Communication Efficient Distributed Tensor Factorization based on Local SGD for Collaborative Health Data Analysis

by

Zhangyi Pan

Li Xiong
Advisor

Department of Computer Science

2020

Communication Efficient Distributed Tensor Factorization based on Local SGD for Collaborative Health Data Analysis

by

Zhangyi Pan

Li Xiong
Advisor

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Computer Science

2020

Abstract

Communication Efficient Distributed Tensor Factorization based on Local SGD for Collaborative Health Data Analysis

By Zhangyi Pan

Tensor factorization is a useful technique for phenotyping, and has proven to be an effective way to approach massive medical data. We can factorize the electronic health records (EHRs) to discover latent clinical concepts that capture interactions among multiple attributes such as medication and diagnosis. One challenge is how to perform high-throughput tensor factorization using EHRs distributed among multiple sites while preserving patient privacy. Federated tensor factorization has been proposed recently in which local sites communicate intermediate factors with differential privacy to a global server without sharing the original data. Existing methods based on Elastic Averaging Stochastic Gradient Descent (EASGD), although has lowered the communication cost by infrequent communications, relies on an auxiliary penalty which leads to inferior converged results. In this thesis, we propose a communication efficient approach based on local Stochastic Gradient Descent, where the local sites only communicate with the global server after several iterations of local updates and do not require the auxiliary penalty. Our experiments using real medical dataset show that the proposed approach can simultaneously achieve better accuracy and lower communication cost than the state-of-the-art approaches.

Communication Efficient Distributed Tensor Factorization based on Local SGD for Collaborative Health Data Analysis

By

Zhangyi Pan

Li Xiong
Advisor

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Computer Science

2020

Acknowledgements

My thanks to Dr.Li Xiong, my advisor, who gives me this precious opportunity to further probe into the field of machine learning. Also thanks to postdoc fellow Jian Lou and PhD student Jing Ma, who gives me a lot of guidance and help throughout the research.

Contents

Chapter 1

Introduction

Nowadays, more and more hospitals choose to store clinical histories in the form of Electronic Health Records (EHRs) [1]. While such an approach benefits both in research and practical aspects, it requires EHRs to be succinct but comprehensive. Many favor phenotyping to transform EHR data into certain medical concepts, i.e. to extract phenotypes. We can then use the extracted phenotypes for multiple downstream data analysis purposes such as predicting mortality or risk of certain diseases and improving treatments.

A common approach for phenotyping is the tensor factorization [2, 3, 4, 5], which extracts phenotypes by factorizing the input tensor into factors. We treat the data as a tensor, and then decompose the tensor into factors that store some aspect of information as well as the correlation between different aspects (e.g. correlated medical treatments and diseases). In such a way, the information appears to be more interpretable and suitable for further utilities and therefore can be used to improve healthcare quality and thereby utilizes biomedical discoveries.

Recent studies also indicate that we can combine EHR data from different hospitals to get a global result and the result would benefit individual hospitals. By incorporating data from other hospitals, local hospitals can have a more comprehensive understanding of certain diseases and medications, therefore achieve a better service for their patients. Nevertheless, issues arise from such global integration: 1) leakage of patients' privacy during intermediary results sharing; and 2) high communication costs between global server and local sites.

To address the privacy issue, we apply zero-concentrated differential privacy technique [6], which outperforms traditional differential privacy technique in terms of tigher compositional ability. These privacy techniques guarantee privacy to a strict standard, thus mitigate privacy leak to a great extent.

To reduce the communication cost, we integrate the idea of Local Stochastic Gradient Descent (Local SGD) [7]. Compared with general Stochastic Gradient Descent (SGD) which updates global results each iteration, Local SGD only communicates after some number of local updates, therefore greatly reduce communication costs. Each time after global communication, the global server sends aggregated information back to local sites and continue local updates. With a comfortable batch size (number of local updates before a global update), we can reduce communication costs and achieve a better outcome. To summarize, our algorithm improves existing methods by: 1) a higher accuracy evaluated under the RMSE loss; 2) better utility when used for downstream machine learning task (e.g.mortality prediction); 3) strong privacy guarantee for patients and 4) lower communication costs. We evaluate our algorithm on the MIMIC III dataset to verify the improved accuracy, communication cost and utility of the proposed method.

Chapter 2

Background and Related Work

In this section, we will introduce the notations we use in this thesis and related works, including tensor factorization, concentrated differential privacy, and Local SGD.

2.1 Notations

Here are some notations we used in this thesis:

Symbols	Descriptions
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product
$\circ$	Outer Product
$*$	Element-wise Product
$\mathcal{N}$	Number of modes
$\mathcal{T}$	Number of local sites
$\mathcal{R}$	Number of ranks
$\mathbf{X}_{(n)}$	Mode-n matricization of tensor $\mathcal{O}$
$\mathcal{X}$, $\mathbf{X}$, $\mathbf{x}$	Tensor, matrix, vector
$\widehat{\mathbf{B}}, \widehat{\mathbf{C}}$	Global factor matrices
$\mathbf{A}^{[t]}$, $\mathbf{B}^{[t]}$, $\mathbf{C}^{[t]}$	Local factor matrices at the t-th site
$\mathcal{X}^{[t]}$	Local tensor at the t-th site
$\mathbf{x}_{i:}$, $\mathbf{x}_{:r}$	Row vector, Column vector

2.2 Tensor Factorization

In this section, we will introduce tensor factorization, which is the main topic of interest in our thesis.

2.2.1 Tensor

A tensor is a multidimensional array [8]. More formally, an N-way or Nth-order tensor is an element of the tensor product of N vector spaces, each of which has its own coordinate system. To be more clear, a first-order tensor is a vector, a second-order tensor is a matrix and the following image shows a third order tensor.

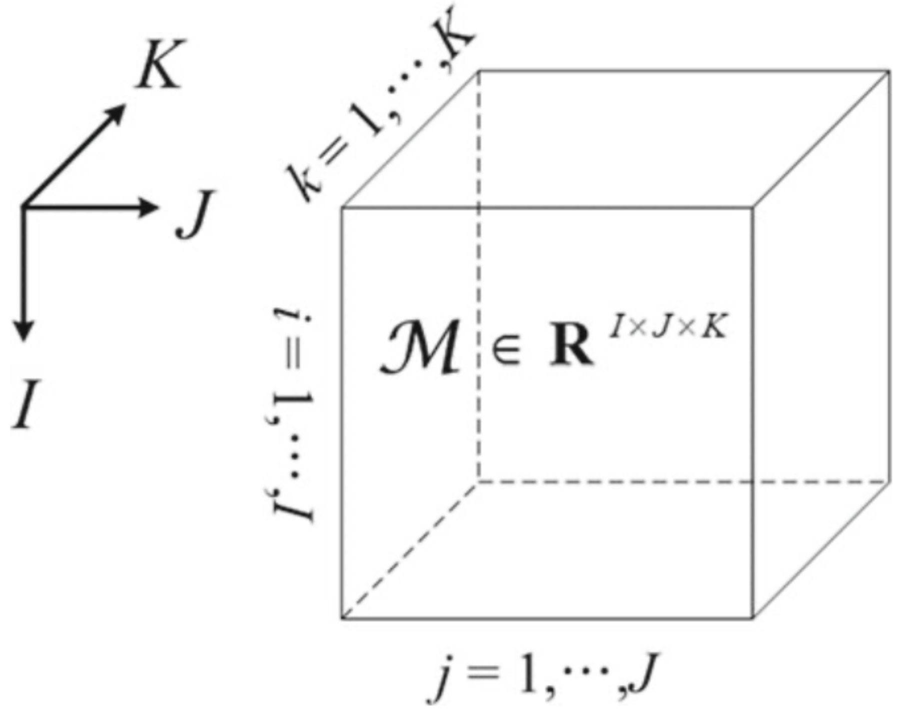

Figure 2.1: A third order tensor [9]

2.2.2 Matrix Products

To begin with, we define the following products which are commonly used in the tensor factorization.

