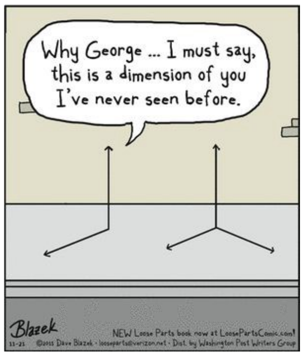
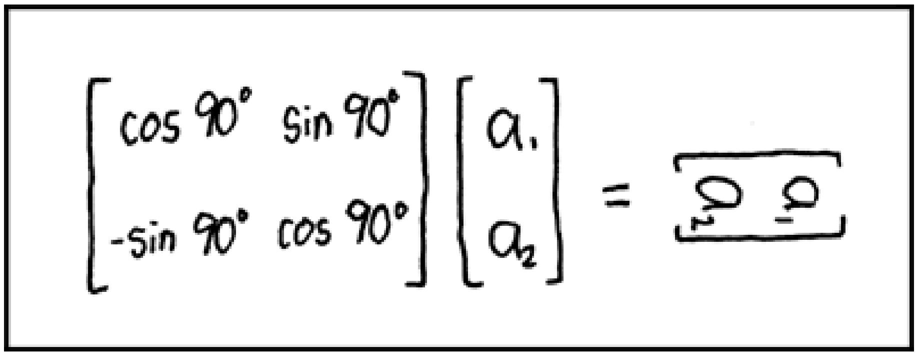
Notes: Computational Linear Algebra


Why George ... I must say, this is a dimension of you I've never seen before.
Blazek
NEW Loose Parts book now at LoosePartsComic.com!



1 Introduction

These notes review the background material from linear algebra and data structures needed to understand data mining. They focus on runtime concepts that you may not have covered in your previous classes, but should hopefully be easy to grasp. You are not required to complete and submit the problems in this packet, but the problems on the (open note) midterm will be very similar, so I strongly recommend you complete all of these problems.

NOTE: If you find any errors in these notes (or any other class material), you can get extra credit by submitting a pull request to github fixing the error.

2 Linear Algebra Basics

Many students struggle in this course with basic linear algebra concepts like the rank of a matrix and eigenvalues. If you would like more background on these topics, I recommend watching the 3blue1brown videos on linear algebra:

`https://www.3blue1brown.com/essence-of-linear-algebra-page`

The eigenvalue video is the most important:

`https://www.youtube.com/watch?v=PFDu9oVAE-g`

3 Big-O/Θ/Ω Notation

You are likely familiar with asymptotic notation from your data structures class, where you used it to measure the time and space complexity of algorithms. In this class, we will also be measuring statistical complexity of algorithms. The statistical complexity will result in formulas a bit more complex than you have likely seen before, and so you will need to be intimately familiar with the formal definitions of asymptotic notation. There are many equivalent definitions, but in this class, we will use the following.

Definition 1. Let f, g be (possibly multivariate) functions from $(\mathbb{R}^+)^d \to \mathbb{R}^+$, where $d > 0$ is an integer that specifies the number of input variables. Then,

1. If $\lim_{\mathbf{x}\to\infty} \frac{f(\mathbf{x})}{g(\mathbf{x})} < \infty$, then we say $f = O(g)$.
2. If $\lim_{\mathbf{x}\to\infty} \frac{f(\mathbf{x})}{g(\mathbf{x})} > 0$, then we say $f = \Omega(g)$.
3. We say that $f = \Theta(g)$ if both $f = O(g)$ and $f = \Omega(g)$.

Intuitively, you should think of O as $\leq$, Ω as $\geq$, and Θ as $=$.

