X}\phi\|=1,\, \|\mathbf{Y}\psi\|=1 \\ \phi^T X^T X \phi_i = 0,\, 1\le i\le j-1 \\ \psi^T Y^T Y \psi_i = 0,\, 1\le i\le j-1}} cos\angle(\mathbf{X}\phi, \mathbf{Y}\psi)$$

$$\lambda_j = cos\angle(\mathbf{X}\phi_j, \mathbf{Y}\psi_j) \quad j = 1, \cdots, p$$

Lemma 4.1.2. *Let* $\mathbf{S_x} = \mathbf{X}^T\mathbf{X}$, $\mathbf{S_y} = \mathbf{Y}^T\mathbf{Y}$, $\mathbf{S_{xy}} = \mathbf{X}^T\mathbf{Y}$ *and* $\mathbf{S_x^{-\frac{1}{2}} S_{xy} S_y^{-\frac{1}{2}}} = \mathbf{UDV}^\top$ *be the singular value decomposition. Then* $\mathbf{\Phi} = \mathbf{S_x^{-\frac{1}{2}} U}$, $\mathbf{\Psi} = \mathbf{S_y^{-\frac{1}{2}} V}$, *and* $\mathbf{\Lambda} = \mathbf{D}$ *where* $\mathbf{\Lambda} = diag(\lambda_1, \cdots, \lambda_p)$.

Lemma 4.1.2 shows that CCA can be solved by first computing the whitening matrices $(\mathbf{X}^T\mathbf{X})^{-\frac{1}{2}}, (\mathbf{Y}^T\mathbf{Y})^{-\frac{1}{2}}$ and then perform a SVD on the whitened covariance matrix $(\mathbf{X}^T\mathbf{X})^{-\frac{1}{2}}\mathbf{X}^T\mathbf{Y}(\mathbf{Y}^T\mathbf{Y})^{-\frac{1}{2}}$. This classical algorithm is feasible and accurate when the data matrices are small but it can be slow and numerically unstable for large scale datasets which are common in modern Natural Language Processing (large corpora [17, 19]) and multi-view learning (abandunt and inexpensive unlabeled data [32]) applications.

Throughout the paper, we call the step of orthonormalizing the columns of $\mathbf{X}$ and $\mathbf{Y}$ **whitening step**. The computational complexity of the classical algorithm is dominated by the whitening step. There are two major bottlenecks,

- Huge matrix multiplication $\mathbf{X}^T\mathbf{X}, \mathbf{Y}^T\mathbf{Y}$ to obtain $\mathbf{S_x}, \mathbf{S_y}$ with computational complexity $O(np_1^2 + np_2^2)$ for general dense $\mathbf{X}$ and $\mathbf{Y}$.
- Large matrix decomposition to compute $\mathbf{S_x}^{-\frac{1}{2}}$ and $\mathbf{S_y}^{-\frac{1}{2}}$ with computational complexity $O(p_1^3 + p_2^3)$ (Even when $\mathbf{X}$ and $\mathbf{Y}$ are sparse, $\mathbf{S_x}, \mathbf{S_y}$ are not necessarily sparse)

Remark 4.1.3. The whitening step dominates because in the SVD step, when applying power iterations, instead of directly computing $\mathbf{S_x}^{-\frac{1}{2}}\mathbf{X}^T\mathbf{Y}\mathbf{S_y}^{-\frac{1}{2}}$, we only need to multiply $\mathbf{S_x}^{-\frac{1}{2}}\mathbf{X}^T\mathbf{Y}\mathbf{S_y}^{-\frac{1}{2}}$ with a thin matrix, which can be efficiently achieved by successively multiplying each huge matrix with the thin matrix.

Remark 4.1.4. Another classical algorithm (built-in function in Matlab) introduced in [9] uses a different way of whitening. It first carrys out a QR decomposition, $\mathbf{X} = \mathbf{Q}_x\mathbf{R}_x$ and $\mathbf{Y} = \mathbf{Q}_y\mathbf{R}_y$ and then performs a SVD on $\mathbf{Q}_x^T\mathbf{Q}_y$, which has the same computational complexity $O(np_1^2 + np_2^2)$ as the algorithm indicated by Lemma 4.1.2. However, it is difficult to exploit sparsity in QR factorization while $\mathbf{X}^T\mathbf{X}, \mathbf{Y}^T\mathbf{Y}$ can be efficiently computed when $\mathbf{X}$ and $\mathbf{Y}$ are sparse.

Besides computational issues, extra $O(p_1^2+p_2^2)$ space is needed to store two whitening matrices $\mathbf{S_x}^{-\frac{1}{2}}$ and $\mathbf{S_y}^{-\frac{1}{2}}$ (typically dense). In high dimensional applications where the number of features is huge, this can be another bottleneck considering the capacity of RAM of personal desktops (10-20 GB). In large distributed storage systems, the extra required space might incur heavy communication cost.

Therefore, it is natural to ask: is there a scalable algorithm that avoids inverting a huge matrix and multiplying two huge matrices? Is it memory efficient? Or even more ambitiously, is there an online algorithm that generates decent approximation given a fixed computational power (e.g. CPU time, FLOP)?

4.1.2 Related Work

Scalability begins to play an increasingly important role in modern machine learning applications and draws more and more attention. Recently lots of promising progress emerged in the literature concerning randomized algorithms for large scale matrix approximations, SVD, and Principal Component Analysis [48, 38, 57, 28]. Unfortunately, these techniques does not directly solve CCA due to the whitening step. Several authors have tried to devise a scalable CCA algorithm. [5] proposed an efficient approach for CCA between two tall and thin matrices ($p_1, p_2 \ll n$) harnessing the recently developed tools, *Subsampled Randomized Hadamard Transform*, which only subsampled a small proportion of the n data points to approximate the matrix product. However, when the size of the features, p_1 and p_2, are large, the sampling scheme does not work. Later, [40] consider sparse design matrices and formulate CCA as iterative least squares, where in each iteration a fast regression algorithm that exploits sparsity is applied.

Another related line of research considers stochastic optimization algorithms for PCA [3, 45, 8], which dates back to [46]. Compared with batch algorithms, they converge faster and have better generalization property. Further, these stochastic algorithms can be applied to streaming setting where data comes sequentially (one pass or several pass) without being stored. As mentioned in [3], stochastic optimization algorithm for CCA is more challenging because of the whitening step and remains an open problem.

4.1.3 Main Contribution

The main contribution of this paper is to directly tackle CCA as a nonconvex optimization problem and propose a novel Augmented Approximate Gradient (*AppGrad*) scheme and its stochastic variant for finding the top k dimensional canonical subspace. Its advantages over state-of-art CCA algorithms are three folds. *Firstly*, *AppGrad* scheme only involves large matrix multiplying a thin matrix of width k and small matrix decomposition of dimension $k \times k$, and therefore to some extent is free from the two bottlenecks. It also benefits if $\mathbf{X}$ and $\mathbf{Y}$ are sparse while classical algorithm still needs to invert the dense matrices $\mathbf{X}^T\mathbf{X}$ and $\mathbf{Y}^T\mathbf{Y}$. *Secondly*, *AppGrad* achieves optimal storage complexity $O(k(p_1 + p_2))$, the space necessary to store the output, compared with classical algorithms which usually require $O(p_1^2 + p_2^2)$ for storing the whitening matrices. *Thirdly*, the stochastic (online) variant of *AppGrad* is especially efficient for large scale datasets if moderate accuracy is desired. It is well-suited to the case when computational resources are limited or data comes as a stream. To the best of our knowledge, it is the first stochastic algorithm for CCA, which gives an affirmative answer to a question left open in [3].

The rest of the paper is organized as follows. In next section, we describe the proposed *AppGrad* scheme in detail and establish its convergence result. Section 3 extends the algorithm to stochastic setting. Extensive real data experiments are presented in section 4. Concluding remarks and future work are summarized in section 5. Proof of the main theorem is relegated to the appendix.

4.2 Algorithm

For simplicity, we first focus on the leading canonical pair (ϕ_1, ψ_1) to motivate the proposed algorithms. Results for general scenario can be obtained in the same manner and will be briefly discussed in the later part of this section.

4.2.1 An Optimization Perspective

Throughout the paper, we assume $\mathbf{X}$ and $\mathbf{Y}$ are of full rank. We use $\|\cdot\|$ for L_2 norm. $\forall\, u \in \mathbb{R}^{p_1},\ v \in \mathbb{R}^{p_2}$, we define $\|u\|_x = (u^T\mathbf{X}^T\mathbf{X}u)^{\frac{1}{2}}$ and $\|v\|_y = (v^T\mathbf{Y}^T\mathbf{Y}v)^{\frac{1}{2}}$, which are norms induced by $\mathbf{X}$ and $\mathbf{Y}$. For the rest of the paper, we will use $(u^T\mathbf{X}^T\mathbf{X}u)^{\frac{1}{2}}, (v^T\mathbf{Y}^T\mathbf{Y}v)^{\frac{1}{2}}$ and their shorthands interchangeably.

To begin with, we recast CCA as an optimization problem [26].

Algorithm 7 CCA via Alternating Least Squares

```
**Input:** Data matrix $\mathbf{X} \in \mathcal{R}^{n\times p_1}, \mathbf{Y} \in \mathcal{R}^{n\times p_2}$ and initialization $(\phi^0, \psi^0)$
**Output :**$(\phi_{\text{ALS}}, \psi_{\text{ALS}})$
**repeat**
    $\phi^{t+1} = \arg\min_{\phi} \frac{1}{2}\|\mathbf{X}\phi - \mathbf{Y}\psi^t\|^2 = \mathbf{S_x^{-1}S_{xy}}\psi^t$
    $\phi^{t+1} = \phi^{t+1}/\|\phi^{t+1}\|_x$
    $\psi^{t+1} = \arg\min_{\psi} \frac{1}{2}\|\mathbf{Y}\psi - \mathbf{X}\phi^t\|^2 = \mathbf{S_y^{-1}S_{yx}}\phi^t$
    $\psi^{t+1} = \psi^{t+1}/\|\psi^{t+1}\|_y$
**until** convergence
```

Lemma 4.2.1. (ϕ_1, ψ_1) *is the solution of*

$$\begin{aligned} &\min \frac{1}{2}\|\mathbf{X}\phi - \mathbf{Y}\psi\|^2 \\ &\textit{subject to} \quad \phi^T\mathbf{X}^T\mathbf{X}\phi = 1,\, \psi^T\mathbf{Y}^T\mathbf{Y}\psi = 1 \end{aligned} \tag{4.2.1}$$

Although (4.2.1) is a nonconvex optimization problem (due to the nonconvex constraint), [26] showed that an alternating minimization strategy (Algorithm 7), or rather iterative least squares, actually converges to the leading canonical pair. However, each update $\phi^{t+1} = \mathbf{S_x^{-1}S_{xy}}\psi^t$ is computationally intensive. Essentially, the alternating least squares acts like a second order method, which is usually recognized to be inefficient for large-scale problems, especially when current estimate is not close enough to the optimum. Therefore, it is natural to ask: is there a valid first order method that solves (4.2.1)? Heuristics borrowed from convex optimization literature give rise to a projected gradient scheme summarized in Algorithm 8. Instead of completely solving a least squares in

Algorithm 8 CCA via Naive Gradient Descent

```
**Input:** Data matrix $\mathbf{X} \in \mathcal{R}^{n \times p_1}, \mathbf{Y} \in \mathcal{R}^{n \times p_2}$, initialization $(\phi^0, \psi^0)$, step size $\eta_1, \eta_2$
**Output :** NAN (incorrect algorithm)
**repeat**
    $\phi^{t+1} = \phi^t - \eta_1 \mathbf{X}^T (\mathbf{X}\phi^t - \mathbf{Y}\psi^t)$
    $\phi^{t+1} = \phi^{t+1} / \|\phi^{t+1}\|_x$
    $\psi^{t+1} = \psi^t - \eta_2 \mathbf{Y}^T (\mathbf{Y}\psi^t - \mathbf{X}\phi^t)$
    $\psi^{t+1} = \psi^{t+1} / \|\psi^{t+1}\|_y$
**until** convergence
```

each iterate, a single gradient step of (4.2.1) is performed and then project back to the constrained domain, which avoids inverting a huge matrix. Unfortunately, the following proposition demonstrates that Algorithm 8 fails to converge to the leading canonical pair.

