

M.Sc. Engg. (CSE) Thesis

# **Fast, Scalable and Geo-Distributed PCA Algorithm for Tall and Wide Big Data Analytics**

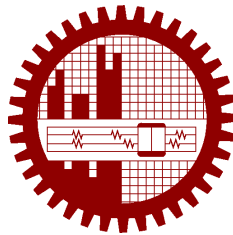
Submitted by

T. M. Tariq Adnan

1017052006

Supervised by

Dr. Muhammad Abdullah Adnan



Submitted to

**Department of Computer Science and Engineering**  
**Bangladesh University of Engineering and Technology**  
Dhaka, Bangladesh

in partial fulfillment of the requirements for the degree of  
Master of Science in Computer Science and Engineering

February 2021

## **Candidate's Declaration**

I, do, hereby, certify that the work presented in this thesis, titled, “Fast, Scalable and Geo-Distributed PCA Algorithm for Tall and Wide Big Data Analytics”, is the outcome of the investigation and research carried out by me under the supervision of Dr. Muhammad Abdullah Adnan, Associate Professor, Department of CSE, BUET.

I also declare that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.


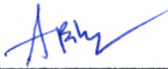
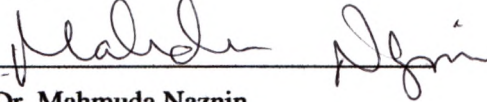
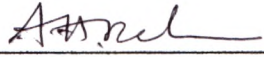

---

T. M. Tariq Adnan

1017052006

The thesis titled “**Fast, Scalable and Geo-Distributed PCA Algorithm for Tall and Wide Big Data Analytics**”, submitted by T. M. Tariq Adnan, Student ID 1017052006, Session October 2017, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents on February 23, 2021.

### Board of Examiners

1.   
\_\_\_\_\_  
Dr. Muhammad Abdullah Adnan  
Associate Professor  
Department of CSE, BUET, Dhaka  
Chairman  
(Supervisor)
2.   
\_\_\_\_\_  
Dr. A.K.M. Ashikur Rahman  
Professor and Head  
Department of CSE, BUET, Dhaka  
Member  
(Ex-Officio)
3.   
\_\_\_\_\_  
Dr. Mahmuda Naznin  
Professor  
Department of CSE, BUET, Dhaka  
Member
4.   
\_\_\_\_\_  
Dr. Atif Hasan Rahman  
Assistant Professor  
Department of CSE, BUET, Dhaka  
Member
5.   
\_\_\_\_\_  
Dr. Khandker Farid Uddin Ahmed  
Professor  
Department of Math, BUET, Dhaka  
Member  
(External)

## **Acknowledgement**

First of all, I would like to express my gratitude to the Almighty for keeping me full of vigor and vitality even during the ongoing Covid-19 pandemic and for allowing me to complete this research work and submit it for the fulfilment of my M.Sc. Engineering thesis.

I am thankful to my thesis supervisor Dr. Muhammad Abdullah Adnan for his continuous effort and guidance during this research. I am really very grateful to him for his enormous patience in times of my countless missteps, support in my work, excellent advices and constant supervision. Especially, his encouragement in my tough times was invaluable.

I thank our honorable head of the department Dr. A.K.M. Ashikur Rahman as well as the other members of my thesis committee Dr. Mahmuda Naznin, Dr. Atif Hasan Rahman, and especially the external member Dr. Khandker Farid Uddin Ahmed.

I also express my gratitude to Md. Mehrab Tanjim who is currently pursuing his PhD degree at the University of California San Diego for his best effort contribution to make this research work successful.

I would also like to express my utmost gratitude to the Department of Computer Science and Engineering (CSE) for the provision of laboratory facilities during the thesis work. I sincerely thank those CSE Office staffs who have provided logistic support to me to successfully complete the thesis work.

Finally, I would like to thank my family, my friends, and all of those who supported me with their appreciable assistance, patience, and suggestions during the course of my thesis.

Dhaka  
February 23, 2021

T. M. Tariq Adnan  
1017052006

# Contents

<b>Candidate’s Declaration</b>	<b>i</b>
<b>Board of Examiners</b>	<b>ii</b>
<b>Acknowledgement</b>	<b>iii</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Algorithms</b>	<b>ix</b>
<b>Abstract</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Underlying Challenges . . . . .	1
1.2 Thesis Objectives and Our Contribution . . . . .	2
1.3 Organization . . . . .	3
<b>2 Motivation</b>	<b>4</b>
2.1 High Dimensionality with Sparsity . . . . .	4
2.2 PCA on Geo-Distributed Datasets . . . . .	5
<b>3 Preliminaries</b>	<b>6</b>
3.1 Big Data . . . . .	6
3.2 Data Analysis . . . . .	6
3.3 Data Analysis Approaches . . . . .	7
3.3.1 Centralized Approach . . . . .	7
3.3.2 Distributed Approach . . . . .	8
3.4 Geo Distributed Data . . . . .	9
3.5 Data Sovereignty . . . . .	9
3.6 Data Cluster and Cluster Computing . . . . .	10
3.6.1 Advantages . . . . .	10
3.6.2 Disadvantages . . . . .	10

3.7	Data Parallelism	11
3.8	Model Parallelism	11
3.9	TensorFlow	11
3.10	<i>Hadoop</i> Cluster	12
3.11	<i>Hadoop</i> MapReduce	12
3.12	Spark Cluster	13
3.13	Dimensionality Reduction	13
3.14	Vector Norms	14
3.15	p-Norm	14
3.16	Frobenius Norm	15
3.17	Normal Distribution	15
3.18	Matrix Diagonalization	16
3.19	Expectation Maximization (EM) Algorithm	17
3.20	Stop Condition	17
<b>4</b>	<b>PCA as a Data Analytic Tool</b>	<b>18</b>
4.1	Assumptions Involved in PCA	19
4.2	PCA and Change of Basis	19
4.3	Eigenvalue Decomposition (EVD)	21
4.3.1	Covariance Matrix	21
4.3.2	Diagonalize the Covariance Matrix	23
4.3.3	Solving PCA in EVD Approach	24
4.4	Practical Approach of EVD	25
4.5	Singular Value Decomposition (SVD)	25
4.5.1	Performing SVD	25
4.5.2	Linking SVD with EVD	27
4.5.3	SVD for Dense Matrices	28
4.5.4	SVD for Sparse Matrices	29
4.6	Stochastic SVD (SSVD)	29
4.7	Computational Complexity	29
4.8	Probabilistic PCA (PPCA)	30
4.8.1	What is PPCA	30
4.8.2	The Probability Model	31
4.8.3	Properties of the Maximum-Likelihood Estimators	31
4.8.4	An EM Algorithm for Probabilistic PCA	32
<b>5</b>	<b>Related Works</b>	<b>34</b>
5.1	MLlib-PCA	34
5.2	Mahout-PCA	35

5.3	sPCA	35
5.4	sSketch-PCA	37
5.5	Geo-Distributed Analytics and Large Parameter	38
<b>6</b>	<b>Our Proposed Algorithm: TallnWide</b>	<b>40</b>
6.1	Handling Tall and Wide Big Data	41
6.2	Communication Efficient Calculation	44
6.3	Validation of Zero-Noise-Limit Probabilistic PCA	46
<b>7</b>	<b>Algorithm and Complexity Analysis</b>	<b>48</b>
7.1	Algorithm Description	48
7.2	Complexity Analysis	50
<b>8</b>	<b>Experimental Design</b>	<b>52</b>
8.1	Spark Implementation	52
8.2	Algorithms Compared	53
8.3	Datasets	55
8.4	Cluster Configuration	57
8.5	Performance Metrics	59
<b>9</b>	<b>Experimental Evaluation</b>	<b>61</b>
9.1	Comparison among Identified Principal Components	61
9.1.1	Reconstruction Error	61
9.1.2	Variance Explained by Principal Components	62
9.2	Evaluation for a Single DC	62
9.2.1	Running Times in a Single DC	65
9.2.2	Scalability	67
9.2.3	Intermediate Data Size	68
9.3	Evaluation for Geo-Distributed DCs	69
9.3.1	Running Times Among Accumulation Strategies	71
9.3.2	Communication Efficiency Across Different Approaches	72
<b>10</b>	<b>Conclusion</b>	<b>74</b>
	<b>References</b>	<b>75</b>
	<b>List of Publications</b>	<b>82</b>
<b>A</b>	<b>Linear Algebra</b>	<b>83</b>
	<b>Index</b>	<b>86</b>

# List of Figures

3.1	Centralized Approach . . . . .	7
3.2	Distributed Approach . . . . .	8
3.3	Data and Model Parallelism . . . . .	12
4.1	A spectrum of possible redundancies in data from the two separate measurements $r_1$ and $r_2$ . The two measurements on the left are uncorrelated because one can not predict one from the other. Conversely, the two measurements on the right are highly correlated indicating highly redundant measurements. . . . .	22
4.2	A typical 2D example. The signal and noise variances $\sigma_{signal}^2$ and $\sigma_{noise}^2$ are graphically represented by the two lines subtending the cloud of data. The largest direction of variance lie along the best-fit line . . . . .	24
4.3	Construction of the matrix form of SVD 4.3 from the scalar form 4.2 according to [1] . . . . .	27
6.1	Generation of $M$ (left), $Z$ (middle), and $XtX$ (right) in parallel fashion among all DCs. For simplicity, operations for mean vector, $\mu$ , are not shown in the middle figure. . . . .	43
6.2	Comparison of idle time among different approaches. Approach 1: Trivial Order, Approach 2: Efficient Order, and Approach 3: Efficient Order w/ Ideal Central DC. . . . .	45
6.3	Matrix dependency diagram of two variations: (a) conventional PPCA (implemented in sPCA) and (b) zero-noise-limit PPCA (implemented in TallnWide) . . . . .	46
9.1	Visualization of projected data on principal subspace identified by the comparing methods. . . . .	63
9.2	Visualization of total captured variance by the top 10 principal components identified by the comparing methods. . . . .	64
9.3	(a) Per iteration running time for all iterative methods. (b) Comparison of running time of iterative methods to converge (5% tolerance) for PubMed dataset. . . . .	66
9.4	Memory consumption in one working node for all methods for Twitter dataset. . . . .	68



# List of Tables

7.1	Comparison of complexities among all methods. Since we divide the parameters into blocks, our space complexity is less than others. Also, despite having the same time complexity w.r.t. sPCA and sSketch-PCA, we reduce the constant factors in running time of TallnWide significantly. . . . .	51
8.1	Summary table of the used Datasets . . . . .	57
8.2	Bandwidth (in MB/s) among geo-distributed DCs. . . . .	59
9.1	Comparison of reconstruction error in percentage . . . . .	62
9.2	Total Variance explained (in percentage) by top 10 Principal Components identified by the comparing methods. . . . .	62
9.3	Effect of $\rho$ on the choice of different number of blocks. . . . .	65
9.4	Comparison of running time (in sec) for TallnWide on different datasets against state-of-the-art library functions: MLib-PCA, Mahout-PCA, and sPCA. For iterative methods, we consider running time to reach convergence (5% tolerance). For full Twitter dataset, we consider 1 iteration. . . . .	67
9.5	TallnWide is space-efficient. It produces less intermediate data than state-of-the-art methods. For a fair comparison, we compare results with only those data where other methods do not face memory issues. . . . .	69
9.6	Comparison of running times (in sec) among different strategies (for <b>single iteration</b> ): gSpark-Accumulation, gHDFS-Accumulation, and TallnWide-Accumulation. . . . .	71
9.7	Average running time (in sec) needed for taking different DC as center using TallnWide-Accumulation strategy. Statistics is shown for AmazonRating and PubMed datasets. . . . .	72

# List of Algorithms

1	TallnWide . . . . .	49
2	Geo-Accumulation . . . . .	50
3	SegmentedZJob . . . . .	53
4	SegmentedYtZnZtZJob . . . . .	54

## Abstract

Principal Component Analysis (PCA) is a widely popular technique for reducing the dimensionality of a dataset. Interestingly, when dimensions of the dataset grow too large, existing state-of-the-art methods for PCA face scalability issue due to the explosion of intermediate data. Moreover, in a geographically distributed environment where most of today's data are originally generated, these methods require unnecessary data transmissions as they apply centralized algorithms for PCA and thus are proven to be inefficient. To solve these problems, we take advantage of the zero-noise-limit Probabilistic PCA model, which provably outputs the correct principal components, and introduce a block-division method for it in order to suppress the explosion of intermediate data efficiently. We employ several optimization ideas such as mean propagation for preserving sparsity, dynamic tuning of the number of blocks to automatically adjust to large dimensions, etc. Additionally, in the geo-distributed environment, we propose a communication efficient solution by reducing idle time, passing only the required parameters, and choosing geographically ideal central datacenter for faster accumulation. We refer to our algorithm as TallnWide. Our empirical evaluation with real datasets shows that TallnWide can successfully handle significantly higher dimensional data ( $10\times$ ) than existing methods, and offer up to  $2.9\times$  improvement in running time in the geo-distributed environment compared to the conventional approaches. For reproducibility and extensibility of our work, we make the source code of TallnWide publicly available at <https://github.com/tmadnan10/TallnWide>.

# Chapter 1

## Introduction

The frequency of data access at the users' end has increased by a large amount for the past few years. With such rapid generation, data that appear in many applications, such as social networks [2], product ratings [3], web documents [4], etc., often become high-dimensional. For researchers, however, big data brings new sets of challenges like how to efficiently and effectively handle big data in commodity computers, how to apply conventional algorithms, how to properly manage big data in distributed setting etc. Unfortunately, most of the machine learning algorithms cannot operate with such a high number of dimensions [5–8]. In a nutshell, to extract any meaningful insight from this big data, we need powerful tools to analyse it, elastic frameworks to cope up with its sheer volume and most importantly we have to rethink and redesign existing algorithms which are most suitable for smaller sized dataset and do not take advantage of clusters. As Principal Component Analysis (PCA) is a widely used technique for dimensionality reduction [9–12], it is often desirable to reduce the dimensions (number of columns) of such higher-dimensional datasets using PCA to make it thin (lower number of dimensions), and then apply other machine learning techniques on it. PCA can also be used for lossy-data compression [13], feature extraction [14], and data visualization [15].

### 1.1 Underlying Challenges

There are several state-of-the-art libraries that offer PCA for distributed clusters, namely: Mahout [16], MLlib [17], sPCA [18], and sSketch [19]. However, PCA algorithms implemented in these libraries are not suitable when dimensions (columns) of data grow proportionately with the number of data samples (rows). For example, Twitter Network [2] presents a dataset of 50M users' followers, which is a  $50M \times 50M$  matrix (dimension,  $D = 50M$ ). These data are quite large concerning both rows (tall) and columns (wide). We refer to such data as *tall and wide big data*. If we run sPCA or Mahout-PCA on it, for computing the first 100 principal components, it takes over **37.25GB of memory** per worker node to store the parameter alone (by parameter,

we mainly indicate the principal components or subspace). For MLib-PCA the situation is much worse. Even for  $D = 100K$ , the parameter takes roughly **74.50GB of memory per worker node**. sSketch-PCA can provide good scalability up to a certain extent, and even then, it faces memory overflow error for datasets with significantly larger dimensions (failed for  $D \approx 10M$ ). In summary, current techniques face **out-of-memory error**, and to the best of our knowledge, there is no existing solution.

On top of data being tall and wide, they are often geographically distributed as almost all the companies (e.g. Twitter, Facebook, etc.) store information in multiple geographic locations to ensure privacy and low latency at the users' end [20–23]. In this setting, we do not have a global view of such distributed data. Therefore, even if we deal with datasets with relatively smaller dimensions for which the parameter fits in the memory, we still need to gather partial datasets that are spread across different geographic locations and run existing methods on this centralized data [20, 24, 25]. Communication over regions is expensive, and some countries even prohibit passing *raw data* across national borders [22, 23].

## 1.2 Thesis Objectives and Our Contribution

Under these circumstances, at present, we need an analytic method that is capable of 1) handling tall and wide big data, and 2) performing a communication-efficient calculation in the geo-distributed environment. In this work, we propose solutions for each of the cases and present a highly scalable algorithm, namely *TallnWide*, for computing PCA. Our main contributions are as follows:

- To manage tall and wide data, we need a solution that can scale up for any arbitrarily large number of dimensions and mitigate the memory overflow error. Block-division is a right candidate solution for such a problem as it allows the computation to get divided into manageable blocks. However, not all techniques of PCA allow computation to be divided into multiple partitions/blocks. Also, dividing the computation is non-trivial because of the interlinked dependencies between matrices and various steps of matrix operations. To solve these challenges, firstly, we identify a variant (zero-noise-limit) of Probabilistic PCA (PPCA) [26], which has a much simpler dependency graph compared to conventional PPCA and can produce results in fewer steps. Because of its simplicity, it also allows the computation to be easily divided into multiple blocks. In this work, we propose a block-division algorithm for it to scale up PCA for any arbitrarily large number of dimensions and mitigate the memory overflow error. To the best of our knowledge, we are the first to give such a method for PCA.
- For a communication-efficient solution in the geo-distributed environment, we propose a scheme that seeks to minimize idle times in computation and avoid bottlenecks of

communication by transmitting only the required parameters, and not the raw data. Also, we give a formula that can choose a geographically ideal central datacenter (DC) for faster accumulation (a central DC is needed to gather partial results from all DCs to produce the final result and transmit it back). Such a scheme saves both computation and communication costs by a significant margin. We refer to the whole algorithm as TallnWide.

- For implementing TallnWide, we consider the popular memory-based distributed framework, Spark [27]. We consider various optimization and effective ideas like tuning the number of blocks dynamically and employing efficient accumulation strategy. Moreover, similar to [18] and [19], we also propagate the mean for sparsity preservation. We run extensive experiments with four real large datasets with varying sizes and values in both geo-distributed (a cluster of DCs from three different regions) and local environments (each DC as a cluster). Our experiment shows that TallnWide is well capable of handling tall and wide big data in a datacenter with commodity computing hardware and can complete execution on datasets with dimensions as high as 50M while existing methods **fail** to run on datasets where dimensions reach to 10M. Additionally, in the geo-distributed environment, our approach offers up to  $2.9\times$  improvement in running time in contrast to other alternatives.

## 1.3 Organization

The rest of the book is organized as follows. In the following Chapter 2, we discuss the primary motivations for proposing TallnWide in details. In Chapter 3, we discussed the basic terms and notations related to our study. In Chapter 4, we discussed the significance of PCA in data analytics and introduce a brief technical background of existing PCA methods. Chapter 5 discusses state-of-the-art implementations of these methods. We also point out their limitations briefly. We present the design of our proposed TallnWide in Chapter 6 and the algorithm with its complexity analysis is presented in Chapter 7. Chapter 8 shows our experimental setup, while Chapter 9 illustrates the evaluation. Finally, Chapter 10 concludes the thesis.

# Chapter 2

## Motivation

As mentioned in the introduction, there are two main problems which motivate us to propose an efficient solution for PCA. The first one is the ever-increasing number of dimensions of data, which also makes it very sparse. This requires an algorithm which can handle an arbitrary number of dimensions and preserve sparsity at the same time. The second one is dataset being by born geographically distributed, which requires a communication efficient solution. We discuss both of them below with illustrative examples and use-cases.

### 2.1 High Dimensionality with Sparsity

With the increasing rate of data generation, the number of features, i.e. attributes per data sample has also been increasing by a large scale during the last decade. This results in the data to be very high dimensional, which eventually contributes to the increment of data sparsity. As we already mentioned, a wide range of fields such as social network [2], health sector [4], e-commerce [3], bio-metric , industries, etc. are often generating high dimensional and consequently sparse data. As an illustrative example, let us consider the user-item interaction matrix, which is a crucial part of building a recommender system in various web platforms. In a user-item interaction matrix, each row denotes a new user, and each column denotes a new item. Typically, for any web service, such as large e-commerce systems, both the number of users and items are in millions, and both increase as the system sees a new user or adds a new item to inventory. However, each user interacts with only a few items resulting in a very sparse matrix. For instance, Amazon product data provided by [3] has  $21M$  users' ratings on a total of  $9.8M$  products. However, 99.99% of this dataset is sparse as most of the users rate only a few items (more details on this dataset is provided in subsection 8.3).

This example shows how dimensions of data can proliferate and how sparsity of the overall dataset can increase along with it. Therefore, efficient data analytic techniques need to be well capable of dealing with the sparsity and higher dimensionality of big data.

## 2.2 PCA on Geo-Distributed Datasets

Nowadays, data are by born geographically distributed, which has evolved the requirement of developing a geo-distributed or federated learning technique. The main principle of federated learning is simple: learn a shared model across multiple decentralized servers storing local datapoints, without exchanging them. This technique not only allows us to build a robust model with data from multiple sources but also ensures critical issues such as data privacy and security. There are many use-cases for such federated learning in various fields such as healthcare systems [28, 29], Financial Services Industry (FSI) [30–32], IoT, telecommunications [33, 34], etc.

For example, in the field of the healthcare systems, one significant usefulness of geo-distributed learning is to develop an analytical tool for measuring the effectiveness of particular treatments against groups of people all around the world. The target of such analysis is to identify the characteristic properties in patients demonstrating better and lower response. A similar analysis can be carried out in order to identify adverse drug reactions on the patients located at different geographic locations [28] or to extract significant insights from geo-distributed healthcare dataset [29]. With the help of geo-distributed data analytics, this kind of global benchmarking can analyze the patients' data at a location close to the collection spot within the geographic boundaries defined by regulatory compliance. Similar use-cases exist for Financial Services Industry (FSI) to preserve the security of the raw data [31] or to detect fraudulent activities [32]. As federated learning is an important and sometimes the only technique for geo-distributed data, over the years, it has caught the researchers' attention to reduce communication requirements [35, 36], or ensure robustness to differential privacy attacks [37]. Currently, a good number of federated machine learning tools are already available [21, 22, 38]. Additionally, to meet the new challenges for running machine learning algorithms, and to provide a generalized framework for running the machine learning tools in a federated setup, Gaia [39] and TernGrad [40] were proposed. However, most of these available machine learning tools and frameworks are not suitable to operate on datasets with significantly higher dimensionality [5–8]. Therefore, when it comes to reducing the dimensionality of the data into a manageable one or perform any other pre-processing steps such as feature extraction [14], lossy-data compression [13], and data visualization [15], etc. in such a geo-distributed environment, the requirement of a federated PCA is certain.

Nevertheless, to the best of our knowledge, in such federated setup described above, there is no implementation of PCA, and out of the necessity, we focus on proposing a communication efficient solution for federated PCA. We discuss more related works in geo-distributed analytics in section 5.5.



# Chapter 3

## Preliminaries

### 3.1 Big Data

Big Data is a term for voluminous amounts of data that has the potential to be mined for information. Big data can be analyzed for insights that lead to better decisions. Big Data is characterized by three main factors: the extremely large volume of data, the wide variety of data types and the velocity at which the data must be processed.

Such voluminous data can come from countless different sources such as business sales records, the collected results of scientific experiments of real-time sensors used in the internet of things (IoT).

Data may exist in wide variety of types, including structured data, such as SQL database stores, as well as unstructured data such as document files or streaming data from sensors. Big data of different types and from different sources may be used together during analysis.

Principal Component Analysis (PCA), which is the main focus of our thesis, is considered as a pre-processing step in Big Data Analytic. It reveals the relations between the different dimensions of data and allows a reduction in dimensionality of the data via low rank approximation.

### 3.2 Data Analysis

According to [41] Data analysis, also known as analysis of data or data analytics, is a process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names, in different business, science, and social science domains.

Data mining is a particular data analysis technique that focuses on modeling and knowledge discovery for predictive rather than purely descriptive purposes, while business intelligence

covers data analysis that relies heavily on aggregation, focusing on business information [30, 32]. In statistical applications data analysis can be divided into descriptive statistics, exploratory data analysis (EDA), and confirmatory data analysis (CDA). EDA focuses on discovering new features in the data and CDA on confirming or falsifying existing hypotheses. Predictive analytics focuses on application of statistical models for predictive forecasting or classification, while text analytics applies statistical, linguistic, and structural techniques to extract and classify information from textual sources, a species of unstructured data. All are varieties of data analysis.

### 3.3 Data Analysis Approaches

Under these circumstances, we can see that in case of extracting some information from the existing data, we may have to cover a good number of data centers located at different geographical area. This has given the introduction of a new data analysis approach, “Distributed Data Analysis”. Therefore, we get the idea of two approaches of data analysis.

#### 3.3.1 Centralized Approach

The centralized approach to data analysis from distributed data centers is to centralize them first. As shown in Figure 3.1, this involves two different steps:

1. **Centralizing step** Data from various data centers are copied into a single data center. It involves recreation of data of all data centers in one location.
2. **Analysis Step** Process of extracting the necessary information from the centralized data takes place in that single data center. Here traditional intra data center technology is sufficient for the analysis purpose.

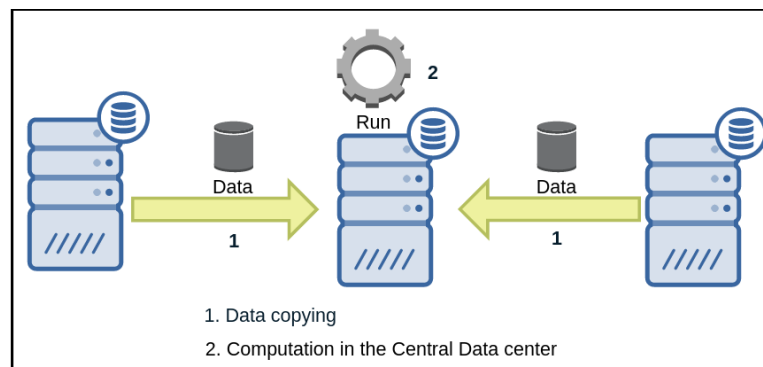


Figure 3.1: Centralized Approach

This centralized approach is predominant in most practical settings. There are mainly two reasons behind its popularity.