Definition 1 *The Kronecker product of matrices $A \in \mathbb{R}^{I\times J}$ and $B \in \mathbb{R}^{K\times L}$ is denoted by$A \otimes B$. The result is a matrix of size (IK) $\times$ (JL) and defined by:*

$$\begin{aligned}\mathbf{A}\otimes\mathbf{B} &= \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix}, \\ &= \begin{bmatrix} \mathbf{a}_1\otimes\mathbf{b}_1 & \mathbf{a}_1\otimes\mathbf{b}_2 & \mathbf{a}_1\otimes\mathbf{b}_3 & \cdots & \mathbf{a}_J\otimes\mathbf{b}_{L-1} & \mathbf{a}_J\otimes\mathbf{b}_L \end{bmatrix},\end{aligned} \tag{2.1}$$

Definition 2 *The Khatri–Rao product is the "matching columnwise" Kronecker product. Given matrices* $\mathbf{A} \in \mathbb{R}^{I \times K}$ *and* $\mathbf{B} \in \mathbb{R}^{J \times K}$ *, their Khatri-Rao product is denoted by* $\mathbf{A} \odot \mathbf{B}$*. The result is a matrix of size* $(IJ) \times K$ *defined by:*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_K \otimes \mathbf{b}_K \end{bmatrix}, \tag{2.2}$$

If $\mathbf{a}$ *and* $\mathbf{b}$ *are vectors, then the Khari-Rao and Kronecker products are identical, i.e.* $\mathbf{A} \otimes \mathbf{B} = \mathbf{A} \odot \mathbf{B}$.

2.2.3 CP Decomposition

The CANDECOMP-PARAFAC (CP) decomposition factorizes a tensor into a sum of component rank-one tensors or vectors [10]. Each vector can represent a certain concept and we want the decomposed tensors to contain information about its concept and also we can multiply to get the correlation between dimensions. Suppose we have a third-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, we want to decompose it as:

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{2.3}$$

where R is referred to as the rank of the CP decomposition. Elementwise, it will be:

$$\mathbf{x}_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}, \tag{2.4}$$

Figure 2.2 will be a good illustration for CP decomposition and R is the number of component rank-one tensors.

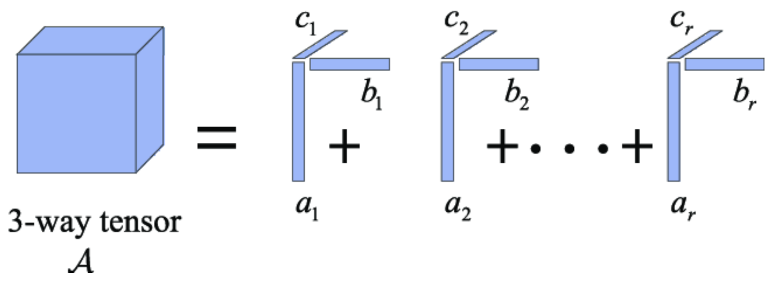

Figure 2.2: CP decomposition for a three-way tensor [11]

We call the combinations of vectors from one certain dimension or rank-one components as factor matrices, i.e. $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_R \end{bmatrix}$. For a third-order tensor, we will decompose it to three factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ and for tensor factorization with a 3-way tensor, we will be coping with the following optimization problem:

$$\min \frac{1}{2} \|\mathbf{\mho} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_F^2, \tag{2.5}$$

which means we want the difference between the original tensor and decomposed matrices to be as small as possible.

2.3 Stochastic Gradient Descent

Gradient Descent is a first-order iterative algorithm to find a local optimum for a differentiable equation. We take the gradient of the equation and iteratively subtract a proportion of the gradient from the equation to get a local minimum. If we want a local maximum, we add a proportion of the gradient each time and we refer to it as gradient ascent. To be more specific, suppose we have a function $\mathcal{F}(x)$ and we have an initial guess x_0, we will then find the local minimum through:

$$x_{n+1} = x_n - \eta_n \nabla \mathcal{F}(x), \tag{2.6}$$

and for a η small enough, we are moving against the gradient, thus reducing our $\mathcal{F}(x)$ value and hopefully x_{n+1} will converge to a local minimum.

Stochastic Gradient Descent (SGD) is a stochastic approximation version of the gradient descent [12]. SGD replaces the gradient in Gradient Descent method with an approximation of it. We want to minimize the sum of the objective function related to each observation:

$$\min_x \mathcal{F}(x) = \min_x \frac{1}{n} \sum_{i=1}^{n} \mathcal{F}_i(x), \tag{2.7}$$

and $\mathcal{F}_i(x)$ denotes the objective function associated with observation i. With such an idea, we can rewrite the optimization problem as:

$$x = x - \eta \nabla \mathcal{F}(x) = x - \frac{\eta}{n} \sum_{i=1}^{n} \nabla \mathcal{F}_i(x), \tag{2.8}$$

in which the η denotes the learning rate. However, problems may arise if we have too much data as we may spend too much on computing some unrelated data. In SGD, we only take a small proportion of data or even one observation each time and update, that is:

$$x = x - \eta \nabla \mathcal{F}_i(x), \tag{2.9}$$

With SGD, we may save a lot of computational costs and speed up the algorithm, but at the cost of a slower convergence rate.