Proposition 4.2.2. *If leading canonical correlation $\lambda_1 \neq 1$ and ϕ_1, ψ_1 are not eigenvectors of $\mathbf{S_x}, \mathbf{S_y}$, $\forall \eta_1, \eta_2 > 0$, the leading canonical pair (ϕ_1, ψ_1) is not a fixed point of the naive gradient scheme in Algorithm 8. Therefore, the algorithm does not converge to (ϕ_1, ψ_1).*

Proof of Proposition 4.2.2. The proof is similar to the proof of Proposition 4.2.5 and we leave out the details here. □

Example 4.2.3 (example for Proposition 4.2.2). Let

$$\mathbf{X} = \begin{pmatrix} 1 & 5 \\ 2 & -6 \\ 3 & 7 \\ 4 & -8 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 9 & 1 \\ 10 & -1 \\ 11 & -1 \\ 12 & 1 \end{pmatrix}$$

The CCA directions for $\mathbf{X}$ and $\mathbf{Y}$ are:

$$\phi = \begin{pmatrix} 0.1871 & 0.0234 \\ 0.0099 & 0.0766 \end{pmatrix}, \psi = \begin{pmatrix} 0.0473 & -0.0010 \\ -0.0104 & -0.4999 \end{pmatrix}$$

We can verify that

$$\phi^\top \mathbf{X}^\top \mathbf{X} \phi = \psi^\top \mathbf{Y}^\top \mathbf{Y} \psi = \mathbf{I}$$

$$\phi^\top \mathbf{X}^\top \mathbf{Y} \psi = \begin{pmatrix} 0.9585 & 0 \\ 0 & 0.1553 \end{pmatrix}$$

Let's check the first pair of CCA directions

$$\phi_1, = \begin{pmatrix} 0.1871 \\ 0.0099 \end{pmatrix}, \psi_1 = \begin{pmatrix} 0.0473 \\ -0.0104 \end{pmatrix}$$

is not a fixed point of the naive projected gradient update:

For naive projected gradient update, let

$$\phi^t = \begin{pmatrix} 0.1871 \\ 0.0099 \end{pmatrix}, \psi^t = \begin{pmatrix} 0.0473 \\ -0.0104 \end{pmatrix}$$

, follow the update rule,

$$\phi^{t+1} = \begin{pmatrix} 0.1871 \\ 0.0099 \end{pmatrix} - s_x\left(\mathbf{X}^\top\mathbf{X}\begin{pmatrix} 0.1871 \\ 0.0099 \end{pmatrix} - \mathbf{X}^\top\mathbf{Y}\begin{pmatrix} 0.0473 \\ -0.0104 \end{pmatrix}\right)$$

$$= \begin{pmatrix} 0.1871 \\ 0.0099 \end{pmatrix} + s_x\begin{pmatrix} -0.2318 \\ 0.0786 \end{pmatrix}$$

Therefore, ϕ^{t+1} isn't parallel to ϕ^t and after the project step (which is essentially a rescale) still $\phi^{t+1} \neq \phi^t$.

The failure of Algorithm 8 is due to the nonconvex nature of (4.2.1). Although every gradient step decreases the objective function, this property no longer persists after projecting to its nonconvex domain $\left\{(\phi, \psi) \,|\, \phi^T\mathbf{X}^T\mathbf{X}\phi = 1, \, \psi^T\mathbf{Y}^T\mathbf{Y}\psi = 1\right\}$ (the normalization step). On the contrary, decreases triggered by gradient descent is always maintained if projecting to a convex region.

Remark 4.2.4. For all the algorithms presented in this section, during the update of ψ^{t+1}, ϕ^t can be replaced by ϕ^{t+1}, simply the difference between gradient descent and block coordinate descent.

4.2.2 *AppGrad* Scheme

As a remedy, we propose a novel Augmented Approximate Gradient (*AppGrad*) scheme summarized in Algorithm 9. It inherits the convergence guarantee of alternating least squares as well as the scalability and memory efficiency of first order methods, which only involves matrix-vector multiplication and only requires $O(p_1 + p_2)$ extra space.

Algorithm 9 CCA via AppGrad

```
Input: Data matrix $\mathbf{X} \in \mathcal{R}^{n\times p_1}, \mathbf{Y} \in \mathcal{R}^{n\times p_2}$, initialization $(\phi^0, \psi^0, \widetilde{\phi}^0, \widetilde{\psi}^0)$, step size $\eta_1, \eta_2$
Output: $(\phi_{\text{AG}}, \psi_{\text{AG}}, \widetilde{\phi}_{\text{AG}}, \widetilde{\psi}_{\text{AG}})$
repeat
    $\widetilde{\phi}^{t+1} = \widetilde{\phi}^t - \eta_1 \mathbf{X}^T(\mathbf{X}\widetilde{\phi}^t - \mathbf{Y}\psi^t)$
    $\phi^{t+1} = \widetilde{\phi}^{t+1} / \|\widetilde{\phi}^{t+1}\|_x$
    $\widetilde{\psi}^{t+1} = \widetilde{\psi}^t - \eta_2 \mathbf{Y}^T(\mathbf{Y}\widetilde{\psi}^t - \mathbf{X}\phi^t)$
    $\psi^{t+1} = \widetilde{\psi}^{t+1} / \|\widetilde{\psi}^{t+1}\|_y$
until convergence
```

AppGrad seems unnatural at first sight but has some nice intuitions behind as we will discuss later. The differences and similarities between these algorithms are subtle but crucial. Compared with the naive gradient descent, we introduce two auxiliary variables $(\widetilde{\phi}^t, \widetilde{\psi}^t)$, an unnormalized version of (ϕ^t, ψ^t). During each iterate, we keep updating $\widetilde{\phi}^t$ and $\widetilde{\psi}^t$ without scaling them to have unit norm, which in turn produces the 'correct' normalized counterpart, (ϕ^t, ψ^t). It turns out that $(\phi_1, \psi_1, \lambda_1\phi_1, \lambda_1\psi_1)$ is a fixed point of the dynamic system $\{(\phi^t, \psi^t, \widetilde{\phi}^t, \widetilde{\psi}^t)\}_{t=0}^{\infty}$.

Proposition 4.2.5. *$\forall \lambda_i > 0$, let $\widetilde{\phi}_i = \lambda_i\phi_i, \widetilde{\psi}_i = \lambda_i\psi_i$, then $(\phi_i, \psi_i, \widetilde{\phi}_i, \widetilde{\psi}_i)$ are the fixed points of AppGrad scheme.*

To prove the proposition, we need the following lemma that characterizes the relations among some key quantities.

Lemma 4.2.6. $\mathbf{S_{xy}} = \mathbf{S_x}\mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Psi}^T\mathbf{S_y}$

Proof of Lemma 4.2.6. By Lemma 4.1.2, $\mathbf{S_x^{-\frac{1}{2}}}\mathbf{S_{xy}}\mathbf{S_y^{-\frac{1}{2}}} = \mathbf{UDV^T}$, where $\mathbf{U} = \mathbf{S_x^{\frac{1}{2}}}\mathbf{\Phi}$, $\mathbf{V} = \mathbf{S_y^{\frac{1}{2}}}\mathbf{\Psi}$ and $\mathbf{D} = \mathbf{\Lambda}$. Then we have $\mathbf{S_{xy}} = \mathbf{S_x^{\frac{1}{2}}}\mathbf{UDV}^T\mathbf{S_y^{\frac{1}{2}}} = \mathbf{S_x}\mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Psi}^T\mathbf{S_y}$. □

Proof of Proposition 4.2.5. Substitute $(\phi^t, \psi^t, \widetilde{\phi}^t, \widetilde{\psi}^t) = (\phi_i, \psi_i, \widetilde{\phi}_i, \widetilde{\psi}_i)$ into the iterative formula in Algorithm 9.

$$\begin{aligned}
\widetilde{\phi}^{t+1} &= \widetilde{\phi}_i - \eta_1(\mathbf{S_x}\widetilde{\phi}_i - \mathbf{S_{xy}}\psi_i) \\
&= \widetilde{\phi}_i - \eta_1(\mathbf{S_x}\widetilde{\phi}_i - \mathbf{S_x}\mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Psi}^T\mathbf{S_y}\psi_i) \\
&= \widetilde{\phi}_i - \eta_1(\mathbf{S_x}\widetilde{\phi}_i - \lambda_i\mathbf{S_x}\phi_i) \\
&= \widetilde{\phi}_i
\end{aligned}$$

The second equality is direct application of Lemma 4.2.6. The third equality is due to the fact that $\mathbf{\Psi}^T\mathbf{S_y}\mathbf{\Psi} = I_p$. Then,

$$\phi^{t+1} = \widetilde{\phi}_i/\|\widetilde{\phi}_i\|_x = \widetilde{\phi}_i/\lambda_i = \phi_i$$

Therefore $(\widetilde{\phi}^{t+1}, \phi^{t+1}) = (\widetilde{\phi}^t, \phi^t) = (\widetilde{\phi}_i, \phi_i)$. A symmetric argument will show that $(\widetilde{\psi}^{t+1}, \psi^{t+1}) = (\widetilde{\psi}^t, \psi^t) = (\widetilde{\psi}_i, \psi_i)$, which completes the proof. □

The connection between *AppGrad* and alternating minimization strategy is not instaneous. Intuitively, when (ϕ^t, ψ^t) is not close to (ϕ_1, ψ_1), solving the least squares completely as carried out in Algorithm 7 is a waste of computational power (informally by regarding it as a second order method, the Newton Step has fast convergence only when current estimate is close to the optimum). Instead of solving a sequence of possibly irrelevant least squares, the following lemma shows that *AppGrad* directly targets at the least squares that involves the leading canonical pair.

