An Online and Unified Algorithm for Projection Matrix Vector Multiplication with Application to Empirical Risk Minimization

Lianke Qin, Zhao Song, Lichen Zhang, Danyang Zhuo

Soham Dutta [1] , Aryama Ghosh [1]
Pratyush Patra [1], Rwitobroto Dey [1]

[1]B.Stat 1st Year, Indian Statistical Institute, Kolkata

Outline

Section 1. About the Paper

Introduction

- Online matrix vector multiplication is a fundamental step and bottleneck in many machine learning algorithms.
- It is defined as follows: given a matrix at the pre-processing phase, at each iteration one receives a query vector and needs to form the matrix-vector product (approximately) before observing the next vector.
- (Qin et al., 2023) studies a particular instance of the problem called the online projection matrix vector multiplication. Via a reduction, they show that it suffices to solve the inverse maintanence problem.

What Matrix Multiplication really is?

- Matrix Multiplication forms a major part in many numerical algorithms. Given two matrices A and B of dimensions $m \times n$ and $n \times p$, a simple looping algorithm to compute AB takes time $\Theta(mnp)$ ($\Theta(n^3)$ for $A, B \in M_n(\mathbb{R})$).
- Since long, efforts have been made to optimize matrix multiplication and reduce the value of ω (multiplying two square matrices of dimension n has a time complexity of n^ω). Notable efforts include those by Strassen (1969) ($\omega = \log_2 7$), Coppersmith and Winograd (1990) ($\omega < 2.3755$), Le Gall (2014) ($\omega < 2.3728639$).

What Matrix Multiplication really is?

The fastest known algorithm is due to Williams et al.(2023) which gives a bound of 2.371522 on ω.

Work of Qin et al.,2023

The problem can be described as follows: Let T denote the number of rounds or the number of vectors we will receive and $A \in M_n(\mathbb{R})$. The goal of Online Matrix Vector Multiplication (OMv) (Henzinger et al., 2015; Larsen and Williams, 2017) is in each round we receive a query vector $h_t \in \mathbb{R}^n$ and output the product Ah_t for each round. The key difficulty of this problem is to output the matrix vector multiplication. Otherwise one could batch the vectors into a matrix and apply fast matrix multiplication algorithms in n^ω time for $T = n$ rounds.

Work of Qin et al., 2023

Qin et al. focuses on a particular case i.e. when A is a projection matrix and refers to this as Online Projection Matrix Vector Multiplication (OPMv). Work has been done to speed up OPMv since long. Vaidya, 1989 gives an algorithm that takes $\mathcal{O}(n^{2.5})$ time. Cohen et al.,2019 modifies the run time to n^{ω} where $\omega \sim 2.373$ by combining it with fast rectangular matrix multiplications. Other noteworthy contributions include those by Lee et al.,2019 and Song and Yu, 2021.

Work of Qin et al.,2023

The work can be summarised as follows:

- An unified solution to OPMv is provided i.e. the approach is compatible with both data oblivious approaches (Lee et al.,2019; Song and Yu, 2021) and data dependent approach (Cohen et al., 2019).
- The running time of Lee et al., 2019 has been significantly improved and an unnatural factor of $\log^6(\log(1/\delta))$ has been shaved off.
- A novel sketching-based dynamic projection inverse data structure has been presented which unifies the algorithm in (Cohen et al., 2019; Lee et al., 2019; Song and Yu, 2021).

Notations and Terminology

- $[n]$ denotes the set $\{1, 2, 3, \cdots, n\}$.
- $\mathbb{E}$ denotes the expectation, Var denotes the variance and $\mathbb{P}$ denotes the probability.
- For a vector x, $||x||$ denotes the L_2 norm.
- 0_n denotes the vector of dimension $n \times 1$ and all entries as 0.
- For a positive diagonal matrix W, we denote by $\sqrt{W}$, $W^{-1/2}$ as the matrix with (i, i)-th entry as $\sqrt{W_{i,i}}$ and $1/\sqrt{W_{i,i}}$ respectively.
- I denotes the identity matrix.
- For a matrix A, $||A||$ denotes its spectral norm and $||A||_F$ denotes its Frobenius norm.