1. There are lots of frameworks that have already been established for centralized learning approach. That is why centralizing the data is the easiest way to reuse existing data analysis frameworks [42–44]
2. Learning algorithms are highly communication intensive. Thus, it is assumed that they will not be properly responsive to cross data center execution.

For these reasons, this centralized approach is consistent with reports on the infrastructures of other large organizations, such as Facebook [25], Twitter [2], and LinkedIn [24].

However, the centralized approach has two shortcomings.

1. While making multiple copy of data at the central data center, it consumes a good amounts of inter data center bandwidth. Since inter data center bandwidth is expensive, it is not easy to increase it according to the necessity [22, 45–47].
2. While creating copy of data, it may be a case that data is crossing national borders. However, in current workd data sovereignty is a growing concern that might create a big limitation in this aspect [48].

### 3.3.2 Distributed Approach

In the distributed approach, raw data is kept in their corresponding data centers. Every data center does a portion of the execution that is only on the data of that data center. The final analysis takes pales by passing small amount of information among the data centers.

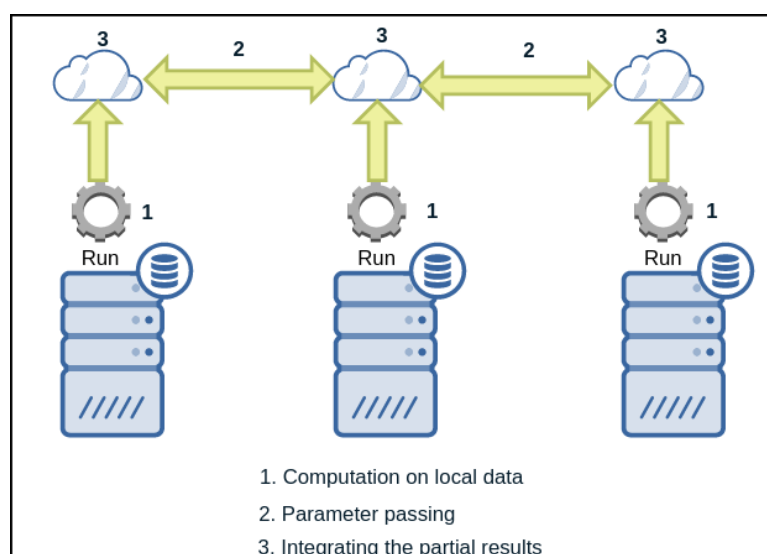


Figure 3.2: Distributed Approach

So according to Figure 3.2, we can see this approach includes three steps:

1. **Local Computation** Whenever the command of starting of any learning process is issued every data center start a partial computation on its own data. They create necessary high level information on data that will be needed in the final computation.
2. **Parameter Passing** Data centers communicate among themselves and share valuable information.
3. **Final Computation** By integrating the partial results data centers make the final computational model.

In this way distributed solutions can achieve much lower cross data centers bandwidth utilization, and thus substantially lower cost for large-scale analysis tasks. As the current world has got a concern about ‘Big Data’ and ‘Data Sovereignty’, the distributed approach seems to be more efficient.

### 3.4 Geo Distributed Data

Over the year concept about data has changed a lot. Only few years ago data were generated locally and computation was done locally also. But now data are generated over the whole world [20–23]. Data are not bound to locality.

Data are by born geographically distributed over the world. That’s why nowadays geo distributed analysis is being popular. Data are not gathered in one place rather algorithms are being designed to run over the geo-distributed data.

### 3.5 Data Sovereignty

Data sovereignty is the concept that information which has been converted and stored in binary digital form is subject to the laws of the country in which it is located [22,23].

Many of the current concerns that surround data sovereignty relate to enforcing privacy regulations and preventing data that is stored in a foreign country from being subpoenaed by the host country’s government.

The wide-spread adoption of cloud computing services, as well as new approaches to data storage including object storage, have broken down traditional geopolitical barriers more than ever before. In response, many countries have regulated new compliance requirements by amending their current laws or enacting new legislation that requires customer data to be kept within the country the customer resides.

Verifying that data exists only at allowed locations can be difficult. It requires the cloud customer to trust that their cloud provider is completely honest and open about where their servers are hosted and adhere strictly to service level agreements (SLAs).

## 3.6 Data Cluster and Cluster Computing

A cluster is a system comprising two or more computers or systems (called nodes) which work together to execute a particular task. They are usually deployed for High Availability (HA) [49, 50] for greater reliability and High performance Computing (HPC) [51–53] to provide greater computational power than a single computer can provide.

The components of a cluster are usually connected to each other through fast local area networks (LANs), with each node running its own instance of an operating system (OS). In most circumstance, all of the nodes use the same hardware and the same OS, although it is possible to user different OS and/or different hardware on each computer.

### 3.6.1 Advantages

Cluster computing provides the following important advantages.

1. Cluster computing is mainly a scalable solution. Resources may be removed or new resources may be added to clusters after the system has already been deployed. They can scale to very large and complex systems with large computing power, not possible with single computers. Also, each of the machines in a cluster can be a complete system, usable for a wide range of other computing applications.
2. Clustering solutions are more fault tolerant. If one of the servers in the system stops working, other servers in the cluster can take the load. If a server in the cluster needs any maintenance, it can be done by stopping it while handing the load over to other servers.
3. Clusters are more cost effective. A cluster will cost much less than a single computer of comparable speed and availability.

### 3.6.2 Disadvantages

Unfortunately, clustering suffers from some limitations as well:

1. As clusters consist of many computers running in parallel, it is obvious that these systems are only efficient in carrying out a specific task if the task can be separated into smaller subtasks that can be completed in parallel. This may not always be possible.
2. The network hardware used to communicate typically has low bandwidth and high latency (as compared to the link between the different processors in a single computer) and this communication complexity usually acts as the bottleneck during execution of a task. Thus, algorithms are required to be designed to run on distributed settings and must be optimized to reduce communication between nodes which may be tough to do.

3. Clusters can become very large and complex, and the maintenance of these large and complex systems could be quite expensive.

## 3.7 Data Parallelism

Data parallelism is a form of parallelization across multiple processors in parallel computing environments. It focuses on distributing the data across different nodes, which operate on the data in parallel. It can be applied on regular data structures like arrays and matrices by working on each element in parallel. It contrasts to task parallelism as another form of parallelism [54–56]. A data parallel job on an array of ‘ $n$ ’ elements can be divided equally among all the processors. Let us assume we want to sum all the elements of the given array and the time for a single addition operation is  $Ta$  time units. In the case of sequential execution, the time taken by the process will be  $n * Ta$  time units as it sums up all the elements of an array. On the other hand, if we execute this job as a data parallel job on 4 processors the time taken would reduce to  $(n/4) * Ta + \text{Merging overhead time units}$ . Parallel execution results in a speed up of 4 over sequential execution. One important thing to note is that the locality of data references plays an important part in evaluating the performance of a data parallel programming model. Locality of data depends on the memory accesses performed by the program as well as the size of the cache.

## 3.8 Model Parallelism

Data Parallelism and Model Parallelism are different ways of distributing an algorithm. These are often used in the context of machine learning algorithms that use stochastic gradient descent to learn some model parameters [57].

In model parallelism algorithm sends the same data to all the cores. Each core is responsible for estimating different parameter(s). Cores then exchange their estimate(s) with each other to come up with the right estimate for all the parameters.

Figure 3.3 shows a basic visualization of data and model parallelism.

## 3.9 TensorFlow

TensorFlow [58] is an open source software library for numerical computation using data flow graphs. As we have used the concept of model parallelism and data parallelism in our geo distributed algorithm, so it is important. Because in *TensorFlow* architecture both the model parallelism and data parallelism have been used.

The flexible architecture of *TensorFlow* allows to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. *TensorFlow* was originally

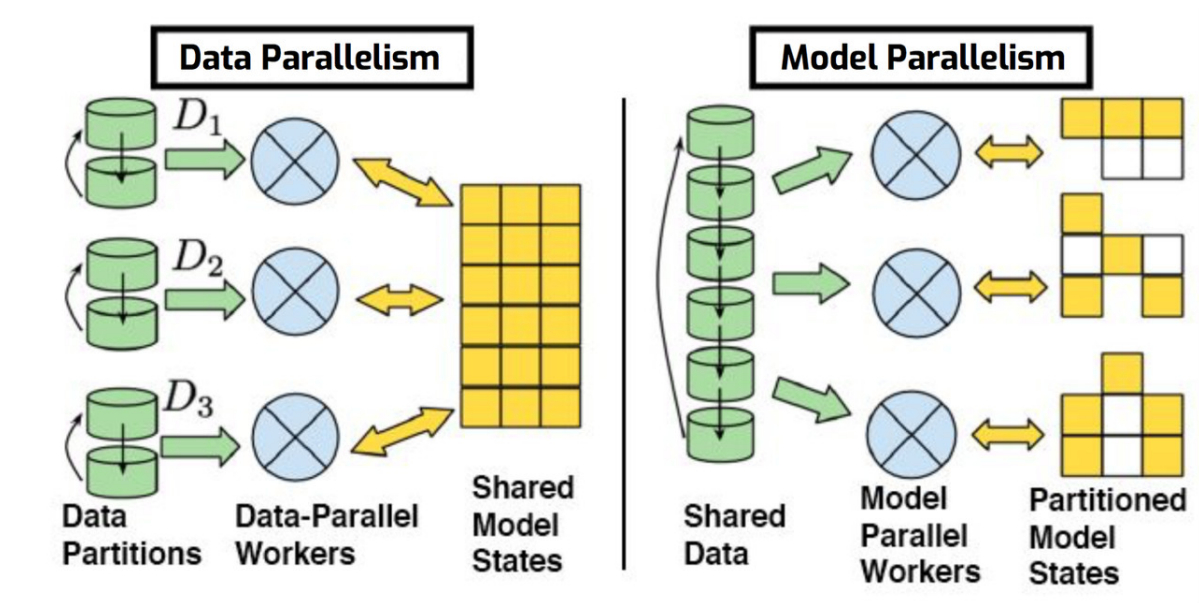


Figure 3.3: Data and Model Parallelism

developed by researchers and engineers working on the *Google Brain Team* within *Google's* Machine Intelligence research organization for the purpose of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in wide variety of other domains as well.

### 3.10 Hadoop Cluster

*Hadoop* [59] is an open source software framework built on two technologies, Linux operation system and Java programming language. It supports the processing and storage of extremely large data sets in a cluster computing environment. It is part of the Apache project sponsored by the Apache Software Foundation.

All the modules in *Hadoop* are designed with the assumption that hardware failures are common occurrences and should be automatically handled by the framework.

### 3.11 Hadoop MapReduce

MapReduce [60] is the heart of *Hadoop*. MapReduce is a processing technique and a programming paradigm for distributed computing based on Java. It allows for massive scalability across hundreds or thousands of servers in a *Hadoop* cluster.

The MapReduce algorithm consists of two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value-pairs). Reduce takes the output from a map as an input and combines

those data tuples into smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

During a MapReduce job, *Hadoop* assigns the Map and Reduce tasks to the appropriate serves in the cluster. The framework manages all the details of data passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.

Most of the computing takes place on nodes with data on local disks that reduces the network traffic.

After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the *Hadoop* server.

## 3.12 Spark Cluster

Spark [61] is an open source processing engine that provides scalable, massively parallel, in-memory execution environment for running analytics applications. It can be thought as an in-memory layer that sits above multiple data stores, where data can be loaded into memory and analyzed in parallel across a cluster.

Much like MapReduce, Spark works to distribute data across a cluster, and process that data in parallel. But unlike MapReduce - which shuffles files around on disk - Spark works in memory, making it much faster at processing data than MapReduce.

Spark includes rebuilt machine-learning algorithms and graph analysis algorithms that are especially written to execute in parallel and in memory. It also supports interactive SQL processing of queries and real-time streaming analytics.

Spark provides programmers with an application programming interface (API) centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. The availability of RDDs facilitates the implementation of both iterative algorithms, that visit their dataset multiple times in a loop, and exploratory data analysis (the repeated database-style querying of data). It should be noted that we also choose Spark as the distributed framework in order to implement our proposed algorithm, TallnWide.

## 3.13 Dimensionality Reduction

Real-world data, such as speech signals, digital photographs, or fMRI scans, usually has a high dimensionality. In order to handle such real-world data adequately, its dimensionality needs to be reduced. Dimensionality reduction is the transformation of high-dimensional data into a meaningful representation of reduced dimensionality. Ideally, the reduced representation



should have a dimensionality that corresponds to the intrinsic dimensionality of the data. The intrinsic dimensionality of data is the minimum number of parameters needed to account for the observed properties of the data. The problem of (nonlinear) dimensionality reduction can be defined as follows. Assume we have a dataset represented in a  $n \times D$  matrix  $\mathbf{X}$  consisting of  $n$  datavectors  $x_i (i \in 1, 2, \dots, n)$  with dimensionality  $D$ . Assume further that this dataset has intrinsic dimensionality  $d$  (where  $d < D$ , and often  $d \ll D$ ). Note that, PCA is one of most widely used technique to reduce the dimensionality of higher dimensional datasets.

### 3.14 Vector Norms

A norm is a function  $\|\cdot\| : \mathbf{V} \rightarrow \mathbb{R}$  which satisfies

$$(I) \quad \|x\| \geq 0, \forall x \in \mathbf{V} \text{ and } \|x\| = 0 \iff x = 0$$

$$(II) \quad \|x + y\| \leq \|x\| + \|y\|, \forall x, y \in \mathbf{V}$$

$$(III) \quad \|\lambda x\| = |\lambda| \|x\|, \text{ for any scalar } \lambda \text{ and } \forall x \in \mathbf{V}$$

Where  $\mathbf{V}$  is known as the vector space. In words, these conditions require that (I) the norm of a nonzero vector is strictly positive, (II) the norm of a vector sum does not exceed the sum of the norms of its parts which is known as the triangle inequality, and (III) scaling a vector scales its norm by the same amount. Thus, norms can be thought of as functions that define the size of a vector.

### 3.15 p-Norm

p-Norm is a family of vector norms, denoted as  $\|\cdot\|_p$ , given by:

$$\|x\|_p = \left( \sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}}$$

The most widely used are the 1-norm, 2-norm (also known as Euclidean norm), and  $\infty$ -norm (also known as sup-norm):

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$$

$$\|x\|_\infty = \max(|x_i|)$$

## 3.16 Frobenius Norm

This is an element-wise norm where the vector norm used is the 2-norm. Each entry of the matrix is considered as a dimension of a vector and 2-norm of the resulting vector is calculated. So we have,

$$\| \mathbf{A} \|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$$

which is equivalent to

$$\| \mathbf{A} \|_F = \sqrt{\text{tr}(\mathbf{A} * \mathbf{A})} = \sqrt{\text{tr}(\mathbf{A}\mathbf{A}^*)}$$

where  $\mathbf{A}$  is an  $m \times n$  matrix,  $a_{ij}$  is the value in the  $i$ -th row and  $j$ -th column of  $\mathbf{A}$ ,  $\mathbf{A}^*$  is the conjugate transpose of  $\mathbf{A}$ , and  $\text{tr}(\mathbf{B})$  is the trace of  $\mathbf{B}$  (the sum of its diagonal entries). Since  $\text{tr}(\mathbf{A}\mathbf{A}^*)$  is the sum of the eigenvalues of  $\mathbf{A}\mathbf{A}^*$ , we have an alternative characterization of the Frobenius norm:

$$\| \mathbf{A} \|_F = \sqrt{\sum_{i=1}^n \lambda_i}$$

## 3.17 Normal Distribution

In probability theory, the normal (or Gaussian) distribution is a very common continuous probability distribution. Normal distributions are important in statistics and are often used in the natural and social sciences to represent real-valued random variables whose distributions are not known [62].

The normal distribution is useful because of the central limit theorem. In its most general form, under some conditions (which include finite variance), it states that averages of samples of observations of random variables independently drawn from independent distributions converge in distribution to the normal, that is, become normally distributed when the number of observations is sufficiently large. Physical quantities that are expected to be the sum of many independent processes (such as measurement errors) often have distributions that are nearly normal. Moreover, many results and methods (such as propagation of uncertainty and least squares parameter fitting) can be derived analytically in explicit form when the relevant variables are normally distributed.

The normal distribution is sometimes informally called the bell curve. However, many other distributions are bell-shaped (such as the Cauchy, Student's  $t$ , and logistic distributions). Even the term Gaussian bell curve is ambiguous because it may be used to refer to some function

defined in terms of the Gaussian function which is not a probability distribution because it is not normalized in that it does not integrate to 1.

The probability density of the normal distribution is

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where:

- $\mu$  is the mean or expectation of the distribution (and also its median and mode).
- $\sigma$  is the standard deviation.
- $\sigma^2$  is the variance.

### 3.18 Matrix Diagonalization

Matrix diagonalization is the process of taking a square matrix and converting it into a special type of matrix—a so-called diagonal matrix—that shares the same fundamental properties of the underlying matrix. Matrix diagonalization is equivalent to transforming the underlying system of equations into a special set of coordinate axes in which the matrix takes this canonical form. Diagonalizing a matrix is also equivalent to finding the matrix's eigenvalues, which turn out to be precisely the entries of the diagonalized matrix. Similarly, the eigenvectors make up the new set of axes corresponding to the diagonal matrix.

The remarkable relationship between a diagonalized matrix, eigenvalues, and eigenvectors follows from the beautiful mathematical identity (the eigen decomposition) that a square matrix  $A$  can be decomposed into the very special form

$$A = PDP^{-1}$$

where  $P$  is a matrix composed of the eigenvectors of  $A$ ,  $D$  is the diagonal matrix constructed from the corresponding eigenvalues, and  $P^{-1}$  is the matrix inverse of  $P$ . According to the eigen decomposition theorem, an initial matrix equation

$$AX = Y$$

can always be written

$$PDP^{-1}X = Y$$

(at least as long as  $P$  is a square matrix), and premultiplying both sides by  $P^{-1}$  gives

$$DP^{-1}X = P^{-1}Y.$$

Since the same linear transformation  $P^{-1}$  is being applied to both  $X$  and  $Y$ , solving the original system is equivalent to solving the transformed system

$$DX' = Y'$$

where  $X' = P^{-1}X$  and  $Y' = P^{-1}Y$ . This provides a way to canonicalize a system into the simplest possible form, reduce the number of parameters from  $n \times n$  for an arbitrary matrix to  $n$  for a diagonal matrix, and obtain the characteristic properties of the initial matrix. This approach arises frequently in physics and engineering, where the technique is oft used and extremely powerful.

### 3.19 Expectation Maximization (EM) Algorithm

Maximum likelihood estimation [63] is an approach to density estimation for a dataset by searching across probability distributions and their parameters.

It is a general and effective approach that underlies many machine learning algorithms, although it requires that the training dataset is complete, e.g. all relevant interacting random variables are present. Maximum likelihood becomes intractable if there are variables that interact with those in the dataset but were hidden or not observed, so-called latent variables.

The expectation-maximization algorithm [64] is an approach for performing maximum likelihood estimation in the presence of latent variables. It does this by first estimating the values for the latent variables, then optimizing the model, then repeating these two steps until convergence. It is an effective and general approach and is most commonly used for density estimation with missing data, such as clustering algorithms like the Gaussian Mixture Model [65].

### 3.20 Stop Condition

In iterative algorithm like EM, main terminating condition considered is the change in desired variable or objective function. For example in PPCA our desired variables are  $W$  and  $\sigma^2$ . To check convergence we can give tolerance limit as input. If magnitude of changes in desired variable is less than our tolerance limit then we consider our algorithm has converged. But this is not most effective when we want to know how much accuracy we have obtained after convergence. Thus in many cases, error is checked after each iteration and algorithm is terminated if error goes down our target limit.

# Chapter 4

## PCA as a Data Analytic Tool

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [1, 66]. The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. As we can use PCA for dimensionality reduction, it can be used as a preprocessing step in many machine learning algorithms that do not perform well with high-dimensional data. Therefore we can easily realize that PCA is an excellent tool for big data analysis.

There are several ways to perform PCA [66,67]. Eigen Value Decomposition (EVD) of covariance matrix and Singular Value Decomposition (SVD) are two basic and most common ways [1]. These methods usually perform better on small datasets on a single machine. But in distributed settings and for big data, they introduce a new set of difficulties; they do not scale well to high dimensional data and are inefficient in terms of computation and communication cost [68]. To overcome these difficulties, two other methods, Stochastic SVD (SSVD) [69] and Probabilistic PCA (PPCA) [70] are used in practice. But unlike the other methods to calculate principal component (i.e. EVD of covariance matrix, SVD, SSVD), PPCA has two important advantages. It has the ability of handling missing data and calculating probabilistic model to generate synthesized data [26]. These features are best suited for big data as missing values are prevalent in this case and a probabilistic model of the data will be more effective for analytical purposes. In this chapter, we try to show in depth the various approaches of performing PCA with their advantages and disadvantages.

## 4.1 Assumptions Involved in PCA

Mathematically, PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate and so on. Therefore, the assumptions involved in PCA are:

### 1. Linearity

Linearity vastly simplifies the problem by restricting the set of potential bases. With this assumption PCA is now limited to re-expressing the data as a linear combination of its basis vectors that is, we need to find the appropriate change of basis.

### 2. Large variances have important structure

This assumption also encompasses the belief that the data has a high Signal-to-Noise Ratio ( $SNR = \frac{\sigma_{signal}^2}{\sigma_{noise}^2}$ ). Hence, principal components with larger associated variances represent interesting structure, while those with lower variances represent noise. Note that this is a strong, and sometimes, incorrect assumption.

### 3. Orthogonal PCs

This assumption provides an intuitive simplification that makes PCA soluble with linear algebra decomposition techniques.

## 4.2 PCA and Change of Basis

The goal of principal component analysis is to identify the most meaningful basis to re-express a data set. The hope is that this new basis will filter out the noise and reveal hidden structure. Determining this fact allows an experimenter to discern which dynamics are important, redundant or noise.

With this rigor we may now state more precisely what PCA asks: *Is there another basis, which is a linear combination of the original basis, that best re-expresses our data set?*

Let  $\mathbf{X}$  be the original data set, where each column is a single sample (or moment in time) of our data set. Here  $\mathbf{X}$  is a  $D \times N$  matrix where  $N$  is the number of samples and  $D$  is the dimension of each sample. Let  $\mathbf{Y}$  be another  $D \times N$  matrix related by a linear transformation  $\mathbf{P}$ .  $\mathbf{X}$  is the original recorded data set and  $\mathbf{Y}$  is a new representation of that data set.

$$\mathbf{Y} = \mathbf{P}\mathbf{X} \tag{4.1}$$

Also let us define the following quantities

- $\mathbf{p}_i$  are the rows of  $\mathbf{P}$

- $\mathbf{x}_i$  are the columns of  $\mathbf{X}$
- $\mathbf{y}_i$  are the columns of  $\mathbf{Y}$

Equation 4.1 represents a change of basis and thus can have many interpretations:

- $\mathbf{P}$  is a matrix that transforms  $\mathbf{X}$  into  $\mathbf{Y}$
- Geometrically,  $\mathbf{P}$  is a rotation and a stretch which again transforms  $\mathbf{X}$  into  $\mathbf{Y}$
- The rows of  $\mathbf{P}$ ,  $\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_D\}$  are a set of new basis vectors for expressing the columns of  $\mathbf{X}$

The latter interpretation is not obvious but can be seen by writing out the explicit dot products of  $\mathbf{P}\mathbf{X}$

$$\mathbf{P}\mathbf{X} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_D \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_N \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{p}_1 \cdot \mathbf{x}_1 & \dots & \mathbf{p}_1 \cdot \mathbf{x}_N \\ \vdots & \ddots & \vdots \\ \mathbf{p}_D \cdot \mathbf{x}_1 & \dots & \mathbf{p}_D \cdot \mathbf{x}_N \end{bmatrix}$$

We can note the form of each column of  $\mathbf{Y}$

$$\mathbf{y}_i = \begin{bmatrix} \mathbf{p}_1 \cdot \mathbf{x}_i \\ \vdots \\ \mathbf{p}_D \cdot \mathbf{x}_i \end{bmatrix}$$

We recognize that each coefficient of  $\mathbf{y}_i$  is a dot-product of  $\mathbf{x}_i$  with the corresponding row in  $\mathbf{P}$ . In other words, the  $j^{\text{th}}$  coefficient of  $\mathbf{y}_i$  is a projection on to the  $j^{\text{th}}$  row of  $\mathbf{P}$ . This is in fact the very form of an equation where  $\mathbf{y}_i$  is a projection on to the basis of  $\{\mathbf{p}_1, \dots, \mathbf{p}_D\}$ . Therefore, the rows of  $\mathbf{P}$  are a new set of basis vectors for representing of columns of  $\mathbf{X}$ .

By assuming linearity the problem reduces to finding the appropriate change of basis. The row vectors  $\{\mathbf{p}_1, \dots, \mathbf{p}_D\}$  in this transformation will become the principal components of  $\mathbf{X}$ . Several questions now arise:

- What is the best way to re-express  $\mathbf{X}$ ?
- What is a good choice of basis  $\mathbf{P}$ ?

These questions must be answered by next asking ourselves what features we would like  $\mathbf{Y}$  to exhibit. Evidently, additional assumptions beyond linearity are required to arrive at a reasonable result. The selection of these assumptions is the subject of the next section.

