Imperial College London

Streaming Algorithms in Principal Components Analysis and Canonical Correlation Analysis

Submission date	14.06.2022
Student's name	Weizhi Liu
Supervisor's name	Dr. Andrew Duncan

This is my own work except where otherwise stated.

Weizhi Liu

Acknowledgement

I thank Dr Andrew Duncan for his continued guidance and advice during my project - it was invaluable to me.

I also thank all my friends and classmates during my whole 4-year undergraduate programme.

Abstract

Principal Components Analysis(PCA) and Canonical Correlation Analysis(CCA) play an essential role in data science problems. They allow people to extract the most important information among a wide range of features and also reduce the dimensions of the dataset so that it could be used in further process. However, as we are stepping into the era of big data, the classical ways of solving PCA and CCA is no longer preferred. Streaming algorithm provides a size-independent approach to compute the stochastic approximation of the PCA and CCA. In this thesis, we will review some other streaming methods and develop a new online CCA algorithm by introducing Lagrange multipliers and stochastic gradient descent(SGD) techniques. We also explore the performance of the method by conducting experiments on various datasets. Finally, we discuss the elements which affect the convergence rate of the method and how to improve them.

Contents

1 Introduction

1.1 Overview of the Project

This project aims to investigate streaming algorithms on Principal Components Analysis and Canonical Correlation Analysis. In the remaining part of this section, we provide a non-mathematical background on the high-dimensional big data problems and the motivation for this project.

Section 2 presents some high-lighted existing work in the area. We firstly demonstrate a offline Principal Components Analysis(PCA) works and the mathematics behind the problem. Then we introduce how Hebb's learning rule [13] is applied to the online PCA and some improvements made by the other research teams in the recent decades, ie Oja++ [1], EigenGame [10]. In the later part of this section, we also introduce the mathematical idea behind the Canonical Correlation Analysis(CCA) and how does it relate to the PCA. Some previous results obtained on the online CCA are also shown in this section, including GenOja [3]. In later subsection, we have also developed high-dimensional GenOja algorithm as an upgrade of GenOja in the $k > 1$ cases.

Section 3 is a new streaming algorithm on the CCA problem developed by us. We modified the online PCA algorithm, and the apply it to the CCA problem by making some modifications and introducing two Lagrange multipliers. We also provide the algorithm and the implementation in this section.

Section 4 shows the experiments we have done on the method. We test our own algorithm and some existing online PCA and CCA algorithms on two datasets and compare the result with the theoretical value we introduced in the Section 2. We also investigate how the learning rate affects the algorithm and also a learning rate control technique that helps to boost the efficiency of the algorithm.

Finally, in Chapter 5, we will summarise the findings of this project and offer some suggestions for further research.

The code associates with thesis is on: https://github.com/WilliamLiu666/M4R_PCA.

1.2 Motivation

1.2.1 Big Data and Streaming Algorithms

Big data, the heart of modern research and industry, refers to enormous data collections with a diverse and complicated structure that are challenging to store, analyse, and visualise for subsequent processes or outcomes[28]. Big data analytics is the study of large volumes of data in order to uncover hidden patterns and hidden relationships. With the rise of big data, particularly in fields like

finance[30], physics and bio-medicine[7], fast and efficient algorithms for data mining and processing are becoming increasingly vital.

The streaming algorithm, one of the popular topics in the big data world, plays an essential role in the model industries. It provides a size-independent approach to solving the issues without historical data. In the real world, due to the limitation of computer capability, it is extremely hard to carry out data analysis by using all the recorded data. Especially in the case of size-dependent algorithms where the computational cost increase exponentially, the storage and memory will run out immediately. For example, in a factory, suppose that each sensor records a new set of data in each millisecond, and there are thousands of detectors in total. Then the factory computer has to process over 1 million data each second. Since the data is coming like an endless stream, the storage would be limited to storing all the previous datasets. In this case, the engineers have to figure out a method that efficiently processes the newly observed data before new data comes. Similarly, millions of people access their web page or their online services for internet companies. If the companies wish to analysis their user data, they have to find out an efficient streaming algorithm to achieve their target. Otherwise, the computational cost will explore rapidly if they include the historical data. Investigating the streaming algorithm helps us solve the above problem and all other fields with a large amount of data required to process.

1.2.2 Principal Components Analysis and Canonical Components Analysis

Principal Components Analysis is a widely used dimension reduction method in data science study. It helps people find out the most important information in the dataset and can be further used in classification, regression, clustering or other data science problems. The target of PCA is to exact as much information as possible from the dataset while using the least number of dimensions. In mathematics, PCA finds the vectors which maximise the variance of the projection onto them[31].

Canonical Correlation Analysis is also useful in dimension reduction problems and often considered as an unsupervised multi-view learning[27][19][5]. Mathematically, CCA is an extension of the PCA as both targets maximising the variance. Unlike PCA which maximises the variance within a single datasets, CCA optimise the correlation between two different dataset. Compared with PCA, the challenge in CCA is that there are two components that need to be updated together with different constraints.

Many data science algorithms require a high signal-to-noise ratio in order to provide valuable and meaningful output. However, the real world data is always noisy and people have to figure a way to discard the noise from the useful information. PCA and CCA are such tools to help people find new variables that

are linear functions of those in the original dataset, that successively maximize variance and that are uncorrelated with each other[15]. A good example would be the CIFAR-10[16] dataset, which is also discussed in later sections. The data is made of 32x32 colour images in 10 classes. Although the data is in the space $\mathbb{R}^{3072}$, the projection of the dataset in the first two principal components plane contains 40% of the information and could provide a clear view in classification.

As described above, the streaming algorithm is an essential tool to solve the problems in the world of big data, while PCA and CCA are significant in the data process. This thesis is aimed to investigate how the streaming algorithm of PCA and CCA works and develop a new approach. The challenges in the recent research are in terms of convergence rate and gap-free[1]. Research groups are trying to solve the problem with a variety of techniques, for example least square [18][17], doubly stochastic estimations [2]. In this thesis, we are mainly focusing on the Oja's learning rule and how stochastic gradient descent helps the application of Oja's rule in CCA.

1.3 Notation

Scalars, vectors and matrices are represented by normal, bold and capital bold letters respectively, e.g. x, $\boldsymbol{x}$, and $\boldsymbol{X}$. The L2-norm of a vector x is denoted by $||\boldsymbol{x}||$. For any matrix X, spectral norm and Frobenius norm are represented by $||\boldsymbol{X}||$ and $||\boldsymbol{X}||_F$ respectively. $x^{(i)}$ represents the value of x at the ith iteration or in the ith sample of the data. Star symbol represents the theoretical optimal value, e.g. x^* is the theoretical value of x such that the function $f(x^*)$ is optimised.

2 Existing Work

2.1 Principal Components Analysis

Suppose there is a dataset $\{\mathbf{x}^{(i)}\}_{i=1}^{m}$ with $\mathbf{x}^{(i)} \in \mathbb{R}^n$. For $p \gg 1$, the samples are in high dimensions with many features, and we wish to find the method that can reduce the number of the features but keep as much information as possible. A popular data analysis technique for dimension reduction is Principal Components Analysis(PCA). PCA is an unsupervised self-guided method that extracts the structural information from the dataset and reduces the dimension of the dataset to k with $k < n$.

Dimensionality reduction is frequently a practical step to help other downstream tasks (such as classification or clustering) struggle to work with very high-dimensional data sets. Many models become more effective and computationally practical by reducing the number of dimensions while losing nothing in terms of accuracy (or other quality measures).

As discussed above, we aim for a description of every data point in terms of basis functions ϕ_j such that

$$\mathbf{x}^{(i)} = \sum_{j=1}^{n} a_j^{(i)} \boldsymbol{\phi}_j, \tag{1}$$

and the basis should be orthogonal

$$\boldsymbol{\phi}_j^T \cdot \boldsymbol{\phi}_k = \delta_{jk} \tag{2}$$

There are many ways to view the dimensionality reduction, one is to approximate $\mathbf{x}^{(i)}$ by

$$\mathbf{x}_k^{(i)} = \sum_{j=1}^{k} a_j^{(i)} \boldsymbol{\phi}_j + \sum_{j=k+1}^{n} b_j \boldsymbol{\phi}_j, \tag{3}$$

where the coefficient b_j is not dependent on i. In another word, the aim is to find $(b_j, \boldsymbol{\phi}_j)$ such that they apply to all samples. To achieve this, we describe each sample $\mathbf{x}^{(i)}$ only through the k coefficients $a_j^{(i)}$. As for the sum of b_j for the rest $p - k$ terms, they are set to constant and the error of the approximation is related to it.

The error under the above assumption can be written as

$$\Delta\mathbf{x}^{(i)} = \mathbf{x}^{(i)} - \mathbf{x}_k^{(i)} = \sum_{j=k+1}^{n} \left(a_j^{(i)} - b_j\right) \boldsymbol{\phi}_j, \tag{4}$$

In order to find the best approximation of the overall dataset, the overall mean

squared error(MSE) is given in the form

$$\begin{aligned}\mathbf{MSE} &= \frac{1}{m}\sum_{i=1}^{m}||\Delta\mathbf{x}^{(i)}||^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=k+1}^{n}\sum_{l=k+1}^{n}\left(a_j^{(i)}-b_j\right)\left(a_l^{(i)}-b_l\right)\boldsymbol{\phi}_j^T\cdot\boldsymbol{\phi}_l \\ &= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=k+1}^{n}\sum_{l=k+1}^{n}\left(a_j^{(i)}-b_j\right)\left(a_l^{(i)}-b_l\right)\delta_{jl} \\ &= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=k+1}^{n}\left(a_j^{(i)}-b_j\right)^2\end{aligned} \tag{5}$$

To optimise the MSE, there are two parameters need to be computed, namely b_j and $\boldsymbol{\phi}_j$. Firstly consider b_j

$$\frac{\partial\mathbf{MSE}}{\partial b_j} = \frac{1}{m}\sum_{i=1}^{m}-2\left(a_j^{(i)}-b_j\right). \tag{6}$$

Assume that b_j^* is the optimum value of b_j, then it satisfies

$$\frac{\partial\mathbf{MSE}}{\partial b_j}\Big|_{b_j^*} = 0, \tag{7}$$

this implies

$$b_j^* = \frac{1}{m}\sum_{i=1}^{m}a_j^{(i)}. \tag{8}$$

As for the optimum basis ϕ_j, before some computation, we can reformulate the equation(5) as

$$\begin{aligned}\mathbf{MSE} &= \sum_{j=k+1}^{n}\frac{1}{m}\sum_{i=1}^{m}\left(a_j^{(i)}-b_j^*\right)^2 \\ &= \sum_{j=k+1}^{n}\frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}^{(i)^T}\cdot\boldsymbol{\phi}_j-\frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)^T}\cdot\boldsymbol{\phi}_j)^2,\end{aligned} \tag{9}$$

$a_j^{(i)}$ can be replaced by $\mathbf{x}^{(i)^T}\cdot\boldsymbol{\phi}_j$ since equation(1)

$$\begin{aligned}\mathbf{x}^{(i)} &= \sum_{j=1}^{n}a_j^{(i)}\boldsymbol{\phi}_j \\ \boldsymbol{\phi}_k^T\cdot\mathbf{x}^{(i)} &= \sum_{j=1}^{n}a_j^{(i)}\boldsymbol{\phi}_k^T\cdot\boldsymbol{\phi}_j \\ \boldsymbol{\phi}_k^T\cdot\mathbf{x}^{(i)} &= \sum_{j=1}^{n}a_j^{(i)}\delta_{jk} \\ \boldsymbol{\phi}_k^T\cdot\mathbf{x}^{(i)} &= a_k^{(i)} \\ \mathbf{x}^{(i)^T}\cdot\boldsymbol{\phi}_k &= a_k^{(i)}\end{aligned} \tag{10}$$

As both terms in the second sum in equation(9) contains $\boldsymbol{\phi}_j$,then

$$\begin{aligned}\mathbf{MSE} &= \sum_{j=k+1}^{n} \frac{1}{m} \sum_{i=1}^{m} \left(\left(\mathbf{x}^{(i)^T} - \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^{(i)^T} \right) \cdot \boldsymbol{\phi}_j \right)^2 \\ &= \sum_{j=k+1}^{n} \frac{1}{m} \left(\boldsymbol{\phi}_j^T \cdot \left(\mathbf{x}^{(i)^T} - \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^{(i)^T} \right) \cdot \left(\mathbf{x}^{(i)^T} - \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^{(i)^T} \right)^T \cdot \boldsymbol{\phi}_j \right) \\ &= \sum_{j=k+1}^{n} \left(\boldsymbol{\phi}_j^T \cdot \boldsymbol{C} \cdot \boldsymbol{\phi}_j \right), \end{aligned} \tag{11}$$

where $\boldsymbol{C}$ is the covariance matrix of $\mathbf{x}$.