Notations and Terminology

- **Sketching Matrix:** A data compression technique where an input matrix A is efficiently represented by a smaller matrix B so that B retains most of the properties of A upto some guaranteed ratio.
- **Unbiased Estimator:** An estimator of a given parameter is said to be unbiased if its expected value is equal to the true value of the parameter.
- L_∞ **norm of a matrix:** Given a matrix A with row sums as $a_1, a_2, \cdots, a_n$, the L_∞ norm of A is given by $\max(a_i)(i \in \{1, 2, \cdots, n\})$.

Section 2. Technical Overview

Technical Overview

Given a projection matrix $P = \sqrt{W} A^T (AWA^T)^{-1} A \sqrt{W}$, it is hard to understand the changes in the form of P due to minor changes in the diagonal matrix W. Instead, one can carefully design a matrix L whose blocks consist of $W^{1/2}$, W^{-1} and A. Using the well-known Schur complement, the inversion L^{-1} contains information for us to "recover" the projection matrix to maintain. By multiplying L^{-1} with a crafted but easy to compute vector v, $L^{-1}v$ will give exactly Ph, the quantity we care about.

Section 3. Dynamic Inverse

Dynamic Inverse

In this section, we provide a detailed discussion of the dynamic inverse maintenance algorithm and its applications to sketching settings.

Our inverse maintenance technology starts from the well-known Schur complement:

Schur Complement

Given four matrices A, B, C, D, we have the following identity assuming that all inverses exist:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix}$$

where

$$\begin{aligned} A_1 &= A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} \\ B_1 &= -A^{-1}B(D - CA^{-1}B)^{-1} \\ C_1 &= -(D - CA^{-1}B)^{-1}CA^{-1} \\ D_1 &= (D - CA^{-1}B)^{-1} \end{aligned}$$

Schur Complement

Schur complement implies that if we carefully design the matrix-to-invert on the left hand side, we can make sure that the target projection matrix, even with sketching matrix and vector is at certain location of the inverse.

Schur Complement

As the simplest example consider

$$L = \begin{bmatrix} U^{-1} & A^T & U^{-\frac{1}{2}} & 0 \\ A & 0 & 0 & 0 \\ 0 & 0 & -I & 0 \\ \left(U^{-\frac{1}{2}}\right)^T & 0 & 0 & -I \end{bmatrix}$$

Schur Complement

Using Schur Complement, L^{-1} can be easily calculated as

$$L^{-1} = \begin{bmatrix} M^{-1} & M^{-1}N & 0 \\ 0 & -I & 0 \\ N^T M^{-1} & N^T M^{-1} & -I \end{bmatrix}$$

where

$$M = \begin{bmatrix} U^{-1} & A^T \\ A & 0 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} U^{-\frac{1}{2}} \\ 0 \end{bmatrix}$$

Schur Complement

If we multiply L^{-1} with a vector in the form of $\begin{bmatrix} 0_{n+d} \\ v \\ v \end{bmatrix}$

we have that

$$L^{-1} \begin{bmatrix} 0_{n+d} \\ v \\ v \end{bmatrix} = \begin{bmatrix} * \\ * \\ \sqrt{U}A^T(AUA^T)^{-1}A\sqrt{U}v \end{bmatrix}$$

Recognize the last entry of the above matrix i.e. $\sqrt{U}A^T(AUA^T)^{-1}A\sqrt{U}v$ to be Pv which is what we wanted.

Woodbury Matrix Identity

The Wodbury Matrix Identity, also known as the Matrix Inversion Lemma or the Sherman-Morrison-Woodbury formula allows easy computation of inverses and solutions to linear equations.
The identity is

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

We use this identity quite a lot in our upcoming algorithm.