## 4.3 Eigenvalue Decomposition (EVD)

Given a matrix  $\mathbf{Y}$  of size  $N \times D$  ( $N$  rows and  $D$  columns), a PCA algorithm obtains  $d$  principal components ( $d \leq D$ ) that explain the most variance (and hence information) of the data in matrix  $\mathbf{Y}$  [1, 71]. To be useful in practice,  $d$  is chosen to be much smaller than  $D$ , that is  $d \ll D$ . In case EVD technique, the target matrix  $\mathbf{V}$  is of dimension  $N \times d$  where the columns of  $\mathbf{V}$  are the principal components of  $\mathbf{Y}$ .

### 4.3.1 Covariance Matrix

How do we quantify and generalize these notions to arbitrarily higher dimensions? Consider two sets of measurements with zero means:

$$\mathbf{A} = \{a_1, a_2, \dots, a_N\}, \mathbf{B} = \{b_1, b_2, \dots, b_N\}$$

where the subscript denotes the sample number. The variance of  $\mathbf{A}$  and  $\mathbf{B}$  are individually defined as,

$$\sigma_A^2 = \frac{1}{N} \sum_i a_i^2$$

$$\sigma_B^2 = \frac{1}{N} \sum_i b_i^2$$

The covariance between  $\mathbf{A}$  and  $\mathbf{B}$  is a straight-forward generalization.

$$\text{covariance of } \mathbf{A} \text{ and } \mathbf{B} \equiv \sigma_{AB}^2 = \frac{1}{N} \sum_i a_i b_i$$

The covariance measures the degree of the linear relationship between two variables. A large positive value indicates positively correlated data. Likewise, a large negative value denotes negatively correlated data. The absolute magnitude of the covariance measures the degree of redundancy. Some additional facts about the covariance:

- $\sigma_{AB}$  is zero if and only if  $\mathbf{A}$  and  $\mathbf{B}$  are uncorrelated (e.g. Figure 4.1, left panel)
- $\sigma_{AB}^2 = \sigma_A^2$  if  $\mathbf{A} = \mathbf{B}$



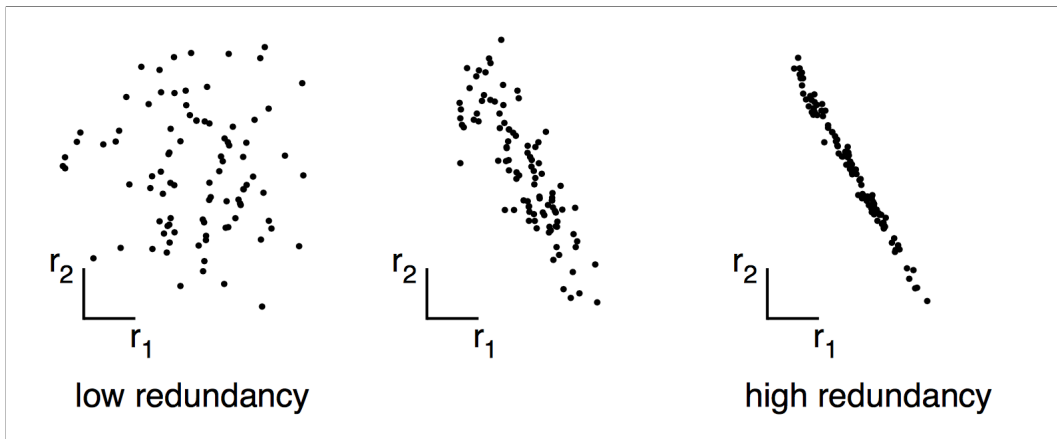


Figure 4.1: A spectrum of possible redundancies in data from the two separate measurements  $r_1$  and  $r_2$ . The two measurements on the left are uncorrelated because one can not predict one from the other. Conversely, the two measurements on the right are highly correlated indicating highly redundant measurements.

We can equivalently convert  $\mathbf{A}$  and  $\mathbf{B}$  into corresponding row vectors.

$$\mathbf{a} = [a_1 \ a_2 \ \dots \ a_n]$$

$$\mathbf{b} = [b_1 \ b_2 \ \dots \ b_n]$$

so that we may express the covariance as a dot product matrix computation

$$\sigma_{ab}^2 \equiv \frac{1}{N} \mathbf{a} \mathbf{b}^T$$

Finally, we can generalize from two vectors to an arbitrary number. Rename the row vectors  $\mathbf{a}$  and  $\mathbf{b}$  as  $\mathbf{x}_1$  and  $\mathbf{x}_2$ , respectively, and consider additional indexed row vectors  $\mathbf{x}_1, \dots, \mathbf{x}_D$ . Define a new  $D \times N$  matrix  $\mathbf{X}$ .

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_D \end{bmatrix}$$

One interpretation of  $\mathbf{X}$  is the following. Each row of  $\mathbf{X}$  corresponds to all measurements of a particular type. Each column of  $\mathbf{X}$  corresponds to a set of measurements from one particular trial. We now arrive at a definition for the covariance matrix  $\mathbf{C}_x$

$$\mathbf{C}_x = \frac{1}{N} \mathbf{X} \mathbf{X}^T$$

Consider the matrix  $\mathbf{C}_x = \frac{1}{N} \mathbf{X} \mathbf{X}^T$ . The  $ij^{th}$  element of  $\mathbf{C}_x$  is the dot product between the vector of the  $i$ th measurement type with the vector of the  $j^{th}$  measurement type. We can summarize several properties of  $\mathbf{C}_x$ :

- $C_x$  is a square symmetric  $D \times D$  matrix (Theorem 2 of Appendix A)
- The diagonal terms of  $C_x$  are the variance of particular measurement types
- The off-diagonal terms of  $C_x$  are the covariance between measurement types.

$C_x$  captures the covariance between all possible pairs of measurements. The covariance values reflect the noise and redundancy in our measurements.

- In the diagonal terms, by assumption, large values correspond to interesting structure
- In the off-diagonal terms large magnitudes correspond to high redundancy

Pretend we have the option of manipulating  $C_x$ . We will suggestively define our manipulated covariance matrix  $C_Y$ . What features do we want to optimize in  $C_Y$ ?

### 4.3.2 Diagonalize the Covariance Matrix

We can summarize the last two sections by stating that our goals are

1. to minimize redundancy, measured by the magnitude of the covariance
2. maximize the signal, measured by the variance

What would the optimized covariance matrix  $C_Y$  look like?

- All off-diagonal terms in  $C_Y$  should be zero. Thus,  $C_Y$  must be a diagonal matrix. Or, said another way,  $Y$  is decorrelated
- Each successive dimension in  $Y$  should be rank-ordered according to variance

There are many methods for diagonalizing  $C_Y$ . It is curious to note that PCA arguably selects the easiest method: PCA assumes that all basis vectors  $\{\mathbf{p}_1, \dots, \mathbf{p}_m\}$  are orthonormal, i.e.  $\mathbf{P}$  is an orthonormal matrix.

In a simple 2D example like in Figure 4.2,  $\mathbf{P}$  acts as a generalized rotation to align a basis with the axis of maximal variance. In multiple dimensions this could be performed by a simple algorithm:

1. Select a normalized direction in  $m$ -dimensional space along which the variance in  $\mathbf{X}$  is maximized. Save this vector as  $\mathbf{p}_1$ .
2. Find another direction along which variance is maximized, however, because of the orthonormality condition, restrict the search to all directions orthogonal to all previous selected directions. Save this vector as  $\mathbf{p}_2$ .
3. Repeat this procedure until  $D$  vectors are selected.

The resulting ordered set of  $\mathbf{p}$ 's are the principal components.

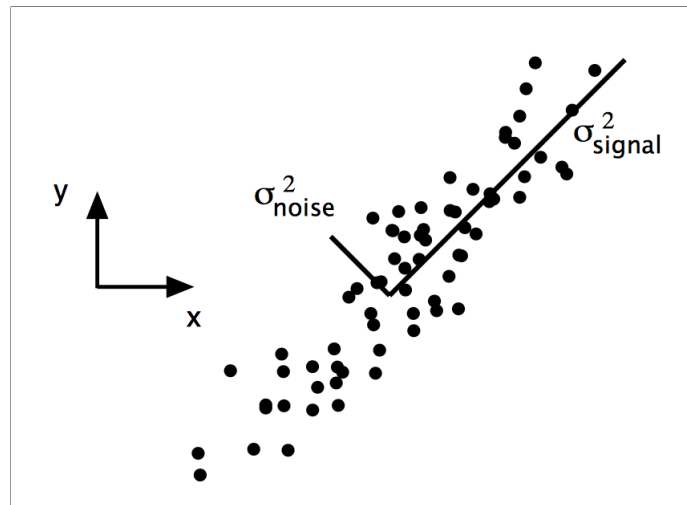


Figure 4.2: A typical 2D example. The signal and noise variances  $\sigma_{signal}^2$  and  $\sigma_{noise}^2$  are graphically represented by the two lines subtending the cloud of data. The largest direction of variance lie along the best-fit line

### 4.3.3 Solving PCA in EVD Approach

We derive our first algebraic solution to PCA based on an important property of eigenvector decomposition. Once again, the data set is  $P$ , an  $D \times N$  matrix, where  $D$  is the number of measurement types and  $N$  is the number of samples. The goal is summarized as follows

Find some orthonormal matrix  $P$  in  $Y = PX$  such that  $C_Y \equiv \frac{1}{N}XX^T$  is a diagonal matrix. The rows of  $P$  are the principal components of  $X$

We begin by rewriting  $C_Y$  in terms of the unknown variable

$$\begin{aligned}
 C_Y &= \frac{1}{N}YY^T \\
 &= \frac{1}{N}(PX)(PX)^T \\
 &= \frac{1}{N}PXX^TP^T \\
 &= P\left(\frac{1}{N}XX^T\right)P^T \\
 C_Y &= PC_XP^T
 \end{aligned}$$

Note that we have identified the covariance matrix of  $X$  in the last line.

Our plan is to recognize that any symmetric matrix  $A$  is diagonalized by an orthogonal matrix of its eigenvectors (by Theorems 3 and 4 from Appendix A). For a symmetric matrix  $A$  Theorem 4 provides  $A = EDE^T$ , where  $D$  is a diagonal matrix and  $E$  is a matrix of eigenvectors of  $A$  arranged as columns.

Now we comes the trick. We select the matrix  $\mathbf{P}$  to be a matrix where each row  $\mathbf{p}_i$  is an eigenvector of  $\frac{1}{N}\mathbf{X}\mathbf{X}^T$ . By this selection,  $\mathbf{P} \equiv \mathbf{E}^T$ . With this relation and Theorem 1 of Appendix A ( $\mathbf{P}^{-1} = \mathbf{P}^T$ ) we can finish evaluating  $\mathbf{C}_Y$ .

$$\begin{aligned} \mathbf{C}_Y &= \mathbf{P}\mathbf{C}_X\mathbf{P}^T \\ &= \mathbf{P}(\mathbf{E}^T\mathbf{D}\mathbf{E})\mathbf{P}^T \\ &= \mathbf{P}(\mathbf{P}^T\mathbf{D}\mathbf{P})\mathbf{P}^T \\ &= (\mathbf{P}\mathbf{P}^T)\mathbf{D}(\mathbf{P}\mathbf{P})^T \\ &= (\mathbf{P}\mathbf{P}^{-1})\mathbf{D}(\mathbf{P}\mathbf{P})^{-1} \\ \mathbf{C}_Y &= \mathbf{D} \end{aligned}$$

It is evident that the choice of  $\mathbf{P}$  diagonalizes  $\mathbf{C}_Y$ . This was the goal for PCA. We can summarize the results of PCA in the matrices  $\mathbf{P}$  and  $\mathbf{C}_Y$ .

- The principal components of  $\mathbf{C}_Y$  are the eigenvectors of  $\mathbf{C}_X = \frac{1}{N}\mathbf{X}\mathbf{X}^T$
- The  $i^{\text{th}}$  diagonal value of  $\mathbf{C}_Y$  is the variance of  $\mathbf{X}$  along  $\mathbf{p}_i$

## 4.4 Practical Approach of EVD

In practice computing PCA of a data set  $\mathbf{X}$  is done in two steps:

1. subtracting off the mean of each measurement type
2. computing the eigenvectors of  $\mathbf{C}_X$

## 4.5 Singular Value Decomposition (SVD)

Let  $\mathbf{X}$  be an arbitrary  $N \times D$  matrix and  $\mathbf{X}^T\mathbf{X}$  be a rank  $r$ , square, symmetric  $D \times D$  matrix.

### 4.5.1 Performing SVD

To do Singular Value Decomposition of matrix  $\mathbf{X}$  let us assume that:

- $\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \dots, \hat{\mathbf{v}}_r\}$  is the set of orthonormal  $D \times 1$  eigenvectors with associated eigenvalues  $\{\lambda_1, \lambda_2, \dots, \lambda_r\}$  for the symmetric matrix  $\mathbf{X}^T\mathbf{X}$

$$(\mathbf{X}^T\mathbf{X})\hat{\mathbf{v}}_i = \lambda_i\hat{\mathbf{v}}_i$$



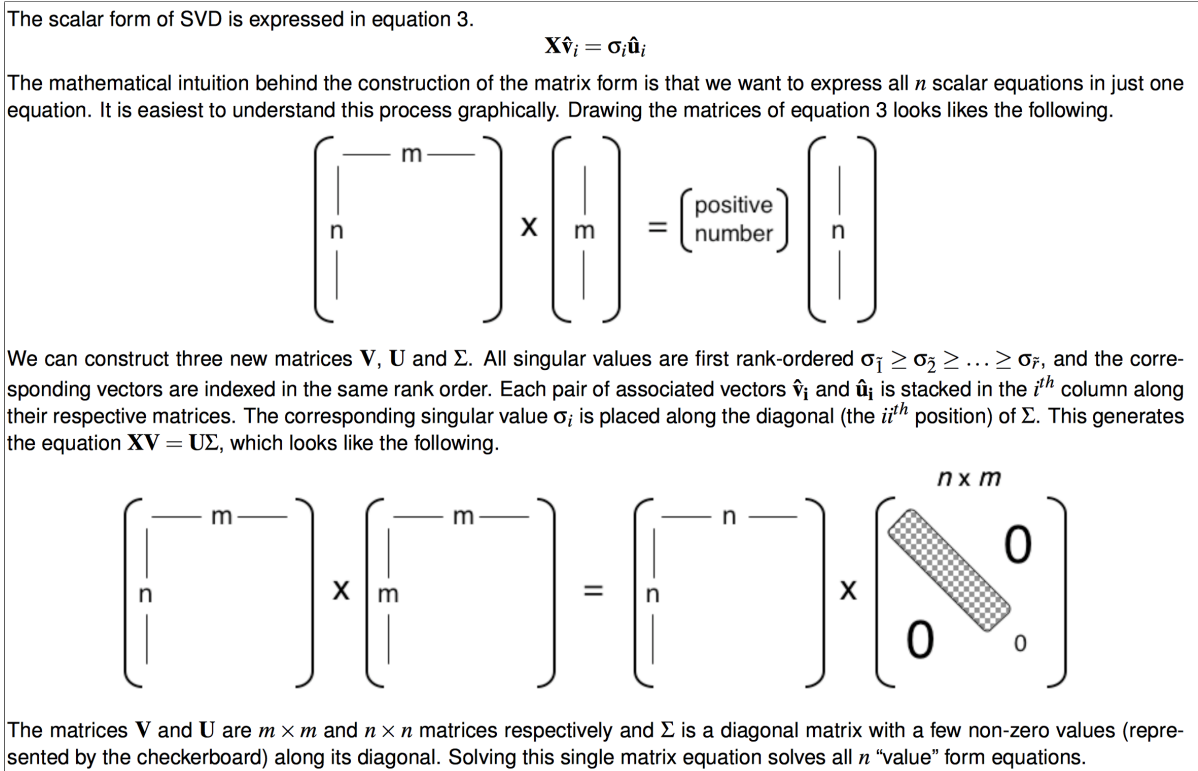


Figure 4.3: Construction of the matrix form of SVD 4.3 from the scalar form 4.2 according to [1]

### 4.5.2 Linking SVD with EVD

How does this link in to the previous *EVD* analysis of PCA? Let us consider the  $N \times D$  matrix,  $\mathbf{X}$ , for which we have a singular value decomposition,  $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ . There is a theorem from linear algebra which says that the non-zero singular values of  $\mathbf{X}$  are the square roots of the nonzero eigenvalues of  $\mathbf{A}\mathbf{A}^T$  or  $\mathbf{A}^T\mathbf{A}$ .

The former assertion for the case  $\mathbf{A}^T\mathbf{A}$  is proven in the following way:

$$\begin{aligned} \mathbf{A}^T\mathbf{A} &= (\mathbf{U}\Sigma\mathbf{V}^T)^T(\mathbf{U}\Sigma\mathbf{V}^T) \\ &= (\mathbf{V}\Sigma^T\mathbf{U}^T)(\mathbf{U}\Sigma\mathbf{V}^T) \\ &= \mathbf{V}(\Sigma^T\Sigma)\mathbf{V}^T \end{aligned}$$

We observe that  $\mathbf{A}^T\mathbf{A}$  is similar to  $\Sigma^T\Sigma$  and thus it has the same eigenvalues. Since  $\Sigma^T\Sigma$  is a square ( $D \times D$ ), diagonal matrix, the eigenvalues are in fact the diagonal entries, which are the squares of the singular values. Note that the non-zero eigenvalues of each of the covariance matrices,  $\mathbf{A}\mathbf{A}^T$  and  $\mathbf{A}^T\mathbf{A}$  are actually identical.

It should also be noted that we have effectively performed an eigenvalue decomposition for the matrix,  $\mathbf{A}^T\mathbf{A}$ . Indeed, since  $\mathbf{A}^T\mathbf{A}$  is symmetric, this is an orthogonal diagonalisation and thus the eigenvectors of  $\mathbf{A}^T\mathbf{A}$  are the columns of  $\mathbf{V}$ . This will be important in making the practical

connection between the SVD and the PCA of matrix  $\mathbf{X}$ , which is what we will do next.

Returning to the original  $N \times D$  data matrix,  $\mathbf{X}$ , let us define a new  $D \times N$  matrix,  $\mathbf{Z}$ :

$$\mathbf{Z} = \frac{1}{\sqrt{N-1}} \mathbf{X}^T$$

Recall that since the  $m$  rows of  $\mathbf{X}$  contained the  $N$  data samples, we subtracted the row average from each entry to ensure zero mean across the rows. Thus, the new matrix,  $\mathbf{Z}$  has columns with zero mean. Consider forming the  $D \times D$  matrix,  $\mathbf{Z}^T \mathbf{Z}$ :

$$\begin{aligned} \mathbf{Z}^T \mathbf{Z} &= \left( \frac{1}{\sqrt{N-1}} \mathbf{X}^T \right)^T \left( \frac{1}{\sqrt{N-1}} \mathbf{X}^T \right) \\ &= \frac{1}{N-1} \mathbf{X} \mathbf{X}^T \\ \text{i.e. } \mathbf{Z}^T \mathbf{Z} &= \mathbf{C}_x \end{aligned}$$

We find that defining  $\mathbf{Z}$  in this way ensures that  $\mathbf{Z}^T \mathbf{Z}$  is equal to the covariance matrix of  $\mathbf{Z}$ ,  $\mathbf{C}_x$ . From the discussion in the previous section, the principal components of  $\mathbf{X}$  (which is what we are trying to identify) are the eigenvectors of  $\mathbf{C}_x$ . Therefore, if we perform a singular value decomposition of the matrix  $\mathbf{Z}^T \mathbf{Z}$ , the principal components will be the columns of the orthogonal matrix,  $\mathbf{V}$ .

The last step is to relate the SVD of  $\mathbf{Z}^T \mathbf{Z}$  back to the change of basis:

$$\mathbf{Y} = \mathbf{P} \mathbf{X}$$

We wish to project the original data onto the directions described by the principal components. Since we have the relation  $\mathbf{V} = \mathbf{P}^T$ , this is simply:

$$\mathbf{Y} = \mathbf{V}^T \mathbf{X}$$

If we wish to recover the original data, we simply compute (using orthogonality of  $\mathbf{V}$ ):

$$\mathbf{X} = \mathbf{V} \mathbf{Y}$$

### 4.5.3 SVD for Dense Matrices

Golub and Kahan [72] introduced a two-step approach for computing SVD: convert the input matrix to a bidiagonal one and then perform SVD on the bidiagonal matrix. Demmel and Kahan [73] improved this approach by adding another step before bidiagonalization, which is QR decomposition. [68] referred to this method as SVD-Bidiag, which has the following three steps for a given matrix  $\mathbf{Y}$ :

1. compute the QR decomposition of  $Y$ , which results in an orthogonal matrix  $Q$  and an upper triangular matrix  $R$
2. transform  $R$  to a bidiagonal matrix  $B$
3. compute SVD on  $B$

The SVD-Bidiag algorithm is implemented in *RScalLAPACK*. Their analysis showed that the computational complexity of the SVD-Bidiag algorithm is dominated by the QR decomposition and bi-diagonalization steps, and is given by  $\mathcal{O}(ND^2 + D^3)$ . Therefore, the SVD-Bidiag algorithm is only suitable when  $D$  is small. More details can be found in [68].

#### 4.5.4 SVD for Sparse Matrices

SVD can be computed efficiently for sparse matrices using Lanczos' algorithm [74], which has a computational complexity of  $\mathcal{O}(Nz^2)$ , where  $z$  is the number of non-zero dimensions (out of  $D$  dimensions). More details can be found in [68].

## 4.6 Stochastic SVD (SSVD)

Randomized sampling techniques have recently gained popularity in solving large-scale linear algebra problems. The work in [69] describes a randomized method to compute approximate decomposition of matrices, which is referred to as stochastic SVD (SSVD). SSVD has two steps:

1. It uses randomized techniques to compute a low-dimensional approximation of the input matrix
2. It performs SVD on the approximation matrix

The accuracy of the results depends on the performance of the randomized techniques and the size of the approximation matrix. Accuracy can be improved through running the randomization step multiple times. Therefore, SSVD has the flexibility of trading off the accuracy of the results with the required computational resources.

## 4.7 Computational Complexity

Computational complexity of SSVD is dominated by the first step, which is  $\mathcal{O}(DNd)$ . This is a much better complexity than the previous techniques, because  $d$  is typically much smaller than  $D$  and is usually a constant. However, SSVD requires exchanging multiple intermediate matrices, which may cause a problem for scalability. The amount of intermediate data can be up to  $\mathcal{O}(\max(Nd, d^2))$  [68].



## 4.8 Probabilistic PCA (PPCA)

In our research work, we mainly focus on the probabilistic approach of PCA. That is why, we present *PPCA* in some kind of details. In this regard, we follow the research work of Tipping and Bishop [70]

### 4.8.1 What is PPCA

We can obtain a probabilistic formulation of PCA by introducing a Gaussian latent i.e. unobserved variable model. This kind of model is highly related to statistical factor analysis. First let us define some quantity:

- $\mathbf{y}$  is a  $D$  dimensional observed data vector
- $\mathbf{x}$  is a  $d$  dimensional latent variable
- $\mathbf{W}$  is a  $D \times d$  transformation matrix
- the columns of  $\mathbf{W}$  are the principal components
- $\boldsymbol{\mu}$  is the dimension wise mean vector of  $\mathbf{y}$ . This parameter allows the data to have non zero mean
- $\boldsymbol{\epsilon}$  is a  $D$ -dimensional zero-mean noise variable
- Here  $\mathbf{x}$ ,  $\boldsymbol{\epsilon}$ ,  $\mathbf{y}$  are normal distributed i.e. Gaussian distributed

$$\begin{aligned}\mathbf{x} &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \\ \mathbf{y} &\sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \sigma^2)\end{aligned}$$

where  $\mathbf{x} \sim \mathcal{N}(\mathbf{u}, \boldsymbol{\Sigma})$  denotes the Normal distribution with  $\mathbf{u}$  mean and  $\boldsymbol{\Sigma}$  covariance matrix.

A latent variable model seeks to relate a  $D$ -dimensional observation vector  $\mathbf{y}$  to a corresponding  $d$ -dimensional vector of latent variables  $\mathbf{x}$  where the relationship is linear:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \quad (4.4)$$

The motivation is that, with  $d < D$ , the latent variables will offer a more parsimonious explanation of the dependencies between the observations. The model parameters may thus be determined by maximum-likelihood, As there is no closed-form analytic solution for finding  $\mathbf{W}$  and  $\sigma^2$ , their values must be obtained via an iterative procedure.