To find the orthonormal $\boldsymbol{\phi}_j$, we need to carry out a constrained optimisation. Hence the Lagrangian is introduced:

$$\mathbf{L} = \sum_{j=k+1}^{n} \left(\boldsymbol{\phi}_j^T \cdot \boldsymbol{C} \cdot \boldsymbol{\phi}_j \right) + \sum_{j=k+1}^{n} \lambda_j \left(1 - \boldsymbol{\phi}_j^T \cdot \boldsymbol{\phi}_j \right), \tag{12}$$

where the λ_j are Lagrange multipliers. In order to optimise this, take derivative of $\mathbf{L}$

$$\frac{\mathbf{dL}}{\mathbf{d}\boldsymbol{\phi}_j} = 2\boldsymbol{C}\boldsymbol{\phi}_j - 2\lambda_j \boldsymbol{\phi}_j \tag{13}$$

Let $\boldsymbol{\phi}_j^*$ be the optimiser of the equation(12), then

$$\frac{\mathbf{dL}}{\mathbf{d}\boldsymbol{\phi}_j}\Big|_{\boldsymbol{\phi}_j^*} = 2\boldsymbol{C}\boldsymbol{\phi}_j^* - 2\lambda_j \boldsymbol{\phi}_j^* = 0. \tag{14}$$

This implies

$$\boldsymbol{C}\boldsymbol{\phi}_j^* = \lambda_j \boldsymbol{\phi}_j^* \tag{15}$$

The equation(15) is an eigenvalue problem with the Lagrangian multipliers λ_j are the eigenvalues of the covariance matrix $\boldsymbol{C}$. As a result, the basis of the eigenvectors of $\boldsymbol{C}$ provides the solution to our problem of finding a basis to expand. To put it another way, if we expand the data on this basis, we will be able to reduce the MSE in the method we previously defined. Therefore,

$$\begin{aligned}\mathbf{MSE} &= \sum_{j=m+1}^{n} \boldsymbol{\phi}_j^{*^T} \boldsymbol{C} \boldsymbol{\phi}_j^* \\ &= \sum_{j=m+1}^{n} \boldsymbol{\phi}_j^{*^T} \lambda_j \boldsymbol{\phi}_j^* \\ &= \sum_{j=m+1}^{n} \lambda_j. \end{aligned} \tag{16}$$

This indicates that the error is the sum of the eigenvalues that are discarded when going from n to k dimensions. Thus, the choice of an optimal k is subsequently decided based on the eigenvalues and the principal components are therefore the eigenvectors with the largest eigenvalue.

With further analysis, the principal components are also the direction in which the projection onto it has the highest variance. Let projection of the samples on a unit length eigenvector $\boldsymbol{r}$ with corresponding eigenvalue λ is given as $\boldsymbol{Xr}$. Since the sample $\boldsymbol{X}$ is zero-mean, $\boldsymbol{Xr}$ is also zero-mean. Thus the variance of the projection is given by

$$\mathbf{Var} = (\boldsymbol{Xr})^T \boldsymbol{Xr} = \boldsymbol{r}^T \boldsymbol{X}^T \boldsymbol{Xr} = \lambda \boldsymbol{r}^T \boldsymbol{r} = \lambda. \tag{17}$$

This shows that the eigenvalue also represents the variance of the projection on the corresponding eigenvector. Therefore, the principal components also maximise the projection variance.

2.2 Oja's Method

As shown in the previous section, the principal components are no more than the eigenvectors of the covariance matrix of the dataset. However, sometimes the eigenvectors cannot be computed directly, or the cost of the computation is too expensive. In this case, a stochastic approximation approach is introduced to the problem. There are various Oja's methods. In this thesis, the Oja's rule refers the Hebb's learning rule[13]. The method proposes an iterative method for approximating the principal components of a dataset, which can be transformed into a stochastic version by altering the samples used in computing the term $\Delta V^{(n)}$. Instead of directly using the whole dataset, the algorithm allows people randomly pick h points from the dataset with $h \ll m$. In the context of a streaming algorithm, the random samples are the newly observed data from the stream.

Suppose that $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is an m-size zero-mean sample with n independent features and $\boldsymbol{V}$ is a matrix in space $\mathbb{R}^{n \times k}$. Suppose $\boldsymbol{V}^*$ is the principal component. Since the principal components are the directions with the maximum projection variance, then

$$\frac{\mathbf{dVar}}{\mathbf{d}\boldsymbol{V}}\Big|_{\boldsymbol{V}^*} = 2\boldsymbol{X}^T \boldsymbol{X} \boldsymbol{V}^* = \mathbf{0}. \tag{18}$$

According to (18), the method iteratively update the eigenvector by

$$\boldsymbol{V}^{(n+1)} = \boldsymbol{V}^{(n)} + \eta \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{V}^{(n)}, \tag{19}$$

and then normalise $\boldsymbol{V}^{(n+1)}$ by

$$\boldsymbol{V}^{(n+1)} = \frac{\boldsymbol{V}^{(n+1)}}{||\boldsymbol{V}^{(n+1)}||}, \tag{20}$$

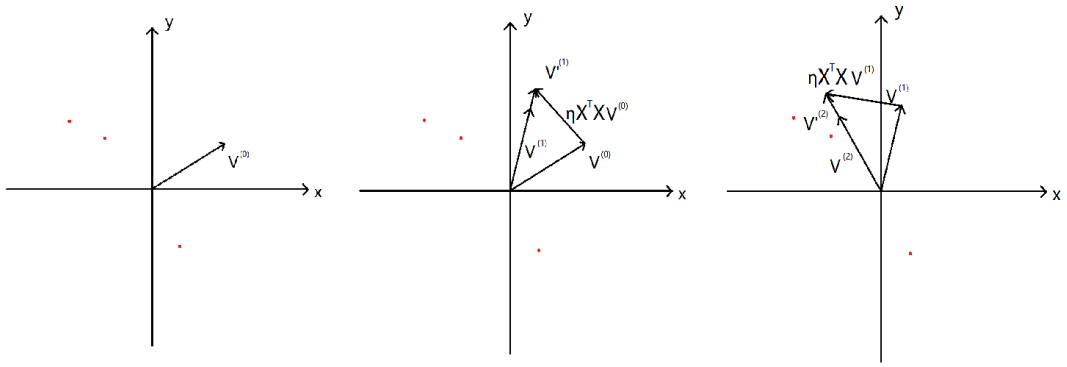

Figure 1: Graph demonstration of offline Oja's rule(all the points are involved in the calculation in each iteration) in the $k = 1$ case. In this example plot, the red points represents the sample points and the vector Vs are the predicted value of V^* in each iteration.

where $\boldsymbol{V}^{(n)}$ is the approximation to the eigenvector at the nth iteration and η is a constant represents the learning rate.

In the Hebbian learning rule, the $\boldsymbol{XV}$ is the post-synaptic term. It assesses the significance of each point. Because the variance quantifies the distance between the sample points, the points further away from the centre will contribute more to the variance. As a result, points with a higher projection value on the vector V are regarded more important, and the corresponding post-synaptic value is higher than the rest. Following the computation of the weights $\boldsymbol{XV}$, the weighted average of all the sample points is given by $\boldsymbol{X}^T\boldsymbol{XV}^{(n)}$, which is the change in $\boldsymbol{V}$. Multiplying with learning rate η and adding to $\boldsymbol{V}^{(n)}$ will push $\boldsymbol{V}^{(n+1)}$ toward the theoretical principal components $\boldsymbol{V}^*$. Since the length is changed after this additive process, the new vector $V^{(n+1)}$ has to be normalised by dividing its length $||\boldsymbol{V}^{(n+1)}||$.

Allen[1] provides a gap-dependent theorem relating to the Oja's method. It claims that for a properly chosen learning rate, the Oja's method would converge in limited steps of iterations after a warm up phase of T_0 steps.

Theorem 1 (Oja's method) *Letting gap* $= \lambda_k - \lambda_{k+1} \in (0, \frac{1}{k}]$ *and* $\Lambda = \sum_{i=1}^{k} \lambda_i \in (0, 1]$, *for every* $\epsilon, p \in (0, 1)$ *define learning rates*

$$T_0 = \tilde{\Theta}\left(\frac{k\Lambda}{gap^2 \cdot p^2}\right)$$
$$T_1 = \tilde{\Theta}\left(\frac{\Lambda}{gap^2}\right) \tag{21}$$

$$\eta_t = \begin{cases} \tilde{\Theta}\left(\frac{1}{gap \cdot T_0}\right) & t \leq T_0 \\ \tilde{\Theta}\left(\frac{1}{gap \cdot T_1}\right) & t \in (T_0, T_0 + T_1] \\ \tilde{\Theta}\left(\frac{1}{gap \cdot (t - T_0)}\right) & t > T_0 + T_1 \end{cases} \tag{22}$$

Let $\boldsymbol{Z}$ be the column orthonormal matrix consisting of all eigenvectors of $\boldsymbol{\Sigma}$ with values no more than λ_{k+1}. Then the output $\boldsymbol{V}^{(T)} \in \mathbb{R}^{d \times k}$ of Oja's algorithm satisfies with probability at least $1 - p$:

for every $T = T_0 + T_1 + \tilde{\Theta}\left(\frac{T_1}{\epsilon}\right)$, it satisfies $||\boldsymbol{Z}^T \boldsymbol{V}^{(T)}||_F^2 \leq \epsilon$.

The above $\tilde{\Theta}$ hides poly-log factors in $\frac{1}{p}$, $\frac{1}{gap}$ and d.

The convergence of the algorithm is measured by the Frobenius norm instead of the spectral norm. The Frobenius norm is a stricter criterion, the convergence in the Frobenius norm implies the convergence in the spectral norm[1].

Algorithm 1: Oja

Input: dataset : $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, **initial vector** : $\boldsymbol{V}_0$, **learning rate** : η
Output: estimated principal component $\hat{\boldsymbol{V}}$
function `Oja's Method`**:**
  $\hat{\boldsymbol{V}} \leftarrow \boldsymbol{V}_0$
  for $t \leftarrow 1, 2, 3, ..., m$ **do**
    $\hat{\boldsymbol{V}}' \leftarrow \hat{\boldsymbol{V}} + \eta \boldsymbol{X}^{t^T} \boldsymbol{X}^t \hat{\boldsymbol{V}}$
    $\boldsymbol{Q}, \boldsymbol{R} \leftarrow QR(\hat{\boldsymbol{V}}')$
    $\boldsymbol{S} \leftarrow sign(diag(R))$
    $\hat{\boldsymbol{V}} \leftarrow \boldsymbol{QS}$
  end
  return $\hat{\boldsymbol{V}}$
end

The QR function is the reduced QR factorization function which makes all of the components orthonormal. The *sign* function is defined as

$$sign(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

This *sign* function is used to make sure that the sign of the vector $\hat{\boldsymbol{V}}$ does not change during the QR factorization.