2.4 Concentrated Differential Privacy

Differential privacy is a de facto notion to preserve individual information [13]. We want the dataset to be available for the public but also prevent personal information leakage. Concentrated Differential Privacy (CDP) is a new variant of the differential privacy [6]. A certain formulation of CDP is zero-concentrated differential privacy (zCDP) which utilizes the Renyi divergence between probability distributions to measure the divergence and requires the privacy loss random variable to be sub-Gaussian. It has been shown to provide tighter privacy composition analysis. To be more specific:

Definition 3 *Two datasets X, Y are said to be neighboring if $|X|$=$|Y|$=n but X and Y differ in one record. [14]*

Definition 4 *We say a randomized mechanism $\mathcal{A}$ is ρ-zero concentrated private if for any two neighboring databases $\mathcal{D}$ and $\mathcal{D}^{'}$ that differs in at most one data entry and for all* $\alpha \in (1, \inf)$*:*

$$D_\alpha(\mathcal{A}(\mathcal{D}) \parallel \mathcal{A}(\mathcal{D}^{'})) \triangleq \frac{1}{\alpha - 1} \log\left(\mathbb{E}[e^{(\alpha-1)L(o)}]\right) \leq \rho\alpha, \tag{2.10}$$

where $D_\alpha(\mathcal{A}(\mathcal{D}) \parallel \mathcal{A}(\mathcal{D}^{'}))$ is called α-Renyi divergence between the distributions $\mathcal{A}(\mathcal{D})$ and $\mathcal{A}(\mathcal{D}^{'})$), and L(o) is the privacy loss random variable which is defined as:

$$L^{(o)}_{(\mathcal{A}(\mathcal{D})\|\mathcal{A}(\mathcal{D}'))} \triangleq log\frac{Pr(\mathcal{A}(\mathcal{D} = o)}{Pr(\mathcal{A}(\mathcal{D}') = o)}, \tag{2.11}$$

We have already talked about how we define privacy guarantees with zCDP. Now the remaining question is how we design a mechanism to satisfy these privacy-preserving guarantees. A very useful privacy mechanism is the Gaussian Mechanism:

Definition 5 *Suppose we have an arbitrary $\epsilon \in (0, 1)$. For $c^2 > 2ln(\frac{1.25}{\delta})$, the Gaussian Mechanism with parameter $\sigma \geq c\Delta_2(\mathcal{A})/\epsilon$, adding a noise with scale to $\mathcal{N}(0, \sigma^2)$ to each component of the algorithm $\mathcal{A}$, is $(\epsilon - \delta)$-deferentially private*

It's also proved that a Gaussian mechanism with noise $\mathcal{N}(0, \sigma^2)$ where $\sigma = \sqrt{1/(2\rho)}\,\Delta_2$ satisfies ρ-zCDP. As a result, we will use Gaussian noise in our experiments to satisfy our proposed concentrated differential privacy.

2.5 Distributed Tensor Factorization

There already exist some methods that deal with distributed tensor factorization. In this section, we will introduce some state-of-art models.

2.5.1 Vanilla Distributed SGD

Vanilla Distributed SGD is a distributed version of SGD, which separates the data into several parts and distributes the partitions to local workers. Then the local worker performs gradient descent on the distributed data and then aggregates their results [15]. The overall algorithm can be described as:

Algorithm 1 Vanilla Distributed SGD

```
    **Input**: c, $\eta$, t
1: for i $\in$ {1,...,t} do in parallel
2:     $\mathbf{v}_i$ = SGD($c_i$, $\eta$)
3: end for
4: Aggregate the results from all the clients by $\mathbf{v} = \frac{1}{t}\sum_{i=1}^{t}\mathbf{v}_i$
5: return v
```

Here c denotes the data and we separate c into k parts and let the local workers compute the results in parallel. Such an approach takes more data into account while runs faster as each local site only computes a proportion of the data.

2.5.2 ADMM Based Approaches

The alternating direction method of multipliers (ADMM) is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which is then easier to handle [16]. It has found wide applications in a number of areas. ADMM algorithm can be applied to lots of machine learning problems, including tensor factorization. The objective function for general ADMM is:

$$\begin{aligned} &\textbf{minimize } \mathcal{F}(x) + \mathcal{G}(z) \\ &\textbf{subject to } Ax + Bz = c \end{aligned} \tag{2.12}$$

A tensor factorization method that incorporates Alternating direction method of multipliers (ADMM) algorithm into the optimization problem is TRIP [17]. Compared with methods based on Distributed SGD, TRIP extends ADMM to tensor factorization and incorporates infrequent communication, thus saves some communication costs. However, to assist infrequent communication, TRIP introduces some auxiliary variables and thus increases the communication cost per iteration. In TRIP, the overall communication cost is reduced but the cost per iteration is larger thus we may be interested in avoiding the auxiliary variables. For each worker t, TRIP will be minimizing the objective function:

$$\begin{aligned} \min \frac{1}{2}\|\mathcal{O}^{[t]} - [\![\mathbf{A}^{[t]}, \mathbf{B}^{[t]}, \mathbf{C}^{[t]}]\!]\|_F^2 + \frac{\lambda}{2}\|\mathbf{I} - \mathbf{B}'\mathbf{B}\|_F^2 + \frac{\lambda}{2}\|\mathbf{I} - \mathbf{C}'\mathbf{C}\|_F^2 \\ \text{s.t. } \mathbf{B} = \mathbf{B}^{[t]}, \mathbf{C} = \mathbf{C}^{[t]} \\ \text{and } \mathbf{B}' = \mathbf{B}, \mathbf{C}' = \mathbf{C} \end{aligned} \tag{2.13}$$

2.5.3 EA-SGD Based Approaches

EA-SGD is a modified version of SGD where the communication and coordination of work among local workers are based on an elastic force that links the parameters they compute with the central server [18]. The overall objective function for EA-SGD can be expressed as:

$$\mathcal{F}(x) + \frac{\gamma^2}{2}(x - \widetilde{x}), \tag{2.14}$$

where $\widetilde{x}$ denotes the global variable.

DPFact extends Elastic Averaging SGD (EA-SGD) to tensor factorization and adds some noises to preserve privacy [19]. DPFact also has infrequent communication and does not need auxiliary variables in communication. As a result, DPFact has a smaller communication cost than TRIP. Nevertheless, to achieve infrequent communication, DPFact introduces some elastic penalties and such penalties may change the objective function for the optimization during updates, thus jeopardizing the overall accuracy and utility. In conclusion, we want a method that infrequently communicates and preserves accuracy and utility and we then propose the usage of Local SGD, an improved version of Vanilla Distributed SGD which communicates after several local updates. DPFact optimizes the objective function defined as following:

$$\min \frac{1}{2}\|\mathbf{\mathcal{O}}^{[t]} - [\![\mathbf{A}^{[t]}, \mathbf{B}^{[t]}, \mathbf{C}^{[t]}]\!]\|_F^2 + \frac{\gamma}{2}\|\mathbf{B}^{[t]} - \widehat{\mathbf{B}}\|_F^2 + \frac{\gamma}{2}\|\mathbf{C}^{[t]} - \widehat{\mathbf{C}}\|_F^2 + \mu\|(\mathbf{A}^{[t]})^\top\|_{2,1}. \tag{2.15}$$

2.6 Local Stochastic Gradient Descent

Distributed SGD seems very appealing in every aspect, but it gives rise to another problem: the high communication costs caused by the communication between the local workers and the server. In each iteration, the local workers will have to share their results and the costs can accumulate through the iterations. To speed up to local update, we want the k to be larger thus each local site will be assigned less data but increase the communication costs. In such a way, local update time decreases while the communication time increases, wasting the benefits achieved from parallel work and stochastic approximation.