### 4.8.2 The Probability Model

The use of the isotropic Gaussian noise model  $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$  for in conjunction with equation (4.4) implies that the  $\mathbf{x}$ -conditional probability distribution over  $\mathbf{y}$ -space is given by:

$$\mathbf{y}|\mathbf{x} \sim \mathcal{N}(\mathbf{W}\mathbf{x} + \boldsymbol{\mu}, \sigma^2 \mathbf{I}) \quad (4.5)$$

With the marginal distribution over the latent variables also Gaussian and conventionally defined by  $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , the marginal distribution for the observed data  $\mathbf{y}$  is readily obtained by integrating out the latent variables and is likewise Gaussian:

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C}) \quad (4.6)$$

where the observation covariance model is specified by  $\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}$ . given  $N$  observations  $\{\mathbf{y}_n\}_1^N$  as the input data, the log likelihood of data is given by:

$$\begin{aligned} \mathcal{L}(\{\mathbf{y}_r\}_1^N) &= \sum_{n=1}^N \ln p(\mathbf{y}_r) \\ &= -\frac{1}{N} \{D * \ln(2\pi) + \ln |\mathbf{C}| + \text{tr}(\mathbf{C}^{-1} * \mathbf{S})\} \end{aligned} \quad (4.7)$$

where  $\mathbf{S}$  is the sample covariance matrix of data  $\{\mathbf{y}_r\}$  given by:

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^N (\mathbf{y}_r - \boldsymbol{\mu})(\mathbf{y}_r - \boldsymbol{\mu})^T \quad (4.8)$$

and  $\text{tr}(\mathbf{M})$  is the trace of matrix  $\mathbf{M}$

### 4.8.3 Properties of the Maximum-Likelihood Estimators

According to [70], the likelihood equation 4.7 is maximized when:

$$\mathbf{W}_{ML} = \mathbf{U}_d \sqrt{\boldsymbol{\Lambda}_d - \sigma^2 \mathbf{I}} \mathbf{R} \quad (4.9)$$

where:

where the  $d$  column vectors in the  $D \times d$  matrix  $\mathbf{U}_d$  are the principal eigenvectors of  $\mathbf{S}_d$ , with corresponding eigenvalues  $\{\lambda_1, \dots, \lambda_d\}$  in the  $d \times d$  diagonal matrix  $\boldsymbol{\Lambda}_d$ , and  $\mathbf{R}$  is an arbitrary  $d \times d$  orthogonal rotation matrix. Thus, from Equation (4.9), the latent variable model defined by Equation (4.4) effects a mapping from the latent space into the principal subspace of the observed data.

It may also be shown that for  $\mathbf{W} = \mathbf{W}_{ML}$ , the maximum-likelihood estimator for  $\sigma^2$  is given by:

$$\sigma_{ML}^2 = \frac{1}{D-d} \sum_{i=d+1}^D \lambda_i \quad (4.10)$$

which has a clear interpretation as the variance ‘lost’ in the projection, averaged over the lost dimensions.

#### 4.8.4 An EM Algorithm for Probabilistic PCA

Probabilistic PCA (PPCA) is an example of a linear Gaussian model [26, 70]. The observed variable  $\mathbf{y}_c$  is related to a linear transformation of the latent variable  $\mathbf{x}$  so that

$$\mathbf{y}_c = \mathbf{W} * \mathbf{x} + \boldsymbol{\sigma}$$

where  $\mathbf{y}_c$  is a  $D$ -dimensional observed or data vector (mean centered),  $\mathbf{W}$  is a  $D \times d$ -dimensional principal subspace,  $\mathbf{x}$  is a  $d$ -dimensional Gaussian latent variable, and  $\boldsymbol{\sigma}$  is a  $D$ -dimensional zero-mean Gaussian distributed noise variable with covariance  $ss * \mathbf{I}$ . So, we can say,

$$\mathbf{x} \sim \mathcal{N}(0, \mathbf{I}), \boldsymbol{\sigma} \sim \mathcal{N}(0, ss * \mathbf{I}), \mathbf{y}_c \sim \mathcal{N}(0, \mathbf{W} * \mathbf{W}^T + ss * \mathbf{I})$$

Given fixed model parameters  $\mathbf{W}$  and  $ss$ , the following can be said about the hidden state  $\mathbf{x}$ , for some observation  $\mathbf{y}_c$ :

$$\begin{aligned} P(\mathbf{x}|\mathbf{y}_c) &= \frac{P(\mathbf{y}_c|\mathbf{x}) * P(\mathbf{x})}{P(\mathbf{y}_c)} \\ &= \frac{\mathcal{N}(\mathbf{y}_c|\mathbf{W} * \mathbf{x}, ss * \mathbf{I}) * \mathcal{N}(\mathbf{x}|0, \mathbf{I})}{\mathcal{N}(\mathbf{y}_c|0, \mathbf{W} * \mathbf{W}^T + ss * \mathbf{I})} \\ &= \mathcal{N}(\mathbf{x}|\boldsymbol{\beta} * \mathbf{y}_c, \mathbf{I} - \boldsymbol{\beta} * \mathbf{W}) \end{aligned} \quad (4.11)$$

where  $\boldsymbol{\beta} = \mathbf{W}^T * (\mathbf{W} * \mathbf{W}^T + ss * \mathbf{I})^{-1}$ . [70] proposed an Expectation Maximization (EM) algorithm which uses the inference (4.11) above in the Expectation (E) step to estimate the unknown state and then choose  $\mathbf{W}$  and the restricted  $ss$  in the Maximization (M) step so as to maximize the expected joint likelihood of the estimated  $\mathbf{x}$  and the observed  $\mathbf{y}_c$ . We can write down corresponding E-step and M-step in matrix formulation as follows at  $k^{\text{th}}$  iteration:

$$\mathbf{E}\text{-step:} \quad \mathbf{M} = (\mathbf{W}^k)^T * \mathbf{W}^k + ss^k * \mathbf{I}; \quad (4.12)$$

$$\mathbf{E}\text{-step:} \quad \mathbf{X} = (\mathbf{Y} \ominus \boldsymbol{\mu}) * \mathbf{W}^k * \mathbf{M}^{-1}; \quad (4.13)$$

$$\mathbf{E}\text{-step:} \quad \mathbf{XtX} = \mathbf{X}^T * \mathbf{X} + N * ss^k * \mathbf{M}^{-1}; \quad (4.14)$$

$$\mathbf{E}\text{-step:} \quad \mathbf{YtX} = (\mathbf{Y} \ominus \boldsymbol{\mu})^T * \mathbf{X}; \quad (4.15)$$

$$\mathbf{M}\text{-step:} \quad \mathbf{W}^{k+1} = \mathbf{YtX} * \mathbf{XtX}^{-1}; \quad (4.16)$$

$$\mathbf{M}\text{-step:} \quad ss_1 = \|\mathbf{Y} \ominus \boldsymbol{\mu}\|_F^2; \quad (4.17)$$

$$ss_2 = \text{trace}(\mathbf{XtX} * (\mathbf{W}^{k+1})^T * \mathbf{W}^{k+1}); \quad (4.18)$$

$$ss_3 = \sum_{n=1}^N \mathbf{X}_n * (\mathbf{W}^{k+1})^T * (\mathbf{Y} \ominus \boldsymbol{\mu})_n^T; \quad (4.19)$$

$$ss^{k+1} = (ss_1 + ss_2 - 2 * ss_3) / N / D \quad (4.20)$$

# Chapter 5

## Related Works

We have discussed the techniques of performing PCA in the preceding chapter. In this Chapter, we give a brief description of various methods which are already implemented based on these techniques to compute PCA. Here we analyze their time and space complexities along with their limitations. Since we are intended to provide a geo-distributed solution of PCA, we only consider the distributed settings, and so we do not discuss any single machine implementation (for example, [75]).

### 5.1 MLlib-PCA

MLlib [17] is a built-in machine learning library in Spark. For PCA, MLlib computes EVD of the covariance matrix of  $Y$  and mainly provides distributed computation of large covariance matrix. We refer to its method as MLlib-PCA.

**Complexity:** The major computational part of MLlib-PCA is the calculation of the covariance matrix. MLlib-PCA is a deterministic algorithm and does not leverage sparsity of the matrix, so for a data matrix with size  $N \times D$ , it takes  $\mathcal{O}(ND \times \min(N, D))$  time to calculate covariance matrix. Also, it creates a large dense matrix of size  $D \times D$ , which it needs to store, and thus its space complexity is  $\mathcal{O}(D^2)$ .

**Limitations:** This method offers the **least scalability** because both  $N$  and  $D$  can be very large. For example, even for a data with vertical dimension  $D = 128k$ , it **faces out-of-memory error**. It is mainly because it needs to store a large intermediate data (covariance matrix) of size  $\mathcal{O}(D^2)$ , and it quickly faces memory overflow error as the number of dimensions grows. MLlib-PCA is designed in a way that it can compute PCA in a distributed cluster with centralized dataset. In the current setup, there is no implementation of MLlib-PCA which can handle dataset that are geographically distributed.

## 5.2 Mahout-PCA

Mahout [16] is another popular library that provides the implementation of various machine learning algorithms for distributed computing. The primitive features of Apache Mahout are listed below.

- The algorithms of Mahout are written on top of Hadoop [59], so it works well in distributed environment. Mahout uses the Apache Hadoop library to scale effectively in the cloud.
- Mahout offers the coder a ready-to-use framework for doing data mining tasks on large volumes of data.
- Mahout lets applications to analyze large sets of data effectively and in quick time.
- Includes several MapReduce enabled clustering implementations such as k-means, fuzzy k-means, Canopy, Dirichlet, and Mean-Shift.
- Supports Distributed Naive Bayes and Complementary Naive Bayes classification implementations.
- Comes with distributed fitness function capabilities for evolutionary programming.
- Includes matrix and vector libraries.

Mahout computes PCA using SSVD for big data. We refer to this as Mahout-PCA.

**Complexity:** In Mahout-PCA, the majority of calculation lies in the matrix multiplication  $Q^T * Y_c$  ( $m \times N$  multiplied by  $N \times D$ ). Thus its computational complexity is  $\mathcal{O}(NDm)$ . The algorithm requires to store  $N \times m$  dimensional matrix  $Q$ . So the total intermediate data can be calculated as  $\mathcal{O}(Nm)$ .

**Limitations:** Mahout-PCA is more scalable than that of MLlib-PCA. This method needs to store a large intermediate data, which is the main bottleneck. As  $N$  is large, transmitting the  $N \times m$  sized intermediate data can take a very long time in communication and a lot of space in memory of each node. Furthermore, it **fails** to compute PCA on a dataset with dimension larger than  $5M$  (see Table 9.4 for experimental outcome) as it fails to store the parameter of dimension  $N \times m$ . As  $N$  increases, the size of intermediate data grows to a point when memory overflow error occurs.

## 5.3 sPCA

Elgamal et al. [18] presents a scalable implementation of Probabilistic PCA for distributed platforms, which they referred to as sPCA.

To improve the performance of the basic EM algorithm, sPCA incorporates several special features, such as

- *Mean Propagation to Leverage Sparsity:* The first optimization they proposed is the mean propagation idea, which preserves and utilizes the sparsity of the input matrix  $\mathbf{Y}$ . PPCA requires the input matrix to be mean-centered, meaning that the mean vector  $\mu$  must be subtracted from each row of the original matrix  $\mathbf{Y}$ . Large scale datasets, however, are mostly sparse, with many zero elements. Sparse matrices can achieve a small disk and memory footprint by storing only nonzero elements, and performing operations only over non-zero elements. Subtracting the non-zero mean from the matrix would make many elements non-zero, so the advantage of sparsity is lost. To avoid the problems of subtracting the mean, they keep the original matrix  $\mathbf{Y}$  and the mean  $\mu$  in two separate data structures. they did not subtract the mean  $\mu$  from  $\mathbf{Y}$ . Rather, they propagate the mean throughout the different matrix operations.
- *Minimizing Intermediate Data:* Intermediate data can slow down the distributed execution of any PCA algorithm, because it needs to be transferred to other nodes for processing to continue. Their second optimization is job consolidation, which means merging multiple distributed jobs into one in order to reduce the communication between these jobs.
- *Efficient Matrix Multiplication:* PPCA requires many matrix multiplications, which are expensive operations in a distributed setting. To appreciate the techniques that sPCA employs to overcome the inefficiency of matrix multiplication, they briefly explain different possible implementations of this operation.
- *Efficient Frobenius Norm Computation:* The PPCA algorithm requires computing the Frobenius norm of the mean-centered input matrix. To solve this problem, they design an algorithm which does not even require creating the dense vector. Many machine learning algorithms compute various norms of matrices. The proposed method for optimizing the computation of the Frobenius norm can be extended to other matrix norms using similar ideas. Thus, this simple optimization can benefit several other machine learning algorithms.

**Complexity:** The major part of computation in sPCA is done on calculating  $\mathbf{X}$  and  $s_{s_3}$  by multiplying  $N \times D$  sized mean-centered data  $\mathbf{Y}_c = (\mathbf{Y} \ominus \boldsymbol{\mu})$  by  $D \times d$  sized parameter  $\mathbf{W}^k / \mathbf{W}^{k+1}$ . So, there are **two** operations which take  $\mathcal{O}(NDd)$ . However, sPCA preserves sparsity in calculations. Therefore, if  $nnz(\mathbf{Y})$  represents the non-zero elements of  $\mathbf{Y}$ , the complexity would be  $\mathcal{O}(nnz(\mathbf{Y}) \times d)$ . Also, in each iteration, the parameter  $\mathbf{W}^k$  has to be passed and stored, resulting in the space complexity to be  $\mathcal{O}(Dd)$ .

**Limitations:** By far, sPCA offers the most scalability. However, similar to the case of MLlib-PCA and Mahout-PCA, it is also vulnerable to limitations. We discuss them from two aspects

that we are targeting in this work.

- The approach is not applicable for tall and wide big data with arbitrary number of dimensions. Though the approach has done some efficient use of memory by reducing the generation of intermediate data, (by requiring to store  $\mathcal{O}(Dd)$  parameter), sPCA too **faces out of memory**. In our single cluster setup, we fail to run sPCA on the AmazonRating dataset with dimension  $21M \times 9.8M$  (see Table 9.4 from Chapter 9 for more detail).
- On top of the scalability issue, there is no implementation of sPCA on geographically distributed big data. In current world where data is distributed across national border to ensure fast access and national data sovereignty, it is necessary to be capable of doing data analysis on such geo-distributed data. However, sPCA did not indicate a process of generating some kind of partial result and later accumulate them to produce the final analytic results on PCA.

## 5.4 sSketch-PCA

sSketch-PCA [19] utilizes the SSVD technique in the computation of PCA which provides a scalable implementation of the Gaussian sketch method and then uses it for PCA. Similar to sPCA, it also uses various optimization ideas, such as

- *Mean propagation for sparsity preservation*: Since most of the big data in real life are sparse and to calculate PCA on these data efficiently, sSketch-PCA do not form mean centered data matrix explicitly, rather it preserves the mean and propagates it in different calculations.
- *Effective job consolidation*: sSketch-PCA provides multiple optimization ideas in order to implement a faster version of QR decomposition, such as treereduce, Cholesky decomposition [76], and their combination where required. They eliminates some unnecessary broadcasts and redundant computations. Their overall optimization reduces the intermediate data generation and provides a faster algorithm.
- *On-the-fly computation*: To avoid explicitly materializing intermediate data, sSketch-PCA computes rows of intermediate data as they are needed and discard them when their use is expired. In this way, it has been successful in minimizing the generation of intermediate data, which eventually results in a better scalability.

**Complexity:** In the sketching phase, in order to generate a Gaussian sketch matrix,  $\Omega$  of dimension  $D \times d$ , numbers are randomly selected from Gaussian distribution with *zero* mean and *unit* variance. The major part of computation in sSketch lies in the generation of the sketched



data matrix  $\mathbf{Z}$  ( $D \times d$ ) by multiplying the sketch matrix with the mean-centered data matrix,  $\mathbf{Y}_c$  ( $N \times D$ ). Similar to sPCA, as sSketch incorporates sparsity preservation using mean propagation, the computation complexity is  $\mathcal{O}(nnz(\mathbf{Y}) \times d)$ , where  $nnz(\mathbf{Y})$  denotes the non-zero entries of  $\mathbf{Y}$ . In each iteration,  $\mathbf{Y}^T * \mathbf{Z}$  of dimension  $D \times d$  and  $\mathbf{Z}^T * \mathbf{Z}$  of dimension  $d \times d$  is stored and transmitted. Therefore, the space complexity is  $\mathcal{O}(Dd)$ .

**Limitations:** sSketch-PCA is the most scalable among the techniques which used SVD to compute PCA. By using various modifications, like avoiding both reduce operation and redundant computations, use of accumulators, and preservation of sparsity of both input and intermediate data, it successfully reduces the constant factors in the running time. However, it still requires a space complexity of  $\mathcal{O}(Dd)$ .

Therefore, similar to the other state-of-the-art methods, as the dimension,  $D$  becomes very high, sSketch also incurs **memory overflow error**. We fail to run sSketch-PCA when the dimension  $D$  reaches to  $10M$  (see the Chapter 9: Experimental Evaluation for more clarification).

For more details on the complexities and limitations of these state-of-the-art techniques, interested readers are encouraged to read the survey provided by Elgamal et al. [68].

## 5.5 Geo-Distributed Analytics and Large Parameter

In recent times, in order to ensure lower latency at the users' end, and to maintain data privacy, most of the companies are establishing their data storage closer to the end users generally bounded by national rules and regulations. Therefore, we can say that nowadays data are by born distributed. This kind of setup has evolved a new demand of a federated or geo-distributed learning technique that can gain any required insights from such geographically sparse data while keeping them at their residing locations.

To handle this challenge, recently, geo-distributed analytics has gained much attention to avoid bottlenecks that are incurred for pulling all geographically distributed data to a central location. For example, [22] proposed Geode for running SQL queries efficiently in the geo-distributed environment. Similarly, to reduce query response time, a system for low latency geo-distributed analytics, namely Iridium, was proposed in [21]. To meet new challenges for running machine learning algorithms in a generalized framework, Gaia and TernGrad was proposed in [39] and [40], respectively. Nevertheless, for PCA, which is a wonderful and widely used tool to reduce the dimensionality of higher dimensional data, to the best of our knowledge, there is no solution in geo-distributed settings. All the methods mentioned above for PCA run on aggregated data in a central DC. That means, the existing methods, which are designed based on centralized algorithms need to gather all the datasets from different geographic locations and centralize them in a single data cluster. However, as we already know that passing raw data across national boundaries has been prohibited by many countries. And even if it is allowed,

such communication across region is very costly and requires high amount of bandwidth.

Additionally, to overcome scalability challenges for the large parameter in other machine learning algorithms, parameter servers have been introduced in [77, 78] for single DC. However, in a geo-distributed environment, dedicating one DC as a parameter server is impractical.

Therefore, in this thesis, we propose a novel solution for handling large parameter for PCA in a single DC as well as in an environment of geographically distributed ones. That means, we address both the scalability issue of the existing PCA methods and the scarcity of an communication-efficient geo-distributed algorithm for PCA. We propose TallnWide, which is a highly scalable algorithm capable of handling arbitrarily large dimensional datasets. Additionally, we provide an efficient communication technique which pass only the PCA parameter to minimize the utilization of inter region B/W. Additionally, with various optimizations, our communication scheme provides maximum parallelism between computation and communication, minimize the overall idles times of CPU, disk and network I/O, and provides faster generation of the final result.

# Chapter 6

## Our Proposed Algorithm: TallnWide

As we have already mentioned that the existing methods of PCA are not scalable enough to handle tall and wide big data and they eventually face out of memory error when the dimensions of the data increase arbitrarily. On top of that, at present world, where most of the data is geographically distributed, there is no existing solution to handle geo-distributed datasets. To address these two issues, we introduce TallnWide, which is a highly scalable geo-distributed implementation of Probabilistic PCA. In this chapter, we discuss our proposed algorithm in details. For a better explanation of our works, we extend the notations we have used in the previous sections. Throughout the remaining part of this thesis we use these terms and notation in order to explain our work. Terms and notations are as follows:

- $S$  is the total number of DCs.  $d$  is our target dimension.
- $N_s$  represents number of rows of the data residing in  $s^{\text{th}}$  DC. So, the total number of rows is  $N = \sum_{s=1}^S N_s$ .
- $\beta$  is the total number of divided blocks of the parameter.
- $D_i$  represents the number of columns in the  $i^{\text{th}}$  block of the data. So, total number of columns  $D = \sum_{i=1}^{\beta} D_i$ .
- $\mathbf{Y}_s$  represents the dataset in  $s^{\text{th}}$  DC,  $\mathbf{Y}_{s,i}$  is the  $i^{\text{th}}$  block of the dataset residing in  $s^{\text{th}}$  DC (size  $N_s \times D_i$ ).  $\mathbf{Y}$  is the overall geo-distributed data, so we have:

$$\mathbf{Y} = \left[ \begin{array}{c|c|c} \mathbf{Y}_{11} & \dots & \mathbf{Y}_{1\beta} \\ \hline \dots & \dots & \dots \\ \hline \mathbf{Y}_{S1} & \dots & \mathbf{Y}_{S\beta} \end{array} \right]$$

We denote such **vertical concatenation** by  $\Xi$  and **horizontal concatenation** by  $\amalg$  so that

$$\mathbf{Y} = \Xi_{s=1}^S \mathbf{Y}_s = \Xi_{s=1}^S \amalg_{i=1}^{\beta} \mathbf{Y}_{s,i}$$

By **concatenation**, we do not mean any **accumulation** or **summation** (for which we use  $\sum$ ), rather we mean stacking on top/side of each other. Here,  $\mathbf{Y}_{s,i}$  is of  $N_s \times D_i$  size.

- $\boldsymbol{\mu} = \amalg_{i=1}^{\beta} \boldsymbol{\mu}_i$  is the mean vector of all columns.
- $\mathbf{W}^k = \Xi_{i=1}^{\beta} \mathbf{W}_i^k$  is the  $D \times d$  size principal components, i.e. our parameter, at  $k^{\text{th}}$  iteration. Here,  $\mathbf{W}_i^k$  is the  $i^{\text{th}}$  block of  $\mathbf{W}^k$  (size  $D_i \times d$ ) at each DC at  $k^{\text{th}}$  iteration.
- $\mathbf{X} = \Xi_{s=1}^S \mathbf{X}_s$  is the  $N \times d$  size latent data, where  $\mathbf{X}_s$  (size  $N_s \times d$ ) is the latent data at  $s^{\text{th}}$  DC.
- In the scenario of tall and wide big data analysis, generally the data  $\mathbf{Y}$  with dimension  $N \times D$  is large and sparse and the parameter  $\mathbf{W}$  with dimension  $D \times d$  is large and dense.

## 6.1 Handling Tall and Wide Big Data

By *Tall and Wide Big Data*, we mean both  $N$  and  $D$  are quite large, and as  $D \approx N$ , parameters for PCA increase proportionately and thus they will not fit into the memory of the slave machines in a single DC. To be specific, as data dimension increases, the amount of intermediate data generated throughout the algorithmic steps becomes so high that the memory overflows and the algorithm fails to generate any concluding result. To overcome this challenge in scalability, we divide the parameter into manageable chunks/blocks to divide/distribute computations among the working nodes in order to get faster result and scale the computation. In this approach, since we need to handle a manageable size of the PCA parameter, which decreases the generation of intermediate data and eventually can minimize the occurrence of memory overflow error. The higher the data dimensions get, we have the scope to increase the division count so that more smaller blocks are generated in order to keep the intermediate data generation in control. In this way, our algorithm is not vulnerable to ever saturates in scalability.

However, splitting the parameter into blocks to perform PCA is a non-trivial task. For example, when we try to isolate matrix computations of EVD for PCA, we reach a dead-end because EVD requires the  $D \times D$  covariance matrix as a whole for decomposition [1]. Similarly, SSVD requires storing of  $N \times m$  dimensional matrix  $\mathbf{Q}$ , and so it too has a high memory requirement. Compared to these techniques, we find that PPCA holds significant promise since it has relatively low memory footprint [68] and there is no additional decomposition involved.

Second part of the challenge is to reduce steps of the algorithm in simple matrix-matrix and/or matrix-vector operations. We have to make sure that the division does not cause unnecessary

overheads or bottlenecks. We also have to divide other intermediate data into blocks as well and resolve inter-dependencies. We should emphasize that due to these various challenges, we have not seen any block-division algorithm for PCA in a distributed platform.

In PPCA, our initial goal is to generate  $\mathbf{W}^k$  first, for  $k^{\text{th}}$  iteration.  $\mathbf{W}^k$  has to reside at each DC as it is the central parameter for the whole algorithm. Since we only want to calculate principal components, we can ignore the noise and only run the EM algorithm with zero-noise (considering the value of noise to be zero) to refine the parameter  $\mathbf{W}^k$ . We refer to this as *zero-noise-limit PPCA*. Later, we will provide proof that zero-noise-limit PPCA indeed outputs the correct principal components.

As noise is zero, we do not need (4.17) - (4.20) and steps of PPCA, (4.12) - (4.16) are changed as follows:

$$\begin{aligned}
 \mathbf{E}\text{-step:} \quad & \mathbf{M} = (\mathbf{W}^k)^T * \mathbf{W}^k; \\
 & \mathbf{X} = (\mathbf{Y} \ominus \boldsymbol{\mu}) * \mathbf{W}^k * \mathbf{M}^{-1}; \\
 \mathbf{E}\text{-step:} \quad & \mathbf{XtX} = \mathbf{X}^T * \mathbf{X}; \\
 & \mathbf{YtX} = (\mathbf{Y} \ominus \boldsymbol{\mu})^T * \mathbf{X}; \\
 \mathbf{M}\text{-step:} \quad & \mathbf{W}^{k+1} = \mathbf{YtX} * \mathbf{XtX}^{-1}
 \end{aligned}$$

Now, we can easily divide the computations into smaller chunks. This proves to be significant and crucial when handling parameter of a bigger size. At a time,  $s^{\text{th}}$  DC works with the  $i^{\text{th}}$  block of the parameter that fits in the memory, i.e. at  $k^{\text{th}}$  iteration, only  $\mathbf{W}_i^k$  fits. When  $k = 1$ ,  $\mathbf{W}^1$  is initialized randomly. Now, block-wise division of computation for  $\mathbf{M}$  looks like this:

$$\mathbf{E}\text{-step:} \quad \mathbf{M} = (\mathbf{W}^k)^T * \mathbf{W}^k = \sum_{i=1}^{\beta} (\mathbf{W}_i^k)^T * \mathbf{W}_i^k$$

This calculation is illustrated as the left figure in Figure 6.1. Each DC generates  $(\mathbf{W}_i^k)^T * \mathbf{W}_i^k$  at a time, and all the results from blocks have to be added locally for getting full result  $\mathbf{M}$ . Note that each DC has the same  $\mathbf{M}$  after this operation.