Lemma 4.2.7. *Let* (ϕ_1, ψ_1) *be the leading canonical pair and* $(\tilde{\phi}_1, \tilde{\psi}_1) = \lambda_1(\phi_1, \psi_1)$. *Then,*

$$\begin{aligned} \widetilde{\phi}_1 &= \arg\min_{\phi} \frac{1}{2}\|\mathbf{X}\phi - \mathbf{Y}\psi_1\|^2 \\ \widetilde{\psi}_1 &= \arg\min_{\psi} \frac{1}{2}\|\mathbf{Y}\psi - \mathbf{X}\phi_1\|^2 \end{aligned} \tag{4.2.2}$$

Proof of Lemma 4.2.7. Let $\phi^* = \arg\min_{\phi} \frac{1}{2}\|\mathbf{X}\phi - \mathbf{Y}\psi_1\|^2$, by optimality condition, $\mathbf{S_x}\phi^* = \mathbf{S_{xy}}\psi_1$. Apply Lemma 4.2.6,

$$\phi^* = \mathbf{S_x}^{-1}\mathbf{S_x}\mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Psi}^T\mathbf{S_y}\psi_1 = \lambda_1\phi_1 = \widetilde{\phi}_1$$

Similar argument gives $\psi^* = \widetilde{\psi}_1$ □

Lemma 4.2.7 characterizes the relationship between leading canonical pair (ϕ_1, ψ_1) and its unnormalized counterpart $(\tilde{\phi}_1, \tilde{\psi}_1)$, which sheds some insight on how *AppGrad* works. The intuition is that (ϕ^t, ψ^t) and $(\widetilde{\phi}^t, \widetilde{\psi}^t)$ are current estimations of (ϕ_1, ψ_1) and $(\widetilde{\phi}_1, \widetilde{\psi}_1)$, and the updates of $(\widetilde{\phi}^{t+1}, \widetilde{\psi}^{t+1})$ in Algorithm 9 are actually gradient steps of the least squares in (4.2.2), with the unknown truth (ϕ_1, ψ_1) approximated by (ϕ^t, ψ^t). In terms of mathematics,

$$\begin{aligned} \widetilde{\phi}^{t+1} &= \widetilde{\phi}^t - \eta_1\mathbf{X}^T(\mathbf{X}\widetilde{\phi}^t - \mathbf{Y}\psi^t) \\ &\approx \widetilde{\phi}^t - \eta_1\mathbf{X}^T(\mathbf{X}\widetilde{\phi}^t - \mathbf{Y}\psi_1) \\ &= \widetilde{\phi}^t - \eta_1\nabla_{\phi}\frac{1}{2}\|\mathbf{X}\phi - \mathbf{Y}\psi_1\|^2|_{\phi=\widetilde{\phi}^t} \end{aligned} \tag{4.2.3}$$

The normalization step in Algorithm 9 corresponds to generating new approximations of (ϕ_1, ψ_1), namely (ϕ^{t+1}, ψ^{t+1}), using the updated $(\widetilde{\phi}^{t+1}, \widetilde{\psi}^{t+1})$ through the relationship $(\phi_1, \psi_1) = (\widetilde{\phi}_1/\|\widetilde{\phi}_1\|_x, \widetilde{\psi}_1/\|\widetilde{\psi}_1\|_y)$.

As equation (4.2.3) suggests, *AppGrad* actually uses approximate gradients. We show that when current estimates enter a neighborhood of the true canonical pair, this approximate gradient scheme is contractive. Without loss of generality, assume $\lambda_{max}(\mathbf{X}^T\mathbf{X})$, $\lambda_{max}(\mathbf{Y}^T\mathbf{Y}) \leq 1$ and $\exists\, L > 1$ such that $\lambda_{min}(\mathbf{X}^T\mathbf{X})$, $\lambda_{min}(\mathbf{Y}^T\mathbf{Y}) \geq L^{-1}$, where λ_{min}, λ_{max} are smallest and largest eigenvalues. It is equivalent to condition number is bounded by L since we can always normalize $\mathbf{X}$ and $\mathbf{Y}$.

Theorem 4.2.8. *Assume* $\Delta\lambda = \lambda_1 - \lambda_2 > 0$, *Set* $\eta_1 = \eta_2 = \Delta\lambda/6C\lambda_1, \forall C > 1$. *Let* $\delta = \frac{(C-1)(\Delta\lambda)^2}{6C^2L\lambda_1^2}$, $\Delta\widetilde{\phi}^t = \widetilde{\phi}^t - \widetilde{\phi}_1, \Delta\widetilde{\psi}^t = \widetilde{\psi}^t - \widetilde{\psi}_1$. *Then when* $\|\Delta\widetilde{\phi}^t\|^2$, $\|\Delta\widetilde{\psi}^t\|^2 \leq \lambda_1\Delta\lambda/2$ *, AppGrad is* δ*-contractive in the sense that*

$$\|\Delta\widetilde{\phi}^{t+k}\|^2 + \|\Delta\widetilde{\psi}^{t+k}\|^2 \leq \Big(1 - \delta\Big)^k \Big(\|\Delta\widetilde{\phi}^t\|^2 + \|\Delta\widetilde{\psi}^t\|^2\Big)$$

Remark 4.2.9. The theorem reveals that the larger is the eigengap $\Delta\lambda$, the broader is the basin of attraction. We didn't try to optimize the conditions above and actually as shown in the experiments, a randomized initialization always suffices to capture most of the correlations.

4.2.3 General Rank-k Case

Following the spirit of rank-one case, *AppGrad* can be easily generalized to compute the top k dimesional canonical subspace as summarized in Algorithm 10. The only dif-

Algorithm 10 CCA via *AppGrad* (Rank-k)

```
Input: Data matrix $\mathbf{X} \in \mathcal{R}^{n\times p_1}, \mathbf{Y} \in \mathcal{R}^{n\times p_2}$, initialization $(\mathbf{\Phi}^0, \mathbf{\Psi}^0\widetilde{\mathbf{\Phi}}^0, \widetilde{\mathbf{\Psi}}^0)$, step size $\eta_1, \eta_2$
Output : $(\mathbf{\Phi}_{\text{AG}}, \mathbf{\Psi}_{\text{AG}}, \widetilde{\mathbf{\Phi}}_{\text{AG}}, \widetilde{\mathbf{\Psi}}_{\text{AG}})$
repeat
  $\widetilde{\mathbf{\Phi}}^{t+1} = \widetilde{\mathbf{\Phi}}^t - \eta_1 \mathbf{X}^T(\mathbf{X}\widetilde{\mathbf{\Phi}}^t - \mathbf{Y}\mathbf{\Psi}^t)$
  SVD: $(\widetilde{\mathbf{\Phi}}^{t+1})^T\mathbf{X}^T\mathbf{X}\widetilde{\mathbf{\Phi}}^{t+1} = \mathbf{U}_x\mathbf{D}_x\mathbf{U}_x^T$
  $\mathbf{\Phi}^{t+1} = \widetilde{\mathbf{\Phi}}^{t+1}\mathbf{U}_x\mathbf{D}_x^{-\frac{1}{2}}\mathbf{U}_x^T$
  $\widetilde{\mathbf{\Psi}}^{t+1} = \widetilde{\mathbf{\Psi}}^t - \eta_2 \mathbf{Y}^T(\mathbf{Y}\widetilde{\mathbf{\Psi}}^t - \mathbf{X}\mathbf{\Phi}^t)$
  SVD: $(\widetilde{\mathbf{\Psi}}^{t+1})^T\mathbf{Y}^T\mathbf{Y}\widetilde{\mathbf{\Psi}}^{t+1} = \mathbf{U}_y\mathbf{D}_y\mathbf{U}_y^T$
  $\mathbf{\Psi}^{t+1} = \widetilde{\mathbf{\Psi}}^{t+1}\mathbf{U}_y\mathbf{D}_y^{-\frac{1}{2}}\mathbf{U}_y^T$
until convergence
```

ference is that the original scalar normalization is replaced by its matrix counterpart, that is to multiply the inverse of the square root matrix $\mathbf{\Phi}^{t+1} = \widetilde{\mathbf{\Phi}}^{t+1}\mathbf{U}_x\mathbf{D}_x^{-\frac{1}{2}}\mathbf{U}_x^T$, ensuring that $(\mathbf{\Phi}^{t+1})^T\mathbf{X}^T\mathbf{X}\mathbf{\Phi}^{t+1} = \mathbf{I}_k$.

Notice that the gradient step only involves a large matrix multiplying a thin matrix of width k and the SVD is performed on a small $k \times k$ matrix. Therefore, the computational complexity per iteration is dominated by the gradient step, of order $O(n(p_1 + p_2)k)$. The cost will be further reduced when the data matrices $(\mathbf{X}, \mathbf{Y})$ are sparse.

Compared with classical spectral agorithm which first whitens the data matrices and then performs a SVD on the whitened covariance matrix, *AppGrad* actually merges these two steps together. This is the key of its efficiency. In a high level, whitening the whole data matrix is not necessary and we only want to whiten the directions than contains the leading CCA subspace. However, these directions are unknown and therefore for two-step procedures, whitening the whole data matrix is unavoidable. Instead, *AppGrad* tries to identify (gradient step) and whiten (normalization step) these directions simultaneously. In this way, every normalization step is only performed on the potential leading k dimensional CCA subspace and therefore only deals with a small $k \times k$ matrix.

Parallel results of Lemma 4.2.1, Proposition 4.2.2, Proposition 4.2.5, Lemma 4.2.7 for this general scenario can be established in a similar manner. Here, to make Algorithm 10 more clear, we state the fixed point result without proof.

Proposition 4.2.10. *Let $\mathbf{\Lambda}_k = diag(\lambda_1, \cdots, \lambda_k)$ be the diagonal matrix of top k canonical correlations and let $\mathbf{\Phi}_k = (\phi_1, \cdots, \phi_k), \mathbf{\Psi}_k = (\phi_1, \cdots, \phi_k)$ be the top k CCA vectors. Also denote $\widetilde{\mathbf{\Phi}}_k = \mathbf{\Phi}_k \mathbf{\Lambda}_k$ and $\widetilde{\mathbf{\Psi}}_k = \mathbf{\Psi}_k \mathbf{\Lambda}_k$. Then for any $k \times k$ orthogonal matrix $\mathbf{Q}$, $(\mathbf{\Phi}_k, \mathbf{\Psi}_k, \widetilde{\mathbf{\Phi}}_k, \widetilde{\mathbf{\Psi}}_k)\mathbf{Q}$ is a fixed point of AppGrad scheme.*

The top k dimensional canonical subspace is identifiable up to a rotation matrix and Proposition 4.2.10 shows that every optimum is a fixed point of *AppGrad* scheme.

4.2.4 Kernelization

Sometimes CCA is restricted because of its linearity and kernel CCA offers an alternative by projecting data into a high dimensional feature space. In this section, we show that *AppGrad* works for kernel CCA as well. Let $K_{\mathcal{X}}(\cdot, \cdot)$ and $K_{\mathcal{Y}}(\cdot, \cdot)$ be Mercer kernels, then there exists feature mappings $f_{\mathcal{X}} : \mathcal{X} \to \mathcal{F}_{\mathcal{X}}$ and $f_{\mathcal{Y}} : \mathcal{Y} \to \mathcal{F}_{\mathcal{Y}}$ such that $K_{\mathcal{X}}(x_i, x_j) = \langle f_{\mathcal{X}}(x_i), f_{\mathcal{X}}(x_j)\rangle$ and $K_{\mathcal{Y}}(y_i, y_j) = \langle f_{\mathcal{Y}}(y_i), f_{\mathcal{Y}}(y_j)\rangle$. Let $\mathbf{F}_{\mathcal{X}} = (f_{\mathcal{X}}(x_1), \cdots, f_{\mathcal{X}}(x_n))^T$ and $\mathbf{F}_{\mathcal{Y}} = (f_{\mathcal{Y}}(y_1), \cdots, f_{\mathcal{Y}}(y_n))^T$ be the compact representation of the objects in the possibly infinite dimensional feature space. Since the top k dimensional canonical vectors lie in the space spaned by the features, say $\mathbf{\Phi}_k = \mathbf{F}_{\mathcal{X}}^T\mathbf{W}_{\mathcal{X}}$ and $\mathbf{\Psi}_k = \mathbf{F}_{\mathcal{Y}}^T\mathbf{W}_{\mathcal{Y}}$ for some $\mathbf{W}_{\mathcal{X}}, \mathbf{W}_{\mathcal{Y}} \in \mathbb{R}^{n\times k}$. Let $\mathbf{K}_{\mathcal{X}} = \mathbf{F}_{\mathcal{X}}\mathbf{F}_{\mathcal{X}}^T, \mathbf{K}_{\mathcal{Y}} = \mathbf{F}_{\mathcal{Y}}\mathbf{F}_{\mathcal{Y}}^T$ be the Gram matrices. Similar to Lemma 4.2.1, kernel CCA can be formulated as

$$\arg\max_{\mathbf{W}_{\mathcal{X}}, \mathbf{W}_{\mathcal{Y}}} \quad \|\mathbf{K}_{\mathcal{X}}\mathbf{W}_{\mathcal{X}} - \mathbf{K}_{\mathcal{Y}}\mathbf{W}_{\mathcal{Y}}\|_F^2$$

$$\text{subject to } \mathbf{W}_{\mathcal{X}}^T\mathbf{K}_{\mathcal{X}}\mathbf{K}_{\mathcal{X}}\mathbf{W}_{\mathcal{X}} = \mathbf{I}_k \quad \mathbf{W}_{\mathcal{Y}}^T\mathbf{K}_{\mathcal{Y}}\mathbf{K}_{\mathcal{Y}}\mathbf{W}_{\mathcal{Y}} = \mathbf{I}_k$$

Following the same logic as Proposition 4.2.10, a similar fixed point result can be proved. Therefore, Algorithm 10 can be directly applied to compute $\mathbf{W}_{\mathcal{X}}, \mathbf{W}_{\mathcal{Y}}$ by simply replacing $\mathbf{X}, \mathbf{Y}$ with $\mathbf{K}_{\mathcal{X}}, \mathbf{K}_{\mathcal{Y}}$.