Thus we use the Local Stochastic Gradient Descent (Local SGD) [7], an extension of SGD and Distributed SGD. In Local SGD, we still separate the work to several clients, just like in Distributed SGD, but this time we locally update several times separately and then communicates only at some certain iterations (typically after some number of iterations). We can express the way that Local SGD evolves as :

$$x_{i+1}^t = \begin{cases} \mathrm{x}_i^t - \eta_i \nabla \mathcal{F}(x_{i+1}^t) & i+1 \notin \mathcal{L}_I \\ \frac{1}{T}\sum_{t=1}^{T}(x_i^t - \eta_i \nabla \mathcal{F}(x_{i+1}^t)) & i+1 \in \mathcal{L}_I, \end{cases} \tag{2.16}$$

where K denotes the number of local sites and η_t denotes the learning rate at iteration t (we allow the learning rate to change from step to step, but we may also keep a global learning rate). T is the set of all indices (iterations) and $\mathcal{L}_T$ is the list of indices that we will do global communicates. If $\mathcal{L}_T$ contains every index, then it will be identical to Parallel SGD and if $\mathcal{L}_T$ contains only the last index, we will only communicate at the very end. Both cases are not ideal so we will need to find a proper gap between communications. If we fix the number of total iterations, our communication costs will be greatly reduced, and if we compute until convergence, we will need to deal with the number of total iterations. With respect to other research, Local SGD converges fast, and as a result, we can conclude that Local SGD reduces the communication cost. The overall algorithm can be described as:

Algorithm 2 Local SGD

```
    Input: I, $\eta$, T
1: for i ∈ I do
2:     for t ∈ {1,...,T} do in parallel
3:         if $i + 1 \in \mathcal{L}_I$ then
4:             $x_{i+1}^t \longleftarrow \frac{1}{T}\sum_{t=1}^{T}(x_i^t - \eta_i \nabla \mathcal{F}(x_{i+1}^t))$
5:         else
6:             $x_{i+1}^t \longleftarrow x_i^t - \eta_i \nabla \mathcal{F}(x_{i+1}^t)$
7:         end if
8:     end for
9: end for
```

In sum, local SGD is a useful extension of the original gradient descent in which we can achieve a speedup while getting rid of excessive communication costs and can deal with a large amount of data.

Chapter 3

Proposed Model

In this chapter, we provide an overview of the algorithm and further details about how we approach the optimization problem.

3.1 Overview

For simplicity, in this chapter, we suppose we have a three-order tensor with modes patients, procedures and diagnoses as the input, even though our algorithm generally applies to tensors with a higher order. For input, we want the format to be (value, mode 1, mode 2, mode 3) and in our case, mode 1, mode 2, mode 3 represents patients, procedures, and diagnoses correspondingly.

Since our algorithm is a distributed one, which means we have local sites and a global server, we have to compute phenotypes while protecting the privacy of the patients in local sites. Though we assume the global server may be trusted, but curiosity may still trigger some privacy probing within the protocols. As a result, the local sites only send **B** and **C** to the global server, which denotes procedure and diagnoses information. With the incorporation of Local SGD, we first set a batch size b and we only globally communicate after b iterations of local updates and we add Gaussian noise to **B** and **C** factor matrices each iteration. After global communication, the server sends the updated global factor matrices back to the local sites and local sites continue their local updates. The process continues until convergence, which means we have little change in local factor matrices.

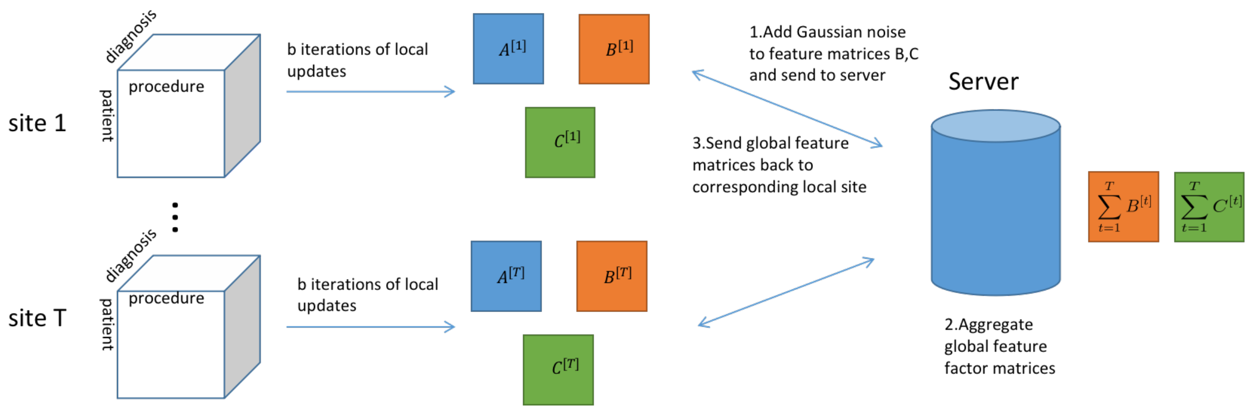
site 1
diagnosis
procedure
patient
b iterations of local updates
$A^{[1]}$
$B^{[1]}$
$C^{[1]}$
site T
diagnosis
procedure
patient
b iterations of local updates
$A^{[T]}$
$B^{[T]}$
$C^{[T]}$
1.Add Gaussian noise to feature matrices B,C and send to server
3.Send global feature matrices back to corresponding local site
Server
$\sum_{t=1}^{T} B^{[t]}$
$\sum_{t=1}^{T} C^{[t]}$
2.Aggregate global feature factor matrices

Figure 3.1: Algorithm Overview

Figure 3.1 is an overview of the whole algorithm. $\mathfrak{O}^{[t]}$ represents the t-th local sites and the local sites compute in parallel, thus the number of sites will not have a significant impact on the local updates time.

3.2 Algorithm

Algorithm 3 LocalTF

```
    Input: $\mathbf{\mathfrak{O}}$, $\tau$, $\eta$, $\gamma$, $\mu$, $\sigma$, $\rho$, b
1: Randomly initialize the global feature matrices $\mathbf{B}$, $\mathbf{C}$ and local feature factor matrices
   $\mathbf{B}^{[t]}$, $\mathbf{C}^{[t]}$ for t $\in \mathfrak{T}$.
2: while $\mathbf{B}^{[t]}$, $\mathbf{C}^{[t]}$ not converge do
3:     if Hospital then
4:         for k = 1, ..., t do
5:             Shuffle tensor elements;
6:             for observation i do
7:                 Update $\mathbf{A}^{[t]}$ using (3.6);
8:                 Update $\mathbf{B}^{[t]}$, $\mathbf{C}^{[t]}$ using (3.10);
9:             end for
10:            Proximal update for ${}_{new}\mathbf{A}^{[t]}$ using (3.7);
11:        end for
12:        Calibrate Gaussian noise matrix $\mathbf{\mathfrak{M}}_B^{[t]}$ and $\mathbf{\mathfrak{M}}_C^{[t]}$ as $\mathcal{N}(0, \frac{\Delta_2^2}{(2\rho)})$ for each factor matrix;
13:        Update factor matrices ${}_{priv}\mathbf{B}^{[t]}$ and ${}_{priv}\mathbf{C}^{[t]}$ using (3.11);
14:        if Every b iterations then
15:            Send ${}_{priv}\mathbf{B}^{[t]}$, ${}_{priv}\mathbf{C}^{[t]}$ to Server;
16:        end if
17:    end if
18:    if Server for every b iterations then
19:        Receive ${}_{priv}\mathbf{B}^{[t]}$, ${}_{priv}\mathbf{C}^{[t]}$ from local hospitals;
20:        Update global $\widehat{\mathbf{B}}, \widehat{\mathbf{C}}$ using (3.12);
21:        Send $\widehat{\mathbf{B}}, \widehat{\mathbf{C}}$ back to local hospitals;
22:    end if
23: end while
```

In Algorithm 1, we separate the problem to the hospital side and the global server side. For hospitals, each hospital computes its own part in parallel and calibrate noises to the feature factor matrices. In every b iterations, the hospitals send their results to the global server. For server, they gather and compute the global information once every b iterations and then send the corresponding results back to the hospitals.