2.3 Oja++

The Oja++ is an efficient, gap-free and global convergent upgrade of the Oja's method [1]. Also the algorithm supports the cases where the number of target

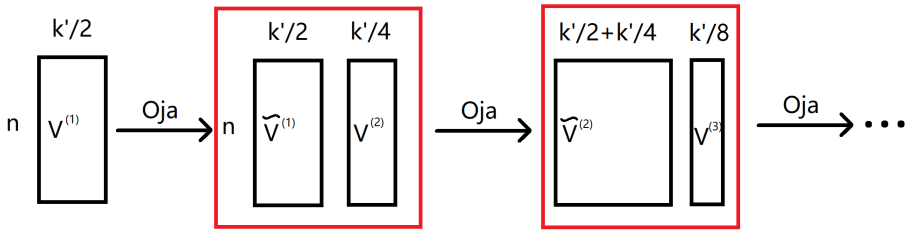

Figure 2: Graph Explanation of Oja++

principal components k is greater than 1.

In the normal Oja's method introduced above, the rate of convergence is given by $O(\frac{\lambda}{gap^2 \cdot \epsilon})$. However, in practice, the eigen-gap is usually very small, making the convergent rate significantly slow[22][29]. The Oja++ algorithm solved the issue as a gap-free method by approximating the upper bound of correlation on $\boldsymbol{V}$ and the abandoned eigenvectors. Allen[1] and Hardt et al.[12] provided a gap-free theorem relating to the Oja's method.

Theorem 2 (gap-free Oja's method) *For every $\rho, \epsilon, p \in (0,1)$, let $\lambda_1, ..., \lambda_m$ be all eigenvalues of Σ that are $> \lambda_k - \rho$, let $\Lambda_1 = \sum_{i=1}^{k} \lambda_i \in (0,1], \Lambda_2 = \sum_{j=k+1}^{k+m} \in (0,1]$, define learning rates*

$$\begin{aligned} T_0 &= \tilde{\Theta}\left(\frac{k \cdot \min\{1, \Lambda_1 + \frac{k\Lambda_2}{p^2}\}}{\rho^2 \cdot p^2}\right) \\ T_1 &= \tilde{\Theta}\left(\frac{\Lambda_1 + \Lambda_2}{\rho^2}\right) \end{aligned} \tag{23}$$

$$\eta_t = \begin{cases} \tilde{\Theta}\left(\frac{1}{\rho \cdot T_0}\right) & t \leq T_0 \\ \tilde{\Theta}\left(\frac{1}{\rho \cdot T_1}\right) & t \in (T_0, T_0 + T_1] \\ \tilde{\Theta}\left(\frac{1}{\rho \cdot T_0}\right) & t > T_0 + T_1 \end{cases} \tag{24}$$

Let $\boldsymbol{W}$ be the column orthonormal matrix consisting of all eigenvectors of $\boldsymbol{\Sigma}$ with values no more than $\lambda_k - \rho$. Then the output $\boldsymbol{V}^{(T)} \in \mathbb{R}^{d \times k}$ of Oja's algorithm satisfies with probability at least $1 - p$:

$$\text{for every } T = T_0 + T_1 + \tilde{\Theta}\left(\tfrac{T_1}{\epsilon}\right), \text{ it satisfies } ||\boldsymbol{W}^T \boldsymbol{V}^{(T)}||_F^2 \leq \epsilon.$$

The above $\tilde{\Theta}$ hides poly-log factors in $\frac{1}{p}, \frac{1}{\rho}$ and d.

Allen-Zhu and Li[1] also improved the gap-free Oja's method to a more efficient algorithm, Oja++.

Theorem 3 (gap-free Oja's method) *Given $\rho \in (0,1)$, Oja++ outputs a column-orthonormal matrix $\boldsymbol{V}^{(t)} \in \mathbb{R}^{d\times k}$ with $||\boldsymbol{W}^T\boldsymbol{V}^{(T)}|| \leq \epsilon$ in $T = \tilde{\Theta}\left(\frac{\lambda_1+...+\lambda_{k+m}}{\rho^2\epsilon}\right)$ iterations*

Figure(7) in the later experiment section shows that the second principal component is much slower than the previous principal components, the first component, and the same for the rest of the principal components. Suppose that we are looking for the top k' principal components. The trick of Oja++ is that instead of finding the all the top k' principal components together in each iteration, the Oja++ starts with the $k = k'/2$ case and expands $\boldsymbol{V}$ to $k = k'/2 + k'/4$, then $k = k'/2 + k'/4 + k'/8$, ... etc.

Algorithm 2: Oja++

Input: **dataset** : $\boldsymbol{X} \in \mathbb{R}^{m\times n}$, **length** : $\{m_i\}_{i=1}^s$, **initial vector** : $\{\boldsymbol{V}_0^i \in \mathbb{R}^{n\times r_i}\}_{i=1}^s$, **learning rate** : $\{\eta^i\}_{i=1}^s$
Output: **estimated principal component** $\hat{\boldsymbol{V}} \in \mathbb{R}^{n\times k}$
function `Oja++`:
  $\hat{\boldsymbol{V}} \leftarrow []$
  for $i \leftarrow 1, 2, 3, ..., s$ **do**
    $\hat{\boldsymbol{V}} \leftarrow [\hat{\boldsymbol{V}}, \boldsymbol{V}_0^i]$
    for $t \leftarrow 1, 2, 3, ...m_i$ **do**
      $\hat{\boldsymbol{V}} \leftarrow \hat{\boldsymbol{V}} + \eta_i \boldsymbol{X}^{t^T}\boldsymbol{X}^t\hat{\boldsymbol{V}}$
      $\boldsymbol{Q}, \boldsymbol{R} \leftarrow QR(\hat{\boldsymbol{V}})$
      $\boldsymbol{S} \leftarrow sign(diag(R))$
      $\hat{\boldsymbol{V}} \leftarrow \boldsymbol{QS}$
    end
  end
  return $\hat{\boldsymbol{V}}$
end

Algorithm(2) is how the Oja++ works. s represent the number blocks considered during the process, and $\sum_{i=1}^s r_i = k$. The length m_i is the estimated number of iterations needed for the corresponding block to converge. Note that we assume that $\sum_{i=1}^s m_i \leq m$, which means that we have sufficient sample points from the stream to make sure that the top k principal components converge.

2.4 Eigen Game

The learning process of finding the eigenvalue and eigenvectors can be considered as a strict-Nash equilibrium[10]. Each eigenvector is regarded as a separate player to maximise its respective utility function. The author builds a convergent sequential algorithm by solving each player's optimization problem in turn to solve the given PPAD-complete problem[11][8].

The most significant difference between this paper and others is that it not only maximises the variance of each principal component but also reduces the correlation within all the principal components. Mathematically, suppose that $\boldsymbol{v} \in \mathbb{R}^{n\times k}$ is the top-k principal components for the dataset $\boldsymbol{X} \in \mathbb{R}^{m\times n}$. Then the target of the learning is to find $\hat{\boldsymbol{v}} \in \mathbb{R}^{n\times k}$ such that it achieves the maximum:

$$\max_{\hat{\boldsymbol{v}}^T\hat{\boldsymbol{v}}=1} \sum_i \boldsymbol{R}_{ii}, \tag{25}$$

and also attains the minimum:

$$\min_{\hat{\boldsymbol{v}}^T\hat{\boldsymbol{v}}=1} \sum_{i\neq j} \boldsymbol{R}_{ij}, \tag{26}$$

where $\boldsymbol{R} = \hat{\boldsymbol{v}}^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}$. Note that $\boldsymbol{R}$ is semi-positive definite, the ideal minimum value is 0.

Theorem 4 (PCA Solution is the Unique strict-Nash Equilibrium[10]) *Assume that the top-k eigenvalues of $\boldsymbol{X}^T\boldsymbol{X}$ are positive and distinct. Then the top-k eigenvectors form the unique strict-Nash equilibrium of the proposed game in*

$$\max_{\hat{\boldsymbol{v}}_i^T\hat{\boldsymbol{v}}_i=1} \{u(\hat{\boldsymbol{v}}_i|\hat{\boldsymbol{v}}_{j<i}) = \hat{\boldsymbol{v}}_i^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}_i - \sum_{j<i} \frac{(\hat{\boldsymbol{v}}_i^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}_j)^2}{\hat{\boldsymbol{v}}_j^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}_j}\} \tag{27}$$

where $u(a_i|b)$ represents that the ith player adjust a_i to maximise a utility condition on b.

By employing the generalized Gram-Schmidt[4], the derivative of the utility function(27) is given by:

$$\begin{aligned} \nabla_{\hat{\boldsymbol{v}}_i} u(\hat{\boldsymbol{v}}_i|\hat{\boldsymbol{v}}_{j<i}) =& 2\boldsymbol{X}^T\boldsymbol{X}[\hat{\boldsymbol{v}}_i - \sum_{j<i} \frac{(\hat{\boldsymbol{v}}_i^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}_j)^2}{\hat{\boldsymbol{v}}_j^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}_j}\hat{\boldsymbol{v}}_j] \\ =& 2\boldsymbol{X}^T[\boldsymbol{X}\hat{\boldsymbol{v}}_i - \sum_{j<i} \frac{(\hat{\boldsymbol{v}}_i^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}_j)^2}{\hat{\boldsymbol{v}}_j^T\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{v}}_j}\boldsymbol{X}\hat{\boldsymbol{v}}_j]. \end{aligned} \tag{28}$$

The first term in the bracket is recognised as the reward, and the second term is the penalty. The gradient also agrees with Oja's algorithms[23][24]. Together, the EigenGame algorithm is developed in Algorithm (3).

Algorithm 3: EigenGame

Input: dataset : $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, **initial vector** : $\boldsymbol{V}_0 \in \mathbb{R}^{n \times k}$, **learning rate** : η

Output: estimated principal component $\hat{\boldsymbol{V}} \in \mathbb{R}^{n \times k}$

function `EigenGame`**:**

$\hat{\boldsymbol{V}} \leftarrow \boldsymbol{V}_0$

for $t \leftarrow 1, 2, 3, ..., m$ **do**

$reward \leftarrow \boldsymbol{X}^{(t)} \hat{\boldsymbol{V}}$

$penalty \leftarrow \sum_{j<i} \frac{(\hat{\boldsymbol{V}}_i^T \boldsymbol{X}^T \boldsymbol{X} \hat{\boldsymbol{V}}_j)^2}{\hat{\boldsymbol{V}}_j^T \boldsymbol{X}^T \boldsymbol{X} \hat{\boldsymbol{V}}_j}$

$\nabla_{\hat{\boldsymbol{V}}_i} \leftarrow 2\boldsymbol{X}^{(t)^T}(reward - penalty)$

$\nabla^R_{\hat{\boldsymbol{V}}_i} \leftarrow \nabla_{\hat{\boldsymbol{V}}_i} - (\nabla^T_{\hat{\boldsymbol{V}}_i} \hat{\boldsymbol{V}}_i)\hat{\boldsymbol{V}}_i$

$\hat{\boldsymbol{V}}_i \leftarrow \hat{\boldsymbol{V}}_i + \eta \nabla^R_{\hat{\boldsymbol{V}}_i}$

$\boldsymbol{Q}, \boldsymbol{R} \leftarrow QR(\hat{\boldsymbol{V}})$

$\boldsymbol{S} \leftarrow sign(diag(R))$

$\hat{\boldsymbol{V}} \leftarrow \boldsymbol{QS}$

end

return $\hat{\boldsymbol{V}}$

end

2.5 Some Other Streaming PCA Methods

As described above, streaming PCA is quite a vast topic in big data. There are also many other versions of the online methods. Many research groups made contributions to the online PCA, and they have developed many different algorithms.