4.3 Stochastic *AppGrad*

Recently, there is a growing interest in stochastic optimization which is shown to have better performance for large-scale learning problems [13, 50, 12]. Especially in the so-called ‘data laden regime’, where data is abundant and the bottleneck is runtime, stochastic optimization dominate batch algorithms both empirically and theoretically. Given these advantages, lots of efforts have been spent on developing stochastic algorithms for nonconvex spectral methods such as principal component analysis [46, 3, 45, 8]. Despite promising progress in PCA, as mentioned in [3], stochastic CCA is more challenging due

Algorithm 11 CCA via Stochastic *AppGrad* (Rank-k)

```
**Input:** Data matrix $\mathbf{X} \in \mathcal{R}^{n\times p_1}$, $\mathbf{Y} \in \mathcal{R}^{n\times p_2}$, initialization $(\mathbf{\Phi}^0, \mathbf{\Psi}^0, \widetilde{\mathbf{\Phi}}^0, \widetilde{\mathbf{\Psi}}^0)$, step size $\eta_{1t}, \eta_{2t}$, minibatch size $m$
**Output :** $(\mathbf{\Phi}_{\text{SAG}}, \mathbf{\Psi}_{\text{SAG}}, \widetilde{\mathbf{\Phi}}_{\text{SAG}}, \widetilde{\mathbf{\Psi}}_{\text{SAG}})$
**repeat**
    Randomly pick a subset $\mathcal{I} \subset \{1, 2, \cdots, n\}$ of size $m$
    $\widetilde{\mathbf{\Phi}}^{t+1} = \widetilde{\mathbf{\Phi}}^t - \eta_{1t}\mathbf{X}_{\mathcal{I}}^T(\mathbf{X}_{\mathcal{I}}\widetilde{\mathbf{\Phi}}^t - \mathbf{Y}_{\mathcal{I}}\mathbf{\Psi}^t)$
    SVD: $(\widetilde{\mathbf{\Phi}}^{t+1})^T\mathbf{X}_{\mathcal{I}}^T\mathbf{X}_{\mathcal{I}}\widetilde{\mathbf{\Phi}}^{t+1} = \mathbf{U}_x^T\mathbf{D}_x\mathbf{U}_x$
    $\mathbf{\Phi}^{t+1} = \widetilde{\mathbf{\Phi}}^{t+1}\mathbf{U}_x^T\mathbf{D}_x^{-\frac{1}{2}}\mathbf{U}_x$
    $\widetilde{\mathbf{\Psi}}^{t+1} = \widetilde{\mathbf{\Psi}}^t - \eta_{2t}\mathbf{Y}_{\mathcal{I}}^T(\mathbf{Y}_{\mathcal{I}}\widetilde{\mathbf{\Psi}}^t - \mathbf{X}_{\mathcal{I}}\mathbf{\Phi}^t)$
    SVD: $(\widetilde{\mathbf{\Psi}}^{t+1})^T\mathbf{Y}_{\mathcal{I}}^T\mathbf{Y}_{\mathcal{I}}\widetilde{\mathbf{\Psi}}^{t+1} = \mathbf{U}_y^T\mathbf{D}_y\mathbf{U}_y$
    $\mathbf{\Psi}^{t+1} = \widetilde{\mathbf{\Psi}}^{t+1}\mathbf{U}_y^T\mathbf{D}_y^{-\frac{1}{2}}\mathbf{U}_y$
**until** convergence
```

to the whitening step and remains an open problem .

As a gradient scheme, *AppGrad* naturally generalizes to the stochastic regime and we summarize it in Algorithm 11. Compared with batch algorithm, only a small subset of samples are used to compute the gradient, which reduces the computational cost per iteration from $O(n(p_1 + p_2)k)$ to $O(m(p_1 + p_2)k)$ ($m = |\mathcal{I}|$ is the size of the minibatch). Empirically, this makes stochastic *AppGrad* much faster than the batch version as we will see in the experiments. Also, for large scale applications when fully calculating the CCA subspace is prohibitive, stochastic *AppGrad* can generate a decent approximation given a fixed computational power, while other algorithms only give a one-shot estimate after the whole procedure is carried out completely. Moreover, when there is a generative model, as shown in [13], due to the tradeoff of statistical and numerical accuracy, fully solving an optimization problem is unnecessary since the statistical error will finally dominate. On the contrary, stochastic optimization directly targets at the generative model and therefore is more statistically efficient.

It is worth mentioning that the normalization step is accomplished using a sampled Gram matrix $\mathbf{X}_{\mathcal{I}}^T\mathbf{X}_{\mathcal{I}}$ and $\mathbf{Y}_{\mathcal{I}}^T\mathbf{Y}_{\mathcal{I}}$. A key observation is that when $|\mathcal{I}| = m = O(k)$, $(\widetilde{\mathbf{\Phi}}^{t+1})^T\mathbf{X}_{\mathcal{I}}^T\mathbf{X}_{\mathcal{I}}\widetilde{\mathbf{\Phi}}^{t+1} \approx (\widetilde{\mathbf{\Phi}}^{t+1})^T\mathbf{X}^T\mathbf{X}\widetilde{\mathbf{\Phi}}^{t+1}$ using standard concentration inequality, because the matrix we want to approximate $(\widetilde{\mathbf{\Phi}}^{t+1})^T\mathbf{X}^T\mathbf{X}\widetilde{\mathbf{\Phi}}^{t+1}$ is a $k \times k$ matrix, while generally $O(p)$ sample is needed to have $\mathbf{X}_{\mathcal{I}}^T\mathbf{X}_{\mathcal{I}} \approx \mathbf{X}^T\mathbf{X}$. As we have argued in previous section, this bonus is a byproduct of the fact that *AppGrad* tries to identify and whiten the directions that contains the CCA subspace simultaneously, or else $O(p)$ samples are necessary for whitening the whole data matrices.

4.4 Experiments

In this section, we present experiments on four real datasets (Mediamill, MNIST, Penn Tree Bank, URL Reputation) to evaluate the effectiveness of the proposed algorithms for computing the top 20 (k=20) dimensional canonical subspace. A short summary of the datasets is in Table 4.1.

Table 4.1: Brief Summary of Datasets

DATASETS	DESCRIPTION	p_1	p_2	n
MEDIAMILL	IMAGE AND ITS LABELS	100	120	$30,000$
MNIST	LEFT AND RIGHT HALVES OF IMAGES	392	392	$60,000$
PENN TREE BANK	WORD CO-OCURRANCE	$10,000$	$10,000$	$500,000$
URL REPUTATION	HOST AND LEXICAL BASED FEATURES	$100,000$	$100,000$	$1,000,000$

4.4.1 Details of Datasets

Mediamill is an annotated video dataset from the Mediamill Challenge [51]. There are 43907 images in total, 30933 for training and 12914 for testing. Each image is a representative keyframe of a video shot annotated with 101 labels and consists of 120 features. CCA is performed to explore the correlation structure between the images and its labels.

MNIST is a database of handwritten digits. CCA is used to learn correlated representations between the left and right halves of the images. Each image has a width and height of 28 pixels and therefore, both views are 392 dimensional features. 60, 000 random samples are used for training and 10, 000 samples for testing.

Penn Tree Bank dataset is extracted from Wall Street Journal, which consists of 1.17

million tokens and a vocabulary size of $43,000$ [37]. CCA has been successfully used on this dataset to build low dimensional word embeddings [17, 19]. The task here is a CCA between words and their context. The rows of $\mathbf{X}$ matrix are indicator vectors of the current word and the rows of $\mathbf{Y}$ are indicators of the word behind. We consider 10, 000 most frequent words to avoid sample sparsity $(p_1 = p_2 = 10,000)$. Both the training set and testing set contains roughly $500,000$ samples.

URL Reputation dataset [41] is extracted from UCI machine learning repository. The dataset contains 2.4 million URLs each represented by 3.2 million features. For simplicity we only use the first $2,000,000$ samples. 38% of the features are host based features like WHOIS info, IP prefix and 62% are lexical based features like Hostname and Primary domain. We run a CCA between a subset of host based features and a subset of lexical based features. In both training and testing set, $\mathbf{X}$ and $\mathbf{Y}$ are of size $1,000,000 \times 100,000$.

4.4.2 Implementations

Evaluation Criterion: The evaluation criterion we use for the first three datasets (Mediamill, MNIST, Penn Tree Bank) is *Proportions of Correlations Captured* (PCC). To introduce this term, we first define *Total Correlations Captured* (TCC) between two matrices to be the sum of their canonical correlations as defined in Lemma 4.1.2. Then, for estimated top k dimensional canonical subspace $\widehat{\mathbf{\Phi}}_k, \widehat{\mathbf{\Psi}}_k$ and true leading k dimensional

CCA subspace $\mathbf{\Phi}_k, \mathbf{\Psi}_k$, PCC is defined as

$$\text{PCC} = \frac{\text{TCC}(\mathbf{X}\widehat{\mathbf{\Phi}}_k, \mathbf{Y}\widehat{\mathbf{\Psi}}_k)}{\text{TCC}(\mathbf{X}\mathbf{\Phi}_k, \mathbf{Y}\mathbf{\Psi}_k)}$$

Intuitively PCC characterizes how much correlation is captured by certain algorithm compared with the true CCA subspace. Therefore, the higher is PCC the better is the estimated CCA subspace. This criterion has some advantages. First, compared with $\|P_{\widehat{\mathbf{\Phi}}_k} - P_{\mathbf{\Phi}_k}\|$ (P_Ω is projection matrix of the column space of Ω), it is more natural and relevant considering that the goal of CCA is to capture most correlations between two data matrices. Second, when the eigengap $\Delta\lambda = \lambda_k - \lambda_{k+1}$ is not big enough, the top k dimensional CCA subspace is ill posed while the correlations captured is well defined.

We use the output of the standard spectral algorithms as the truth $(\mathbf{\Phi}_k, \mathbf{\Psi}_k)$ to calculate the denominator of PCC. However, for URL Reputation dataset, the number of samples and features are too large for the algorithm to compute the true CCA subspace in a reasonable amount of time and instead we compare the numerator $\text{TCC}(\mathbf{X}\widehat{\mathbf{\Phi}}_k, \mathbf{Y}\widehat{\mathbf{\Psi}}_k)$ (monotone w.r.t. PCC) for different algorithms.