Now, we divide the computation of  $\mathbf{X}$  as follows:

$$\begin{aligned}
 \mathbf{E}\text{-step:} \quad \mathbf{X} &= \overline{\bigoplus}_{s=1}^S \mathbf{X}_s = \overline{\bigoplus}_{s=1}^S (\mathbf{Y}_s \ominus \boldsymbol{\mu}) * \mathbf{W}^k * \mathbf{M}^{-1} \\
 &= \left( \overline{\bigoplus}_{s=1}^S \sum_{i=1}^{\beta} (\mathbf{Y}_{s,i} * \mathbf{W}_i^k \ominus \boldsymbol{\mu}_i * \mathbf{W}_i^k) \right) * \mathbf{M}^{-1} \\
 &= \left( \overline{\bigoplus}_{s=1}^S \sum_{i=1}^{\beta} \mathbf{Z}_{s,i} \right) * \mathbf{M}^{-1} = \left( \overline{\bigoplus}_{s=1}^S \mathbf{Z}_s \right) * \mathbf{M}^{-1}
 \end{aligned}$$

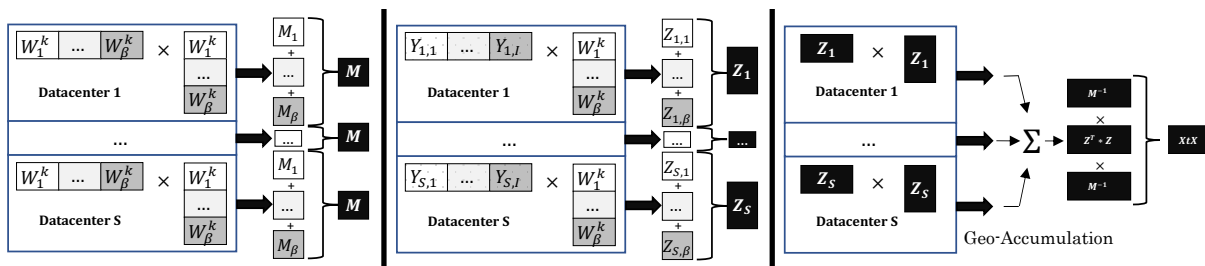


Figure 6.1: Generation of  $M$  (left),  $Z$  (middle), and  $XtX$  (right) in parallel fashion among all DCs. For simplicity, operations for mean vector,  $\mu$ , are not shown in the middle figure.

where,  $Z_{s,i} = (Y_{s,i} * W_i^k \ominus \mu_i * W_i^k)$  and  $Z_s = (Y_s * W^k \ominus \mu * W^k)$ .

Computation of  $Z_s$  is illustrated as the middle figure in Figure 6.1. Interestingly, we need not form full  $X$ , we need only  $Z_s$  at each DC and multiply it by  $M^{-1}$  (which each DC already has) for computing  $XtX$  (illustrated as the right figure in Figure 6.1):

**E-step:**

$$\begin{aligned}
 XtX &= X^T * X \\
 &= \left( \bigoplus_{s=1}^S X_s^T \right) * \left( \bigoplus_{s=1}^S X_s \right) \\
 &= \sum_{s=1}^S X_s^T * X_s \\
 &= M^{-1} * \left( \sum_{s=1}^S Z_s^T * Z_s \right) * M^{-1}
 \end{aligned}$$

For the last stage of the expectation, we need not form  $YtX$  explicitly. Instead, we can go directly to the maximization stage and derive the parameter:

**M-step:**

$$\begin{aligned}
 W^{k+1} &= \left( \left( \bigoplus_{s=1}^S (Y_s \ominus \mu) \right)^T * \left( \bigoplus_{s=1}^S X_s \right) \right) * XtX^{-1} \\
 &= \left( \left( \bigoplus_{s=1}^S \prod_{i=1}^{\beta} (Y_{s,i} \ominus \mu_i) \right)^T * \left( \bigoplus_{s=1}^S Z_s \right) \right) * M^{-1} * XtX^{-1} \\
 &= \sum_{s=1}^S \left( \bigoplus_{i=1}^{\beta} (Y_{s,i}^T * Z_s * MXtX - \mu_i^T * z_s * MXtX) \right)
 \end{aligned}$$

where vector  $z_s$  denotes the sum of all rows of  $Z_s$  and  $MXtX = M^{-1} * XtX^{-1}$ . Notice that, since subtracting the mean vector from the data matrix results in creating a dense matrix, at each step, we consider operations with mean vector  $\mu$  separately for preserving sparsity in the input matrix  $Y$ . We refer to this as *mean propagation* [68]. Note that their notation  $\sum_{s=1}^S$  signifies that just similar to the intermediate data  $Z_s$ , this derivation of the parameter  $W^{k+1}$  at any  $k^{th}$

iteration requires the geo-accumulation of the partial parameters generated at each DC. The following Section demonstrate how to introduce a communication efficient accumulation process to generate the final result form the partially generated ones in a faster way.

## 6.2 Communication Efficient Calculation

The order at which we perform the matrix operations for our block-division plays a significant role in communication efficiency. In the block-division method, the partial results generated at each DC will be gathered by a central DC, and the aggregated results will be transmitted back to all other DCs. We refer to this process as **Geo-Accumulation** or simply **accumulation**. We now present various approaches of geo-accumulation to transfer our intermediate results.

**Approach 1: Trivial Order.** At the current order of computing  $W^{k+1}$  mentioned in the **M Step** in the earlier section, the following happens:

$$W^{k+1} = \underbrace{\sum_{s=1}^S \left( \underbrace{\sum_{i=1}^{\beta} (Y_{s,i}^T * Z_s * MXtX - \mu_i^T * z_s * MXtX)}_{\text{Geo-Accumulation of partial results}} \right)}_{\text{Partial results of all blocks from all DCs}} \quad (6.1)$$

As  $W^k$  has been horizontally partitioned into  $\beta$  blocks, in this order, each DC will generate partial results for each block. After that, all the partial results from every DC will be accumulated by the central DC and multiplied by  $MXtX$  to get the final result  $W^{k+1}$ . In the current order, raw data need not be transmitted, but it creates a tremendous amount of idle time in worker nodes. To see why let us imagine the case for a single DC. As at a time only one block fits into the memory, the worker nodes of a single DC produces partial results for  $i = 1$  first, and save it to the secondary storage, for making room for next segments (i.e.,  $i = 2$ ) in memory. In this fashion, we first have to generate (incurring CPU time) and store (incurring Disk I/O) all partial results for each block in each DC. When it is finished, we have to retrieve (which has a Disk I/O) each block from each DC for accumulation (which has a Network I/O) by the central DC. After accumulation, the aggregated result of each block will be transmitted back to each DC and stored again. This is shown as Approach 1 in Figure 6.2. It is undoubtedly an in efficient approach. We are intended to maximize the parallelism and reduce the amount of idle times on our resources. That is why we prefer the following efficient order of accumulation.

**Approach 2: Efficient Order.** Now, we consider the reversed order of the computation, as mentioned in (6.1) above. The reversed order follows:

$$W^{k+1} = \underbrace{\sum_{i=1}^{\beta} \left( \underbrace{\sum_{s=1}^S (Y_{s,i}^T * Z_s * MXtX - \mu_i^T * z_s * MXtX)}_{\text{Full result for all blocks}} \right)}_{\text{Geo-Accumulation for full result of } i^{\text{th}} \text{ block}} \quad (6.2)$$

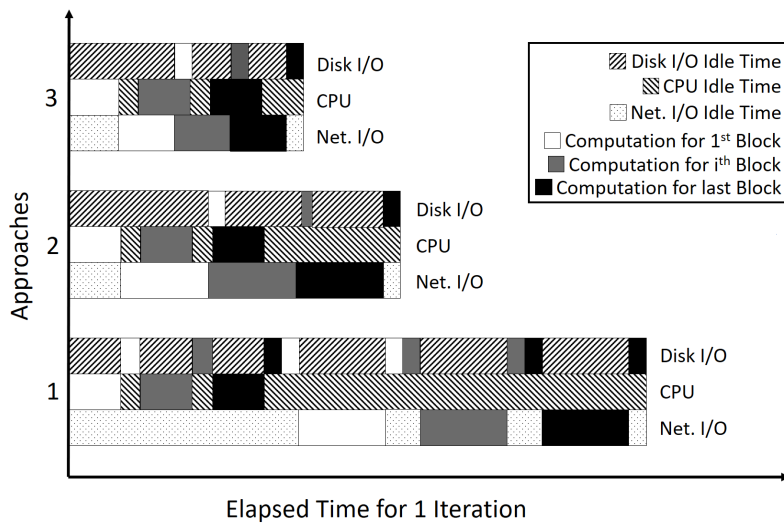


Figure 6.2: Comparison of idle time among different approaches. Approach 1: Trivial Order, Approach 2: Efficient Order, and Approach 3: Efficient Order w/ Ideal Central DC.

In this order, we can accumulate partial results for each block from each DC after they are generated. While we accumulate for one block, we can start the calculation for the next block because there is no dependency. This is shown as Approach 2 in Figure 6.2. That means when any DC is finished with the generation of  $W_1$  it sends the partial  $W$  for accumulation and start the generation of the second block  $W_2$ . In this fashion, the computation of any  $i^{th}$  block of  $W$  can be run in parallel with the accumulation of the  $(i - 1)^{th}$  block. In contrast to Approach 1 from Figure 6.2, we can see that Approach 2 reduces the idle time significantly. This kind of highly efficient ordering technique which enables parallelism in computation and transmission is known as Grouped Aggregate Pushdown [79] and very popular in the database research community.

**Approach 3: Efficient Order w/ Ideal Central DC.** In addition to Approach 2, as every DC has to send their partial results to a central DC for accumulation, we should choose such a DC as the central one for which the slowest link is maximum among all the candidate DCs. Mathematically, let us first assume that we have a symmetric B/W matrix  $B$  where each cell  $B_{uv}$  denotes B/W between DC  $u$  and DC  $v$  ( $u, v \in \{1, \dots, S\}$ ). For every DC  $u$ , we first determine the slowest B/W link from its connections to all other DCs  $v$  ( $v \neq u, v \in \{1, \dots, S\}$ ). Then from a set of such B/Ws for every DC  $u$ , we can select the DC for which the following is true:

$$\operatorname{argmax}_{u \in \{1, \dots, S\}} \left( \min \{ B_{uv} | v \neq u, v \in \{1, \dots, S\} \} \right) \quad (6.3)$$

If there are multiple DCs, we can select any of them. We refer this selected central DC for accumulation as ideal central DC. This is shown as Approach 3 in Figure 6.2, and theoretically by using this ideal central DC the time required to collect the partial results from other DCs and the redistribution task should be the minimum, and thus it offers the fastest accumulation. Later, we will validate the merit of this final approach through our experiment.



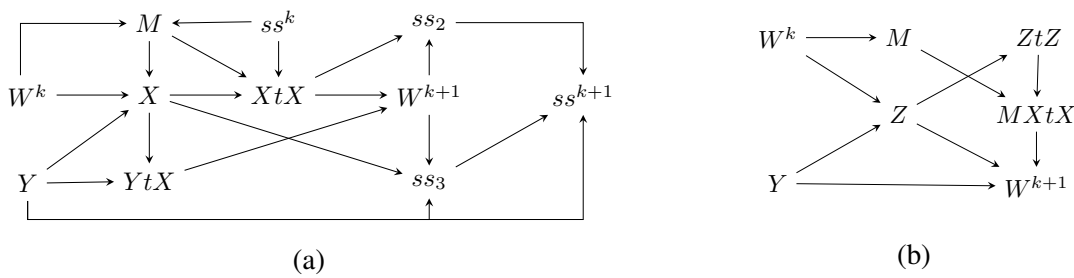


Figure 6.3: Matrix dependency diagram of two variations: (a) conventional PPCA (implemented in sPCA) and (b) zero-noise-limit PPCA (implemented in TallnWide)

## 6.3 Validation of Zero-Noise-Limit Probabilistic PCA

In this section, we validate zero-noise-limit PPCA over the conventional PPCA according to [26]. We first show that zero-noise-limit PPCA produces the correct principal components. We present a formal proof for this claim.

**Lemma 6.3.1** *PCA is a limiting case of the linear-Gaussian model in (4.11) as the covariance of the noise  $\sigma$  becomes infinitesimally small and equal in all directions, i.e.  $\sigma = \lim_{ss \rightarrow 0} ss * \mathbf{I}$ .*

*Proof.* For  $\sigma = \lim_{ss \rightarrow 0} ss * \mathbf{I}$ , inference (4.11) becomes:

$$\begin{aligned}
 P(\mathbf{x}|\mathbf{y}_c) &= \mathcal{N}(\mathbf{x}|\beta * \mathbf{y}_c, \mathbf{I} - \beta * \mathbf{W}); \\
 \beta &= \lim_{ss \rightarrow 0} \mathbf{W}^T * (\mathbf{W} * \mathbf{W}^T + ss * \mathbf{I})^{-1} \\
 P(\mathbf{x}|\mathbf{y}_c) &= \mathcal{N}(\mathbf{x} | (\mathbf{W}^T * \mathbf{W})^{-1} * \mathbf{W}^T * \mathbf{y}_c, 0) \\
 &= \delta(\mathbf{x} - (\mathbf{W}^T * \mathbf{W})^{-1} * \mathbf{W}^T * \mathbf{y}_c)
 \end{aligned}$$

Since the noise has become infinitesimal, the posterior over states collapses to a single point, and the covariance becomes zero. This has the effect of making the likelihood of a point  $\mathbf{y}_c$  dominated solely by the squared distance between it and its reconstruction  $\mathbf{W} * \mathbf{x}$ . But the directions of the columns of a matrix which minimize this error are known as the *principal components*. As columns of  $\mathbf{W}$  have this property, we have our desired output.  $\square$

This version of PPCA is what we have referred to as *zero-noise-limit PPCA*. And we just have designed the block-division EM-algorithm for this version. For more details and correctness of the EM-steps, interested readers are encouraged to read [26].

This zero-noise-limit PPCA has significant advantage over conventional PPCA. To explain why, let us consider the scenario where we calculate noise  $ss$  in our steps according to Equations 4.17-4.20. For  $ss$ , we have to calculate  $ss_1$  (which needs to be calculated only once),  $ss_2$  and

$ss_3$ . Among them, the calculation of  $ss_3$  is the most complicated:

$$ss_3 = \sum_{n=1}^N \sum_{s=1}^S \left( \mathbf{X}_{s,n}^T * \left( \prod_{i=1}^{\beta} (\mathbf{W}_i^{k+1})^T * \sum_{i=1}^{\beta} (\mathbf{Y}_{s,i,n}^T \ominus \boldsymbol{\mu}_i^T) \right) \right)$$

We can see that the calculation of  $ss_3$  in  $k^{\text{th}}$  iteration is dependent on the calculation of entirely accumulated updated parameter for the next iteration, i.e.,  $\mathbf{W}^{k+1}$ . This unnecessarily complicates the computation and introduces inter-dependency between steps and reduces the scope for maximization of parallelism. To illustrate, let us contrast the dependency diagram of our zero-noise-limit PPCA and the conventional PPCA. From Figure 6.3, it is evident that conventional PPCA has additional complications while zero-noise-limit PPCA requires simple and fewer steps.

With this validation, we conclude our overall design for the algorithm TallnWide. We have introduced a novel block division method for the *zero-noise-limit PPCA* which is capable of handling tall and wide big data without facing memory out-of-memory error. Our block division method minimizes the generation of intermediate data and the merit of zero-noise-limit PPCA is the simplification of the overall computation. In case of geo-distributed environment, we identify an efficient ordering of computation and communication so that the maximum parallelism is achieved and the overall execution time can be minimized. We also provide a theoretical formula to identify an ideal central DC which is assigned to carry out the responsibility of aggregating the partial results and redistribute the accumulated one to each of the DCs. In the following Chapter, we provide a formal algorithmic pseudo code of TallnWide along with its time and space complexity.

# Chapter 7

## Algorithm and Complexity Analysis

In this section, firstly, we give a detailed description of our two main algorithms, namely TallnWide, and Geo-Accumulation, which are shown in Algorithm 1, and 2, respectively. Algorithm TallnWide shows the necessary steps of performing PCA on tall and wide big data in a geographically distributed environment. On the other hand, Algorithm Geo-Accumulation performs the distributed accumulation task using the detected ideal DC. It should be noted that TallnWide algorithm is designed for a generalized geo-distributed environment which is capable of handling tall and wide datasets irrespective of the data being geo-distributed or not. TallnWide can be run in two different modes: i) On a single cluster by setting the number of servers  $S$  to zero. We refer to this mode of execution as the *standalone mode*. ii) On geographically distributed clusters where each datacenter holds its own version of data. In summary, TallnWide is capable of handling tall and wide big data in both standalone and in geo-distributed modes. The computational, communication, and space complexities are analyzed in the latter part of this section.

### 7.1 Algorithm Description

We summarize the basic steps of TallnWide in Algorithm 1. To reduce communication time for the first iteration, we only send the seed for a random number generator so that each DC can generate the same  $\mathbf{W}^1$  locally. In Line 1, “normRand(D,d)” produces random Gaussian matrix of size  $D \times d$  with the given seed. In Line 2 and 3, “detIdealDC(B)” and “detNumBlock( )” respectively determines the ideal central DC “ $iDC$ ” from B/W matrix  $\mathbf{B}$  and number of blocks  $\beta$ . The steps of determining the ideal central DC are designed according to the equation 6.3. In the next line, “mean(Y)” outputs the mean matrix  $\boldsymbol{\mu}$ . Line 5 and 6 respectively initialize the two matrices  $\mathbf{M}$  (size  $d \times d$ ) and  $\mathbf{Z}_s$  (size  $N_s \times d$ ) with zero values. “load( $\mathbf{W}^k, i$ )” loads  $i^{\text{th}}$  block from  $\mathbf{W}^k$  in Line 10. We derive necessary intermediate data, namely  $\mathbf{M}$ ,  $\mathbf{Z}$ ,  $\mathbf{Z}t\mathbf{Z}$  and  $\mathbf{M}XtX$  (see Figure 6.3(b)), for calculating  $\mathbf{W}^{k+1}$  in Line 11, 13, 15 and 16. Line 18 to

**Algorithm 1** TallnWide

---

**Input** : Data matrix  $\mathbf{Y}$  of dimension  $N \times D$ ,  
target dimension  $d$ ,  
number of servers  $S$ ,  
B/W matrix  $\mathbf{B}$

**Output:** Principal Component  $\mathbf{W}$

- 1  $\mathbf{W}^1 = \text{normRand}(D, d)$   $\triangleright \mathcal{O}_c(1)$
- 2  $iDC = \text{detIdealDC}(\mathbf{B})$
- 3  $\beta = \text{detNumBlock}()$
- 4  $\boldsymbol{\mu} = \text{mean}(\mathbf{Y})$   $\triangleright \mathcal{O}_c(D)$
- 5  $\mathbf{M} = \text{newMatrix}(d, d)$
- 6  $\mathbf{Z}_s = \text{newMatrix}(N_s, d)$
- 7 **for each iteration**  $k \leftarrow 1$  **to**  $K$  **do**
- 8   **for each DC**  $s \in \{1, \dots, S\}$  **do**
- 9     **for each block**  $i \leftarrow 1$  **to**  $I$  **do**
- 10       $\mathbf{W}_i^k = \text{load}(\mathbf{W}^k, i)$
- 11       $\mathbf{M} += (\mathbf{W}_i^k)^T * \mathbf{W}_i^k$   $\triangleright \mathcal{O}_t(D_i d^2)$
- 12       $\mathbf{Z}_m = \boldsymbol{\mu}_i * \mathbf{W}_i^k$
- 13       $\mathbf{Z}_s += \mathbf{Y}_{s,i} * \mathbf{W}_i^k \ominus \mathbf{Z}_m$   $\triangleright \mathcal{O}_t(\text{nnz}(\mathbf{Y}_{s,i})d)$
- 14     **end**
- 15      $\mathbf{ZtZ} = \text{Geo-Accumulation}(\mathbf{Z}_s^T * \mathbf{Z}_s, iDC)$   $\triangleright \mathcal{O}_c(d^2)$
- 16      $\mathbf{MXtX} = \mathbf{M}^{-1} * (\mathbf{M}^{-1} * \mathbf{ZtZ} * \mathbf{M}^{-1})^{-1}$   $\triangleright \mathcal{O}_t(d^2)$
- 17     **end**
- 18     **for each block**  $i \leftarrow 1$  **to**  $\beta$  **do**
- 19      **for each DC**  $s \in \{1, \dots, S\}$  **do**
- 20        $\mathbf{W}_{s,i}^{k+1} = \mathbf{Y}_{s,i}^T * \mathbf{Z}_s * \mathbf{MXtX} - \boldsymbol{\mu}_i^T * \mathbf{z}_s * \mathbf{MXtX}$   $\triangleright \mathcal{O}_t(\text{nnz}(\mathbf{Y}_{s,i})d)$
- 21        $\mathbf{W}_i^{k+1} = \text{Geo-Accumulation}(\mathbf{W}_{s,i}^{k+1}, iDC)$   $\triangleright \mathcal{O}_c(D_i d)$  and  $\mathcal{O}_s(D_i d)$
- 22      **end**
- 23     **end**
- 24     **if converged then**
- 25        $\mathbf{W} = \mathbf{W}^{k+1}$
- 26       *stop*
- 27     **end**
- 28 **end**
- 29 **return**  $\mathbf{W}$

---

**Algorithm 2** Geo-Accumulation

---

**Input** : Partial result  $\mathbf{A}$  in Matrix form,  
 $iDC$  which denotes the ID of Ideal DC

```

1  $myID = loadID()$ 
2 if ( $myID == iDC$ ) then
3    $DCList = \{1, \dots, S\} - \{myID\}$ 
4   while (not done for every  $s \in DCList$ ) do
5     if  $notifiedFrom(s)$  then
6        $addNewProcess(\mathbf{A} = \mathbf{A} + \mathbf{A}_s)$ 
7       remove  $s$  from  $DCList$ 
8     end
9   end
10  for each DC  $s \in DCList$  do
11     $sendData(\mathbf{A}, s)$ 
12  end
13 end
14 else
15   $sendData(\mathbf{A}, iDC)$ 
16   $notifyMaster(iDC)$ 
17 end

```

---

Line 23 refer to necessary operations for (6.2). “Geo-Accumulation(Matrix  $\mathbf{A}$ ,  $iDC$ )” performs geo-distributed accumulation of any data matrix  $\mathbf{A}$  using “ $iDC$ ” in Line 15 and 21. Finally, we check convergence in Line 24. It should be mentioned that this TallnWide algorithm shows the steps done in a generalized platform-independent environment. However, we describe the detailed computation of two of our significant steps in Spark distributed environment in the following section. Algorithm 2 shows the process of accumulating partial results in details. If the DC itself is the ideal central DC, then it accumulates all the partial results that are already received from the other DCs. The “ $notifiedFrom(s)$ ” function determines whether any **Done** notification is received from DC  $s$  or not. However, upon receiving  $\mathbf{A}_s$  (the partial result of full data  $\mathbf{A}$  in matrix form received from any DC  $s$ ), the ideal DC starts a new process to accumulate it and remove  $s$  from the consideration list. At the end of the accumulation process, the ideal central DC redistributes the final result to each of the DCs. On the other hand, if the DC is not the ideal central DC itself, then it concludes its accumulation task by sending the partial result  $\mathbf{A}$  which is to be accumulated to the central DC and by notifying (sending a **Done** notification) it about the task.

## 7.2 Complexity Analysis

We denote time, space and communication complexities by  $\mathcal{O}_t$ ,  $\mathcal{O}_s$ , and  $\mathcal{O}_c$ , respectively. Now we show the line-wise complexity in Algorithm 1. Since we are only sending the seed for a random number generator, Line 1 has a communication complexity of  $\mathcal{O}_c(1)$ . Line 4 sends the partial mean value at each datacenter, and the aggregated mean generation demands the use of geo-distributed bandwidth which has the communication complexity of  $\mathcal{O}_c(D)$ . Both  $M$  and

$M\mathbf{X}t\mathbf{X}$  are generated redundantly in each of the datacenters with computational complexities of  $\mathcal{O}_t(D_i d^2)$  and  $\mathcal{O}_t(d^2)$ , respectively.

Table 7.1: Comparison of complexities among all methods. Since we divide the parameters into blocks, our space complexity is less than others. Also, despite having the same time complexity w.r.t. sPCA and sSketch-PCA, we reduce the constant factors in running time of TallnWide significantly.