3.3 Worker Side Update

For local updates, we will generally be solving the following optimization problem:

$$\min \frac{1}{2}\|\mathcal{O}^{[t]} - [\![\mathbf{A}^{[t]}, \mathbf{B}^{[t]}, \mathbf{C}^{[t]}]\!]\|_F^2 + \mu\|(\mathbf{A}^{[t]})^\top\|_{2,1}, \tag{3.1}$$

For the patient factor matrix, we will add a $l_{2,1}$-norm to regularize the matrix. For the feature matrices, we will be adding privacy terms. Because we are applying different terms to different matrices, we will be talking about patient and feature factor matrices separately.

3.3.1 Patient Factor Matrix

For some local site t, we will be updating the patient factor matrix $\mathbf{A}^{[t]}$ through minimizing an objective function with respect to local tensors and the $l_{2,1}$-norm, which is:

$$\min_{\mathbf{A}^{[\mathbf{t}]}} \underbrace{\frac{1}{2}\|\mathcal{O}^{[t]} - [\![\mathbf{A}^{[t]}, \mathbf{B}^{[t]}, \mathbf{C}^{[t]}]\!]\|_F^2}_{\mathcal{F}} + \underbrace{\mu\|(\mathbf{A}^{[t]})^\top\|_{2,1}}_{\mathcal{H}}, \tag{3.2}$$

We incorporate the $l_{2,1}$-norm because it does help the model but also makes the optimization non-differentiable. As a result, we treat the problem as a combination of two functions, the differentiable $\mathcal{F}$ and non-differentiable $\mathcal{H}$. In such a way, we can approach the local optimization problem with the proximal gradient method [20]. With proximal gradient method, we can iteratively update the $\mathbf{A}^{[t]}$ with the proximal operator:

$$_{new}\mathbf{A}^{[t]} = \mathbf{prox}_{\eta\mathcal{H}}(\mathbf{A}^{[t]} - \eta\nabla\mathcal{F}(\mathbf{A}^{[t]})), \tag{3.3}$$

where η denotes the learning rate and the proximal operator can be computed through:

$$\mathbf{prox}_{\eta\mathcal{H}}(\Theta) = \underset{\Theta}{\operatorname{argmin}}(\frac{1}{2\eta}\|\Theta - \widehat{\Theta}\| + \mathcal{H}(\Theta)), \tag{3.4}$$

where $\widehat{\Theta}$ denotes the updated matrix, which is the formula inside the proximal operator in (4.3). We can get the closed form solution for the proximal operator by:

$$\mathbf{prox}_{\eta\mathcal{H}}(\Theta) = \widehat{\Theta_{r:}}(1 - \frac{\mu}{\|\widehat{\Theta_{r:}}\|_2})_+, \tag{3.5}$$

After some calculations, we can get the update rule for the matrix with respect to each row, which is:

$$\mathbf{a}_{i:}^{[t]} \leftarrow \mathbf{a}_{i:}^{[t]} - \eta[(\mathbf{a}_{i:}^{[t]}(\mathbf{b}_{j:}^{[t]} * \mathbf{c}_{k:}^{[t]})^\top - \mathbf{\mho}_{ijk}^{[t]})(\mathbf{b}_{j:}^{[t]} * \mathbf{c}_{k:}^{[t]})], \tag{3.6}$$

After updating all entries, we will use the proximal operator to update the patient factor matrix $\mathbf{A}^{[t]}$ by:

$$_{new}\mathbf{A}^{[t]} = \mathbf{prox}_{\eta\mathcal{H}}(\mathbf{A}^{[t]}), \tag{3.7}$$

3.3.2 Feature Factor Matrix

For the feature factor matrices, we locally update them through the objective functions:

$$\begin{aligned} \min_{\mathbf{B^{[t]}}} f_b &= \frac{1}{2}\|\mathbf{\mho}^{[t]} - [\![\mathbf{A}^{[t]}, \mathbf{B}^{[t]}, \mathbf{C}^{[t]}]\!]\|_F^2 \\ \min_{\mathbf{C^{[t]}}} f_c &= \frac{1}{2}\|\mathbf{\mho}^{[t]} - [\![\mathbf{A}^{[t]}, \mathbf{B}^{[t]}, \mathbf{C}^{[t]}]\!]\|_F^2 \end{aligned}, \tag{3.8}$$

We then take the partial derivatives with respect to row vector $\mathbf{b}_{j:}^{[t]}$ and $\mathbf{c}_{k:}^{[t]}$ respectively, and we will get:

$$\begin{aligned}\frac{\partial f_b}{\partial \mathbf{b}_{j:}^{[t]}} &= [(\mathbf{a}_{i:}^{[t]}(\mathbf{b}_{j:}^{[t]} * \mathbf{c}_{k:}^{[t]})^\top - \mathfrak{O}_{ijk}^{[t]})(\mathbf{a}_{i:}^{[t]} * \mathbf{c}_{k:}^{[t]})] \\ \frac{\partial f_c}{\partial \mathbf{c}_{k:}^{[t]}} &= [(\mathbf{a}_{i:}^{[t]}(\mathbf{b}_{j:}^{[t]} * \mathbf{c}_{k:}^{[t]})^\top - \mathfrak{O}_{ijk}^{[t]})(\mathbf{a}_{i:}^{[t]} * \mathbf{b}_{j:}^{[t]})],\end{aligned} \tag{3.9}$$

and we will update the factor matrices by rows adding the partial derivatives:

$$\begin{aligned}\mathbf{b}_{j:}^{[t]} &\leftarrow \mathbf{b}_{j:}^{[t]} - \eta \frac{\partial f_b}{\partial \mathbf{b}_{j:}^{[t]}} \\ \mathbf{c}_{k:}^{[t]} &\leftarrow \mathbf{c}_{k:}^{[t]} - \eta \frac{\partial f_c}{\partial \mathbf{c}_{k:}^{[t]}},\end{aligned} \tag{3.10}$$