Initialization We initialize $(\mathbf{\Phi}^0, \mathbf{\Psi}^0)$ by first drawing $i.i.d.$ samples from standard Gaussian distribution and then normalize such that $(\mathbf{\Phi}^0)^T \mathbf{S}_{\mathbf{x}} \mathbf{\Phi}^0 = I_k$ and $(\mathbf{\Psi}^0)^T \mathbf{S}_{\mathbf{y}} \mathbf{\Psi}^0 = I_k$

Stepsize For both *AppGrad* and stochastic *AppGrad*, a small part of the training set is held out and cross-validation is used to choose the step size adaptively.

Regularization For all the algorithms, a little regularization is added for numerical stability which means we replace Gram matrix $\mathbf{X}^T\mathbf{X}$ with $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$ for some small positive λ.

Oversampling Oversampling means when aiming for top k dimensional subspace, people usually computes top $k + l$ dimesional subspace from which a best k diemsional subspace is extracted. In practice, $l = 5 \sim 10$ suffices to improve the performance. We only do a oversampling of 5 in the URL dataset.

4.4.3 Summary of Results

For the first three datasets (Mediamill, MNIST, Penn Tree Bank), both in-sample and out-of-sample PCC are computed for *AppGrad* and Stochastic *AppGrad* as summarized in Figure4.1. As you can see, both algorithms nearly capture most of the correlations compared with the true CCA subspace and stochastic *AppGrad* consistently achieves same PCC with much less computational cost than its batch version. Moreover, the larger is the size of the data, the bigger advantage will stochastic *AppGrad* obtain. One thing to notice is that, as revealed in Mediamill dataset, out-of-sample PCC is not necessarily less than in-sample PCC because both denominator and numerator will change on the hold out set.

For URL Reputation dataset, as we mentioned earlier, classical algorithms fails on a typical desktop. The reason is that these algorithms only produce a one-shot estimate after the whole procedure is completed, which is usually prohibitive for huge datasets. In this scenario, the advantage of online algorithms like stochastic *AppGrad* becomes crucial. Further, the stochastic nature makes the algorithm cost-effective and generate

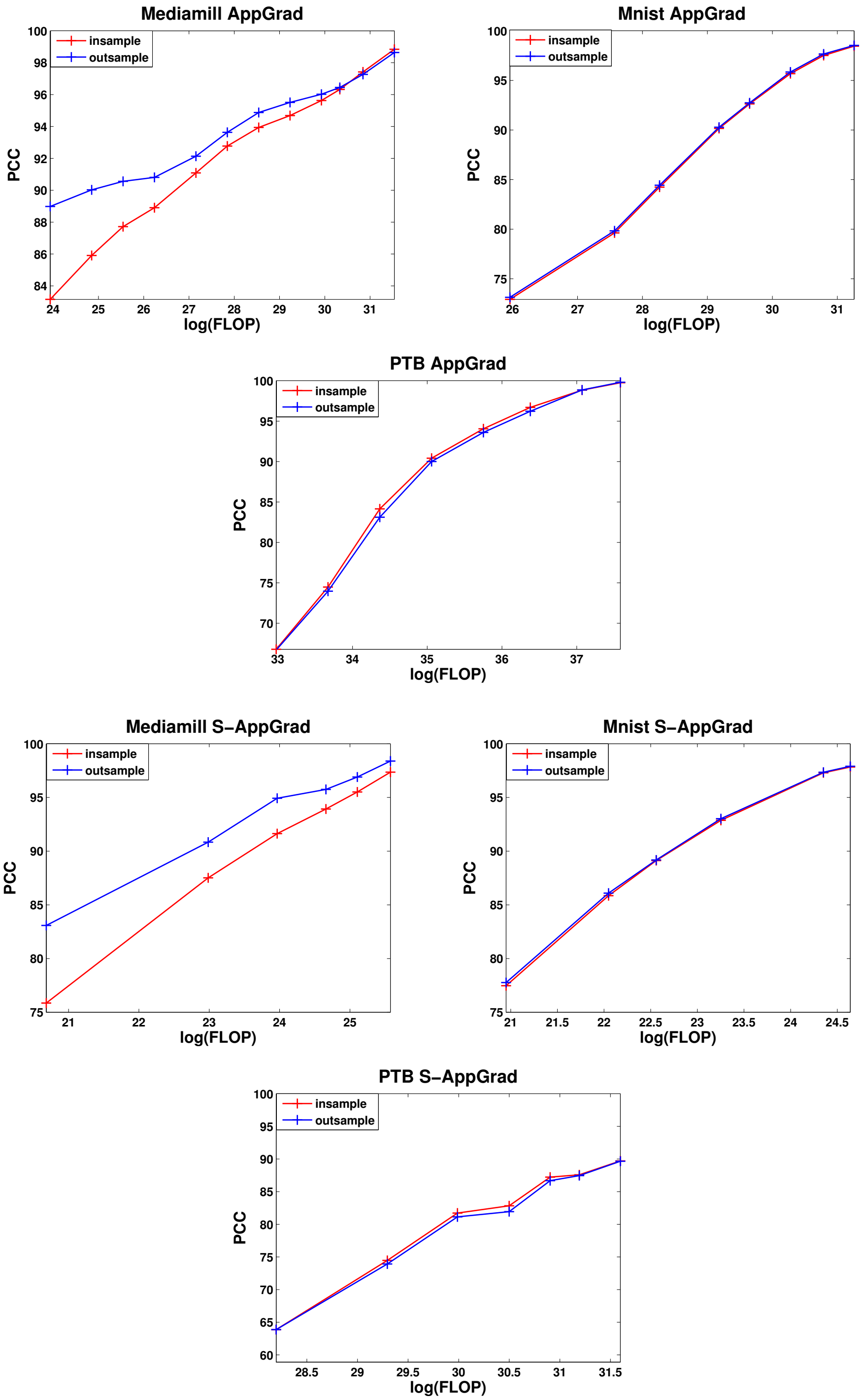

Figure 4.1: Proportion of Correlations Captured (PCC) by *AppGrad* and stochastic *AppGrad* on different datasets

decent approximations given fixed computational resources (e.g. FLOP). As revealed by Figure 4.2, as the number of iterations increases, stochastic *AppGrad* captures more and more correlations.

Since the true CCA subspaces for URL dataset is too slow to compute, we compare our algorithm with some naive heuristics which can be carried out efficiently in large scale and catches a reasonable amount of correlation. Below is a brief description of them.

- No whitening (NW-CCA): directly perform SVD on the unwhitened covariance matrix $\mathbf{X}^T\mathbf{Y}$. This strategy is also used in [56]
- Diagnoally whitening (DW-CCA) [40]: avoid inverting matrices by approximating $(\mathbf{X}^T\mathbf{X})^{-\frac{1}{2}}$ and $(\mathbf{Y}^T\mathbf{Y})^{-\frac{1}{2}}$ with $(\text{diag}(\mathbf{X}^T\mathbf{X}))^{-\frac{1}{2}}$ and $(\text{diag}(\mathbf{Y}^T\mathbf{Y}))^{-\frac{1}{2}}$.
- Whitening the leading m Principal Component Directions (PCA-CCA): First compute the leading m dimensional principal component subspace and project the data matrices $\mathbf{X}$ and $\mathbf{Y}$ to the subspace, denote them $\mathbf{U}_x$ and $\mathbf{U}_y$. Then compute the top k dimensional CCA subspace of the pair $(\mathbf{U}_x, \mathbf{U}_y)$. At last, transform the CCA subspace of $(\mathbf{U}_x, \mathbf{U}_y)$ back to the CCA subspace of orginal matrix pair $(\mathbf{X}, \mathbf{Y})$. Specifically for this example, we choose $m = 1200$ (log(FLOP)=35, dominating the computational cost of Stochastic *AppGrad*) .

Remark 4.4.1. For all the heuristics mentioned above, SVD and PCA steps are carried out using the randomized algorithms developed in [28]. For PCA-CCA, as the number of Principal Components (m) increases, more correlation will be captured but the computational cost will also increase. When $m = p$, PCA-CCA is reduced to the orginal CCA.

Essentially, all the heuristics are incorrect algorithms and try to approximately whiten the data matrices. As suggested by Figure 4.2, stochastic *AppGrad* significantly captures much more correlations.

4.5 Conclusions and Future Work

In this paper, we present a novel first order method, *AppGrad*, to tackle large scale CCA as a nonconvex optimization problem. This bottleneck-free algorithm is both memory efficient and computationally scalable. More importantly, its online variant is well-suited to practical high dimensional applications where batch algorithm is prohibitive and data laden regime where data is abundant and runtime is main concern.

Further, *AppGrad* is flexible and structure information can be easily incorporated into the algorithm. For example, if the canonical vectors are assumed to be sparse [56, 23], a thresholding step can be added between the gradient step and normalization step to obtain sparse solutions. On the contrary, it is hard to add sparse constraint to the original CCA formulation which is a generalized eigenvalue problem. Heuristics in [56] avoid this by simply skipping the whitening procedure (NW-CCA). [23] resorts to semidefinite relaxation and therefore is not scalable. *AppGrad* with thresholding works really well in simulations and we leave its theoretical properties for future research.

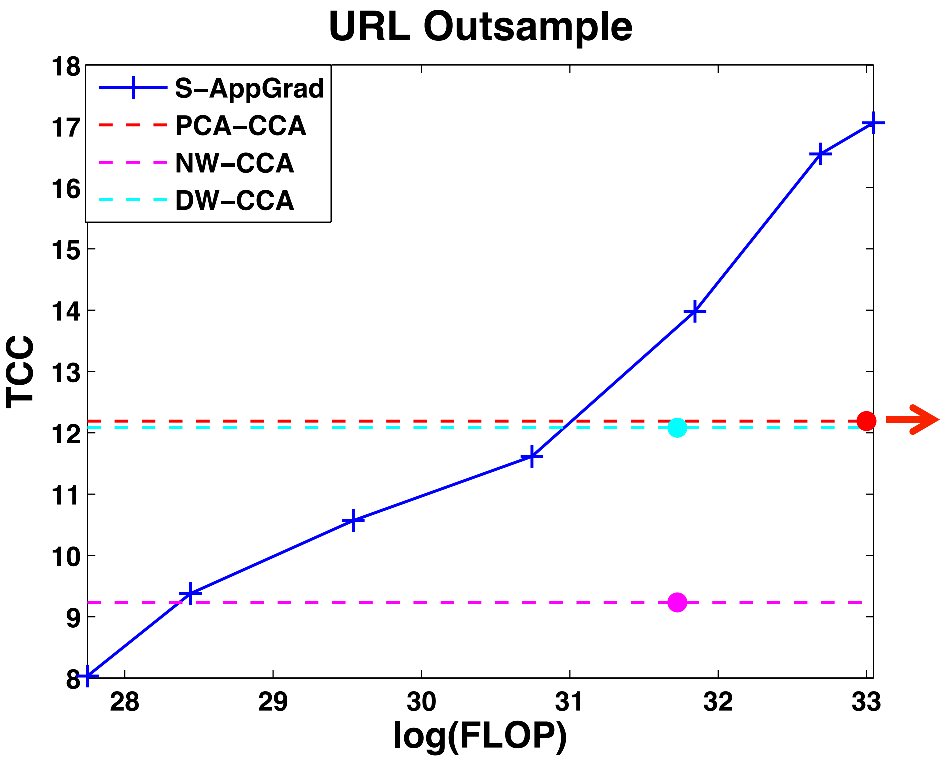

Figure 4.2: Total Correlations Captured (TCC) by NW-CCA, DW-CCA, PCA-CCA and stochastic *AppGrad* on URL dataset. The dash lines indicate TCC for those heuristics and the colored dots denote corresponding computational cost. Red arrow means log(FLOP) of PCA-CCA is more than 33.