Method	Time Complexity	Space Complexity
MLlib-PCA	$\mathcal{O}_t(ND \times \min(N, D))$	$\mathcal{O}_s(D^2)$
Mahout-PCA	$\mathcal{O}_t(NDm)$	$\mathcal{O}_s(Nm)$
sPCA	$\mathcal{O}_t(\text{nnz}(\mathbf{Y}) \times d)$	$\mathcal{O}_s(Dd)$
sSketch-PCA	$\mathcal{O}_t(\text{nnz}(\mathbf{Y}) \times d)$	$\mathcal{O}_s(Dd)$
TallnWide	$\mathcal{O}_t(\text{nnz}(\mathbf{Y}) \times d)$	$\mathcal{O}_s(D_i d)$

However, only major computation in each iteration is the full calculation of  $\mathbf{Y} * \mathbf{W}$  and  $\mathbf{Y}^T * \mathbf{Z}$  (aggregated result from Line 13 and Line 20) which takes  $\mathcal{O}_t(\text{nnz}(Y) \times d)$  time (we **preserve sparsity** in multiplication in Line 13 and 20 by mean propagation). Similarly, in each iteration, only major data for storing is the block of the PCA parameter, i.e.  $\mathbf{W}_i$  (size  $D_i \times d$ ) in Line 21. So the space complexity is  $\mathcal{O}_s(D_i d)$ . Note that  $D_i < D$ . Finally, accumulation time of  $\mathbf{W}^{k+1}$  in Line 21 dominates others, so communication complexity is  $\mathcal{O}_c(D_i d)$ . As other methods are not geo-distributed, we only compare time and space complexities of TallnWide in Table 7.1. Note that, as we do not calculate noise, we reduce the constant factors in computation time by a significant fraction. Also, because of block-division, our space complexity is less than others, and consequently, we do not face the out-of-memory error.

# Chapter 8

## Experimental Design

As we have already mentioned in the Introduction that we implemented our TallnWide algorithm on a very popular distributed framework Spark. In this chapter, first we provide a detailed discussion of Spark implementation of our algorithm. After that, we discuss the setup that we use to run our experiments. In the following chapter, we discuss the experimental outcome.

### 8.1 Spark Implementation

The main goal of our algorithm is to calculate PCA in a geographically distributed setup. To achieve a distributed environment, we consider the distributed framework Spark [27]. In this section, we show spark implementation based computation steps to generate necessary intermediate data during the lifetime of our main TallnWide algorithm (Algorithm 1).

Algorithm 3 shows how we achieve the task of generating the intermediate data  $\mathbf{Z}$  (Line 13 from Algorithm 1) in Spark environment. In Line 1, the *zip* operation combines the data matrix  $\mathbf{Y}$  with the intermediate matrix  $\mathbf{Z}$  of the previous iteration (in case of the first iteration, the input  $\mathbf{Z}$  is the zero-filled initial matrix). The map operation in Line 3, which runs for each  $i^{th}$  row of  $\mathbf{Y}_n\mathbf{Z}$  can be carried out in parallel by each of the worker nodes. For each row of  $\mathbf{Y}_n\mathbf{Z}$ , in Line 4, the temporary variable  $Y_i$  retrieves the corresponding row from data matrix  $\mathbf{Y}$ . Only those values that fall within the horizontal dimension range from *start* to *end* are extracted. On the other hand, at Line 5, the temporary variable  $Z_i$  retrieves the corresponding row from the previous intermediate data matrix  $\mathbf{Z}$ .

After that,  $Y_i$  is multiplied by the principal subspace matrix  $\mathbf{W}^k$  of the previous iteration to generate a quantity *dotRes*. Finally, in Line 7, this *dotRes* is added to the variable  $Z_i$ , while the corresponding row from  $\mathbf{Z}_m$  is subtracted. When all the worker nodes are done with the map operation, a new version of the intermediate data  $\mathbf{Z}$  is generated and returned.

Generation of our parameter  $\mathbf{W}$  (Line 20 from Algorithm 1) can be divided into two parts: computing  $(\mathbf{Y}_{s,i}^T * \mathbf{Z}_s - \boldsymbol{\mu}_i^T * \mathbf{z}_s)$  which we refer to as generating  $\mathbf{Y}t\mathbf{Z}$  and multiplying it with

**Algorithm 3** SegmentedZJob

---

**Input** : Principal Subspace Matrix  $\mathbf{W}^k$ ;  
 Data Matrix  $\mathbf{Y}$ ;  
*start* which denotes the starting Index for Block I;  
*end* which denotes the ending Index for Block I;  
 Intermediate Matrix  $\mathbf{Z}$  from previous iteration;  
 $\mathbf{Z}_m$  which denotes the mean( $\mathbf{Y}$ ) multiplied by  $\mathbf{W}$

**Output:** Updated Intermediate Matrix  $\mathbf{Z}$

```

1  $\mathbf{Y}_n\mathbf{Z} = \mathbf{Y}.zip(\mathbf{Z})$ 
2  $\mathbf{ZSum} = accumulator(newMatrix(N_s, d))$ 
3  $\mathbf{Y}_n\mathbf{Z}.map\{(Y_n\mathbf{Z})_i \Rightarrow$                                  $\triangleright$  Runs for each  $i^{th}$  row of  $\mathbf{Y}_n\mathbf{Z}$ ;
4    $Y_i = (Y_n\mathbf{Z})_i.arg0().range(start, end)$ 
5    $Z_i = (Y_n\mathbf{Z})_i.arg1()$ 
6    $dotRes = Y_i \times \mathbf{W}^k$ 
7    $\mathbf{ZSum}.add(Z_i + dotRes - (Z_m)_i)$ 
8 }
9  $\mathbf{Z} = \mathbf{ZSum}.value()$ 
10 return  $\mathbf{Z}$ 
```

---

previously generated  $\mathbf{M}\mathbf{X}\mathbf{t}\mathbf{X}$ . Algorithm 4 provides the mechanism of generating  $\mathbf{Y}\mathbf{t}\mathbf{Z}$  and  $\mathbf{Z}\mathbf{t}\mathbf{Z}$  (a part of  $\mathbf{M}\mathbf{X}\mathbf{t}\mathbf{X}$ ) in Spark environment. Similar to Algorithm 3, here the map operation is run for each  $i^{th}$  row of  $\mathbf{Y}_n\mathbf{Z}$  and the operation is carried out by each of the worker nodes for their corresponding data segment which has been distributed by Spark. Note that,  $\mathbf{Z}\mathbf{t}\mathbf{Z}$  is generated only for the first iteration while new  $\mathbf{Y}\mathbf{t}\mathbf{Z}$  is generated at every iteration.

## 8.2 Algorithms Compared

The available state-of-the-art methods are already discussed in Chapter 5. Their existing limitations are also discussed briefly. The major issues of the existing methods are that they are not scalable with the increased dimension of the datasets and also none of the existing methods can provide an implementation for geo-distributed datasets. Therefore, to overcome these two concerns, we have proposed TallnWide, which is a highly scalable geo-distributed implementation of zero-noise model PPCA. In this section, we mention the methods that we consider as competitors of TallnWide and therefore, we opt to show the merit evaluation of our proposed method against these state-of-the-art methods. For computing principal components, we compare the following five methods:

- TallnWide: Our implementation of the algorithm TallnWide.
- sPCA: A scalable implementation of PPCA [18].
- Mahout-PCA: Mahout implementation of PCA [16].
- MLlib-PCA: PCA implementation in MLlib [17].
- sSketch-PCA: PCA implementation using a scalable sketching technique [19].



**Algorithm 4** SegmentedYtZnZtZJob

---

**Input** : Iteration number  $k$ ;  
 Data Matrix  $\mathbf{Y}$ ;  
 $\boldsymbol{\mu}$  which denotes mean( $\mathbf{Y}$ );  
 $D_i$  which denotes the No of Rows in  $i^{th}$  segment of  $\mathbf{W}$ ;  
 Target Dimension  $d$ ;  
 Intermediate Matrix  $\mathbf{Z}$ ;  
 $start$  which denoted the starting Index for Block I;  
 $end$  which denotes the ending Index for Block I

**Output:** Intermediate Data  $\mathbf{ZtZ}$ ;  
 Intermediate Data  $\mathbf{YtZ}$

```

1  $\mathbf{Y}_n\mathbf{Z} = \mathbf{Y}.zip(\mathbf{Z})$ 
2 if ( $k == 1$ ) ▷ Iteration for the 1st Segment of  $\mathbf{W}$ 
3 then
4    $\mathbf{YtZSum} = accumulator(newMatrix(D_i, d))$ 
5    $\mathbf{ZtZSum} = accumulator(newMatrix(D_i, d))$ 
6    $\mathbf{Y}_n\mathbf{Z}.map\{(Y_n\mathbf{Z})_i \Rightarrow$  ▷ Runs for each  $i^{th}$  row of  $\mathbf{Y}_n\mathbf{Z}$ 
7      $Z_i = (Y_n\mathbf{Z})_i.arg1().range(start, end)$ 
8      $(YtZ)_i = Y_i^T \times Z_i - \boldsymbol{\mu}^T \times Z_i$ 
9      $(ZtZ)_i = Z_i^T \times Z_i$ 
10     $\mathbf{YtZSum}.add((YtZ)_i)$ 
11     $\mathbf{ZtZSum}.add((ZtZ)_i)$ 
12  }
13   $\mathbf{YtZ} = \mathbf{YtZSum}.value()$ 
14   $\mathbf{ZtZ} = \mathbf{ZtZSum}.value()$ 
15  return  $\mathbf{YtZ}, \mathbf{ZtZ}$ 
16 end
17 else
18    $\mathbf{YtZSum} = accumulator(newMatrix(D_i, d))$ 
19    $\mathbf{Y}_n\mathbf{Z}.map\{(Y_n\mathbf{Z})_i \Rightarrow$  ▷ Runs for each  $i^{th}$  row of  $\mathbf{Y}_n\mathbf{Z}$ 
20      $Y_i = (Y_n\mathbf{Z})_i.arg0().range(start, end)$ 
21      $Z_i = (Y_n\mathbf{Z})_i.arg1()$ 
22      $(YtZ)_i = Y_i^T \times Z_i - \boldsymbol{\mu}^T \times Z_i$ 
23      $\mathbf{YtZSum}.add((YtZ)_i)$ 
24  }
25   $\mathbf{YtZ} = \mathbf{YtZSum}.value()$ 
26  return  $\mathbf{YtZ}$ 
27 end

```

---

The implementations of Mahout-PCA, as well as MLlib-PCA, are highly optimized in their respective platforms. On the other hand, the sPCA implementation on Spark was the first distributed PCA technique of its time. It has added some advanced features and has been quite successful in outperforming most of the close competitors. With all this sPCA was very capable of handling higher dimensional dataset but up to a certain extent after which it incurs memory overflow error. Finally, sSketch-PCA utilizes a scalable implementation of Gaussian sketching method along with various optimization techniques in order to achieve high scalability. sSketch-PCA is highly optimized with various techniques similar to sPCA, such as mean propagation, efficient sparse matrix operations, and effective job consolidation to minimize intermediate data. sSketch-PCA also provides two other scalable methods for deriving singular value and 2-norm of reconstruction error, both of which are used for data analysis purpose. Just similar to sPCA, sSketch-PCA is also implemented on the popular Spark framework for distributed platform. As we already have mentioned that despite of being more scalable than the other competitive methods, sSketch-PCA also faces out-of memory error when the dimension of dataset reaches to  $10M$ . Since we are intended to provide a justification that our proposed TallnWide is capable of handling arbitrarily large dimensional datasets, therefore, we find the algorithms as mentioned above to be the best options for comparing with TallnWide to establish its merit. For uniform comparison, we make all PCA algorithms to compute the top 10 principal components.

On the other hand, there is no existing geo-distributed implementation of PCA that we can consider to compare with TallnWide. Therefore, to evaluate the merit of our accumulation scheme, we discuss some possible alternative accumulation strategies and then finally provide a demonstration that our implemented efficient accumulation scheme along with ideal central datacenter for aggregating partial results yields the best results.

### 8.3 Datasets

As we are aware that the strength of evaluation of every proposed method is largely dependant on the use of suitable datasets. Therefore, to establish the merit of our proposed TallnWide, we use five real datasets. All of these datasets are quite diverse in terms of size, dimensions, sparsity, and data values. Since some of these datasets are so higher dimensional that the comparing methods fails to execute on them, we have to make the datasets suitable to be executed on by all the comparing methods. Therefore, we generate various subsets of the five datasets so that we can assess the scalability and performance of comparing PCA algorithms with increasing data sizes. The following list describes the datasets by showing their source, dimension and . If the number of rows and columns of a dataset are  $N$  and  $D$  respectively, then the sparsity of that dataset is measured as follows:

$$\text{sparsity} = \frac{\# \text{ of zero elements}}{(N * D)}$$

- **PubMed:** PubMed is a sparse, bag-of-word representation of medical documents from the U.S. National Library of Medicine. [4] provides a matrix from this dataset where the rows represent the documents, and the columns represent the words (values are either 0 or 1). Its size is  $8,200,000 \times 141,043$ . The number of non zero elements in this dataset is 483,450,157, and the sparsity is 0.9995. We also consider a small subset of this dataset with a dimension of 2000, referred to as PubMed2K.
- **AmazonRating:** Amazon.com, Inc is an American multinational technology company based in Seattle, Washington, which focuses on e-commerce, cloud computing, digital streaming, and artificial intelligence. It is considered one of the Big Five companies in the U.S. information technology industry, along with Google, Apple, Microsoft, and Facebook. The company has been referred to as “one of the most influential economic and cultural forces in the world”, as well as the world’s most valuable brand. The market place provided by Amazon is the most popular e-commerce website throughout the whole universe. The products available there can be rated by any customer as soon as he buy them. Amazon product data provided by [3] contains product reviews and meta-data from Amazon. It is a sparse matrix of size  $21M \times 9.8M$ , and the values are between 0 and 5. The value at the cell  $[i][j]$  represents the rating given by the user  $i$  to the product  $j$ . The number of non zero elements in this dataset is 82,676,840, and the sparsity is 0.9999. We also consider two subsets: AmazonRating2K (size  $6.6M \times 2K$ ) and AmazonRating50K (size  $6.6M \times 50K$ ).
- **SiftFeature:** The scale-invariant feature transform (SIFT) [80] is a feature detection algorithm in computer vision to detect and describe local features in images. SIFT keypoints of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. From the full set of matches, subsets of keypoints that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches. The determination of consistent clusters is performed rapidly by using an efficient hash table implementation of the generalised Hough transform. Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed model verification and subsequently outliers are discarded. Finally the probability that a particular set of features indicates the presence of an object is computed, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

In order to generate the SiftFeature Dataset, We download all the images from ImageNet [81] and from each image, we extract SIFT features and store them in a matrix

Table 8.1: Summary table of the used Datasets

Dataset	No of Rows	No of Columns	Cell $[i][j]$
PubMed	8.2M	141,043	= 1 if word $j$ is present in document $i$
AmazonRating	21M	9.8M	= the rating given by the user $i$ to the product $j$
Twitter	61M	61M	= 1 if user $i$ is followed by user $j$
SiftFeature	4.5M	128	= a real value representing SIFT feature
Fashion-MNIST	70K	784	= a real value Each sample is a $28 \times 28$ grayscale image

form. It is a dense matrix of size  $4455091 \times 128$  (with sparsity value of 0.2272), and each element is a real value.

- **Twitter:** Twitter is an American microblogging and social networking service on which users post and interact with messages known as “tweets”. Registered users can post, like and retweet tweets, but unregistered users can only read them. Users access Twitter through its website interface, through Short Message Service (SMS) or its mobile-device application software (“app”). Tweets were originally restricted to 140 characters, but was doubled to 280 for non-CJK languages in November 2017. Audio and video tweets remain limited to 140 seconds for most accounts. We take a  $50M \times 50M$  dataset from [2] which is a matrix of Twitter users. If the user  $i$  is followed by the user  $j$ , then the cell  $[i][j]$  holds a 1. It can clearly be observed that this is a very sparse matrix and sparsity value is 0.9999. From this dataset, we primarily consider two subsets: Twitter1K (size  $59670 \times 1K$ ) and Twitter10M (size  $10M \times 10M$ ). We also make several other low dimensional subsets as needed (not mentioned as separate names).
- **Fashio-MNIST:** Fashion-MNIST is a dataset of [82] images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a  $28 \times 28$  grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

The Table 8.1 summarizes the datasets that we consider to evaluate the merit of our proposed TallnWide in light of the comparing method.

## 8.4 Cluster Configuration

As we have already motioned in Chapter 7 that Algorithm 1 TallnWide is a generalized algorithm which designed for a geographically distributed datasets. However, we can execute the algorithm in a Standalone Mode by zeroing the number of DCs i.e.,  $S = 0$ . Therefore, in order to evaluate the merit of TallnWide, we need to run our algorithm in two different environments:

- firstly, in a single cluster (*Standalone Mode*) to assess the capability of handling arbitrarily large dimensional datasets. In this setup data is residing in a single DC with instance of commodity hardware, and the dimension of the dataset is arbitrarily large. We need to verify the running time i.e. speed, scalability and space efficiency of our algorithm in such kind of environmental setup. ii) secondly, in geo-distributed environment, where multiple clusters are located at different geographic locations and they have their own version of dataset. Even if the dataset is not higher dimensional, we still are not capable of manage such setup when it comes to perform PCA since no PCA algorithm is geo-distributed. In this kind of environment, partial results generated by each cluster, they are aggregated by a central datacenter and then redistributed. Therefore, we can see that there is a heavy involvement of network I/O involvement which we need to execute in an efficient manner.

The first experiment mainly focuses on computational capability, which can provide efficient memory utilization. On the other hand, the later one needs efficient inter datacenter communication scheme. In this section, we describe the cluster configuration to create the above mentioned two environments.

**Single DC Configuration for Standalone Mode:** We run the experiments for a single DC (*standalone mode*) by creating a cluster on the Amazon EMR. Our cluster consists of:

- 8 Amazon EC2 m3.xlarge instances, each of which contains:
  - eight vCPU,
  - 15 GB of memory, and
  - 80 GB SSD storage
- Each node has the following software installed:
  - emr-5.7.0,
  - Hadoop 2.7.3,
  - Spark 2.1.1,
  - Ganglia 3.7.2, and
  - Mahout 0.13.0.
  - MLlib library is included in Spark.

Point to be noted that in case of Standalone Mode, the main focus is to assess the scalability of TallnWide. Therefore, we execute the algorithm on very higher dimensional dataset which requires the algorithm to execute for significantly large amount of time. FOr this reason, to cut off the monetary cost, cheaper Amazon AWS instances are considered during cluster setup.

Table 8.2: Bandwidth (in MB/s) among geo-distributed DCs.

DC/DC	Ireland	N. Virginia 1	N. Virginia 2	Oregon 1	Oregon 2
Ireland	-	11.35MB/s	10.95MB/s	9.25MB/s	11.60MB/s
N. Virginia 1	11.35MB/s	-	81.30MB/s	18.05MB/s	18.20MB/s
N. Virginia 2	10.95MB/s	81.30MB/s	-	17.50MB/s	17.70MB/s
Oregon 1	9.25MB/s	18.05MB/s	17.5MB/s	-	126.95MB/s
Oregon 2	11.60MB/s	18.20MB/s	17.70MB/s	126.95MB/s	-

**Geo-distributed DCs Configuration:** For simulating geo-distributed environment, we consider three geographical regions, namely North Virginia, Oregon, and Ireland. In North Virginia and Oregon, we consider two separate zones, which are referred to as North Virginia 1 & 2, and Oregon 1 & 2. In Ireland, we only consider one zone referred to as Ireland. That means we consider five different geographic locations. Table 8.2 shows the inter-DC B/Ws. In each zone, to create a cluster consisting

- 3 Amazon EC2 m3.2xlarge instances each of them having
  - 16 vCPU,
  - 30 GB of memory, and
  - 160GB SSD
- Each node has the following software installed:
  - emr-5.7.0,
  - Hadoop 2.6,
  - Spark 2.0.0,
  - Ganglia 3.7.2,

Note that we do not install Mahout and MLlib in geo-distributed environment since we do not use them for geo-distributed manner because of their lacking to offer one. Depending on the implementation, we deploy different kinds of geo-distributed clusters on these 15 instances (described later).

## 8.5 Performance Metrics

As we have already described that we consider two different environment to evaluate TallnWide. To evaluate the merit of our algorithm, in case of different environments, we consider different set of performance metrics.

- In case of *standalone mode* evaluation, we consider three performance metrics:
  - running time to achieve the desired accuracy,
  - scalability, and
  - intermediate data size.

In addition, we also show the effect of the number of blocks on the running time and scalability. For checking convergence, we use the following criteria:

$$\Delta \mathbf{W} = \max\left(|\mathbf{W}^k - \mathbf{W}^{k+1}| / (\epsilon + \max(|\mathbf{W}^{k+1}|))\right)$$

where  $\mathbf{W}$  is our principal component. Here  $|\cdot|$  denotes the absolute value of the term, and  $\max$  returns the maximum element of a matrix. When we say 5% tolerance for convergence, we mean changes in elements of  $\mathbf{W}$  or  $\Delta \mathbf{W}$  is less than or equal to 5%. Here  $\epsilon$  is a small number to avoid any division-by-zero error.

- On the other hand, in case of geo-distributed evaluation, we mainly focus on communication efficiency. We show the changes in running time in our various accumulation strategies. Then we show the communication efficiency among different approaches mentioned in section 6.2.

# Chapter 9

## Experimental Evaluation

In this chapter, we present an in-depth comparison of TallnWide with other algorithms based on the performance metrics mentioned in the preceding section. At the beginning of this chapter, we show the comparison among the principal components identified by the comparing methods. After that, we compare the considering methods from other aspects.

### 9.1 Comparison among Identified Principal Components

In this section, we illustrate the similarity of the principal components identified by the comparing methods. By showing proper data visualization, the reconstruction error, and the proportion of explained variance, we justify that TallnWide, despite of omitting the noise, is capable of providing the principal components without any significant compromise in correctness.

#### 9.1.1 Reconstruction Error

We first run the comparing methods on some selected smaller dimensional datasets and identified the top 10 principal components. After that, we project the original data on the subspace with maximum variance, and compute the reconstruction error. For each of our used datasets, Table 9.1 is showing the reconstruction errors for the principal subspace identified by each of the comparing methods. Even though TallnWide incurs slightly larger reconstruction error, the rate is still within an acceptable tolerance (the maximum reconstruction error of TallnWide is incurred for AmazonRating dataset with  $1M$  of rows and  $2K$  of columns, which is just larger than 1%). Since TallnWide with zero noise model PPCA is capable of handling arbitrarily large dimensional datasets, such marginal loss in projected data is acceptable.

We demonstrate the figures of two different fashion elements from Fashion-MNIST dataset in Figure 9.1. On both figures, (a) illustrates the real  $28 \times 28$  pixel image from the dataset. On the other hand, (b)-(e) are drawn using the projected data on the principal subspace identified by the



Table 9.1: Comparison of reconstruction error in percentage

Datasets	Size	MLlib-PCA	Mahout-PCA	sPCA	TallnWide
Fashion-MNIST	$60K \times 784$	$8.98 \times 10^{-8}$	$2.91 \times 10^{-6}$	$5.5 \times 10^{-1}$	$8.9 \times 10^{-1}$
PubMed1M2K	$1M \times 2K$	$3.65 \times 10^{-4}$	$1.25 \times 10^{-2}$	$1.11 \times 10^{-1}$	$2.13 \times 10^{-1}$
SiftFeature	$4.5M \times 128$	$3.22 \times 10^{-8}$	$2.08 \times 10^{-6}$	$4.51 \times 10^{-3}$	$1.03 \times 10^{-2}$
AmazonRating1M2K	$1M \times 2K$	$6.35 \times 10^{-4}$	$4.89 \times 10^{-2}$	$3.64 \times 10^{-1}$	$1.21 \times 10^0$

Table 9.2: Total Variance explained (in percentage) by top 10 Principal Components identified by the comparing methods.

Datasets	Size	MLlib-PCA	Mahout-PCA	sPCA	TallnWide
Fashion-MNIST	$60K \times 784$	71.97	71.97	71.70	71.04
PubMed1M2K	$1M \times 2K$	49.09	49.09	48.94	48.70
SiftFeature	$4.5M \times 128$	32.69	32.69	32.41	32.23
AmazonRating1M2K	$1M \times 2K$	77.43	77.43	77.33	77.11

comparing methods. We can clearly observe that TallnWide, despite of very marginally higher reconstruction error, is capable of providing a similar projected data, which is suitable to apply on any kind of data analytics.

### 9.1.2 Variance Explained by Principal Components

Now we show the comparison among the percentage of variance of the considering datasets captured by the top 10 principal components identified by our comparing methods. Table 9.2 shows the experimental results. A better understandable graph plot is shown on Figure 9.2. It is evident that TallnWide is capable of capturing similar amount of variance in contrast to the other state-of-the-art methods, and to be specific, it is more than 99% similar.

## 9.2 Evaluation for a Single DC

Now we come to the performance evaluation of TallnWide from other aspects. In this section, we first show the efficiency of our block-division method in TallnWide algorithm for a single DC. We show that TallnWide is both fast and scalable compared to the other methods.