Hardt et al.[12] modified the Oja's algorithm by introducing blocks techniques. However, their result has disadvantage in the gap-dependence. Li et al. [17] also solved the block variant of Oja's method in $k > 2$ cases. Shamir[29] developed a locally efficient version of the Oja's method but his algorithm highly depends on the initial choice of $\boldsymbol{V}$. If the initial matrix is uncorrelated with the eigenvectors, the method will suffer from a significantly low convergent rate.

2.6 Canonical Correlation analysis

The Canonical Correlation Analysis (CCA) is a Principal Components Analysis extension[14]. CCA explores the elements that represent the most relevant correlated features rather than looking for the most critical component of a specific dataset.

In mathematics, assume that there are two datasets, $\boldsymbol{X} \in \mathbb{R}^{m\times p}$ and $\boldsymbol{Y} \in \mathbb{R}^{m\times q}$, both are zero-mean and m-size. The target of CCA is to find two normalised basis $\boldsymbol{u}$ and $\boldsymbol{v}$ such that

$$\max_{\boldsymbol{u},\boldsymbol{v}} \quad \mathbf{Cov}\left(\boldsymbol{X}^T\boldsymbol{u}, \boldsymbol{Y}^T\boldsymbol{v}\right). \tag{29}$$

Then the problem can be reformulated to a maximisation problem by introducing the Lagrange multiplier to it

$$L = \mathbf{Cov}\left(\boldsymbol{X}^T\boldsymbol{u}, \boldsymbol{Y}^T\boldsymbol{v}\right) + \lambda_1\left(\mathbf{Var}\left(\boldsymbol{X}^T\boldsymbol{u}\right) - 1\right) + \lambda_2\left(\mathbf{Var}\left(\boldsymbol{Y}^T\boldsymbol{v}\right) - 1\right). \tag{30}$$

Since both $\boldsymbol{X}$ and $\boldsymbol{Y}$ are zero-mean data. By replacing the variance and covariance with dot products,

$$L = \boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{v} + \lambda_1\left(\boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} - 1\right) + \lambda_2\left(\boldsymbol{v}^T\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{v} - 1\right). \tag{31}$$

Taking derivative of (31) with respect to a and b,

$$\frac{\partial L}{\partial \boldsymbol{u}} = \boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{v} - \rho_1\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u}, \tag{32}$$

$$\frac{\partial L}{\partial \boldsymbol{v}} = \boldsymbol{Y}^T\boldsymbol{X}\boldsymbol{u} - \rho_2\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{v}, \tag{33}$$

where $\rho_1 = -\frac{\lambda_1}{2}$ and $\rho_2 = -\frac{\lambda_2}{2}$. In order to reach the optimised point, the derivative is set to zero and the following relationship is obtained

$$\rho_1\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} = \boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{v}, \tag{34}$$

$$\rho_2\boldsymbol{Y}^T\boldsymbol{Y}\boldsymbol{v} = \boldsymbol{Y}^T\boldsymbol{X}\boldsymbol{u}. \tag{35}$$

This can Formalised in matrix equation as

$$\begin{pmatrix} 0 & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & 0 \end{pmatrix}\begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} = \begin{pmatrix} \boldsymbol{C}_{11} & 0 \\ 0 & \boldsymbol{C}_{22} \end{pmatrix}\begin{pmatrix} \rho_1\boldsymbol{u} \\ \rho_2\boldsymbol{v} \end{pmatrix}, \tag{36}$$

where C_{12} is the covariance of data set $\boldsymbol{X}$ and $\boldsymbol{Y}$, $\boldsymbol{C}_{11}$ and $\boldsymbol{C}_{22}$ are the variance of X and Y respectively. Solving the equation (34) by multiplying a vector a to both side, we obtain

$$\rho_1\boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{v} \tag{37}$$

Since $\boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{u} = 1$, then

$$\rho_1 = \boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{v} \tag{38}$$

which is the maximised correlation. Similarly, solving (35),

$$\rho_2 = \boldsymbol{v}^T\boldsymbol{Y}\boldsymbol{X}\boldsymbol{u}. \tag{39}$$

Note that

$$\rho_1 = \boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{v} = \left(\boldsymbol{v}^T\boldsymbol{Y}\boldsymbol{X}\boldsymbol{u}\right)^T = \rho_2^T = \rho_2 = \rho. \tag{40}$$

Therefore, by rearrange the equation(37), $\boldsymbol{u} = \frac{\boldsymbol{C}_{11}^{-1}\boldsymbol{C}_{12}\boldsymbol{v}}{\rho}$ and $\boldsymbol{v} = \boldsymbol{C}_{22}^{-1/2}\boldsymbol{f}$. Substituting into equation(40)

$$\rho^2 \boldsymbol{f} = \boldsymbol{C}_{22}^{-1/2}\boldsymbol{C}_{21}\boldsymbol{C}_{11}^{-1}\boldsymbol{C}_{21}\boldsymbol{C}_{22}^{-1/2}\boldsymbol{f}. \tag{41}$$

This indicates that ρ^2 is an eigenvector of $\boldsymbol{C}_{22}^{-1/2}\boldsymbol{C}_{21}\boldsymbol{C}_{11}^{-1}\boldsymbol{C}_{21}\boldsymbol{C}_{22}^{-1/2}$. Since

$$\boldsymbol{C}_{22}^{-1/2}\boldsymbol{C}_{21}\boldsymbol{C}_{11}^{-1}\boldsymbol{C}_{21}\boldsymbol{C}_{22}^{-1/2} = \left(\boldsymbol{C}_{11}^{-1/2}\boldsymbol{C}_{21}\boldsymbol{C}_{22}^{-1/2}\right)^T \left(\boldsymbol{C}_{11}^{-1/2}\boldsymbol{C}_{21}\boldsymbol{C}_{22}^{-1/2}\right), \tag{42}$$

ρ is the eigenvector of $\boldsymbol{C}_{11}^{-1/2}\boldsymbol{C}_{21}\boldsymbol{C}_{22}^{-1/2}$.

Finally, the theoretical way to compute the canonical correlation component is to find the eigenvector of the matrix $\boldsymbol{C}_{11}^{-1/2}\boldsymbol{C}_{21}\boldsymbol{C}_{22}^{-1/2}$ and then find the inverse square root of the covariance matrices. However, when computing the matrix inverse and square root, the computational cost increases dramatically for a large data set. As a result, most recent research has focused on assessing optimization error for the generalised eigenvalue problem using the power iteration method[32][2].

2.7 GenOja

Gen-Oja [3] as a streaming CCA algorithm is a way to solve the issue. The method is an iterative technique in which two associated sequences are updated at each time step. The idea starts from the theoretical CCA problem, by substituting equation(40) into the matrix equation (36),

$$\begin{pmatrix} 0 & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} = \rho \begin{pmatrix} \boldsymbol{C}_{11} & 0 \\ 0 & \boldsymbol{C}_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix}. \tag{43}$$

By a simple matrix inverse and multiplication, the equation(43) can be transformed to an Oja-like matrix problem:

$$\rho \boldsymbol{l} = \boldsymbol{A}\boldsymbol{l} \tag{44}$$

where $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{C}_{11} & 0 \\ 0 & \boldsymbol{C}_{22} \end{pmatrix}^{-1} \begin{pmatrix} 0 & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & 0 \end{pmatrix}$ and $\boldsymbol{l} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix}$

However, computing the inverse of matrix is computationally expensive especially in the case of high-dimensional dataset. Therefore, an extract step is required to solve the equation (43). Multiplying both side by a transpose of $\boldsymbol{l}$ gives

$$\boldsymbol{l}^T \boldsymbol{B}(\rho \boldsymbol{l}) - \boldsymbol{l}^T \boldsymbol{A}\boldsymbol{l} = 0 \tag{45}$$

Let

$$J = \boldsymbol{l}^T \boldsymbol{B}(\rho \boldsymbol{l}) - \boldsymbol{l}^T \boldsymbol{A}\boldsymbol{l} \tag{46}$$

minimisation of J can be done by an iterative SGD method with derivative

$$\frac{dJ}{d\boldsymbol{l}} = \boldsymbol{B}\boldsymbol{w} - \boldsymbol{A}\boldsymbol{l} \tag{47}$$

Therefore, iteratively updating two connected sequences $(\boldsymbol{w}^t, \boldsymbol{l}^t)$ at the same time is how the GenEig algorithm works (in algorithm (4)). In the first step of the algorithm, a stochastic gradient descent with constant step-size is used to update $\boldsymbol{w}^t$ in order to minimise J. The second step employs Oja's algorithm to update $\boldsymbol{l}^t$.

Algorithm 4: GenOja streaming CCA

Input: X, Y, η_1, η_2
Output: canonical correlation components
function `GenOja algorithm`**:**
  Generate random w^0, l^0
  for $t \leftarrow 1, 2, 3, ..., N$ **do**
    $x \leftarrow X[t]$
    $y \leftarrow Y[t]$
    $w^t \leftarrow w^{t-1} - \eta_1(yw^{t-1} - xl^{t-1})$
    $v^t \leftarrow l^{t-1} + \eta_2 w^t$
    $l^t \leftarrow \frac{l^t}{||l^t||_2}$
  end
  return l^N
end

2.7.1 GenOja with High-dimensional Output

In the previous section, the GenOja method with 1D output is introduced. Although the algorithm solves the streaming CCA problem with a highly efficient method, it does not provide additional canonical components other than the first one. Sometimes the second, the third, or even more components are required to solve specific issues. In such a case, a high-dimensional streaming CCA is required. The method should solve the maximisation problem:

$$\arg\max Cov(\langle a_i, X\rangle, \langle b_i, Y\rangle) \tag{48}$$

under two constraints:

$$\textbf{Constraint(1)} \quad \boldsymbol{u}_i \perp \boldsymbol{u}_1, .., \boldsymbol{u}_{i-1} \textbf{ and } \boldsymbol{v}_i \perp \boldsymbol{v}_1, ..., \boldsymbol{v}_{i-1} \tag{49}$$

$$\textbf{Constraint(2)} \quad Var(\boldsymbol{X}\boldsymbol{u}_i) = Var(\boldsymbol{Y}\boldsymbol{v}_i) = 1 \tag{50}$$

In comparison to the original problem, a new constraint (50) has been added. Because of this requirement, all components are orthogonal to one another. In other words, if the first canonical correlation component $\boldsymbol{u}_1, \boldsymbol{v}_1$ in space $\mathbb{R}^n$ and $\mathbb{R}^m$ has already been determined, the second canonical correlation component $\boldsymbol{u}_2$ and $\boldsymbol{v}_2$ are vectors in space $\mathbb{R}^n \backslash \boldsymbol{u}_1$ and $\mathbb{R}^m \backslash \boldsymbol{v}_1$ on which the projections of X and Y are maximised.

A QR factorization is used in the process, as described in Oja's approach, to ensure that each component is orthogonal to the others. The GenOja technique, on the other hand, combines the components of two datasets into a single matrix. We cannot carry out the factorization directly in this scenario, as that would orthogonalise a meaningless matrix. Before the QR factorization, a prepossess is to partition $\boldsymbol{U}$ and $\boldsymbol{V}$ and orthogonalise them. In the final step, concatenate $\boldsymbol{U}$ and $\boldsymbol{V}$ to generate a new $\boldsymbol{M}$.