Problem 1. Complete each equation below by adding the symbol O if $f = O(g)$, Ω if $f = \Omega(g)$, or Θ if $f = \Theta(g)$. The first row is completed for you as an example.

f(n)		g(n)
1	$=$	$O(n)$
$3n \log n$	$=$	n^2
1	$=$	$1/n$
$\log_2 n$	$=$	$\log_3 n$
$\log n$	$=$	$\frac{1}{\log n}$
$5 \cdot 10^{30}$	$=$	$\log n$
$\log n$	$=$	$\log(n^2)$
2^n	$=$	3^n
$\frac{1}{n}$	$=$	$\sqrt{\frac{1}{n}}$
$\log n$	$=$	$(\log n)^2$

Problem 2. Complete each equation below by adding the symbol O if $f = O(g)$, Ω if $f = \Omega(g)$, or Θ if $f = \Theta(g)$. If f cannot be related to g using asymptotic notation, draw a slash through the equals sign. The first row is completed for you as an example.

f(a,b,c)		g(a,b,c)
ab	$=$	$\Omega(b)$
a^2b	$=$	ab^2
$a \log b$	$=$	a^b
a^2bc^3	$=$	$a^2b^2c^3$
$\frac{a}{b}$	$=$	$\frac{a}{b^2}$
$\frac{a}{b}$	$=$	$(\frac{a}{b})^2$
a^b	$=$	b^a
a^b	$=$	$(\log a)^c$
a^b	$=$	$(1+c)^a$

The main advantage of asymptotic notation is that it lets us write complex formulas in simpler forms, focusing only on the "most important" parts.

Problem 3. Simplify the following expressions:

1. $O\left(n^3 + 5n^2 \log n + \log n\right)$

2. $O\left(ab^2 + 3a^2b + ab + 10b\right)$

3. $O\left(a + b + c + ab + bc + abc\right)$

4. $O\left(\frac{1}{n} + \frac{1}{n^2}\right)$

5. $O\left(\frac{1}{n} + \frac{1}{nm} + \frac{1}{m}\right)$

4 BLAS/LAPACK

The *Basic Linear Algebra Subprogram* (BLAS) and *Linear Algebra Package* (LAPACK) are the primary low-level tools of computational linear algebra. These high performance libraries are written in Assembly, C, Fortran, and CUDA. All high level programming languages (R, Python, Matlab, Julia, etc.) convert your code into basic BLAS/LAPACK function calls, and rely on these libraries for good performance.

There are many different versions of BLAS/LAPACK depending on:

1. The format of matrices (dense, sparse, diagonal, toplitz, etc.)

2. The computer architecture (Intel vs ARM, 8/16/32/64/128 bit, parallel CPUs, GPUs, clusters, etc.)

In this class, we will be using the PyTorch library, and not directly interfacing with BLAS/LAPACK. PyTorch supports Dense and Sparse formats on both parallel CPU and CUDA architectures.

It is still important to understand the high-level operations supported by BLAS/LAPACK in order to understand the performance of the code you write, but there are a LOT of details that we will not get into here. You can find the details of the BLAS/LAPACK APIs at

`https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top/blas-and-sparse-blas-routines/blas-routines/blas-level-1-routines-and-functions/cblas-asum.html#cblas-asum`

but you will not need to reference this link at all.

4.1 Notation

We will typically use capital letters (A, B, C) to denote matrices, bold lower case letters $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ to denote vectors, and lower case italics letters (n, m, o) to denote scalars.

4.2 Dense vs Sparse Matrices

A sparse matrix is any matrix with a large number of zero entries relative to the total number of entries. There is no strict cut-off here, but it's typical for a sparse matrix to have $< 1\%$ of its entries be zero. The *sparsity* of a matrix is the total number of non-zero entries. We will be using $\text{nnz}(A)$ to denote the sparsity of $A : \mathbb{R}^{m\times n}$.

Problem 4. What is the sparsity of the following matrix?

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \tag{1}$$

There are many data structures for storing sparse matrices, including:

1. Dictionary of keys (DOK),
2. List of lists (LIL),
3. Coordinate list (COO),
4. Compressed sparse row (CSR, CRS or Yale format), and
5. Compressed sparse column (CSC or CCS).

In this class, we will not cover the details of the different formats. Pytorch implements both the COO and CSR formats, and you are responsible for understanding when to use each of these formats.[1] From here on out the notes always assume that a sparse matrix is stored in either COO or CSR format.

4.3 Memory Usage

If A is stored as a sparse matrix, the memory usage is $\Theta(\text{nnz}(A))$. If A is stored as a dense matrix, the memory usage is $\Theta(mn)$.