**Effect of Number of Blocks:** In case of executing TallnWide in a single cluster Standalone mode, our main intention is to justify the claim that TallnWide can handle arbitrarily large dimensional datasets. As we have already mentioned in Section 6 that we divide the PCA

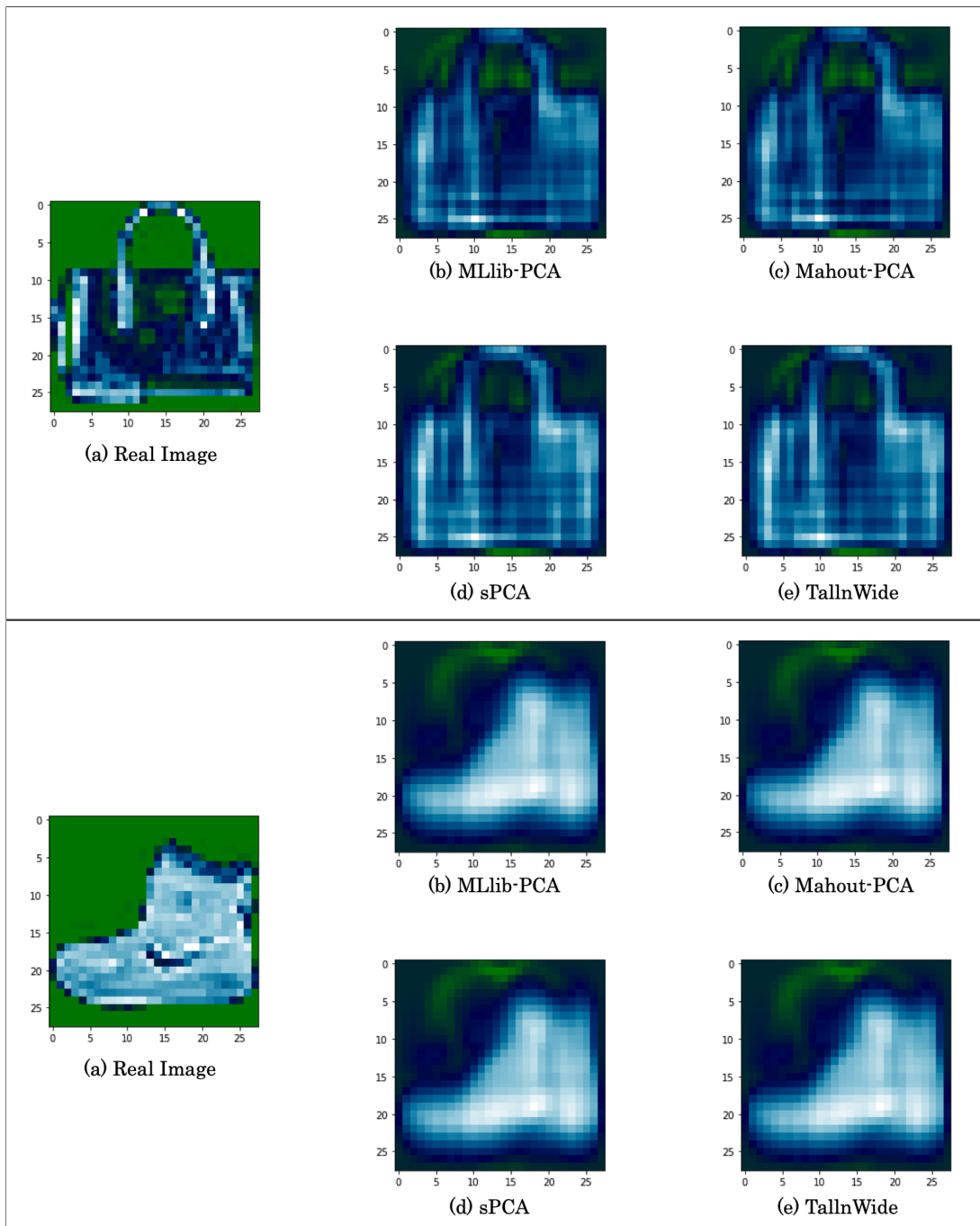


Figure 9.1: Visualization of projected data on principal subspace identified by the comparing methods.

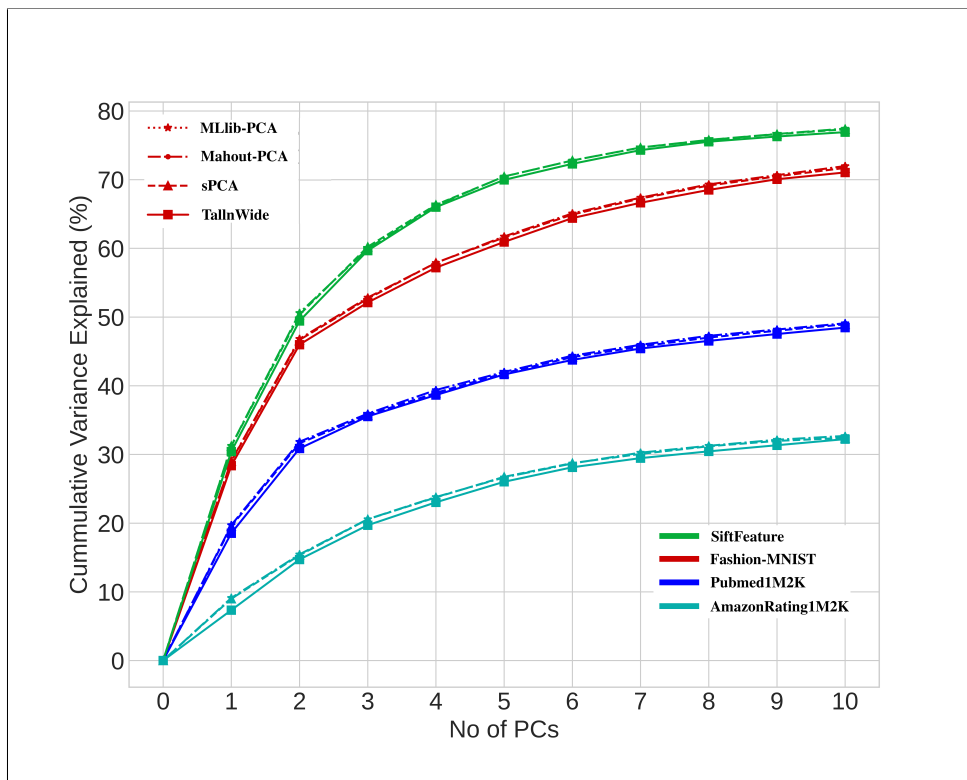


Figure 9.2: Visualization of total captured variance by the top 10 principal components identified by the comparing methods.

parameter  $\mathbf{W}$  into  $\beta$  blocks so that at each time, TallnWide has to handle a portion of data which is manageable in size. It will eventually reduce the amount of intermediate data size and overcome the memory overflow error. The choice of number of blocks plays the major role in the scalability concern. A wrong choice of the number of blocks can either lead to the occurrence of usual memory overflow error or a significant amount of unnecessary disk input-output overhead. Therefore, it is required to choose this number wisely so that it is the smallest number which is sufficient to minimize the out-of-memory error. However, we omit the derivation of the optimum block number in this thesis, and identify the optimum number in an empirical way. In our implementation, we do not fix the number of blocks for the division. Instead, we dynamically choose the number of blocks by this formula:

$$\beta = \lceil (\rho * D * d * 8) / (\min Mem * 1024^2) \rceil$$

where  $\beta$  is the number of blocks,  $\rho$  is the tuning parameter,  $\min Mem$  is the least free memory (in MB) among working nodes. Table 9.3 shows the effect of  $\rho$  on the number of blocks for AmazonRating and Twitter10M datasets. Depending on  $\rho$ , the number of blocks varies. In this table, the sizes of the blocks are shown in approximate values. These sizes are measured during the runtime of the spark code by comparing the approximate free and utilized memories observed before and after the generation of a single block of the PCA parameter,  $\mathbf{W}$ . One point to be noted that these block sizes do not provide any direct indication of the failure or success in

Table 9.3: Effect of  $\rho$  on the choice of different number of blocks.

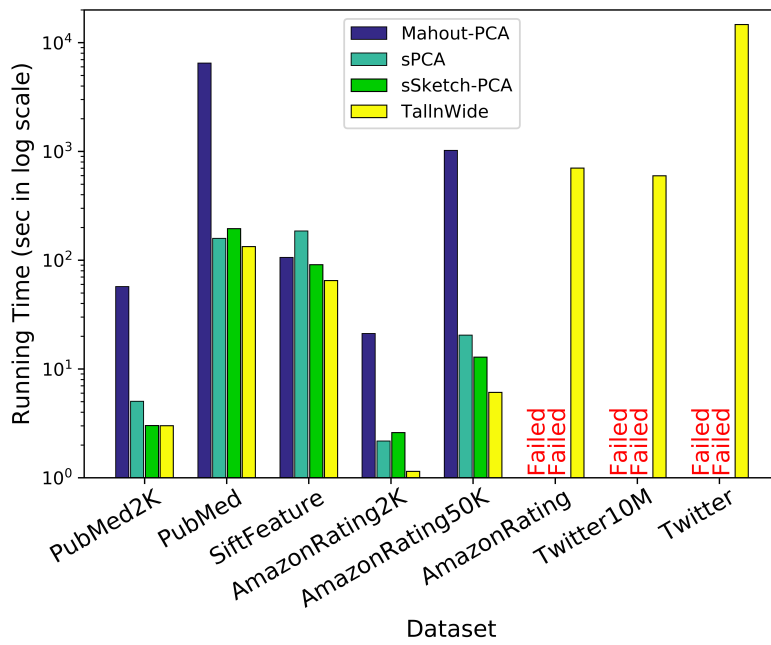
$\rho$	AmazonRating			Twitter10M		
	Number of blocks ( $\beta$ )	Block Size (MB)	Time per iteration (sec)	Number of blocks ( $\beta$ )	Block Size (MB)	Time per iteration (sec)
10	1	2918	<b>Failed</b>	2	2213	<b>Failed</b>
30	2	1628	591.70	3	1811	597.54
40	3	1105	742.47	4	1425	877.31
60	4	917	1444.22	5	1007	1635.71

execution rather they directly impact the size of intermediate data generation. In order to ensure that memory overflow error does not occur, we keep the block size manageable. To do so, we keep the number of blocks to be larger than a certain threshold value. However, if we work with more blocks, we have to encounter additional overheads for Disk I/O. Therefore, our goal is to minimize the number of blocks as much as possible. In our observation,  $\rho = 30$  yields the best result, so we have used  $\rho = 30$  for TallnWide. However, determining the efficient value of  $\rho$  to keep the count of blocks minimum while ensuring the mitigation of overflow error is out of the scope of this thesis and is kept as future work.

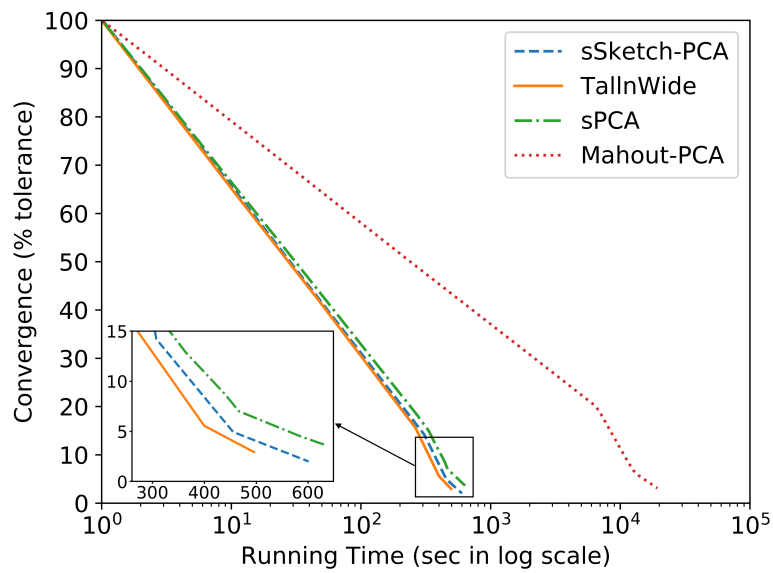
### 9.2.1 Running Times in a Single DC

To compare running times of our method against all other techniques, we first show that as an iterative method TallnWide takes less time to finish one iteration, compared to other iterative methods. Later, we show that when all the iterative techniques converge, TallnWide takes the least amount of time as well. For each method, to get results faster and reduce monetary cost, we consider deriving top 10 principal components. Point to be noted that all the methods we consider for comparison involve randomization in their approaches. This may result in either a decent starting point, which tends to converge earlier or any average starting point, which needs more iterations to converge. For this reason, in order to maintain fairness among the methods, we use per iteration time comparison.

Figure 9.3(a) shows per iteration running times for all the iterative methods for all datasets (except Twitter1K for which logarithmic running time is negative). In all cases, TallnWide takes the least amount of time. For example, for AmazonRating50K dataset, TallnWide takes only 6.10 seconds while Mahout-PCA, sPCA, and sSketch-PCA take 1021.25, 20.47, and 12.86 seconds respectively. For other datasets, the results are similar. Furthermore, when the number of dimensions is too high, Mahout-PCA, sPCA, and sSketch-PCA fail to run. For example, for AmazonRating ( $D = 9.8M$ ), Twitter10M ( $D = 10M$ ), and Twitter datasets ( $D = 50M$ ), they fail, whereas TallnWide performs smoothly, and takes 742.47, 597.54, and 14687.62 seconds per iteration, respectively.



(a)



(b)

Figure 9.3: (a) Per iteration running time for all iterative methods. (b) Comparison of running time of iterative methods to converge (5% tolerance) for PubMed dataset.

Table 9.4: Comparison of running time (in sec) for TallnWide on different datasets against state-of-the-art library functions: MLib-PCA, Mahout-PCA, and sPCA. For iterative methods, we consider running time to reach convergence (5% tolerance). For full Twitter dataset, we consider 1 iteration.

Datasets	Size	MLlib-PCA	Mahout-PCA	sPCA	sSketch-PCA	TallnWide
PubMed2K	$8.2M \times 2K$	94.54	57.27	35.32	24.21	21.11
PubMed	$8.2M \times 141K$	<b>Fail</b>	19491.51	634.90	585.37	533.91
SiftFeature	$4.5M \times 128$	107.09	1801.52	556.10	182.10	194.93
AmazonRating2K	$6.6M \times 2K$	58.58	63.54	10.88	12.99	5.731
AmazonRating50K	$6.6M \times 50K$	<b>Fail</b>	2042.49	102.343	64.33	48.76
AmazonRating	$21M \times 9.8M$	<b>Fail</b>	<b>Fail</b>	<b>Fail</b>	<b>Fail</b>	6339.77
Twitter1K	$50K \times 1K$	26.63	22.86	1.18	1.12	1.10
Twitter10M	$10M \times 10M$	<b>Fail</b>	<b>Fail</b>	<b>Fail</b>	<b>Fail</b>	2987.69
Twitter (1 iteration only)	$50M \times 50M$	<b>Fail</b>	<b>Fail</b>	<b>Fail</b>	<b>Fail</b>	14687.62

We now show that TallnWide takes less time for convergence. In Figure 9.3(b), running times of iterative methods to converge (5% tolerance) for PubMed dataset is shown. We can observe that TallnWide takes the least amount of time (evident from the zoomed segment) to converge compared to others (533.91 seconds vs 585.37, 634.90, and 19491.51 seconds) and offers 1.1 – 37× better performance. Table 9.4 shows full running times of all the methods for all datasets. For Twitter, we show the results of the first iteration only (to reduce the monetary cost). Observe that, due to the high dimension, MLib-PCA fails to run on full PubMed dataset. If we take a smaller dataset, TallnWide still takes less time. For example, for PubMed2K MLib-PCA, Mahout-PCA, sPCA, and sSketch-PCA take 94.54, 57.27, 35.32, and 24.21 seconds to finish while TallnWide takes only 21.11 seconds (a factor of 1.2 – 4.5× improvement). Results for other datasets are similar, and TallnWide offers better performance (1.1 – 42×) in running time in most of the cases. However, only for the dense matrix SiftFeature, MLib-PCA takes the least time. It is because MLib-PCA is a deterministic algorithm, and it does not have any overhead for sparsity preserving calculation. However, here the dimension is relatively small, which is not a case with big data, for which MLib-PCA fails to perform.

### 9.2.2 Scalability

TallnWide does not face out-of-memory error as dimensions of data increase arbitrarily. To show this, we work with different dimensions of Twitter datasets such as 1K, 2K, 4K, 6K, 8K, 10K, 100K, 1M, 5M, 10M, 20M and 50M, and record the memory consumption using Ganglia [83] for each method. Figure 9.4 shows the result. As expected MLib-PCA faces out-of-memory error after 6K dimensions. Mahout-PCA and sPCA too face such error after handling up to 5M dimensions while sSketch-PCA can prolong its scalability further but eventually fails when the dimension reaches close to 10M (from Table 9.4, we can see that sSketch-PCA fails for datasets

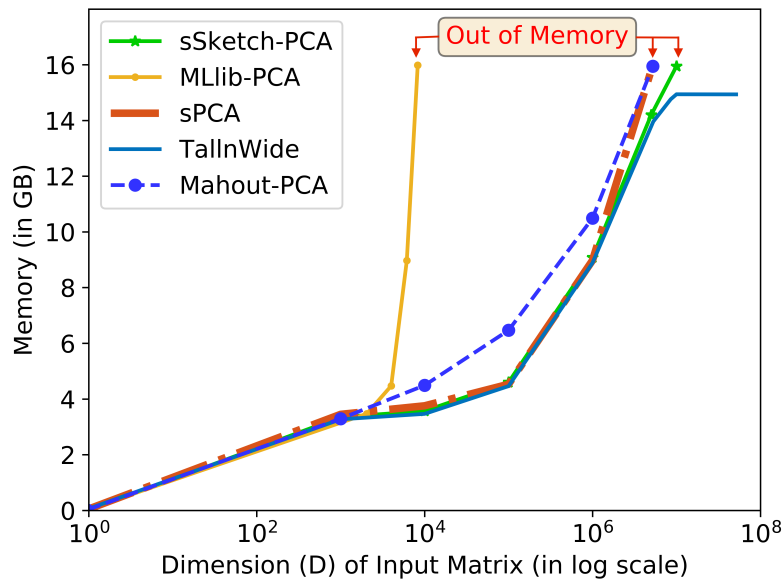


Figure 9.4: Memory consumption in one working node for all methods for Twitter dataset.

with dimension  $9.8M$ ,  $10M$ , and  $50M$ ). However, due to the merit of block-division, TallnWide does not face such a problem. Thus, from this figure and Table 9.4, we can say that TallnWide successfully handles up to  $10\times$  dimensions which bears the testimony of our claim that it can handle an arbitrarily large number of dimensions.

### 9.2.3 Intermediate Data Size

We have already discussed that one of the main motivations of proposing TallnWide is to overcome the memory overflow error that is incurred by the existing methods when they are executed on datasets with arbitrarily large dimensions. As the dimensions of the PCA parameter continues to increase, the size of the intermediate data eventually explodes and the out-of-memory error happens. In order to address this serious challenge, we introduce a novel block division method in TallnWide, which divides the PCA parameter into manageable size so that the intermediate data size can be kept in control. In this final part of the evaluation in a single DC, we present an experimental justification that TallnWide is memory efficient, and due to the controlled size of intermediate data, it can successfully mitigate the memory overflow error.

We measure intermediate data size generated by all the comparing algorithms for PubMed, AmazonRating50K, and SiftFeature datasets. We choose these datasets because they allow other methods to run without facing out-of-memory errors. Table 9.5 shows the experimental results. Along with the zero-noise model, which reduces the generation of intermediate data, our TallnWide offers a significantly lower memory utilization by providing a block division of the main parameter (principal subspace  $W$ ). Although we may incur some computational overhead due to partitioning, the total running time is not impacted by that much as we showed earlier. From the table, we can see, in case of MLib-PCA, the intermediate data size grows very

Table 9.5: TallnWide is space-efficient. It produces less intermediate data than state-of-the-art methods. For a fair comparison, we compare results with only those data where other methods do not face memory issues.

Technique	PubMed	AmazonRating50K	SiftFeature
Mlib-PCA	$\geq 150\text{GB}$	$\geq 150\text{GB}$	896KB
Mahout-PCA	3.65GB	1.05GB	0.67GB
sPCA	198MB	66MB	143KB
sSketch-PCA	151MB	53MB	140KB
TallnWide	131MB	45MB	128KB

quickly and eventually goes beyond 150 GB for our all higher-dimensional datasets. It exceeds the total aggregated memory of the cluster. Since the intermediate data size for Mahout-PCA depends on the input row size  $N$ , which is typically very big in our case, it also generates a lot of intermediate data. For the PubMed dataset, Mahout-PCA, sPCA and sSketch-PCA generate 3.65GB, 198MB and 151MB of intermediate data respectively, whereas TallnWide generates only 131 MB, a factor of  $29\times$ ,  $1.5\times$ , and  $1.2\times$  **reductions** respectively. In case of SiftFeature which has a row size of about 4 million but only has 128 dimensions, we see our method offers a huge reduction of memory compared to Mahout-PCA (0.67 GB vs 128 KB) and also generates a slightly lower amount of intermediate data than the closest competitors sPCA (143KB vs 128KB) and sSketch-PCA (140KB vs 128KB).

### 9.3 Evaluation for Geo-Distributed DCS

In the preceding sections, we have justified the merit of TallnWide in a single DC Standalone mode. We have established the claim that TallnWide can handle datasets with a arbitrary number of dimensions without incurring any memory overflow error. Our novel block division method can partition the PCA parameter into blocks of manageable blocks so that at every time the generation of intermediate data can be kept in control. Now in this section, we move to the geo-distributed environment. Here we have multiple datacenters located at different geographic locations. All the datacenters have their own version of data, and passing raw data across national border is costly and sometimes even prohibited. Therefore, no centralized method of computing PCA can be applicable in such scenario, and we do not have any geo-distributed solution for this kind of scenario. Out of this requirement, we design our algorithm TallnWide in such a way that it can handle geo-distributed datasets in a communication-efficient manner. Since there is no current state-of-the-art which can be used as a baseline to compare our geo-distributed accumulation scheme, we ourselves provide three different possible alternative accumulation scheme which can be easily implemented using the existing tools and frameworks. After some comparative discussion and experimental evaluations, we conclude that our custom accumulation



scheme is capable of providing the most efficient communication. We first describe three different accumulation strategies for the implementation of Geo-Accumulation method in Algorithm 1.

- *gSpark-Accumulation*: This is a naïve strategy where we implement the accumulation of the partial parameters in a centralized approach using the out-of-the-box solution provided by Spark. For this, we deploy Spark in the geo-distributed environment, which we call gSpark. As we have mentioned in the cluster configuration section that we choose five different geographic regions and at each region we launch three Amazon AWS instances. Therefore, in total we have 15 instances in our hand to utilize. To deploy the centralized geo-distributed cluster we use all the 15 instances as a single DC. Our single cluster for this accumulation scheme consists of 1 master along with 14 slaves. The slaves are located at different (distant) geo-distributed locations. This accumulation method demands the passing of raw data among the slaves in order to achieve parallel computational capability. As we know that inter regional B/W is typically not that higher as the intra regional one. Moreover, this kind of B/W is very costly. Undoubtedly, this centralized gSpark violates the basic concern of not passing the raw data across the regional border in order to minimize communication cost and data privacy. Despite of this, we keep this setup in consideration in order to evaluate the outcome and nullify the possibility of using such a setup experimentally.
- *gHDFS-Accumulation*: As passing raw data across regional border poses much concern, to avoid passing raw data, we discard the centralized approach, and consider 3 instances from each zone as a single DC. In each DC, we set up a Spark cluster (1 master and 2 slaves). In addition, we set up another Spark cluster taking all the master instances from each DC, i.e., a *cluster of masters*, in order to accumulate the parameter only. After that, we deploy HDFS in a geo-distributed fashion, which we call gHDFS. In this setting, each DC stores partial results in gHDFS and notify the master of the cluster of masters, or *master of masters*, to start accumulation. Master of masters reads partial results from and stores the final result in the same gHDFS. Since gHDFS is accessible from all the DCs, each DC can read the accumulated parameter at the beginning of the next iteration.
- *TallnWide-Accumulation*: In gHDFS, all DCs have one shared file system which does not depict the real scenario of a geo-distributed environment. So, now in our approach, we discard gHDFS and also the cluster of masters. To do so, we make the assumption that the masters from each DC are able to communicate among themselves in a real application via Secure Shell Script (*SSH*) for passing the partial parameters only. We provide our custom accumulation using the native file system, and *Linux* shell script. In this strategy, master from one specific DC is elected in to be the central DC which handles the responsibility of the accumulation task. The TallnWide algorithm is executed on the dataset of each of the datacenters. As soon as any DC (which is not the central DC) is finished with

Table 9.6: Comparison of running times (in sec) among different strategies (for **single iteration**): gSpark-Accumulation, gHDFS-Accumulation, and TallnWide-Accumulation.

Dataset	gSpark-Accumulation	gHDFS-Accumulation	TallnWide-Accumulation
AmazonRating	1547.14	1195.44	525.56
PubMed	212.45	180.13	137.34

the generation of the partial PCA parameter  $W$ , it sends it to the master of the ideal DC using *SSH*. Whenever, the central DC is also finished the generation of its own version of partial  $W$  and also have received all the partial  $W$ s from other DCs, it accumulates it and redistributes the accumulated version to each of the DCs using *SSH* again.

All of the instructions for gSpark and gHDFS, and code for all strategies are publicly available in the provided *GitHub* link.

### 9.3.1 Running Times Among Accumulation Strategies

Now we show that among the three accumulation strategies introduced in the preceding section the TallnWide-Accumulation provides the best performance when it comes to the speed of accumulation process. To keep costs low, we use two datasets, namely AmazonRating and PubMed, to validate the merit of TallnWide-Accumulation scheme. We execute our TallnWide algorithm by setting the number of geo-distributed clusters on these two datasets. Each of our five different geo-distributed clusters generates a partial result of the principal subspace  $W$  based on their own datasets. After that these partial results are aggregated and redistributed by using the *gHDFS-Accumulation* and *TallnWide-Accumulation*. Note that the other accumulation strategy i.e. *gSpark-Accumulation* is mainly a out-of-the-box centralized approach and in this case, TallnWide is executed by setting the number of DCs to one ( $S = 1$  in Algorithm 1) and considering a single datacenter using all the 15 instances. Table 9.6 shows the results of the strategies for the datasets. The results are shown for running one iteration only in order to cut off the monetary cost. Notice that since gSpark has to transmit and shuffle raw data across geo-distributed instances, gSpark-Accumulation is slow. gHDFS has additional overhead for initialization and replication, and therefore gHDFS-Accumulation takes longer time too. However, our final strategy, TallnWide-Accumulation, operates faster because it does not need to pass raw data and does not have any additional overhead and offers  $1.3 - 2.9\times$  better performance than the other mentioned strategies.