Algorithm 5: GenOja streaming CCA with multiple-dimensional output

Input: X, Y, n, η_1, η_2
Output: canonical correlation components
function `GenOja algorithm`**:**
  Generate random $W^{(0)}, M^{(0)}$
  for $t \leftarrow 1, 2, 3, ..., N$ **do**
    $x \leftarrow X[t]$
    $y \leftarrow Y[t]$
    $W^{(t)} \leftarrow W^{(t-1)} - \eta_1(yW^{(t-1)} - xM^{(t-1)})$
    $M^{(t)} \leftarrow M^{(t-1)} + \eta_2 W^{(t)}$
    $\begin{pmatrix} U^{(t)} \\ V^{(t)} \end{pmatrix} \leftarrow M^{(t)}$
    $U^{(t)}, R \leftarrow QR(U^{(t)})$
    $V^{(t)}, R \leftarrow QR(V^{(t)})$
    $M^{(t)} \leftarrow \begin{pmatrix} U^{(t)} \\ V^{(t)} \end{pmatrix}$
  end
  return $M^{(t)}$
end

2.8 Some Other Streaming CCA algorithms

Many other online CCA algorithms have also been developed in the past decade. Lu et al.[18] and Ma er al.[20] solved the optimisation problem by introducing least square method. Ge et al.[9] developed CCALin algorithm which solves the $k > 1$ case with global convergence. Allen-Zhu et la.[2] employing double stochastic method to the oline problem and developed their gap-free LazyCCA algorithm.

3 A New Streaming Approach to CCA

In the world of big data, an enormous amount of data is needed to process. Thus the efficiency of the method is one of the most desirable properties of the algorithm. The online CCA algorithm is an ideal replacement since the traditional CCA approach is significantly inefficient in big data problems. This section will present a newly developed streaming CCA approach to solving the above issue.

The method follows the equation(31), where the Lagrange Multiplier is introduced to the optimisation problem as the constraint of the basis. Since maximising L is equivalent to minimizing $-L$. The minimising process can be achieved by the gradient descent method. Since the data is coming in as an endless stream, the stochastic gradient descent(SGD) method is an ideal way to solve the problem.

3.1 Gradient Descent and Stochastic Gradient Descent

Gradient descent (GD) is an iterative first-order optimisation process for locating a function's local minimum and maximum and is commonly used in data science problems as a tool to optimise the loss functions[26].

For a given loss function, $L(\theta|\boldsymbol{X})$ with a known gradient $\nabla_\theta L(\theta|\boldsymbol{X})$. Suppose that θ^* minimise the loss function L. Then θ^* satisfies:

$$\nabla_\theta L(\theta^*|\boldsymbol{X}) = 0. \tag{51}$$

The value of θ^* can be approximated by an iterative relationship:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta^{(i)}} L(\theta^{(i)}|\boldsymbol{X}), \tag{52}$$

where η is the learning rate. In each step, the value of $\theta^{(i)}$ is updated according to the derivative of L at $\theta^{(i-1)}$. The step size decreases as $\theta^{(i)}$ getting closer to the theoretical minimum value θ^*.

However, the computational cost of the gradient descent is linearly dependent on the size of $\boldsymbol{X}$. Any sample size-dependent methods are not suitable for the big problem due to the limitation in the computer capability. An alternative way of solving the gradient descent is the stochastic gradient descent(SGD). The SGD uses random samples compared to the standard gradient descent method in each iteration. Instead of computing $L(\theta^{(i)}|\boldsymbol{X})$, in SGD we only have to compute $L(\theta^{(i)}|\boldsymbol{X}^{(i)})$.

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta^{(i)}} L(\theta^{(i)}|\boldsymbol{X}^{(i)}) \tag{53}$$

The cost is decreased to $\frac{||\boldsymbol{X}^{(i)}||}{||\boldsymbol{X}||}$ of the original cost.

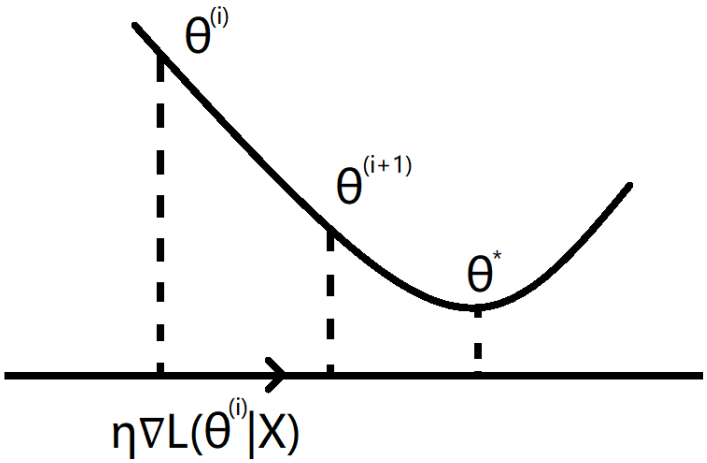

Figure 3: Graph demonstration of an arbitrary step in the gradient descent method. Suppose that $\hat{\theta}$ is at the value $\theta^{(i)}$. Since the gradient at $\theta^{(i)}$ is negative, the first step moves in the positive direction with step size $\eta \nabla_{\theta^{(i)}} L(\theta^{(i)}|\boldsymbol{X})$

Algorithm 6: SGD

Input: $\theta^{(0)}, \eta$
Output: $\theta^{(N)}$
function `SGD`**:**
  for $t \leftarrow 0, 1, 2, 3, ..., N$ **do**
    $\boldsymbol{X}^{(t)} \leftarrow$ **random points in** $\boldsymbol{X}$
    $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta L(\theta^{(t)}|\boldsymbol{X}^{(t)})$
  end
  return $\theta^{(N)}$
end

3.2 Algorithm Explanation

In the streaming situation, SGD is the most appropriate optimization strategy. Because streaming data arrives in real time, the computers or the servers will run out of storage for the entire dataset, and the computing cost will rise exponentially as more data arrives. As a result, a stochastic strategy that only uses newly observed data in the stream, such as SGD, would be more appropriate and efficient. Instead of computing a new L based on the previous sample, the computer only has to process the freshly arriving data from the stream to update the target parameters, thanks to the stochastic technique.

In order to obtain the greatest value by employing the SGD technique, it is necessary to find the derivative of the four parameters, $\boldsymbol{u}, \boldsymbol{v}, \lambda_1$ and λ_2, from equation (31):

$$\frac{\partial L}{\partial \boldsymbol{u}} = \boldsymbol{X}^T \boldsymbol{Y} \boldsymbol{v} + 2\lambda_1 \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{u}, \tag{54}$$

$$\frac{\partial L}{\partial \boldsymbol{v}} = \boldsymbol{Y}^T \boldsymbol{X} \boldsymbol{u} + 2\lambda_2 \boldsymbol{Y}^T \boldsymbol{Y} \boldsymbol{v}, \tag{55}$$

$$\frac{\partial L}{\partial \lambda_1} = \boldsymbol{u}^T \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{u} - 1, \tag{56}$$

$$\frac{\partial L}{\partial \lambda_2} = \boldsymbol{v}^T \boldsymbol{Y}^T \boldsymbol{Y} \boldsymbol{v} - 1. \tag{57}$$

Since we want the maximised value instead of minimised value, we define the loss function as $-L$. With the negative sign, the SGD solves the maximization problem instead of minimization. By substituting the derivatives in the previous part into equation(53),

$$\boldsymbol{u}^{(n+1)} = \boldsymbol{u}^{(n)} + \eta_1 \left(\boldsymbol{X}^{(n)^T} \boldsymbol{Y}^{(n)} \boldsymbol{v}^{(n)} + 2\lambda_1^{(n)} \boldsymbol{X}^{(n)^T} \boldsymbol{X}^{(n)} \boldsymbol{u}^{(n)} \right), \tag{58}$$

$$\boldsymbol{v}^{(n+1)} = \boldsymbol{v}^{(n)} + \eta_1 \left(\boldsymbol{Y}^{(n)^T} \boldsymbol{X}^{(n)} \boldsymbol{u}^{(n+1)} + 2\lambda_2^{(n)} \boldsymbol{Y}^{(n)^T} \boldsymbol{Y}^{(n)} \boldsymbol{v}^{(n)} \right), \tag{59}$$

$$\lambda_1^{(n+1)} = \lambda_1^{(n)} + \eta_2 \left(\boldsymbol{u}^{(n+1)^T} \boldsymbol{X}^{(n)^T} \boldsymbol{X}^{(n)} \boldsymbol{u}^{(n+1)} - 1 \right), \tag{60}$$

$$\lambda_2^{(n+1)} = \lambda_2^{(n)} + \eta_2 \left(\boldsymbol{v}^{(n+1)^T} \boldsymbol{Y}^{(n)^T} \boldsymbol{Y}^{(n)} \boldsymbol{v}^{(n+1)} - 1 \right). \tag{61}$$

3.3 Algorithm

Algorithm 7: SGD Lagrange CCA

Input: $X \in \mathbb{R}^{m \times n_1}, Y \in \mathbb{R}^{m \times n_2}, \eta_1, \eta_2$
Output: canonical correlation components
function `SGD Lagrangian CCA`**:**
  Generate random $u^{(0)}, v^{(0)}$
  Initialise λ_1, λ_2
  for $t \leftarrow 1, 2, 3, ..., m$ **do**
    $x \leftarrow X[t]$
    $y \leftarrow Y[t]$
    $u^{(t+1)} \leftarrow u^{(t)} + \eta_1(x^T y v^{(t)} + \lambda_1(x^T x u^{(t)}))$
    $v^{(t+1)} \leftarrow v^{(t)} + \eta_1(y^T x u^{(t+1)} + \lambda_2(y^T y v^{(t)}))$
    $\lambda_1^{(t+1)} \leftarrow \lambda_1^{(t)} + \eta_2(u^{(t+1)^T} x^T x u^{(t+1)} - I)$
    $\lambda_2^{(t+1)} \leftarrow \lambda_2^{(t)} + \eta_2(v^{(t+1)^T} y^T y v^{(t+1)} - I)$
  end
  return $u^{(m)}, v^{(m)}, \lambda_1^{(m)}, \lambda_2^{(m)}$
end

3.4 Implementation

The algorithm is implemented in Python as it provides a robust matrix computation library $NumPy$. To boost the efficiency of the computer, some repeat occurred matrices are only computed once and stored as a variable, for example, the covariance and variance matrix of $\boldsymbol{X}^{(t)}$ and $\boldsymbol{Y}^{(t)}$. The variance and covariance matrices are computed by invoking the $Numpy.outer()$ function, which performs an efficient vector outer product.

The algorithm only includes a single for loop with m iterations. The dimensions of $\boldsymbol{x}$ and $\boldsymbol{y}$ are n_1 and n_2 respectively. The covariance matrix $x^T y$ has dimension $n_1 \times n_2$. The time complexity of computing this matrix is $O(n_1 n_2)$. Similarly, the variance matrices of $\boldsymbol{x}$ and $\boldsymbol{y}$ have shapes $n_1 \times n_1$ and $n_2 \times n_2$. Therefore, the computational cost of the matrices is $O(n_1^2)$ and $O(n_2^2)$ respectively.

There are four updating steps in the algorithm. Suppose we have already computed and stored the variance and covariance matrices of $\boldsymbol{x}$ and $\boldsymbol{y}$. Then, the cost for $\boldsymbol{x}^T\boldsymbol{y}\boldsymbol{v}^{(t)}$ in the first step is $O(n_1 n_2)$ and $\boldsymbol{x}^T\boldsymbol{x}\boldsymbol{u}^{(t)}$ is $O(n_1^2)$. Therefore, the overall cost of updating $\boldsymbol{u}$ is $O(n_1^2 + n_1 n_2)$. Similarly, computing new $\boldsymbol{v}$ cost $O(n_1 n_2 + n_2^2)$. As for λ_1 and λ_2, $\boldsymbol{u}^T\boldsymbol{x}^T\boldsymbol{x}\boldsymbol{u}$ and $\boldsymbol{v}^T\boldsymbol{y}^T\boldsymbol{y}\boldsymbol{v}$ cost $O(n_1)$ and $O(n_2)$.