Appendices

Appendix A

Appendix for Chapter 3

A.1 Proof of Theorem 3.3.1

Continue to use the notations from the main paper.

Proof. Let's focus on variable $\mathbf{X}$:

Let

$$\mathbf{A} = \tilde{\mathbf{C}}_{xy}\tilde{\mathbf{C}}_{xy}^\top = \mathbf{U}\mathbf{D}\mathbf{V}^\top\mathbf{V}\mathbf{D}\mathbf{U}^\top = \mathbf{U}\mathbf{D}^2\mathbf{U}^\top$$

and $\mathbf{B} = \mathbf{C}_{xx}^{\frac{1}{2}}\mathbf{G}$. So the columns of $\mathbf{U}$ are eigenvectors of $\mathbf{A}$. Let $\mathbf{A}^{t_1}\mathbf{B} = \mathbf{Q}_{t_1}\mathbf{R}_{t_1}$ be the QR decomposition of $\mathbf{A}^{t_1}\mathbf{B}$. Easy to check

$$\mathbf{X}_{t_1} = (\mathbf{H}_\mathbf{X}\mathbf{H}_\mathbf{Y})^{t_1}\mathbf{X}\mathbf{G} = \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{A}^{t_1}\mathbf{B} = \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{Q}_{t_1}\mathbf{R}_{t_1}$$

Note that $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{Q}_{t_1}$ is an orthonormal matrix, so $(\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{Q}_{t_1})\mathbf{R}_{t_1}$ actually gives the QR decomposition of $\mathbf{X}_{t_1}$, i.e. $\mathbf{X}_{k_{\text{cca}}} = \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{Q}_{t_1}$.

By theorem 28.1 in [53], the columns of $\mathbf{Q}_{t_1}$ will converge to $\mathbf{U}_1$ as long as the two regularity condition hold which can be implied by our assumptions (see equation 28.4 and 28.5 in [53] for details). Therefore $\mathbf{X}_{k_{\text{cca}}} = \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{Q}_{t_1}$ converges to $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1$ which are the top canonical variables of $\mathbf{X}$. The argument for $\mathbf{Y}$ is the same. □

A.2 Randomized Algorithm for Finding Top Singular Vectors

Here we briefly describe a fast randomized algorithm which finds the top singular vectors of the data matrices as mentioned in section 3.4 of the main paper. In fact our regression based algorithms only need an orthonormal basis of the subspace spanned by the top left singular vectors of $\mathbf{X}$ and $\mathbf{Y}$ instead of the singular vectors themselves. We stick with top singular vectors in the statements and proofs of the main paper since its mathematically cleaner. All the mathematical properties of the algorithms mentioned in our paper carry through if we replace the top singular vectors with any orthonormal basis of the same subspace which is computationally more convenient because regression is a projection onto a certain subspace. Algorithm 12 is an randomized algorithm for finding this orthonormal basis, based on the idea of random subspace finder introduced by [28].

Remark A.2.1. For numerical stability reasons, in step 2 we perform QR decomposition every time after multiply with $(\mathbf{X}^\top\mathbf{X})$ as suggested by [28]. More intuitions and theoretical details of the algorithm can be found in [28].

Algorithm 12 Random Subspace Finder

```
Input : Matrix $\mathbf{X} \in n \times p_1$, target dimension $k_{\text{pc}}$, number of power iterations $i$.
Output : $\mathbf{U}_1 \in n \times k_{\text{pc}}$, an orthonormal basis of the span of the top $k_{\text{pc}}$ left singular vectors of $\mathbf{X}$, $Q_2$, an orthonormal basis of the span of the top $k_{\text{pc}}$ right singular vectors of $\mathbf{X}$
1.Generate random matrices $R_2 \in p_1 \times k_{\text{pc}}$ with i.i.d standard Gaussian entries.
2.Estimate the span of top $k_{\text{pc}}$ right singular vectors of $\mathbf{X}$ by $A_2 = (\mathbf{X}^\top \mathbf{X})^i R_2$.
3.Use QR decomposition to compute $Q_2 \in p_1 \times k_{\text{pc}}$ which is an orthonormal basis of the column space of $A_2$.
4.Compute the span of top $k_{\text{pc}}$ left singular vectors of $\mathbf{X}$ by $A_1 = \mathbf{X} Q_2$.
5. Use QR decomposition of $A_1$ to compute an orthonormal basis $\mathbf{U}_1$
```

A.3 Gradient Descent with Optimal Step Size

Algorithm 13 gives a detailed description of the Gradient Descent algorithm we used in LING in section 3.1 of the main paper which is explained in detail by [1].

Remark A.3.1. An implementation detail that worth mentioning is that after every iteration of GD (Algorithm 13), we actually project the coefficient $\beta_{2,t}$ onto the orthogonal complement of V_1 the columns of which consists top k_{pc} right singular vector of X (it's obtained by the randomized algorithm while computing U_1 as described in Algorithm 12 above), i.e. we set $\beta_{r,t+1} = \beta_{r,t+1} - V_1 V_1^\top \beta_{r,t+1}$ at the end of every iteration. The projection step significantly increases the performance of LING and our CCA algorithm. The intuition of projection step can be easily seen from the the proof of Theorem 2 (see

Algorithm 13 Gradient Descent with Optimal Step Size (GD)

```
Goal : Solve the LS problem $\min_{\beta_r \in \mathcal{R}^p} \|X\beta_r - Y_r\|^2$.
Input : $X \in n \times p$, $Y_r \in n \times 1$, number of gradient iterations $t_2$, an initial vector $\beta_{r,0}$ (We always initialize with $0$ vector)
Output : $\beta_{r,t_2}$, regression coefficients after $t_2$ iterations.
for $t = 0$ to $t_2 - 1$ do
    $Q = 2X^\top X$
    $w_t = 2X^\top Y_r - Q\beta_{r,t}$
    $s_t = \frac{w_t^\top w_t}{w_t^\top Q w_t}$. $s_t$ is the step size which makes the target function decrease the most in direction $w_t$.
    $\beta_{r,t+1} = \beta_{r,t} + s_t \cdot w_t$.
end for
```

next section) which is addressed in remark A.4.2 of this appendix.

A.4 Error Analysis of LING

This section gives a detailed proof of theorem 3.4.2 in the main paper. Here we continue to use the definition and other notations in the main paper and above sections.

Lemma A.4.1. Sub-optimality Bound for GD

Assume X is full rank with singular values $\lambda_1, \lambda_2..\lambda_p$. Let

$$f(\beta_r) = \frac{1}{2}\beta_r^\top Q\beta_r - 2Y_r^\top X\beta_r + Y_r^\top Y_r$$

be the target function value we want to minimize in regression. Assume f^* is the minimum value of the target function. Let $\beta_{r,t}$ be the coefficient after t iterations when initializing with 0 vector. The sub-optimality of $\beta_{r,t}$ which is defined as $f(\beta_{r,t}) - f^*$ satisfies:

$$\frac{f(\beta_{r,t+1}) - f^*}{f(\beta_{r,t}) - f^*} \leq \left(\frac{\lambda_{k_{pc}+1}^2 - \lambda_p^2}{\lambda_{k_{pc}+1}^2 + \lambda_p^2}\right)^2$$

Proof. Let X have the singular value decomposition

$$X = [U_1, U_2]\begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix}[V_1, V_2]^\top$$

where U_1, V_1 are top k_{pc} singular vectors.

First we claim that if initialize with 0 vector, $\beta_{r,t}, w_t$ will always be in the span of V_2. This is easy to see by recursion. Assume $\beta_{r,t}$ is in the span of V_2, by Algorithm

13, $w_t = 2X^\top Y_r - 2X^\top X\beta_{r,t}$. Note that Y_r is orthogonal to U_1 since it's the residual of Y after projecting onto U_1. So both $X^\top Y_r$ and $X^\top X\beta_{r,t}$ lives in the span of V_2 and also w_t lives in the span of V_2. Therefore $\beta_{r,t+1} = \beta_{r,t} + s_t w_t$ also lives in the span of V_2. If we start with 0 which is in the span of V_2, $\beta_{r,t}$, w_t will stay in the span of V_2 forever.

By taking derivatives of the target function we have

$$f^* = -2Y_r^\top XQ^{-1}X^\top Y_r + Y_r^\top Y_r$$

So we have

$$\begin{aligned}
& f(\beta_{r,t}) - f^* \\
= \quad & \frac{1}{2}\beta_{r,t}^\top Q\beta_{r,t} - 2Y_r^\top X\beta_{r,t} + 2Y_r^\top XQ^{-1}X^\top Y_r \\
= \quad & (Q\beta_{r,t} - 2X^\top Y_r)^\top \frac{1}{2}Q^{-1}(Q\beta_{r,t} - 2X^\top Y_r) \\
= \quad & \frac{1}{2}w_t^\top Q^{-1}w_t \qquad \text{(A.4.1)}
\end{aligned}$$

$$\begin{aligned}
& f(\beta_{r,t+1}) - f(\beta_{r,t}) \\
= & \frac{1}{2}(\beta_{r,t} + s_t w_t)^\top Q(\beta_{r,t} + s_t w_t) \\
- & 2Y_r^\top X(\beta_{r,t} + s_t w_t) \\
- & \frac{1}{2}\beta_{r,t}^\top Q\beta_{r,t} + 2Y_r^\top X\beta_{r,t} \\
= & \frac{1}{2}s_t^2 w_t^\top Q w_t \\
+ & s_t w_t^\top Q\beta_{2,t} - 2s_t w_t^\top X^\top Y_r \\
= & \frac{1}{2}s_t^2 w_t^\top Q w_t - s_t w_t^\top w_t \\
= & -\frac{(w_t^\top w_t)^2}{2w_t^\top Q w_t}
\end{aligned} \tag{A.4.2}$$

With above equation we have

$$\begin{aligned}
& \frac{f(\beta_{r,t+1}) - f^*}{f(\beta_{r,t}) - f^*} \\
= & \ 1 - \frac{f(\beta_{r,t}) - f(\beta_{r,t+1})}{f(\beta_{r,t}) - f^*} \\
= & \ 1 - \frac{\frac{(w_t^\top w_t)^2}{2w_t^\top Q w_t}}{\frac{1}{2}w_t^\top Q^{-1} w_t} \\
= & \ 1 - \frac{(w_t^\top w_t)^2}{(w_t^\top Q w_t)(w_t^\top Q^{-1} w_t)}
\end{aligned} \tag{A.4.3}$$

Since w_t always lives in the span of V_2, let $z_t = V_2^\top w_t$, we have

$$\frac{(w_t^\top w_t)^2}{(w_t^\top Q w_t)(w_t^\top Q^{-1} w_t)} = \frac{(z_t^\top z_t)^2}{(z_t^\top \Lambda_2^2 z_t)(z_t^\top \Lambda_2^{-2} z_t)}$$

By Kantorovich Inequality [1], $\frac{(z_t^\top z_t)^2}{(z_t^\top D_2^2 z_t)(z_t^\top D_2^{-2} z_t)} \geq \frac{4(\lambda_{k_{\text{pc}}+1}\lambda_p)^2}{(\lambda_{k_{\text{pc}}+1}^2 + \lambda_p^2)^2}$. Plug into equation A.4.3 we have

$$\frac{f(\beta_{2,t+1}) - f^*}{f(\beta_{2,t}) - f^*} \leq \left(\frac{\lambda_{k_{\text{pc}}+1}^2 - \lambda_p^2}{\lambda_{k_{\text{pc}}+1}^2 + \lambda_p^2}\right)^2$$

□

Remark A.4.2. From the proof it's clear that keeping the gradient w_t and coefficient $\beta_{2,t}$ in the span of V_2 is curtail to the fast convergence to the GD algorithm. When U_1 consists exactly the top left singular vectors, we proved above that $w_t, \beta_{2,t}$ will always stay in the span of V_2. However, in practice U_1 in computed by Algorithm 12 which is only an approximate of the top left singular vectors. In order to compensate the error of the randomized algorithm, we project the coefficient $\beta_{2,t}$ back to the span of V_2 after every iteration of GD, as illustrated in remark A.3.1 of the appendix.