And we will update τ rounds locally, then we will add the Gaussian noises to the feature factor matrices before we send them to the central server. We will be adding zero mean Gaussian noise with a standard deviation of $\sigma = \frac{\Delta_2^2}{2\rho}$, thus we will be adding a Gaussian noise matrix $\mathbf{M}$ to each feature factor matrix for privacy purpose and a detailed privacy analysis will be provided in section 3.5, the noise will be denoted as:

$$\begin{aligned}{}_{priv}\mathbf{B}^{[t]} &\leftarrow \mathbf{B}^{[t]} + \mathbf{M}_B^{[t]}, \\ {}_{priv}\mathbf{C}^{[t]} &\leftarrow \mathbf{C}^{[t]} + \mathbf{M}_C^{[t]},\end{aligned} \tag{3.11}$$

3.4 Server Side Update

After local updates, we will be updating the global matrices according to the objective function for the feature factor matrices $\mathbf{B}$ and $\mathbf{C}$, which will be:

$$\begin{aligned}\widehat{\mathbf{B}} &\leftarrow \widehat{\mathbf{B}} + \eta \sum_{t=1}^{T} \gamma({}_{priv}\mathbf{B}^{[t]} - \widehat{\mathbf{B}}), \\ \widehat{\mathbf{C}} &\leftarrow \widehat{\mathbf{C}} + \eta \sum_{t=1}^{T} \gamma({}_{priv}\mathbf{C}^{[t]} - \widehat{\mathbf{B}}),\end{aligned} \tag{3.12}$$

This update incorporates the information collected from the local sites and gets a global phenotype that best represents all the local information. The global server will then send the updated global matrices back to the local sites.

3.5 Privacy Analysis

In this section, we want to analyze the privacy budget for our algorithm compared with DPFact.

Theorem 1 *The algorithm 1 is (ϵ, δ)-differentially private if we choose the input privacy budget for each factor matrix per epoch as*

$$\rho = \frac{\epsilon^2}{8Elog(1/\delta)}, \tag{3.13}$$

where E denotes the number of iterations when the algorithm is converged

The proof for this theorem can be found in DPFact [19]. The standard deviation of our Gaussian noise will be $\sigma = \sqrt{1/(2\rho)}\,\Delta_2$ and our privacy is closely related with our convergence rate. We can apply the translation between (ϵ, δ)-differential privacy and concentrated ρ-CDP based on section 2.3. We fix our ϵ and δ of each iteration in our method and we incorporate the ϵ and δ in each iteration from DPFact. Thus if our E is smaller than DPFact, we will have a smaller standard deviation than DPFact. With a smaller standard deviation, we have less perturbation and therefore achieve a better utility under the same privacy budget.

Chapter 4

Experiments

We evaluate our algorithm with MIMIC III dataset and we evaluate the accuracy measured by the RMSE metric. Also, we calculate and compare the communication cost. We also evaluate the accuracy of utility, which we use the mortality prediction. We named our method as LocalTF, which represents local SGD and tensor factorization.

4.1 Dataset

In this section, we will elaborate the dataset we use and what we have done to get the final data used in experiments.

4.1.1 Data Description

We use MIMIC-III dataset for evaluation. MIMIC-III (Medical Information Mart for Intensive Care III) is a publicly available database of health-related data collected from over forty thousand patients in care units of the Beth Israel Deaconess Medical Center between 2001 and 2012. It contains information about patient information(no name), procedures, medications, diagnoses and mortality (used for mortality).

Though MIMIC-III consists of lots of information, we only concern some certain tables, that is ADMISSIONS, ICUSTAYS, DIAGNOSES_ICD, PROCEDURES_ICD, D_ICD_DIAGNOSES, D_ICD_PROCEDURES, and PATIENTS.

The ADMISSIONS table provides information regarding patients' admissions to the hospital. It has its key as the HADM_ID, which represents each admission to hospital. It also has many useful attributes like SUBJECT_ID, which link us to patient information. We also use the HOSPITAL_EXPIRE_FLAG, which denotes whether the patient died in this admission and we use this for utility prediction.

The ICUSTAYS table contains information about each ICU stay. The key of ICUSTAYS is ICUSTAY_ID, and we can link to ADMISSIONS with HADM_ID and link to PATIENTS with SUBJECT_ID. We also use the attribute FIRST_CAREUNIT to separate the data into different ICU units. DIAGNOSES_ICD and PROCEDURES_ICD are the tables that store the information of the diagnoses and procedures corresponding to admissions. Either table contains attributes as Row_ID, SUBJECT_ID, HADM_ID, SEQ_NUM and ICD9_CODE. We can use the HADM_ID to find the corresponding admission and ICD9_Code represents the detailed description of the diagnoses and procedures. SEQ_NUM denotes the order of the procedures or the diagnoses. D_ICD_DIAGNOSES and D_ICD_PROCEDURES are the tables with the detailed diagnoses and procedures information. We relate this description to DIAGNOSES_ICD and PROCEDURES_ICD with ICD9_CODEs.

4.1.2 Pre-processing

We use mySQL to select the hospital admissions with 202 procedures and 316 diagnoses that appeared most frequently in the database. We order the procedures and diagnoses based on the number of different admissions in which they are related to and assign the order as the new key. In the experiments, we only select admissions with those frequent procedures and diagnoses and abandon the others. We join the ICUSTAYS, ADMISSIONS, DIAGNOSES_ICD, and PROCEDURES_ICD to combine all the information we need. We then separate the ICU stays into 6 local tensors, each representing stays in different ICU units. We want to count how many times the same patient has been given identical diagnoses and procedures and the processed data should look like:

Count	Patient ID	Medication ID	Diagnosis ID
1	1	10	3
1	1	8	7
3	2	10	4
2	2	11	7
5	3	1	8

We join the DIAGNOSES_ICD and PROCEDURE_ICD with the ADMISSIONS based on HADM_ID. We also find all the SUBJECT_ID in our used data and get the corresponding mortality tag from ADMISSIONS attribute HOSPITAL_EXPIRE_FLAG. We want our patient id to be consecutive distinct integers, thus we select all the distinct SUBJECT_ID and assign the ranking to them as the new PATIENT_ID. We also select all distinct frequent procedures and diagnoses then assign them consecutive distinct id number as medication id and diagnosis id. In such a way, the constructed tensors will be one-to-one corresponded to the patients, procedures, and diagnoses and will be easier for us to analyze. To achieve a dataset with the format in the table shown above, we group the overall table by distinct (Patient id, Medication id, Diagnosis id) pair the count how many different times they appeared in the data. With this dataset as input, the resulting tensor should be of size 40662 patient * 202 procedures * 316 diagnoses. Since we also want to do mortality prediction with the patient factor matrix, we keep a dictionary with corresponding SUBJECT_ID and the new patient id. We then relate the patient id to the mortality flag achieved from ADMISSIONS. We can then use the resulting patient factor matrix and the tag for each patient (each row represents a patient and the row number is the patient id) to fit into a logistic regression model to do prediction and evaluate the result.

4.2 Implementation Details

We implement the algorithm in Matlab, along with Tensor Toolbox 2.6 package [21] for tensor-related functionalities. To simulate the distributed nature of local hospitals, we use the Parallel Computing toolbox provided by Matlab. For utility prediction, we use the logistic regression model support by Scikit learn in Python. We split the data into train and test with a ratio 8:2.