Without losing generality, let $n_1 > n_2$. Then we have $1 \ll n_1 < n_1 \ll m$. The overall time complexity is therefore the sum of the above introduced cost times the number of iterations, m. Therefore, the time cost is $O(mn_1^2) + O(mn_2^2) + O(mn_1 n_2)$.

4 Experiments

This section includes some experiments on previously introduced online CCA and PCA algorithms. The tests include the behaviour of different methods and analysis of learning rates.

4.1 PCA Experiments

4.1.1 Synthetic Two Moons Dataset

The first experiment is carried out on a synthetic two moons dataset. Two moons[25] is commonly used in classification researches. The original two moons data is in space $\mathbb{R}^{m\times 2}$. In order to make the data in high dimensional space, firstly, a zero matrix in $\mathbb{R}^{m\times 100}$ is combined with the dataset and then randomly rotate the data. This gives us a dataset in space $\mathbb{R}^{m\times 102}$. Since rotation does not affect the variance, the maximum variance obtained is the same as the original two moons data.

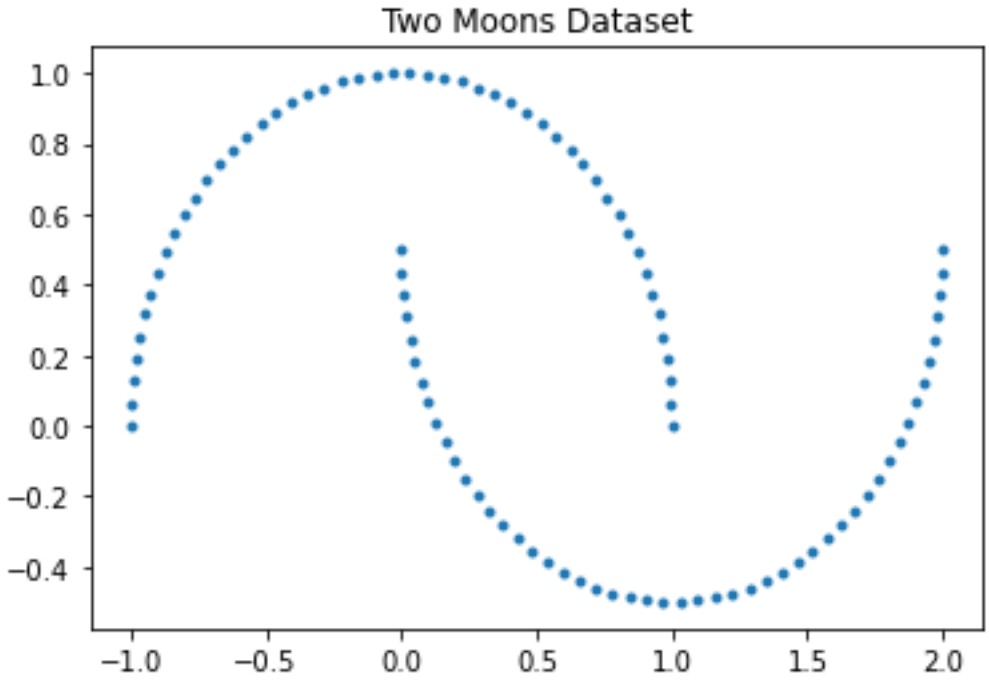

Figure 4: Example of two moons data. The data is separated into two semi-circular curves which are mutually surrounded by the other.

4.1.2 CIFAR-10 Dataset

The CIFAR-10 is a set of 5000 coloured images in 10 different classes. Each picture has 32×32 pixels, which means that data in space $\mathbb{R}^{50000\times 32\times 32\times 3}$. In order to perform matrix multiplication, the CIFAR-10 is flattened to $\mathbb{R}^{5000\times 3072}$.

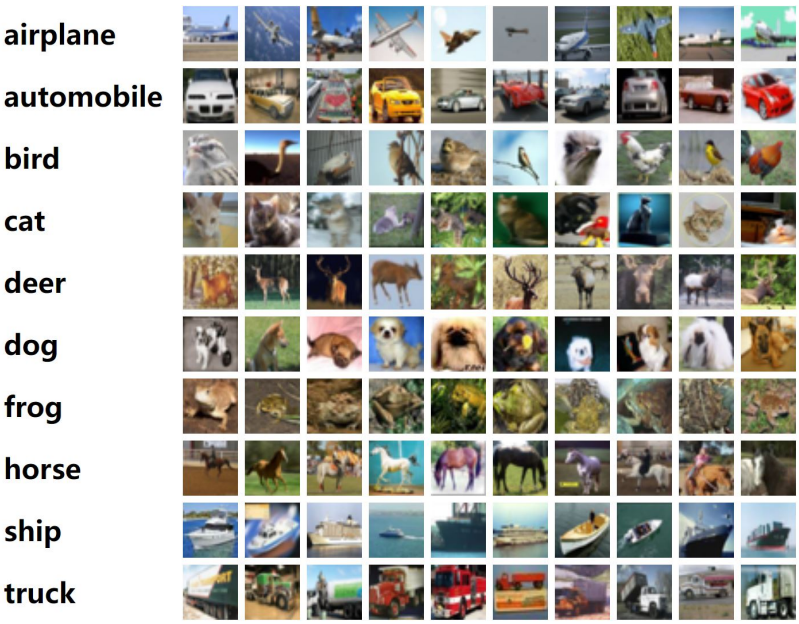

Figure 5: Examples of the CIFAR-10 dataset. There are 10 classes in total, 6 classes of animals and 4 classes of transportation tools.

CIFAR-10 is a good test dataset in PCA due to its variance ratio. Although each sample has 3072 features, the first eigenvalue makes up nearly 30% of the total, and the top 3 eigenvalues contribute 50%. In other words, approximately half of the information of CIFAR-10 is contained in the first three principal components.

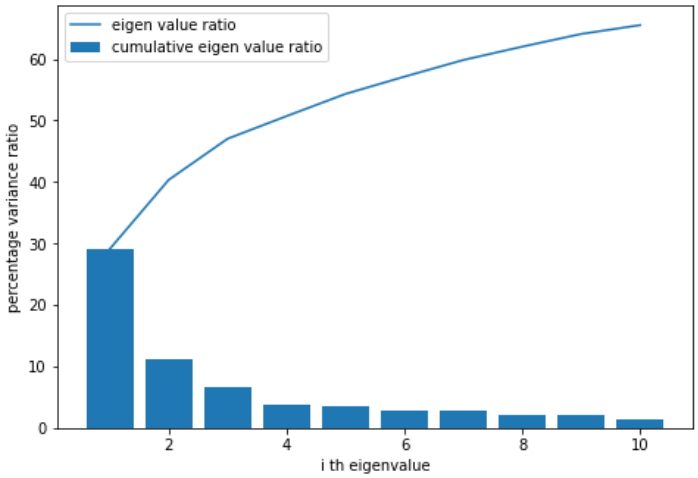

Figure 6: Eigenvaule ratio of CIFAR-10

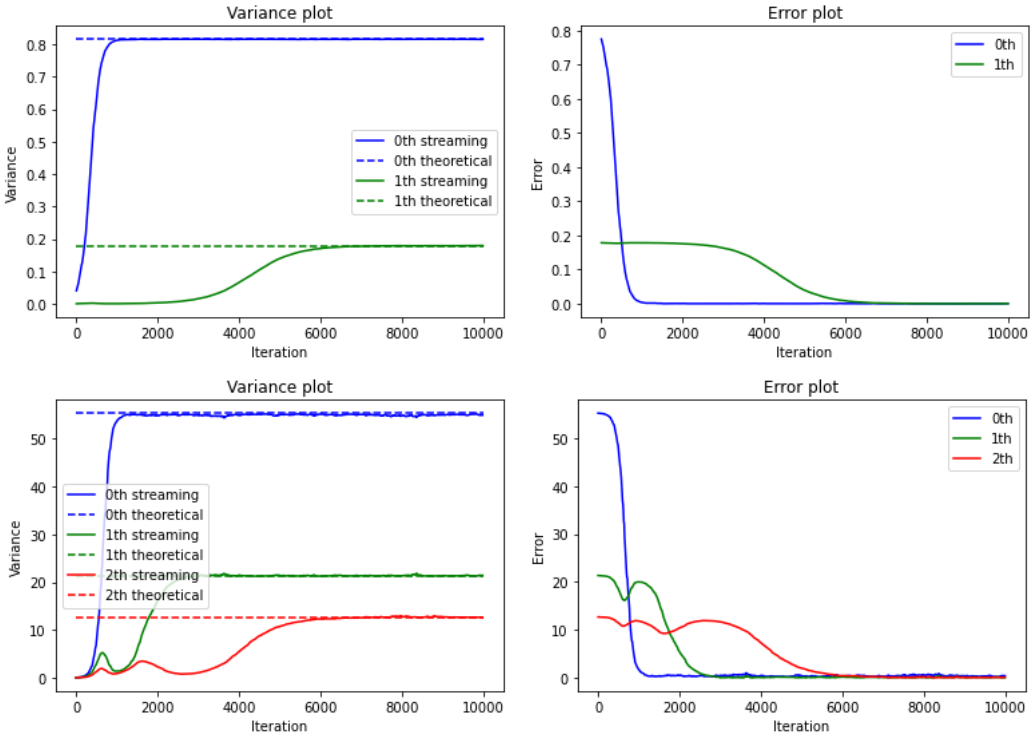

Figure 7: Performance of the Oja's method on the CIFAR-10 dataset. The top two plots are experiments on the synthetic two moons dataset while the two plots on the bottom are the CIFAR-10 plots. The left plots demonstrates the trend of the variance and the right plots shows how the error decreases as the number of iterations increases.

4.1.3 PCA Results

Figure(7) shows the behaviour of the error and the variance of the Oja's rule on two moons and CIFAR-10 respectively. The Oja's learning rule performs well on both test datasets. There is a clear view of the overall convergent trend in all plots. The theoretical maximum variance is reached in a finite number of iterations.

4.2 CCA Experiments

4.2.1 Synthetic Dataset

The first dataset is a synthetic dataset, $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{30000\times 6}$. The data is generated in the following way: Firstly, generate three independent random vector, v_1, v_2, v_3, each vector is 50000-sized and all the entries are independently generated from a standard normal distribution. Then combine the vectors together

$$v = (v_1, 0.5v_1, 0.25v_1, 0.7v_2, 0.3v_2, v_3).$$

Two independent random noise, ϵ_1, ϵ_2, are added to v separately,

$$X = \epsilon_1 + v,$$

$$Y = \epsilon_2 + v.$$

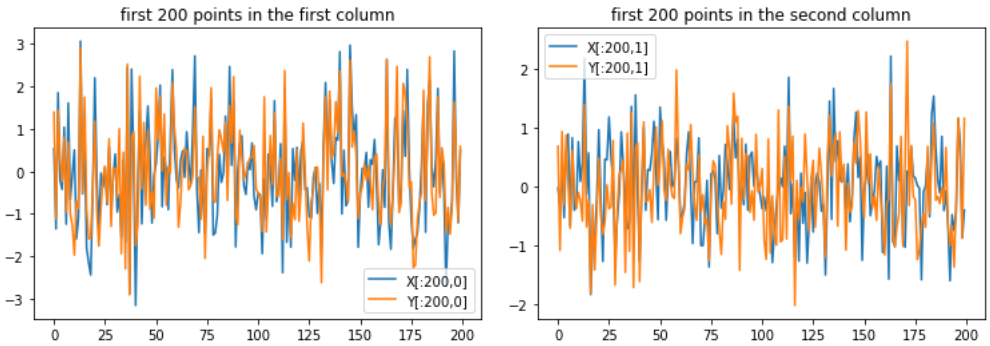

Figure 8: Synthetic dataset. The left plot shows the trend of $X[:200, 0], Y[:200, 0]$ and the right plot shows the trend of $X[:200, 1], Y[:200, 1]$. Since X and Y are generated from the same distribution but added with different noise, they showed a similar trend from the plot.