A.4.1 Proof of Theorem 3.4.2

With the above lemma we can proof theorem 3.4.2 in the main paper

Proof. The optimality of Y^* implies that $Y^* - Y$ is orthogonal to the span of X and in particular is orthogonal to $\hat{Y}_{t_2} - Y^*$. By pythagorean theorem

$$\|\hat{Y}_{t_2} - Y\|^2 - \|Y^* - Y\|^2 = \|\hat{Y}_{t_2} - Y^*\|^2$$

On the other hand

$$\|\hat{Y}_{t_2} - Y\|^2 = \|(X\beta_{r,t_2} + Y_1) - (Y_r + Y_1)\|^2 = \|X\beta_{r,t_2} - Y_r\|^2$$

and

$$\begin{aligned} \|Y^* - Y\|^2 &= \|(Y_1 + U_2U_2^\top Y_r) - (Y_1 + Y_r)\|^2 \\ &= \|U_2U_2^\top Y_r - Y_r\|^2 \end{aligned}$$

Easy to see $\|U_2U_2^\top Y_r - Y_r\|^2 = f^*$

Put above equations and lemma A.4.1 together,

$$\begin{aligned}
\|\hat{Y}_{t_2} - Y^*\|^2 &= \|\hat{Y}_{t_2} - Y\|^2 - \|Y^* - Y\|^2 \\
&= \|X\beta_{r,t_2} - Y_r\|^2 - \|U_2U_2^\top Y_r - Y_r\|^2 \\
&= f(\beta_{r,t_2}) - f^* \\
&= \left(\frac{\lambda_{k_{\mathrm{pc}}+1}^2 - \lambda_p^2}{\lambda_{k_{\mathrm{pc}}+1}^2 + \lambda_p^2}\right)^{2t_2} (f(\beta_{r,t_2}) - f^*)
\end{aligned}$$

□

A.5 Error Analysis of L-CCA

This section gives detailed proof of Theorem 3 in the main paper. The next lemma gives an easy way of computing distance between subspaces the proof of which is in theorem 2.6.1 [24].

Lemma A.5.1. *Let*

$$\mathbf{W} = [\underset{k}{\mathbf{W}_1}, \underset{n-k}{\mathbf{W}_2}] \quad \mathbf{Z} = [\underset{k}{\mathbf{Z}_1}, \underset{n-k}{\mathbf{Z}_2}]$$

are $n \times n$ orthonormal matrices, dist$(\mathbf{W}_1, \mathbf{Z}_1) = \|\mathbf{W}_1^\top Z_2\|_2 = \|\mathbf{W}_2^\top Z_1\|_2$

Now let's state the key lemma for error analysis of L-CCA (below we continue to use the notation used in the main paper and supplementary):

Lemma A.5.2. *Let $\mathbf{X}_t$, $\mathbf{Y}_t$ be the* **LING** *output in every iteration defined in Algorithm 3 of the main paper. Let $\mathbf{Y}_t = \mathbf{H}_\mathbf{Y}\hat{\mathbf{X}}_{t-1} + \Delta_{y,t}$, $\mathbf{X}_t = \mathbf{H}_\mathbf{X}\hat{\mathbf{Y}}_t + \Delta_{x,t}$ where $\Delta_{x,t}, \Delta_{y,t}$ denotes* *the error of LING compared with the exact LS solution. Assume* $\|\Delta_{x,t}\|_2, \|\Delta_{y,t}\|_2 \leq \epsilon$ *for every* t*. Then the distance between subspace spanned top* k_{cca} *canonical variables and the subspace returned by L-CCA is bounded by*

$$dist(\hat{\mathbf{X}}_{t_1}, \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1) \leq C_1 \left(\frac{d_{k_{cca}+1}}{d_{k_{cca}}}\right)^{2t_1} + C_2 \frac{d_{k_{cca}}^2}{d_{k_{cca}}^2 - d_{k_{cca}+1}^2}\epsilon$$

where C_1*,* C_2 *are constants.*

Proof. Let's focus on the t^{th} iteration. Note that QR decomposition is essentially a change of basis, so we have $\hat{\mathbf{X}}_t = \mathbf{X}_t\mathbf{R}_{x,t}$ and $\hat{\mathbf{Y}}_t = \mathbf{Y}_t\mathbf{R}_{y,t}$ for some non-singular matrix $\mathbf{R}_{x,t}, \mathbf{R}_{y,t}$.

First represent $\hat{\mathbf{X}}_t$ in terms of $\hat{\mathbf{X}}_{t-1}$ and errors of LING :

$$\begin{aligned}
\hat{\mathbf{X}}_t &= \mathbf{X}_t\mathbf{R}_{x,t} \\
&= (\mathbf{H}_\mathbf{X}\hat{\mathbf{Y}}_t + \Delta_{x,t})\mathbf{R}_{x,t} \\
&= (\mathbf{H}_\mathbf{X}\mathbf{Y}_t\mathbf{R}_{y,t} + \Delta_{x,t})\mathbf{R}_{x,t} \\
&= (\mathbf{H}_\mathbf{X}(\mathbf{H}_\mathbf{Y}\hat{\mathbf{X}}_{t-1} + \Delta_{y,t})\mathbf{R}_{y,t} + \Delta_{x,t})\mathbf{R}_{x,t} \\
&= \mathbf{H}_\mathbf{X}\mathbf{H}_\mathbf{Y}\hat{\mathbf{X}}_{t-1}\mathbf{R}_{y,t}\mathbf{R}_{x,t} + \mathbf{H}_\mathbf{X}\Delta_{y,t}\mathbf{R}_{y,t}\mathbf{R}_{x,t} \\
&\quad +\Delta_{x,t}\mathbf{R}_{x,t} \qquad \text{(A.5.1)}
\end{aligned}$$

Let $\mathbf{H}_\mathbf{X}\Delta_{y,t} + \Delta_{x,t}\mathbf{R}_{y,t}^{-1} = \Delta_t$, together with equation A.5.1 we have

$$\hat{\mathbf{X}}_t = (\mathbf{H}_\mathbf{X}\mathbf{H}_\mathbf{Y}\hat{\mathbf{X}}_{t-1} + \Delta_t)\mathbf{R}_{y,t}\mathbf{R}_{x,t} \qquad \text{(A.5.2)}$$

For simplicity assume there exist $C_0 > 1$ s.t. $\|\mathbf{R}_{y,t}^{-1}\|_2 \leq (C_0 - 1)$ for all t, we have

$$\begin{aligned}
\|\Delta_t\|_2 &\leq \|\mathbf{H}_{\mathbf{X}}\Delta_{y,t}\|_2 + \|\Delta_{x,t}\mathbf{R}_{y,t}^{-1}\|_2 \\
&\leq \|\Delta_{y,t}\|_2 + (C_0 - 1)\|\Delta_{x,t}\|_2 \\
&\leq C_0\epsilon \qquad \text{(A.5.3)}
\end{aligned}$$

Now define $\mathbf{U}_t = (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \hat{\mathbf{X}}_t$. Since $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}$ and $\hat{\mathbf{X}}_t$ have orthonormal columns and $\hat{\mathbf{X}}_t$ lives in the column space of $\mathbf{X}$ (follows from the definition of the LING algorithm), the columns of matrix $\mathbf{U}_t$ is actually orthonormal. It's also easy to check from the definition that

$$\text{dist}(\hat{\mathbf{X}}_t, \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1) = \text{dist}(\mathbf{U}_t, \mathbf{U}_1) \qquad \text{(A.5.4)}$$

From now on we can bound $\text{dist}(\mathbf{U}_t, \mathbf{U}_1)$ instead. Let $\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2]$, define

$$\mathbf{U}^\top \mathbf{U}_t = \begin{pmatrix} \mathbf{U}_1^\top \\ \mathbf{U}_2^\top \end{pmatrix} \mathbf{U}_t = \begin{pmatrix} \mathbf{W}_{1,t} \\ \mathbf{W}_{2,t} \end{pmatrix} \qquad \text{(A.5.5)}$$

From lemma A.5.1, $\text{dist}(\mathbf{U}_t, \mathbf{U}_1) = \|\mathbf{W}_{2,t}\|_2$. Now let's track the quantity $\|\mathbf{W}_{2,t}(\mathbf{W}_{1,t})^{-1}\|_2$ which will eventually help us bounding $\|\mathbf{W}_{2,t}\|_2$. Recall that $\mathbf{A} = \tilde{\mathbf{C}}_{xy}\tilde{\mathbf{C}}_{xy}^\top = \mathbf{U}\mathbf{D}\mathbf{V}^\top\mathbf{V}\mathbf{D}\mathbf{U}^\top = \mathbf{U}\mathbf{D}^2\mathbf{U}^\top$.

$$
\begin{aligned}
\begin{pmatrix} \mathbf{W}_{1,t} \\ \mathbf{W}_{2,t} \end{pmatrix} = \mathbf{U}^\top \mathbf{U}_t & \\
= \;& \mathbf{U}^\top (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \hat{\mathbf{X}}_t \\
= \;& \mathbf{U}^\top (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top (\mathbf{H}_\mathbf{X}\mathbf{H}_\mathbf{Y}\hat{\mathbf{X}}_{t-1} + \Delta_t)\mathbf{R}_{y,t}\mathbf{R}_{x,t} \\
= \;& \mathbf{U}^\top \mathbf{A}^2 (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \hat{\mathbf{X}}_{t-1}\mathbf{R}_{y,t}\mathbf{R}_{x,t} \\
& + \mathbf{U}^\top (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \Delta_t \mathbf{R}_{y,t}\mathbf{R}_{x,t} \\
= \;& \mathbf{U}^\top (\mathbf{A}^2\mathbf{U}_{t-1} + (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \Delta_t)\mathbf{R}_{y,t}\mathbf{R}_{x,t} \qquad \text{(A.5.6)}
\end{aligned}
$$

Note that

$$
\mathbf{U}^\top \mathbf{A}^2 \mathbf{U}_{t-1} = \mathbf{D}^2 \mathbf{U}^\top \mathbf{U}_{t-1} = \begin{pmatrix} \mathbf{D}_1^2 \mathbf{W}_{1,t-1} \\ \mathbf{D}_2^2 \mathbf{W}_{2,t-1} \end{pmatrix} \qquad \text{(A.5.7)}
$$

and let

$$
\begin{aligned}
\begin{pmatrix} \Delta_{1,t} \\ \Delta_{2,t} \end{pmatrix} &= \begin{pmatrix} \mathbf{U}_1^\top \\ \mathbf{U}_2^\top \end{pmatrix} (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \Delta_t \\
&= \begin{pmatrix} \mathbf{U}_1^\top (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \Delta_t \\ \mathbf{U}_2^\top (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \Delta_t \end{pmatrix} \qquad \text{(A.5.8)}
\end{aligned}
$$

Together with equation A.5.3 we have the following norm bound for $i = 1, 2$

$$
\|\Delta_{i,t}\|_2 = \|\mathbf{U}_1^\top (\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}})^\top \Delta_t\|_2 \leq \|\Delta_t\|_2 \leq C_0 \epsilon \qquad \text{(A.5.9)}
$$

because $\mathbf{U}_i$, $\mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}$ both have orthonormal columns. plug equation A.5.7 A.5.8 into