4.3 Baselines

We use two baselines methods for comparison, CP-ALS and DPFact.
CP-ALS is an existing centralized method supported by Matlab package tensor toolbox. It uses the alternating least square approach to solve the tensor decomposition. DPFact [19] is federated tensor factorization and It uses the same privacy method and has the same privacy guarantee as our method (LocalTF). It uses the Elastic Averaging SGD to approach the optimization problem. It also adds penalty terms to penalize the difference between local feature factor matrices and the global feature factor matrices to minimize the gap between local sites and global server.

4.4 Parameters

We have a lot of input parameters in our algorithm and each of these will greatly affect the results. We will use grid search for those parameters, including quadratic penalty parameter γ, $l_{2,1}$ regularization term μ, learning rate η and number of sites $\mathfrak{T}$. We will fix the rank $\mathfrak{R}$ to 50 in order to catch more features.

4.4.1 $l_{2,1}$ regularization term μ

The regularization term μ regulates the sparsity of the output factors. Since different local sites (which means different ICU units in our dataset) have different sparsity, we will give different values to different local sites. A small μ exerts little effect on the sparsity regularization while a high μ might cancel off too many columns thus jeopardizing the accuracy of the phenotypes. Since the similarity between datasets of our algorithm and DPFact, we incorporate the μ values of DPFact, which is $\mu = [1, 1.8, 3.2, 1.8, 1.5, 0.6]$ for TSICU, SICU, MICU, CSRU, CCU, NICU respectively.

4.4.2 Learning rate η

The learning rate η is also referred to as step size, denoting how fast we "learn" in each step. η conveys a trade-off between time and optimal, where a small η takes too long to finish and a large might result in sub-optimal results and instability. After gird searching through $\eta = [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$, we choose $\eta = 10^{-2}$ as our learning rate.

4.4.3 Number of Sites $\mathfrak{T}$

The number of sites generally affect communication cost, so we compare the communication cost with respect to the number of sites and we find the communication costs increase proportionally with respect to the number of sites. Since we have 6 different ICU units, we pick $\mathfrak{T} = 6$ in our final experiment setting.

4.4.4 Number of Local Updates per Communication b

The Number of Local updates per communication is a very important variable in our method. For a small B value, we have little effect on reducing communication costs. For example, if we choose b=1, then Local SGD has no difference with the Vanilla Distributed SGD and the communication cost is not reduced. For a large b value, it may take too long to local update and converge, and if the convergence is too slow, we will have to communicate more iterations thus increase the communication costs. Overall we will have to find a proper B value so we grid search through $b = [2, 3, 5, 8, 10]$ and we choose $b = 3$ for experiments.

4.5 Results

In this section, we will present the results from our experiments in multiple aspects, including accuracy, communication costs, utility and convergence rate.

4.5.1 Accuracy

We evaluate our accuracy with Root Mean Square Error (RMSE). We set all the parameters with respect to section 5.3 and we pick 6 local sites, each representing a different ICU unit. Since we randomly initialize our matrices, the results vary each, so we test for 5 times for each algorithm and calculate the average and variance of those results

	CP-ALS	DPFact	Local TF	DPfact w/o $l_{2,1}$norm	Local TF w/o $l_{2,1}$norm
Average	1.29164	0.7734	0.7442	1.21252	1.13328
Variance	6.46E-7	2.44E-5	5.32E-6	2.54E-5	5.99E-5

We can see that LocalTF achieves better accuracy compared with the other two methods, especially CP-ALS. CP-ALS has the lowest variance, which means it has the best stability while LocalTF appears to be more stable than DPFact. Overall, LocalTF has the lowest RMSE but ranks the second in the aspect of stability. DPFact has a relatively acceptable RMSE but has poor stability. CP-ALS performs poorly when speaking of accuracy but gives a stable result, probably because CP-ALS does not have regularization and that may jeopardize the RMSE result. In order to compare more directly with CP-ALS, we also test for DPFact and our algorithm without $l_{2,1}$ regularization and see the results. In later sections, we will be talking about convergence and we are also comparing the convergence with the results from CP-ALS, DPFact without $l_{2,1}$ regularization and our algorithm without $l_{2,1}$ regularization. With the results of the methods without $l_{2,1}$ regularization, we can see LocalTF still has the best performance, but the gap between CP-ALS and LocalTF becomes much smaller. From these results, we can say that the $l_{2,1}$ norm plays a crucial role in reducing the RMSE loss, and after removing the $l_{2,1}$ regularization, the three methods are overall comparable when speaking of accuracy, while LocalTFs turns out to be the best. We can see that the variance of DPFact and LocalTF stays at the same level regardless of the $l_{2,1}$ norm and CP-ALS always gives the most stable result.

4.5.2 Communication Costs

In this section, we will be comparing the communication costs of DPFact and LocalTF. Since the accumulated costs should be proportional to the cost of a single worker, to be simple, we will analyze the communication costs between a certain worker and the central server. We measure how much information we will have to communicate, which is the bytes of the **B** and **C**. And the communication costs will be:

	DPFact	LocalTF	Distributed SGD
Total Number of Communication Rounds	25	21	35
Cost Per Iteration	207200	207200	207200
Total Communication Costs (Bytes)	5180000	4351200	7252000

We can see that the two methods have close communication costs, and our method achieves a slightly smaller cost. Since DPFact also reduces its communication costs in its algorithm, we think the result of our LocalTF is acceptable. Compared with other SGD methods, the communication costs of DPFact and LocalTF are much smaller.

4.5.3 Utility

Even though we have evaluated our selected methods with RMSE, it does not mean our tensor factorization method works perfectly because we have to see how much information is "stored" in our tensors. We want to see how predictive can our tensors be, thus we evaluate our methods through utility. We use the mortality prediction task to measure utility, in which we use the result patient factor matrix as the features and we get the mortality target from the MIMIC III database corresponding to each patient in our data. We evaluate the prediction result with the AUC score.

	CP-ALS	DPFact	LocalTF	DPFact w/o l21	LocalTF w/o l21
Average	0.63658	0.62668	0.63144	0.54736	0.5612

We can see that CP-ALS has the highest prediction AUC score, while LocalTF has a relatively high AUC score compared with DPFact. It kind of contradicts our instinct, because CP-ALS has the worst RMSE. It can be explained as CP-ALS do not have the $l_{2,1}$ regularization thus the RMSE could be higher than the other two models with regularization. Overall, LocalTF achieves a better predictive power than the DPFact. After we remove the $l_{2,1}$ norm, we can see that the results our DPFact and LocalTF goes down significantly, and LocalTF still gives a better result than the DPFact. It can be seen that all these methods perform not as good as the CP-ALS in utility aspect and since CP-ALS is an idealized case, it's nearly impossible for LocalTF to exceed the utility of CP-ALS, thus we think our utility result is overall acceptable.

4.5.4 Convergence

In this section, we will be comparing the convergence rate of those three methods. To be fair, we compare the convergence rate of CP-ALS, DPFact and LocalTF without $l_{2,1}$ norm, as CP-ALS does not support $l_{2,1}$ norm.