4.2.2 EMNIST Dataset

The EMNIST dataset is a collection of handwritten character digits extracted from NIST Special Database 19 and transformed into a 28x28 pixel image format with a dataset structure identical to the MNIST dataset[6]. Although the data is in the space $\mathbb{R}^{784}$, there is a significant eigen gap indicating the suitability of dimension reduction methods.

Compared with the commonly used MNIST data, the EMNIST includes more classes, not only the digits from 0 to 9 but also the alphabet characters in both lower and upper cases. Also, the EMNIST dataset is balanced so that similar numbers of samples are assigned to each class (shown in figure(9)).

In order to conduct CCA on a group of images, a common way is to select two specific areas in the picture[21] and carry out CCA to find the relationship

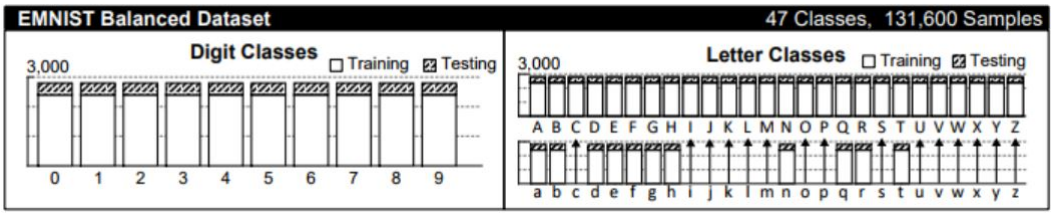

Figure 9: The EMNIST dataset. There are 47 classes in total. Some of the upper case charactors are removed due to the high similarity with the lower cases.

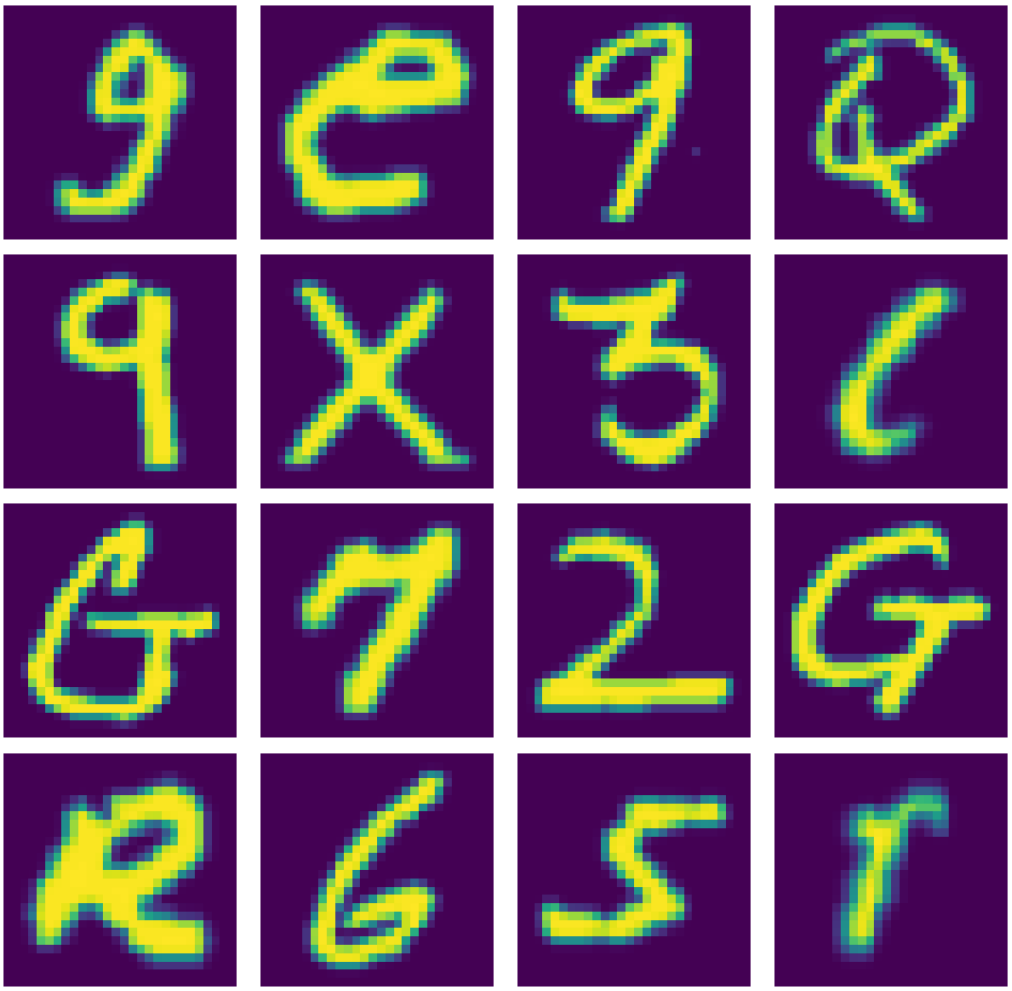

Figure 10: Some samples of EMNIST

between two groups of pixels. In this experiment, the two groups are the 300th-335th pixels and the 400th-435th pixels. The reason for choosing such groups is that these two groups are not adjacent to each other, which makes the correlation too big and too far away, making the two groups uncorrelated. Also, the group size is 36, which is suitable for a personal computer but not too small to indicate the algorithm's performance.

4.2.3 CCA Results

Figure(11) shows the behaviour of the SGD CCA algorithm. The values of λ_1 and λ_2 oscillate with a decreasing trend in the amplitude as the number of iterations increases. Also, the variance shows an increase towards the maximum variance and gets close to it in a finite number of iterations

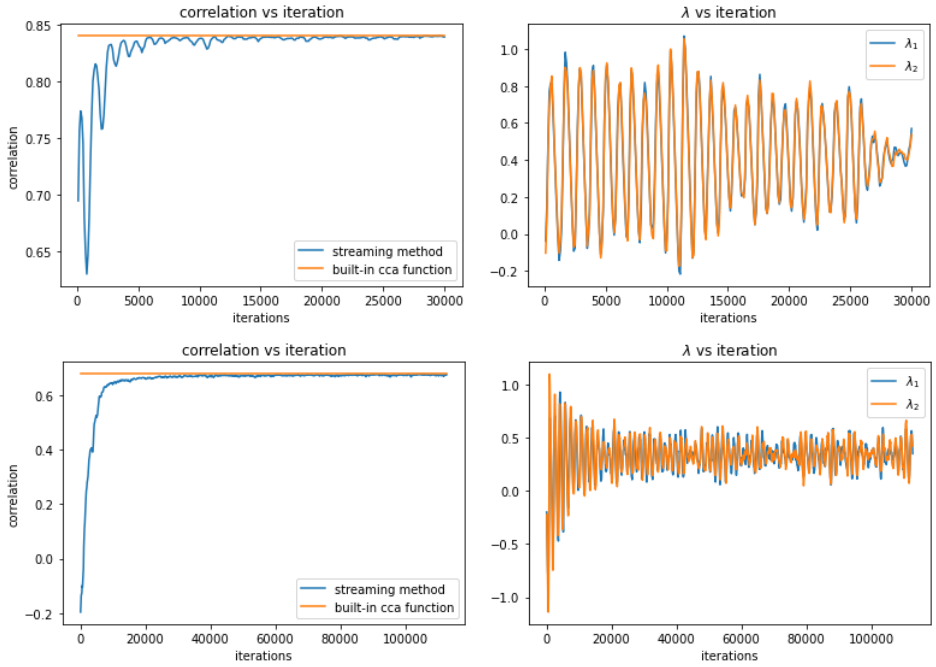

Figure 11: SGD CCA results. The first row is the experiments on the synthetic dataset while the second row is EMNIST. The first column of plots are the correlation plots and the second columns represents the trend of λ_1 and λ_2.

Figure(12) illustrates the performance of the GenOja algorithm in $k = 3$ case. All there components show an increasing trend towards the corresponding theoretical variance. However, in the synthetic dataset, the second and the third canonical correlation components reach the value of the first component before converging to its theoretical value. Also, in the EMNIST data, the components are not completely converged before reaching the maximum number of iterations(the sample size).

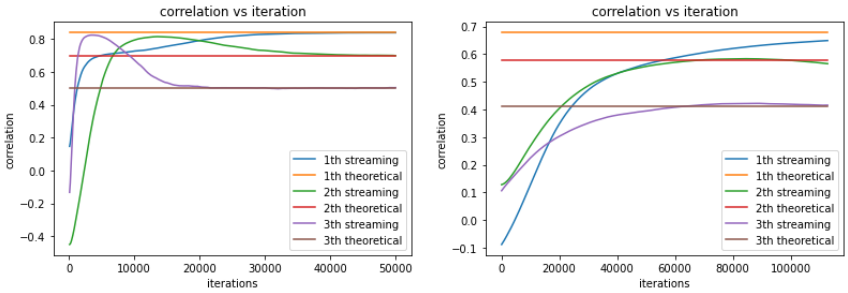

Figure 12: The left plot is the correlation curve of the GenOja algorithm with $k = 3$ on the synthetic dataset. The right plot is the correlation plot on the EMNIST with $k = 3$.

4.2.4 Learning Rate

Learning rate is an essential parameter in the machine learning problem. It controls the step size in each iteration. On the one hand, a small learning rate would lead to a slow convergence rate. On the other hand, if the learning rate is too large, the target parameter will fluctuate around the theoretical value instead of getting close to it.

Heatmap is a helpful tool for analysing the influence of multiple learning rates on the method. For each plot, the number of iterations is fixed to focus on the effects of different learning rates. Each axis represents different choices of the corresponding learning rate, and the log absolute error is shown as temperature in the plot. Since the error is supposed to lay in the range $(0, 2)$, taking log makes the change more obvious. The lighter colour in the heatmap refers to a significant error.

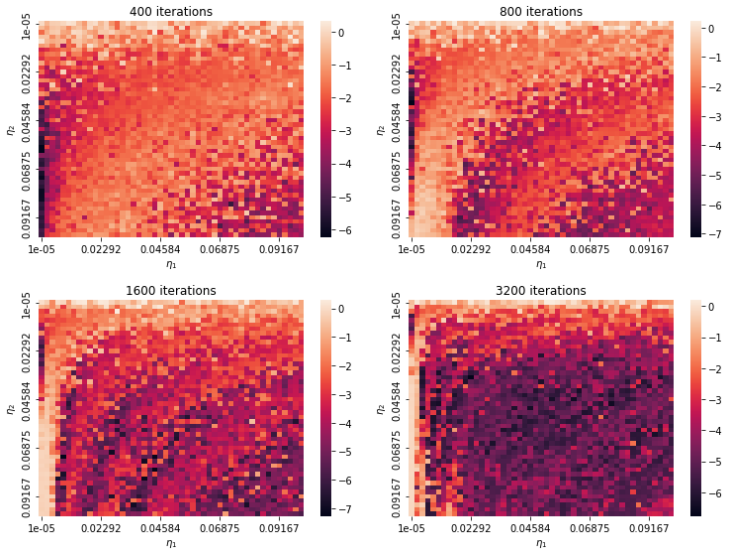

Figure 13: Heatmap of log absolute errors on the synthetic dataset with SGD CCA algorithm.