The Sketching Matrices

We can also add sketching matrix, or even the vector in the correct location of the L matrix to make sure its inverse provides what we want. For example, for left and right sketch, let $R \in \mathbb{R}^{n \times n}$ denote the batched sketching matrices, then we have

The Sketching Matrices

$$L_{left} = \begin{bmatrix} U^{-1} & A^T & U^{-\frac{1}{2}} & 0 & 0 \\ A & 0 & 0 & 0 & 0 \\ 0 & 0 & -I & 0 & 0 \\ \left(U^{-\frac{1}{2}}\right)^T & 0 & 0 & -I & 0 \\ 0 & 0 & 0 & R & I \end{bmatrix}$$

The Sketching Matrices

$$L_{right} = \begin{bmatrix} U^{-1} & A^T & U^{-\frac{1}{2}} & 0 & 0 \\ A & 0 & 0 & 0 & 0 \\ 0 & 0 & -I & 0 & R^T \\ \left(U^{-\frac{1}{2}}\right)^T & 0 & 0 & -I & R^T \\ 0 & 0 & 0 & 0 & I \end{bmatrix}$$

The Sketching Matrices

Note

$$L_{left}^{-1} \cdot \begin{bmatrix} 0_{n+d} \\ v \\ v \\ 0_n \end{bmatrix} = \begin{bmatrix} * \\ R_t \sqrt{U} A^T (AUA^T)^{-1} A \sqrt{U} v \end{bmatrix}$$

where I_t is a diagonal matrix whose $\frac{n(t-1)}{T}$-th to $\frac{nt}{T}$-th diagonal entries are 1 and other diagonal entries are 0 such that $I_t R = R_t$.

The Sketching Matrices

Similarly,

$$L_{right}^{-1} \cdot \begin{bmatrix} 0_{3n+d} \\ I_t R v \end{bmatrix} = \begin{bmatrix} * \\ \sqrt{U} A^T (AUA^T)^{-1} A \sqrt{U} R_t^T R_t v \end{bmatrix}$$

where I_t is a diagonal matrix whose $\frac{n(t-1)}{T}$-th to $\frac{nt}{T}$-th diagonal entries are 1 and other diagonal entries are 0 such that $I_t R = R_t$.

Section 4. Our Work

Understanding of the Paper

Although, we cannot say that we have understood the paper thoroughly and clearly, we toiled hard working our way through each and every minute detail trying to better our understanding of the concepts and methods used in the paper. Though we say that we have understood the algebra behind the problem, we are still unable to grasp the motivation behind a lot of results and concepts used here. The essential idea or motivation behind the construction of the sketching matrices is still unclear to us. As we read the paper several times, we have often wondered as to how the author came up with such a construction and why is it so effective.

Understanding of the Paper

The algorithms at the end of the paper are well-informed and clear almost in its entirety. However, there are certain scopes of confusion as the author has not explicitly mentioned the use of a and b which he has assumed as the two threshold values. Further, b being a real number in the interval $(0, 1)$, it is quite confusing as to how the order of the matrices X and Y are $(4n + d) \times n^b$ or as to how the order of matrix Q is defined to be $n^b \times n^b$. Though, we understand that a and b are related to the complexity of the code, but we fail to grasp as to how they, as a variable, play a role in the algorithm.

How we went about the project

We have tried to implement the code in Python. In doing so, we have extensively used the `numpy` library in Python. More specifically, we have used the `numpy.linalg` module which is specifically designed to help in calculations involving linear algebra in Python. In doing so, we have written a code that implements Algorithm 1 in the paper (sketching on the left). Algorithm 5 (sketching on the right) can also be implemented in the same way. Algorithm 6 has been implemented within Algorithm 1. We have not been able to implement the corrections of the algorithms which were originally suggested by Song and Yu, 2021 and Lee et al., 2019.

Algorithm

Hi

The Code

The code we have written has been uploaded to the following GitHub link. The code has been commented extensively in order to ease the understanding of the reader.

Bibliography