Table 9.7: Average running time (in sec) needed for taking different DC as center using TallnWide-Accumulation strategy. Statistics is shown for AmazonRating and PubMed datasets.

Dataset	Approach 2			Approach 3	
	Ireland	North Virginia 1	North Virginia 2	Oregon 1	Oregon 2 (Ideal)
AmazonRating	594.34	537.57	542.56	546.45	<b>525.56</b>
PubMed	158.31	138.55	141.53	149.44	<b>137.34</b>

### 9.3.2 Communication Efficiency Across Different Approaches

As for now, we have shown that our designed *TallnWide-Accumulation* strategy can easily outperform all other possible alternatives and can achieve the fastest accumulation across geo-distributed DCs. Using TallnWide-Accumulation strategy, we now establish the merit of using Equation (6.3) for Approach 3 (Efficient Order w/ Ideal Central DC) over Approach 2 (Efficient Order) in Figure 6.2 mentioned in Section 6.2. To juggle our memory a bit, it is worth re-mentioning that Approach 2 is designed in a way that a central DC is randomly selected to handle the task of aggregating the partial results and redistributing the final result. The task is carried out in an efficient order so that a maximum parallelism can be achieved. Whenever any block of  $W$  is generated, it is sent for geo-accumulation and that particular DC continues with the generation of the next block. In this way, the idle times in CPU, Network, and Disk I/O can be minimized. On the other hand, Approach 3 is more aggressive to achieve a faster accumulation. It determines an ideal DC according to Equation 6.3 mentioned in Section 6.2. An ideal master is theoretically expected to be faster in communicating with the masters of other DCs. In this section, we experimentally justify the validity of the concept of choosing an ideal central DC for accumulation. We do not experimentally validate Approach 1 (Trivial Order) because of straightforward observation and cutting down the monetary cost of running EC2 instances. Table 8.2 shows the B/W (a symmetric matrix) between every pair of DCs. From this table and Equation (6.3), we derive the slowest B/Ws for all masters from each DC:

$$B/W_{min} = \{9.25, 11.35, 10.95, 9.25, 11.60\}$$

For Oregon 2, we get the maximum from this set which is 11.60MB/s. So, using Oregon 2 as the central DC for accumulation should yield a faster result. To evaluate this claim, we run the *TallnWide-Accumulation* scheme by selecting each of the DCs as our central DC for accumulation, and observe the time requirement to perform one iteration of Algorithm 1 on datasets *AmazonRating* and *Pubmed*. The results are shown in Table 9.7. We can clearly observe that for both the datasets, Oregon 2 takes the least amount of time which perfectly aligns with our theoretical observation according to Equation 6.3.

We have seen that in single DC configuration, our results show that TallnWide offers high

---

scalability, and also capable of generate faster results. With our introduced novel block division method, we can minimize the generation of intermediate data and thus successfully minimize the memory overflow error. In geo-distributed setup, we have shown that our accumulation strategy along with the determination of ideal central DC for accumulation yield faster accumulation than the other alternatives. In summary, we have justified all our claims of minimizing all the challenges in executing PCA algorithm on geo-distributed tall and wide datasets have been justified with proper and convincing experimental results.

# Chapter 10

## Conclusion

In this thesis, we have proposed a new algorithm, referred to as TallnWide, to meet the challenges of geo-distributed tall and wide big data. We have addressed two basic challenges associated with the existing methods of PCA computation. The state-of-the-art fail to meet the demand of tall and wide big data since they are not scalable to handle arbitrarily large dimensional datasets. Moreover, in the recent world, where data are by born geographically distributed, they fail to offer any federated learning for PCA techniques.

In order to solve the scalability issue, we have devised a novel block-division EM algorithm for PCA, and we have demonstrated both theoretically and experimentally that compared to the state-of-the-art techniques, our method is highly scalable (handles  $10\times$  higher dimension) and offers better performance ( $1.1 - 42\times$  faster).

We have also given a better solution in our algorithm for the geo-distributed environment, which does not require passing raw data and provides faster result by selecting the datacenter with optimum communication capability to handle to responsibility of the accumulation of partial parameters. We show that our accumulation strategy, coupled with our communication efficient calculation yields up to  $2.9\times$  faster result than other alternatives.

In our work, we have dynamically derived the number of blocks to be required to minimize the memory overflow error while keeping other overheads to minimum. This derivation involves the monitoring of memory usage during runtime and a tunable parameter. However, we have kept the derivation of the optimum value of the tunable parameter as future work. In future, we intend to give an optimal block-partition scheme and provide a technique for perturbation of values of the parameter for privacy preservation.

For reproducibility and extensibility of our work, we make the source code of TallnWide available at <https://github.com/tmadnan10/TallnWide>.

# References

- [1] J. Shlens, “A tutorial on principal component analysis,” *arXiv preprint arXiv:1404.1100*, 2014.
- [2] H. Kwak, C. Lee, H. Park, and S. Moon, “What is twitter, a social network or a news media?,” in *Proceedings of the 19th international conference on World wide web*, pp. 591–600, 2010.
- [3] J. McAuley, “Amazon product data.(2014),” 2014.
- [4] M. Lichman *et al.*, “UCI machine learning repository,” 2013.
- [5] J. Fan, F. Han, and H. Liu, “Challenges of big data analysis,” *National science review*, vol. 1, no. 2, pp. 293–314, 2014.
- [6] X. Fu, K. Huang, E. E. Papalexakis, H.-A. Song, P. P. Talukdar, N. D. Sidiropoulos, C. Faloutsos, and T. Mitchell, “Efficient and distributed algorithms for large-scale generalized canonical correlations analysis,” in *2016 IEEE 16th international conference on data mining (ICDM)*, pp. 871–876, IEEE, 2016.
- [7] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, “The rise of “big data” on cloud computing: Review and open research issues,” *Information systems*, vol. 47, pp. 98–115, 2015.
- [8] C.-J. Hsieh, S. Si, and I. S. Dhillon, “Communication-efficient distributed block minimization for nonlinear kernel machines,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 245–254, 2017.
- [9] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.
- [10] C. Ding and X. He, “K-means clustering via principal component analysis,” in *Proceedings of the twenty-first international conference on Machine learning*, p. 29, 2004.
- [11] B. Guo, C. Hou, F. Nie, and D. Yi, “Semi-supervised multi-label dimensionality reduction,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 919–924, IEEE, 2016.

- [12] Q. Wang and V. Megalooikonomou, “A dimensionality reduction technique for efficient time series similarity analysis,” *Information systems*, vol. 33, no. 1, pp. 115–132, 2008.
- [13] Q. Du and J. E. Fowler, “Hyperspectral image compression using jpeg2000 and principal component analysis,” *IEEE Geoscience and Remote sensing letters*, vol. 4, no. 2, pp. 201–205, 2007.
- [14] J. M. Porta, J. J. Verbeek, and B. J. Kröse, “Active appearance-based robot localization using stereo vision,” *Autonomous Robots*, vol. 18, no. 1, pp. 59–80, 2005.
- [15] A. N. Gorban, B. Kégl, D. C. Wunsch, A. Y. Zinovyev, *et al.*, *Principal manifolds for data visualization and dimension reduction*, vol. 58. Springer, 2008.
- [16] A. Mahout, “What is apache mahout?, copyright© 2014 the apache software foundation, licensed under the apache license, version 2.0.”
- [17] X. Meng, “Mllib: Scalable machine learning on spark,” in *Spark Workshop April*, 2014.
- [18] T. Elgamal, M. Yabandeh, A. Abounaga, W. Mustafa, and M. Hefeeda, “spca: Scalable principal component analysis for big data on distributed platforms,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 79–91, 2015.
- [19] M. M. Tanjim and M. A. Adnan, “ssketch: A scalable sketching technique for pca in the cloud,” in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 574–582, 2018.
- [20] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy, “The unified logging infrastructure for data analytics at twitter,” *arXiv preprint arXiv:1208.4171*, 2012.
- [21] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, “Low latency geo-distributed data analytics,” *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 421–434, 2015.
- [22] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, “Global analytics in the face of bandwidth and regulatory constraints,” in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 323–336, 2015.
- [23] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, K. Karanasos, J. Padhye, and G. Varghese, “Wanalytics: Geo-distributed analytics for a data intensive world,” in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 1087–1092, 2015.

- [24] A. Auradkar, C. Botev, S. Das, D. De Maagd, A. Feinberg, P. Ganti, L. Gao, B. Ghosh, K. Gopalakrishna, B. Harris, *et al.*, “Data infrastructure at linkedin,” in *2012 IEEE 28th International Conference on Data Engineering*, pp. 1370–1381, IEEE, 2012.
- [25] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, “Data warehousing and analytics infrastructure at facebook,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 1013–1020, 2010.
- [26] S. T. Roweis, “Em algorithms for pca and spca,” in *Advances in neural information processing systems*, pp. 626–632, 1998.
- [27] A. Spark, “Apache spark: Lightning-fast cluster computing,” URL <http://spark.apache.org>, pp. 2168–7161, 2016.
- [28] O. Choudhury, Y. Park, T. Salonidis, A. Gkoulalas-Divanis, I. Sylla, *et al.*, “Predicting adverse drug reactions on distributed health data using federated learning,” in *AMIA Annual Symposium Proceedings*, vol. 2019, p. 313, American Medical Informatics Association, 2019.
- [29] J. Xu and F. Wang, “Federated learning for healthcare informatics,” *arXiv preprint arXiv:1911.06270*, 2019.
- [30] E. U. Directive, “Payment services (PSD 2) - directive,” URL [https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366\\_en](https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366_en), 2014.
- [31] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, and Q. Yang, “Secureboost: A lossless federated learning framework,” *arXiv preprint arXiv:1901.08755*, 2019.
- [32] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, “Ffd: A federated learning based method for credit card fraud detection,” in *International Conference on Big Data*, pp. 18–32, Springer, 2019.
- [33] S. R. Pokhrel, “Towards efficient and reliable federated learning using blockchain for autonomous vehicles,” *Computer Networks*, p. 107431, 2020.
- [34] A. M. Elbir and S. Coleri, “Federated learning for vehicular networks,” *arXiv preprint arXiv:2006.01412*, 2020.
- [35] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.



- [36] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.
- [37] R. C. Geyer, T. Klein, and M. Nabi, “Differentially private federated learning: A client level perspective,” *arXiv preprint arXiv:1712.07557*, 2017.
- [38] S. N. Akter and M. A. Adnan, “Weightgrad: Geo-distributed data analysis using quantization for faster convergence and better accuracy,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 546–556, 2020.
- [39] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, “Gaia: Geo-distributed machine learning approaching {LAN} speeds,” in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pp. 629–647, 2017.
- [40] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, “Terngrad: Ternary gradients to reduce communication in distributed deep learning,” in *Advances in neural information processing systems*, pp. 1509–1519, 2017.
- [41] “Data analytics on wikipedia.”
- [42] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pp. 15–28, 2012.
- [43] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, “Distributed graphlab: A framework for machine learning in the cloud,” *arXiv preprint arXiv:1204.6078*, 2012.
- [44] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 583–598, 2014.
- [45] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, “Aggregation and degradation in jetstream: Streaming analytics in the wide area,” in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pp. 275–288, 2014.
- [46] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, “Inter-datacenter bulk transfers with netstitcher,” in *Proceedings of the ACM SIGCOMM 2011 conference*, pp. 74–85, 2011.

- [47] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” 2008.
- [48] M. Rost and K. Bock, “Privacy by design and the new protection goals,” *DuD, January*, vol. 2009, pp. 1–9, 2011.
- [49] J. Gray and D. P. Siewiorek, “High-availability computer systems,” *Computer*, vol. 24, no. 9, pp. 39–48, 1991.
- [50] O. M. Gadir, K. Subbanna, A. R. Vayyala, H. Shanmugam, A. P. Bodas, T. K. Tripathy, R. S. Indurkar, and K. H. Rao, “High-availability cluster virtual server system,” Sept. 13 2005. US Patent 6,944,785.
- [51] G. Hager and G. Wellein, *Introduction to high performance computing for scientists and engineers*. CRC Press, 2010.
- [52] A. J. Plaza and C.-I. Chang, *High performance computing in remote sensing*. CRC Press, 2007.
- [53] K. Dowd and C. Severance, “High performance computing,” 2010.
- [54] J. Subhlok, J. M. Stichnoth, D. R. O’hallaron, and T. Gross, “Exploiting task and data parallelism on a multicomputer,” in *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 13–22, 1993.
- [55] D. Tarditi, S. Puri, and J. Oglesby, “Accelerator: using data parallelism to program gpus for general-purpose uses,” *ACM SIGPLAN Notices*, vol. 41, no. 11, pp. 325–335, 2006.
- [56] H. E. Bal and M. Haines, “Approaches for integrating task and data parallelism,” *IEEE concurrency*, vol. 6, no. 3, pp. 74–84, 1998.
- [57] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, “Megatron-lm: Training multi-billion parameter language models using gpu model parallelism,” *arXiv preprint arXiv:1909.08053*, 2019.
- [58] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.
- [59] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, Ieee, 2010.

- [60] J. Dittrich and J.-A. Quiané-Ruiz, “Efficient big data processing in hadoop mapreduce,” *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2014–2015, 2012.
- [61] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, *et al.*, “Spark: Cluster computing with working sets,” *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.
- [62] D. G. Altman and J. M. Bland, “Statistics notes: the normal distribution,” *Bmj*, vol. 310, no. 6975, p. 298, 1995.
- [63] I. J. Myung, “Tutorial on maximum likelihood estimation,” *Journal of mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003.
- [64] G. J. McLachlan and T. Krishnan, *The EM algorithm and extensions*, vol. 382. John Wiley & Sons, 2007.
- [65] D. A. Reynolds, “Gaussian mixture models,” *Encyclopedia of biometrics*, vol. 741, 2009.
- [66] I. T. Jolliffe, “Principal components in regression analysis,” in *Principal component analysis*, pp. 129–155, Springer, 1986.
- [67] G. Strang, *Introduction to linear algebra*, vol. 3. Wellesley-Cambridge Press Wellesley, MA, 1993.
- [68] T. Elgamal and M. Hefeeda, “Analysis of pca algorithms in distributed environments,” *arXiv preprint arXiv:1503.05214*, 2015.
- [69] N. P. Halko, *Randomized methods for computing low-rank approximations of matrices*. PhD thesis, University of Colorado at Boulder, 2012.
- [70] M. E. Tipping and C. M. Bishop, “Probabilistic principal component analysis,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.
- [71] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, “Mlbase: A distributed machine-learning system,” in *Cidr*, vol. 1, pp. 2–1, 2013.
- [72] G. H. Golub and C. F. Van Loan, “Matrix computations. vol. vol. 3,” 2012.
- [73] J. Demmel and W. Kahan, “Accurate singular values of bidiagonal matrices,” *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 5, pp. 873–912, 1990.
- [74] V. H. Andez, J. E. R. AN, and A. É. T. AS, “A robust and efficient parallel svd solver based on restarted lanczos bidiagonalization,” *Electronic Transactions on Numerical Analysis*, vol. 31, pp. 68–85, 2008.

- [75] F. Research, “Fast randomized svd,” URL <https://research.fb.com/fast-randomized-svd>, 2014.
- [76] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, “Communication-optimal parallel and sequential qr and lu factorizations,” *SIAM Journal on Scientific Computing*, vol. 34, no. 1, pp. A206–A239, 2012.
- [77] A. Smola and S. Narayanamurthy, “An architecture for parallel topic models,” *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 703–710, 2010.
- [78] Y. S. L. Lee, M. Weimer, Y. Yang, and G.-I. Yu, “Dolphin: Runtime optimization for distributed machine learning,” in *Proc. of ICML ML Systems Workshop*, 2016.
- [79] B. Dageville, B. Ghosh, R. Chen, T. Cruanes, and M. Zait, “Parallel partition-wise aggregation,” Aug. 17 2010. US Patent 7,779,008.
- [80] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1150–1157, Ieee, 1999.
- [81] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, IEEE, 2009.
- [82] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” 2017.
- [83] M. L. Massie, B. N. Chun, and D. E. Culler, “The ganglia distributed monitoring system: design, implementation, and experience,” *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

# List of Publications

T.M. Tariq Adnan, Md. Mehrab Tanjim, Muhammad Abdullah Adnan, *Fast, Scalable and Geo-Distributed PCA for Big Data Analytics*, Elsevier Journal on Information Systems, Volume – 98, Pagee – 101710, 2021. [paper link: <https://doi.org/10.1016/j.is.2020.1017101>.]

# Appendix A

## Linear Algebra

This section provides proofs of a few unapparent theorems in linear algebra, which are crucial to this thesis.

### **The inverse of an orthogonal matrix is its transpose.**

Let  $\mathbf{A}$  be an  $m \times n$  orthogonal matrix where  $a_i$  is the  $i^{\text{th}}$  column vector. The  $ij^{\text{th}}$  element of  $\mathbf{A}^T \mathbf{A}$  is

$$(\mathbf{A}^T \mathbf{A})_{ij} = a_i^T a_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Therefore, because  $\mathbf{A}^T \mathbf{A} = \mathbf{I}$ , it follows that  $\mathbf{A}^{-1} = \mathbf{A}^T$ .

### **For any matrix $\mathbf{A}$ , $\mathbf{A}^T \mathbf{A}$ and $\mathbf{A} \mathbf{A}^T$ are symmetric.**

$$(\mathbf{A} \mathbf{A}^T)^T = \mathbf{A}^{TT} \mathbf{A}^T = \mathbf{A} \mathbf{A}^T$$

$$(\mathbf{A}^T \mathbf{A})^T = \mathbf{A}^T \mathbf{A}^{TT} = \mathbf{A}^T \mathbf{A}$$

### **A matrix is symmetric if and only if it is orthogonally diagonalizable.**

Because this statement is bi-directional, it requires a two-part “if-and-only-if” proof. One needs to prove the forward and the backwards “if-then” case.

Let us start with the forward case. If  $A$  is orthogonally diagonalizable, then  $A$  is a symmetric matrix. By hypothesis, orthogonally diagonalizable means that there exists some  $E$  such that  $A = EDE^T$ , where  $D$  is a diagonal matrix and  $E$  is some special matrix which diagonalizes  $A$ . Let us compute  $A^T$ .

$$A^T = (EDE^T)^T = E^{TT}D^TE^T = EDE^T = A$$

Evidently, if  $A$  is orthogonally diagonalizable, it must also be symmetric.

The reverse case is more involved and less clean so it will be left to the reader. In lieu of this, hopefully the “forward” case is suggestive if not somewhat convincing.

## **A symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors.**

Let  $A$  be a square  $n \times n$  symmetric matrix with associated eigenvectors  $\{e_1, e_2, \dots, e_n\}$ . Let  $E = [e_1 e_2 \dots e_n]$  where the  $i^{\text{th}}$  column of  $E$  is the eigenvector  $e_i$ . This theorem asserts that there exists a diagonal matrix  $D$  such that  $A = EDE^T$ .

This proof is in two parts. In the first part, we see that any matrix has the special property that all of its eigenvectors are not just linearly independent but also orthogonal, thus completing our proof.

In the first part of the proof, let  $A$  be just some matrix, not necessarily symmetric, and let it have independent eigenvectors (i.e. no degeneracy). Furthermore, let  $E = [e_1 e_2 \dots e_n]$  be the matrix of eigenvectors placed in the columns. Let  $D$  be a diagonal matrix where the  $i$ th eigenvalue is placed in the  $i$ th position.

We will now show that  $AE = ED$ . We can examine the columns of the right-hand and left-hand sides of the equation.

$$\text{Let hand side: } AE = [Ae_1 Ae_2 \dots Ae_n]$$

$$\text{Right hand side: } ED = [e_1 \lambda_1 e_1 e_2 \lambda_2 e_2 \dots e_n \lambda_n e_n]$$

Evidently, if  $AE = ED$  then  $Ae_i = \lambda_i e_i$  for all  $i$ . This equation is the definition of the eigenvalue equation. Therefore, it must be that  $AE = ED$ . A little rearrangement provides  $A = EDE^{-1}$ , completing the first part of the proof.

For the second part of the proof, we show that a symmetric matrix always has orthogonal eigenvectors. For some symmetric matrix, let  $\lambda_1$  and  $\lambda_2$  be distinct eigenvalues for eigenvectors  $e_1$  and  $e_2$ .

$$\begin{aligned}
\lambda_1 \mathbf{e}_1 \cdot \mathbf{e}_2 &= (\lambda_1 \mathbf{e}_1)^T \mathbf{e}_2 \\
&= (\mathbf{A} \mathbf{e}_1)^T \mathbf{e}_2 \\
&= \mathbf{e}_1^T \mathbf{A}^T \mathbf{e}_2 \\
&= \mathbf{e}_1^T \mathbf{A} \mathbf{e}_2 \\
&= \mathbf{e}_1^T (\lambda_2 \mathbf{e}_2) \\
\lambda_1 \mathbf{e}_1 \cdot \mathbf{e}_2 &= \lambda_2 \mathbf{e}_1 \cdot \mathbf{e}_2
\end{aligned}$$

By the last relation we can equate that  $(\lambda_1 - \lambda_2) \mathbf{e}_1 \cdot \mathbf{e}_2 = 0$ . Since we have conjectured that the eigenvalues are in fact unique, it must be the case that  $\mathbf{e}_1 \cdot \mathbf{e}_2 = 0$ . Therefore, the eigenvectors of a symmetric matrix are orthogonal.

Let us back up now to our original postulate that  $\mathbf{A}$  is a symmetric matrix. By the second part of the proof, we know that the eigenvectors of  $\mathbf{A}$  are all orthonormal (we choose the eigenvectors to be normalized). This means that  $\mathbf{E}$  is an orthogonal matrix so by theorem 1,  $\mathbf{E}^T = \mathbf{E}^{-1}$  and we can rewrite the final result.

$$\mathbf{A} = \mathbf{E} \mathbf{D} \mathbf{E}^T$$

Thus, a symmetric matrix is diagonalized by a matrix of its eigenvectors.



# Index

- adverse drug reactions, 5
- Amazon AWS, 58
- Amazon EMR, 58
- Amazon product data, 4
- bag-of-word, 56
- Big Data, 6
- Block-division, 2
- business information, 7
- Canopy, 35
- central limit theorem, 15
- centralized approach to data analysis, 7
- Change of Basis, 19
- cloud computing, 9
- commodity computing hardware, 3
- Complementary Naive Bayes, 35
- convergence, 67
- covariance matrix, 18
- Data mining, 6
- Data parallelism, 11
- Data sovereignty, 9
- data visualization, 1
- dense matrix, 57
- deterministic algorithm, 67
- differential privacy attacks, 5
- Dimensionality reduction, 13
- dimensionality reduction, 1
- Dirichlet, 35
- Distributed Data Analysis, 7
- Distributed Naive Bayes, 35
- Efficient Order, 72
- Eigen Value Decomposition (EVD), 18
- eigenvector, 16
- evolutionary programming, 35
- Expectation-Maximization (EM) algorithm, 17
- extensibility, 74
- fault tolerant, 10
- feature extraction, 1
- features, 7
- federated learning, 5
- Financial Services Industry (FSI), 5
- fraudulent activities, 5
- Frobenius Norm, 36
- fuzzy k-means, 35
- Gaia, 5, 38
- Ganglia, 58
- Geode, 38
- geographically distributed, 5
- gHDFS, 70
- GitHub, 71
- gSpark, 70
- Hadoop, 12
- healthcare systems, 5
- High Availability, 10
- High performance Computing, 10
- high-dimensional, 1

- Ideal Central DC, 72
- ImageNet, 56
- IoT, 5
- Iridium, 38
- k-means, 35
- knowledge discovery, 6
- latent variable, 32
- linear Gaussian model, 32
- lossy-data compression, 1
- low latency, 2
- low rank approximation, 6
- Mahout, 35
- MapReduce, 12
- Matrix diagonalization, 16
- Maximum likelihood estimation, 17
- Mean Propagation, 36
- Mean-Shift, 35
- memory efficient, 68
- microblogging, 57
- MLlib, 34
- Model Parallelism, 11
- Normal distribution, 15
- On-the-fly computation, 37
- orthogonal transformation, 18
- out-of-memory error, 2
- p-Norm, 14
- pre-processing, 6
- Principal Component Analysis (PCA), 1
- Probabilistic PCA (PPCA), 18
- product reviews, 56
- raw data, 2
- recommender system, 4
- reproducibility, 74
- resilient distributed dataset (RDD), 13
- SIFT, 56
- Singular Value Decomposition (SVD), 18
- Spark, 3, 13
- sparse, 4
- sparsity, 55
- sparsity preservation, 3
- sPCA, 35
- sSketch-PCA, 37
- Stochastic SVD (SSVD), 18
- structured data, 6
- subtask, 10
- tall and wide big data, 1
- TallnWide, 2
- TallnWide-Accumulation, 71
- telecommunications, 5
- TensorFlow, 11
- TernGrad, 5, 38
- unstructured data, 6
- variance, 21
- zero-noise-limit, 2

Generated using Postgraduate Thesis L<sup>A</sup>T<sub>E</sub>X Template, Version 1.03. Department of  
Computer Science and Engineering, Bangladesh University of Engineering and  
Technology, Dhaka, Bangladesh.

This thesis was generated on Tuesday 2<sup>nd</sup> March, 2021 at 6:42pm.