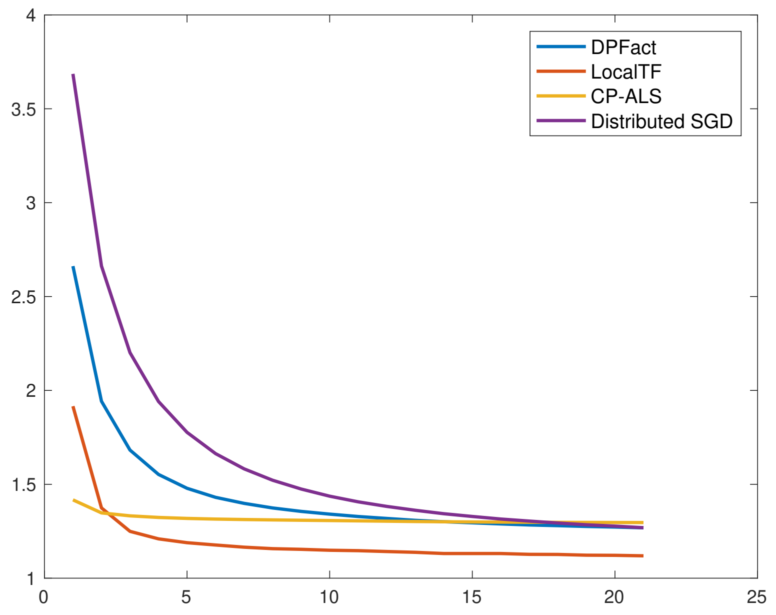

Figure 4.1: Convergence Rate

In this graph, we can see that CP-ALS converges the fastest and LocalTF and DPFact achieve close convergence rate and LocalTF converges slightly faster. We do care about convergence rate because communication costs are also highly related to the number of iterations. Again, though CP-ALS converges in a marvelous speed, it's an idealized method and it gives the best result as perfect as we can. LocalTF achieves a convergence rate at least as good as DPFact, thus we can conclude LocalSGD converges as fast as DPFact, or even slightly faster.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

LocalTF is a useful tensor factorization method that guarantees strict privacy and reduces communication costs. With multiple local sites and a central server, LocalTF can protect the privacy of patients from local sites and also has a great impact on reducing communication costs. As such, we will not worry too much about have too many local sites and wastes too many resources in communication. Our model out-beats DPFact, another well-designed Privacy-Preserving tensor factorization method, in both accuracy and utility aspect. By testing on MIMIC III dataset, we showed that LocalTF is more useful in predictive tasks like mortality prediction. LocalTF also achieved comparable results when compared with CP-ALS, an idealized method, in the aspect of utility test. Thus we can conclude, LocalTF method can be applied in real world applications and will give comfortable results. Overall, LocalTF performs pretty well in communication costs, accuracy, and real world applications.

5.2 Future Work

We still have some future works remaining to be done. First, we want to test LocalTF on some other datasets related to healthcare data. MIMIC III dataset is a comprehensive real world medical dataset, and the results from this dataset should be very representative and applicable to other datasets. However, we are still interested in how our methods apply to other datasets and such experiments will be more comprehensive. We may also test our method with synthetic dataset and see how it performs in such settings and we want to have a good result in synthetic dataset as well. We want to test on synthetic datasets because we may be interested in generalizations such as higher-order (more than 3) tensor factorization and such dataset might be hard to find. In such settings, synthetic dataset might be a proper choice.

Another possible future work is to consider how we can further reduce the communication cost per iteration. Currently, both DPFact and LocalTF work on minimizing the number of iterations we communicate, and few researches have explores how we can minimize the costs for each iteration. If we can reduce the cost each iteration, the communication cost can be further reduced and since this is quite another aspect of the problem, the result might be more optimistic.

Currently, we assume all the workers are honest and moral but in reality, there could be some malicious workers who intentionally upload tampered results to the central server, thus jeopardize the global results. Nowadays, we have many machine learning algorithms that deal with such malicious attack, we want to see if any of these is applicable to LocalSGD to deal with adversarial attacks without greatly affecting our current accuracy and utility.

Bibliography

[1] Tracy D. Gunter and Nicholas P. Terry. The Emergence of National Electronic Health Record Architectures in the United States and Australia: Models, Costs, and Questions. *PMC*, 2005.

[2] Joyce Ho, Joydeep Ghosh, and J. Sun. Marble: High-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2014.

[3] Joyce C. Ho, Joydeep Ghosh, Steve R. Steinhubl, Walter F. Stewart, Joshua C. Denny, Bradley A. Malin, and Jimeng Sun. Limestone: High-throughput candidate phenotype generation via tensor factorization. *Journal of Biomedical Informatics*, 52:199 – 211, 2014. Special Section: Methods in Clinical Research Informatics.

[4] Joyce Ho, Jin-Mann Lin, Brian Gurbaxani, Jimeng Sun, and Joydeep Ghosh. Uncovering medication usage patterns of patients with chronic fatigue syndrome via nonnegative tensor factorization. 01 2015.

[5] Ian Davidson, Sean Gilpin, Owen Carmichael, and Peter Walker. Network discovery via constrained tensor analysis of fmri data. 08 2013.

[6] Cynthia Dwork and Guy N. Rothblum. Concentrated Differential Privacy. *arXiv e-prints*, page arXiv:1603.01887, March 2016.

[7] Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

[8] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009.

[9] Yongwon Jeong. Speaker adaptation in the maximum a posteriori framework based on the probabilistic 2-mode analysis of training models. *EURASIP Journal on Audio, Speech, and Music Processing*, 2013, 12 2013.

[10] Conditions For and Richard A. Harshman. Foundations of the parafac procedure: Models and conditions for an ”explanatory ” multimodal factor analysis by.

[11] Lele Wang, Kun Xie, Thabo Semong, and Huibin Zhou. Missing data recovery based on tensor-cur decomposition. *IEEE Access*, PP:1–1, 11 2017.

[12] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2010.

[13] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3â4):211–407, 2014.

[14] Masooma Iftikhar, Qing Wang, and Yu Lin. Publishing differentially private datasets via stable microaggregation, 2019.

[15] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-SGD: Distributed SGD with Quantization, Sparsification, and Local Computations. *arXiv e-prints*, page arXiv:1906.02367, June 2019.

[16] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[17] Yejin Kim, Jimeng Sun, Hwanjo Yu, and Xiaoqian Jiang. Federated tensor factorization for computational phenotyping. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 887–895, New York, NY, USA, 2017. Association for Computing Machinery.

[18] Sixin Zhang, Anna Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 685–693, Cambridge, MA, USA, 2015. MIT Press.

[19] Jing Ma, Qiuchen Zhang, Jian Lou, Joyce C. Ho, Li Xiong, and Xiaoqian Jiang. Privacy-preserving tensor factorization for collaborative health data analysis. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 1291–1300, New York, NY, USA, 2019. Association for Computing Machinery.

[20] Ingrid Daubechies, Michel Defrise, and Christine Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constrains. *Communications on Pure and Applied Mathematics*, 57, 11 2004.

[21] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015.