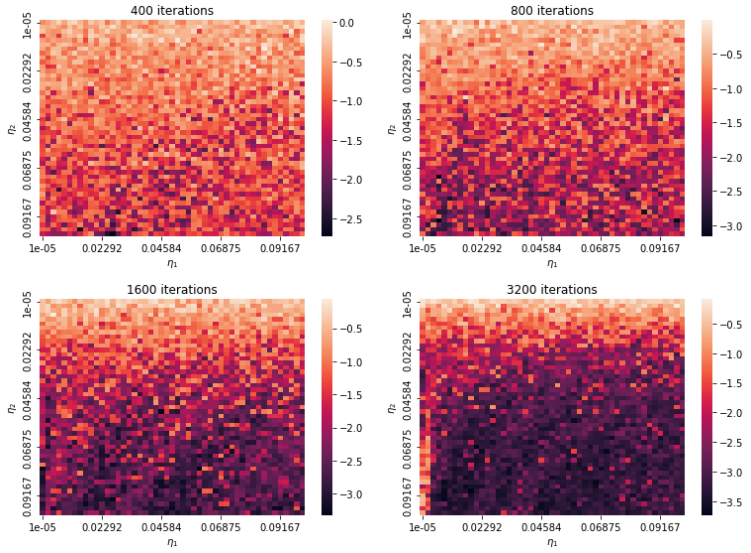

Figure 14: Heatmap of log absolute errors on the EMNIST dataset with SGD CCA algorithm.

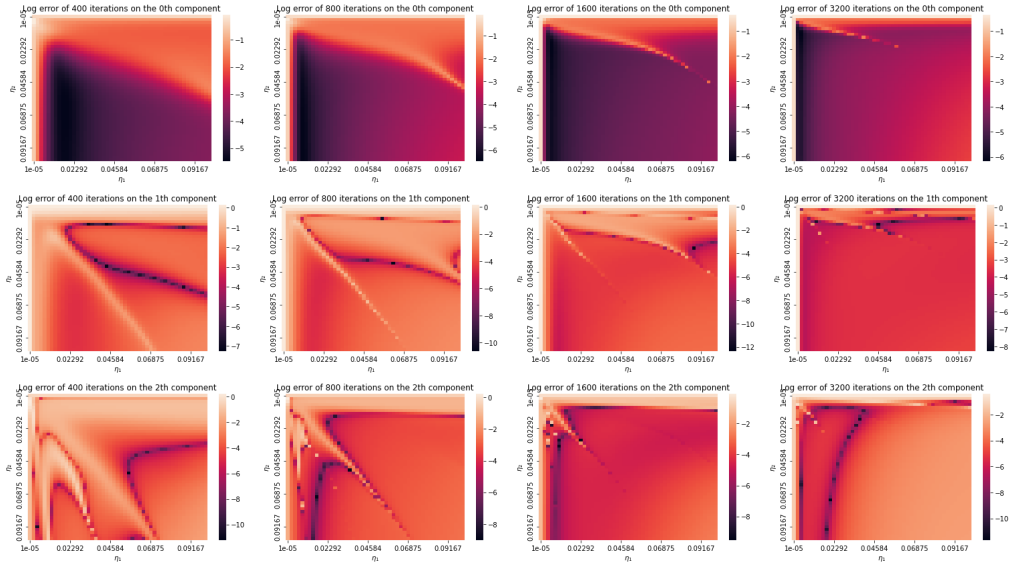

Figure 15: Heatmap of GenOja method on the synthetic dataset with $k = 3$

For both methods, the region of low error moves toward the left-top corner as the number of iterations increases. This is the same as expected. A low learning rate provides a smooth and accurate but slow convergence. In the heatmap of the GenOja algorithm, there is a repeating shift between high and low errors in the second and the third canonical components, especially for the EMNIST data at 3200 iterations. The change of sign causes this alternating pattern during the abs operation on the absolute error. As the number of iterations increases, the estimated value approaches the theoretical value and fluctuates around it.

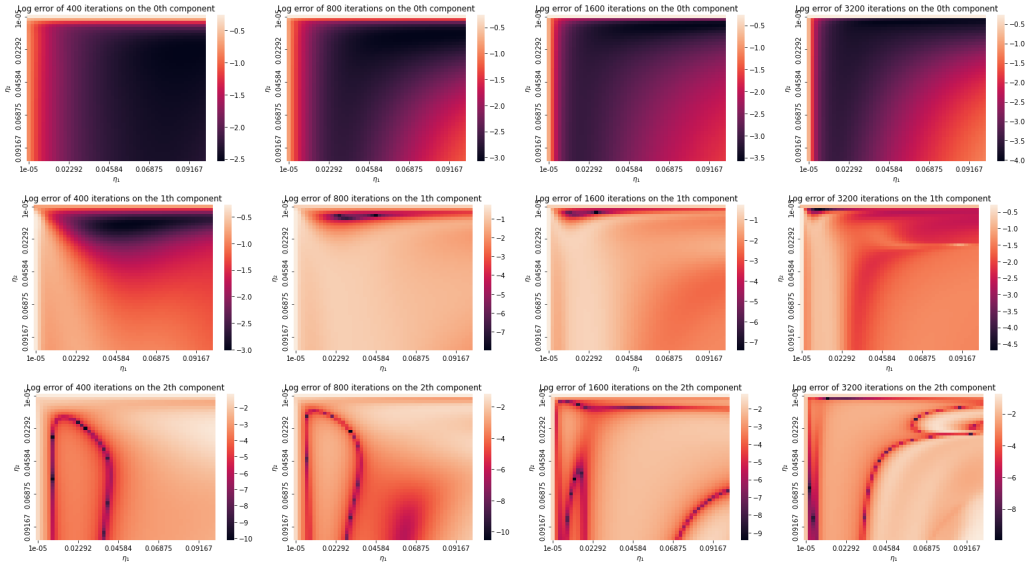

Figure 16: Heatmap of GenOja method on the EMNIST dataset with $k = 3$

A surprising phenomenon is that a small area of high error appears on the left bottom corner of the SGD CCA heatmap. This may be because of the un-convergence of the λ parameters in SGD (shown in figure(17)). As the convergence of the SGD technique is highly dependent on the choice learning rate and starting point, a bad value of η_2 may drive the parameter to a non-convex region of the function. Thus, in order to avoid the above problem, the value of η_2 should be set to a relatively small value.

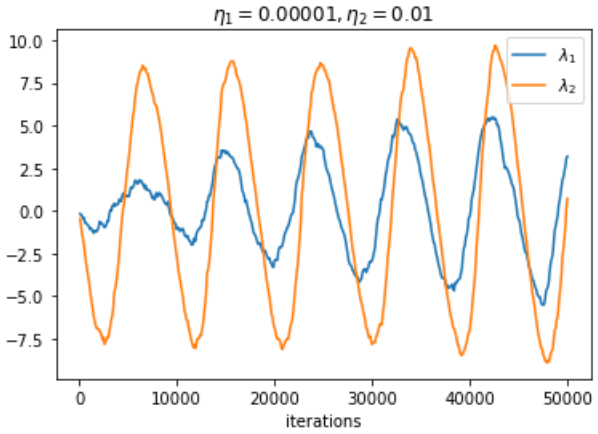

Figure 17: Trend of λ_1 and λ_2 in small η_1 and large η_2. Surprisingly, the amplitude of λ_1 is increasing instead of decreasing. Also, the amplitude of λ_2 shows a slight increase.

5 Conclusion and Further Discussion

We will wrap off this project by discussing the work we have done so far, as well as some recommendations for future research on the subject.

The main contribution of this paper is that we developed a totally new streaming CCA algorithm by employing the Lagrange multiplier and SGD technique. We also carried out some experiments to show the validation of the method and how do the learning rates affect the convergence rate.

Unfortunately, our method has limitation in the output dimensions. We have only worked out the $k = 1$ algorithm and some further work may required to extend the method into a higher dimensional output. Also, we have not deeply instigate the convergence rate and the order of error in this method.

Ideas for the further work:

- Solving the SGD problem in $k > 1$ cases. Some similar work has done in PCA i.e. EigenGame[10]. This technique may also be used in CCA.
- Finding the theoretical mathematical convergence rate of the SGD CCA and investigating the order of the error in the SGD CCA algorithm.
- Investigating the performance of the algorithm on more dataset. In this thesis, we only provide experiment on two different equal-size data. Some further experiment can be carried out on more complicated settings.

6 References

References

[1] Zeyuan Allen-Zhu and Yuanzhi Li. First efficient convergence for streaming k-pca: a global, gap-free, and near-optimal rate, 2016.

[2] Zeyuan Allen-Zhu and Yuanzhi Li. Doubly accelerated methods for faster cca and generalized eigendecomposition. *ArXiv*, abs/1607.06017, 2017.

[3] Kush Bhatia, Aldo Pacchiano, Nicolas Flammarion, Peter L Bartlett, and Michael I Jordan. Gen-oja: Simple & efficient algorithm for streaming generalized eigenvector computation. *Advances in neural information processing systems*, 31, 2018.

[4] Peter G Casazza and Gitta Kutyniok. A generalization of gram–schmidt orthogonalization generating all parseval frames. *Advances in Computational Mathematics*, 27(1):65–78, 2007.

[5] 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, 2009.

[6] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.

[7] Graham Cormode and Ke Yi. *Small Summaries for Big Data*. Cambridge University Press, 2020.

[8] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[9] Rong Ge, Chi Jin, Sham Kakade, Praneeth Netrapalli, and Aaron Sidford. Efficient algorithms for large-scale generalized eigenvector computation and canonical correlation analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2741–2750. JMLR.org, 2016.

[10] Ian Gemp, Brian McWilliams, Claire Vernade, and Thore Graepel. Eigengame: Pca as a nash equilibrium, 2020.

[11] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

[12] Moritz Hardt and Eric Price. The noisy power method: A meta algorithm with applications, 2013.

[13] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[14] Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

[15] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] Chun-Liang Li, Hsuan-Tien Lin, and Chi-Jen Lu. Rivalry of two families of algorithms for memory-restricted streaming pca. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 473–481, Cadiz, Spain, 09–11 May 2016. PMLR.

[18] Yichao Lu and Dean P Foster. large scale canonical correlation analysis with iterative least squares. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[19] Yong Luo, Dacheng Tao, Kotagiri Ramamohanarao, Chao Xu, and Yonggang Wen. Tensor canonical correlation analysis for multi-view dimension reduction. *IEEE transactions on Knowledge and Data Engineering*, 27(11):3111–3124, 2015.

[20] Zhuang Ma, Yichao Lu, and Dean Foster. Finding linear structure in large datasets with scalable canonical correlation analysis. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 169–178, Lille, France, 07–09 Jul 2015. PMLR.

[21] Zihang Meng, Rudrasis Chakraborty, and Vikas Singh. An online riemannian pca for stochastic canonical correlation analysis, 2021.

[22] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[23] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[24] Erkki Oja. Neural networks, principal components, and subspaces. *International journal of neural systems*, 1(01):61–68, 1989.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[27] Jan Rupnik and John Shawe-Taylor. Multi-view canonical correlation analysis. In *Conference on data mining and data warehouses (SiKDD 2010)*, pages 1–4, 2010.

[28] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 42–47, 2013.

[29] Ohad Shamir. Convergence of stochastic gradient descent for pca. In *International Conference on Machine Learning*, pages 257–265. PMLR, 2016.

[30] Avanidhar Subrahmanyam. Big data in finance: Evidence and challenges. *Borsa Istanbul Review*, 19(4):283–287, 2019.

[31] Cheng Tang. Exponentially convergent stochastic k-pca without variance reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

[32] Weiran Wang, Jialei Wang, Dan Garber, Dan Garber, and Nati Srebro. Efficient globally convergent stochastic optimization for canonical correlation analysis. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.