A.5.6 we have

$$\begin{pmatrix} \mathbf{W}_{1,t} \\ \\ \mathbf{W}_{2,t} \end{pmatrix} = \begin{pmatrix} \mathbf{D}_1^2\mathbf{W}_{1,t-1} + \Delta_{1,t} \\ \\ \mathbf{D}_2^2\mathbf{W}_{2,t-1} + \Delta_{2,t} \end{pmatrix} \mathbf{R}_{y,t}\mathbf{R}_{x,t} \tag{A.5.10}$$

Equation A.5.10 directly implies

$$\begin{aligned}
&\|\mathbf{W}_{2,t}(\mathbf{W}_{1,t})^{-1}\|_2 \\
&= \|(\mathbf{D}_2^2\mathbf{W}_{2,t-1} + \Delta_{2,t})(\mathbf{D}_1^2\mathbf{W}_{1,t-1} + \Delta_{1,t})^{-1}\|_2 \\
&\leq \|(\mathbf{D}_2^2\mathbf{W}_{2,t-1})(\mathbf{D}_1^2\mathbf{W}_{1,t-1})^{-1}\|_2 \\
&+C_3(\|\Delta_{1,t}\|_2 + \|\Delta_{2,t}\|_2) \\
&\leq \|\mathbf{D}_2^2\|_2\|\mathbf{W}_{2,t-1}\mathbf{W}_{1,t-1}^{-1}\|_2\|\mathbf{D}_1^{-2}\|_2 + 2C_3C_0\epsilon \\
&= \frac{d_{k_{\text{cca}}+1}^2}{d_{k_{\text{cca}}}^2}\|\mathbf{W}_{2,t-1}\mathbf{W}_{1,t-1}^{-1}\|_2 + 2C\epsilon
\end{aligned} \tag{A.5.11}$$

where $C = C_0C_3$ are all constants independent of t. Note that in the first inequality, we ignore the higher order error term $\|\Delta_{1,t}\|_2 \cdot \|\Delta_{2,t}\|_2$.

Recursively apply equation A.5.11 t_1 times leads to

$$\begin{aligned}
& \|\mathbf{W}_{2,t_1}(\mathbf{W}_{1,t_1})^{-1}\|_2 \\
\leq \quad & \|\mathbf{W}_{2,0}(\mathbf{W}_{1,0})^{-1}\|_2 \left(\frac{d_{k_{\text{cca}}+1}}{d_{k_{\text{cca}}}}\right)^{2t_1} \\
+ \quad & \sum_{j=0}^{t_1-1} \left(\frac{d_{k_{\text{cca}}+1}}{d_{k_{\text{cca}}}}\right)^{2j} 2C\epsilon \\
= \quad & \|\mathbf{W}_{2,0}(\mathbf{W}_{1,0})^{-1}\|_2 \left(\frac{d_{k_{\text{cca}}+1}}{d_{k_{\text{cca}}}}\right)^{2t_1} \\
+ \quad & \frac{d_{k_{\text{cca}}}^2}{d_{k_{\text{cca}}}^2 - d_{k_{\text{cca}}+1}^2} C\epsilon
\end{aligned} \tag{A.5.12}$$

From equation A.5.4 we have

$$
\begin{aligned}
& \operatorname{dist}(\hat{\mathbf{X}}_{t_1}, \mathbf{X}\mathbf{C}_{xx}^{-\frac{1}{2}}\mathbf{U}_1) = \operatorname{dist}(\mathbf{U}_{t_1}, \mathbf{U}_1) \\
= \; & \|\mathbf{W}_{2,t_1}\|_2 \\
\leq \; & \|\mathbf{W}_{2,t_1}(\mathbf{W}_{1,t_1})^{-1}\|_2 \qquad \text{(A.5.13)}
\end{aligned}
$$

The last inequality is because $\|\mathbf{W}_{1,t_1}\|_2 \leq 1$. Put equation A.5.12 A.5.13 together completes the proof. □

Finally, use results of Theorem 3.4.2 in the main paper to bound ϵ in the above lemma directly proves Theorem 3.4.5 in the main paper.

Bibliography

[1] Marina A.Epelman. Rate of convergence of steepest descent algorithm. Lecture Notes, 2007.

[2] Nir Ailon and Edo Liberty. Fast dimension reduction using rademacher series on dual bch codes. Technical report, 2007.

[3] R Arora, A Cotter, K Livescu, and N Srebro. Stochastic optimization for pca and pls. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 861–868. IEEE, 2012.

[4] Andreas Artemiou and Bing Li. On principal components and regression: a statistical explanation of a natural phenomenon. *Statistica Sinica*, 19(4):1557–1565, 2009.

[5] Haim Avron, Christos Boutsidis, Sivan Toledo, and Anastasios Zouzias. Efficient dimensionality reduction for canonical correlation analysis. In *ICML (1)*, pages 347–355, 2013.

[6] Haim Avron, Petar Maymounkov, and Sivan Toledo. Blendenpik: Supercharging lapack's least-squares solver. *SIAM J. Sci. Comput.*, 32(3):1217–1236, April 2010.

[7] Francis R. Bach and Michael I. Jordan. A probabilistic interpretation of canonical correlation analysis. Technical report, University of California, Berkeley, 2005.

[8] Akshay Balsubramani, Sanjoy Dasgupta, and Yoav Freund. The fast convergence of incremental pca. In *Advances in Neural Information Processing Systems*, pages 3174–3182, 2013.

[9] ke Björck and Gene H Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of computation*, 27(123):579–594, 1973.

[10] Matthew B Blaschko and Christoph H Lampert. Correlational spectral clustering. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[11] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[12] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[13] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[14] Kamalika Chaudhuri, Sham M Kakade, Karen Livescu, and Karthik Sridharan. Multi-view clustering via canonical correlation analysis. In *Proceedings of the 26th annual international conference on machine learning*, pages 129–136. ACM, 2009.

[15] Xi Chen, Han Liu, and Jaime G Carbonell. Structured sparse canonical correlation analysis. In *International Conference on Artificial Intelligence and Statistics*, pages 199–207, 2012.

[16] Paramveer Dhillon, Yichao Lu, Dean P. Foster, and Lyle Ungar. New subsampling algorithms for fast least squares regression. In *Advances in Neural Information Processing Systems 26*, pages 360–368. 2013.

[17] Paramveer S. Dhillon, Dean Foster, and Lyle Ungar. Multi-view learning of word embeddings via cca. In *Advances in Neural Information Processing Systems (NIPS)*, volume 24, 2011.

[18] Paramveer S. Dhillon, Dean P. Foster, Sham M. Kakade, and Lyle H. Ungar. A risk comparison of ordinary least squares vs ridge regression. *Journal of Machine Learning Research*, 14:1505–1511, 2013.

[19] Paramveer S. Dhillon, Jordan Rodu, Dean P. Foster, and Lyle H. Ungar. Two step cca: A new spectral method for estimating vector models of words. In *Proceedings of the 29th International Conference on Machine learning*, ICML'12, 2012.

[20] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Sampling algorithms for l2 regression and applications. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 1127–1136, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[21] Petros Drineas, Michael W. Mahoney, S. Muthukrishnan, and Tamás Sarlós. Faster least squares approximation. *CoRR*, abs/0710.1435, 2007.

[22] Dean P. Foster, Sham M. Kakade, and Tong Zhang. Multi-view dimensionality reduction via canonical correlation analysis. Technical report, 2008.

[23] Chao Gao, Zongming Ma, and Harrison H Zhou. Sparse cca: Adaptive estimation and computational barriers. *arXiv preprint arXiv:1409.8565*, 2014.

[24] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[25] Gene. H Golub and Hongyuan Zha. The canonical correlations of matrix pairs and their numerical computation. Technical report, Computer Science Department, Stanford University, 1992.

[26] Gene H Golub and Hongyuan Zha. *The canonical correlations of matrix pairs and their numerical computation*. Springer, 1995.

[27] Isabelle Guyon, Asa Ben Hur, Steve Gunn, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press, 2004.

[28] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, May 2011.

[29] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, May 2011.

[30] Nathan Halko, Per-Gunnar Martinsson, Yoel Shkolnisky, and Mark Tygert. An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific Computing*, 33(5):2580–2594, 2011.

[31] David R. Hardoon, Sandor Szedmak, Or Szedmak, and John Shawe-taylor. Canonical correlation analysis; an overview with application to learning methods. Technical report, 2007.

[32] Cibe Hariharan and Shivashankar Subramanian. Large scale multi-view learning on mapreduce. 2014.

[33] H Hotelling. Relations between two sets of variables. *Biometrika*, 28:312–377, 1936.

[34] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[35] Ian Jolliffe. *Principal Component Analysis. Encyclopedia of Statistics in Behavioral Science*. John Wiley & Sons, 2005.

[36] Sham M. Kakade and Dean P. Foster. Multi-view regression via canonical correlation analysis. In *In Proc. of Conference on Learning Theory*, 2007.

[37] Michael Lamar, Yariv Maron, Mark Johnson, and Elie Bienenstock. SVD and Clustering for Unsupervised POS Tagging. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 215–219, Uppsala, Sweden, 2010. Association for Computational Linguistics.

[38] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[39] Yichao Lu, Paramveer S. Dhillon, Dean Foster, and Lyle Ungar. Faster ridge regression via the subsampled randomized hadamard transform. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[40] Yichao Lu and Dean P Foster. large scale canonical correlation analysis with iterative least squares. In *Advances in Neural Information Processing Systems*, pages 91–99, 2014.

[41] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *In Proc. of the International Conference on Machine Learning (ICML*, 2009.

[42] Ping Ma, Michael W. Mahoney, and Bin Yu. A statistical perspective on algorithmic leveraging. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 91–99, 2014.

[43] Brian McWilliams, David Balduzzi, and Joachim Buhmann. Correlated random features for fast semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 440–448, 2013.

[44] Brian McWilliams, Gabriel Krummenacher, Mario Lucic, and Joachim M. Buhmann. Fast and robust least squares estimation in corrupted linear models. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 415–423, 2014.

[45] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. Memory limited, streaming pca. In *Advances in Neural Information Processing Systems*, pages 2886–2894, 2013.

[46] Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106(1):69–84, 1985.

[47] Vladimir Rokhlin and Mark Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences*, 105(36):13212–13217, September 2008.

[48] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.

[49] G. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proc. 15th International Conf. on Machine Learning*, pages 515–521. Morgan Kaufmann, San Francisco, CA, 1998.

[50] Shai Shalev-Shwartz and Nathan Srebro. Svm optimization: inverse dependence on training set size. In *Proceedings of the 25th international conference on Machine learning*, pages 928–935. ACM, 2008.

[51] Cees GM Snoek, Marcel Worring, Jan C Van Gemert, Jan-Mark Geusebroek, and Arnold WM Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 421–430. ACM, 2006.

[52] Liang Sun, Shuiwang Ji, and Jieping Ye. A least squares formulation for canonical correlation analysis. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1024–1031, New York, NY, USA, 2008. ACM.

[53] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997.

[54] Joel A. Tropp. Improved analysis of the subsampled randomized hadamard transform.

[55] Javier Vía, Ignacio Santamaría, and Jesús Pérez. A learning algorithm for adaptive canonical correlation analysis of several data sets. *Neural Netw.*, 20(1):139–152, January 2007.

[56] Daniela M Witten, Robert Tibshirani, and Trevor Hastie. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, page kxp008, 2009.

[57] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.

[58] Lu Yichao and Dean P. Foster. Fast ridge regression with randomized principal component analysis and gradient descent. *Arxiv, preprint*, 2014.

[59] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML 2004: Proceedings of the twenty-first International Conference on Machine Learning. OMNIPRESS*, pages 919–926, 2004.