Problem 5. Prove or provide a counter example. For both sparse and dense matrices, the storage requirements are $O(mn)$.

[1] The documentation is at `https://pytorch.org/docs/stable/sparse.html`.

Problem 6. A computer has 8GB of RAM. What is the largest dense vector of 64bit floating point numbers that can fit in memory? What about 32bit?

4.4 BLAS Level 1

BLAS Level 1 routines are for linear algebra operations that only involve vectors. These include the dot product, vector norms, addition, and scalar multiplication.

These are fully implemented in PyTorch for both sparse and dense vectors. For dense vectors of length n, the runtime of all operations is $\Theta(n)$. For sparse vectors, all of operations of a single vector $\mathbf{x}$ take time $\Theta(\text{nnz}(\mathbf{x}))$ and all operations of two vectors $\mathbf{x}$ and $\mathbf{y}$ take time $\Theta(\text{nnz}(\mathbf{x}) + \text{nnz}(\mathbf{y}))$.

Problem 7. What is the runtime of the following expressions:

1. $5 \cdot \mathbf{x}$

2. $0 \cdot \mathbf{x}$

3. $\mathbf{x}^T\mathbf{y}$

4. $\|\mathbf{x}\|_1$

Problem 8. What is the formula for the following vector norms:

1. $\|\mathbf{x}\|_1 =$

2. $\|\mathbf{x}\|_2 =$

3. $\|\mathbf{x}\|_\infty =$

4. $\|\mathbf{x}\|_p =$

4.5 BLAS Level 2

BLAS Level 2 routines are for vector-matrix operations. The main examples are the products $A\mathbf{x}$ and $\mathbf{x}A$. Up to constant factors, these operations have the same run times as the BLAS level 3 operations, so we will only discuss the level 3 operations below.

4.6 BLAS Level 3

BLAS Level 3 routines are for matrix-matrix operations. Let $A : \mathbb{R}^{m\times n}$ and $B : \mathbb{R}^{n\times o}$. Note that when $o = 1$, then B is a vector, and so BLAS Level 3 operations can be used to implement BLAS Level 2 operations.

For dense matrices, matrix multiplication AB takes time $O(mno)$. This is often referred to as a "cubic" runtime because when the multiplied matrices are square, and thus $m = n = o$, the runtime is $O(n^3)$. There are good algorithms that take potentially much less time than this, with the best known taking time $O(n^{2.373})$. No BLAS implementation that I know of actually implements this algorithm due to the very high constant factor, and Strassen's algorithm with runtime $\Theta(n^{\log_2 7}) \approx \Theta(n^{2.807})$ is the most commonly used algorithm. Currently, we know that matrix multiplication takes at least time $\Omega(n^2 \log n)$, and it is an open problem what the optimal runtime for matrix multiplication is. The following wikipedia link has more details:

`https://en.wikipedia.org/wiki/Matrix_multiplication#Computational_complexity`

For sparse matrices, PyTorch has only limited support for BLAS Level 2/3. In particular, PyTorch allows the first matrix to be either sparse or dense, but the second matrix must be dense. The runtime in the sparse case is $O(\text{nnz}(A)no)$. The output is always a dense tensor.

For both sparse and dense matrices, the runtime of computing the transpose is $\Theta(1)$. The runtime of addition is linear in the memory usage of the vectors.

For the following problems, always assume that the dimensions match so that the expressions are well defined. You will have to specify the dimensions.

Problem 9. What is the runtime of the following expressions? You should report the answer for every possible combination of sparse/dense matrices.

1. $A + B$

2. AB

3. $A\mathbf{x}$

4. $\mathbf{x}A$

Problem 10. Many systems that implement sparse matrices offer only limited support in the same way that PyTorch does. The reason is that the sparsity of the matrix product AB is impossible to predict from the sparsity of just A and B. In this problem, you will prove this fact by example. Your task is to construct $m \times m$ matrices A and B such that

$$\text{nnz}(AB) = O(1) \qquad \text{and} \qquad \text{nnz}(BA) = \Omega(m^2). \tag{2}$$

Problem 11. (Optional) Strassen's algorithm is a famous divide and conquer algorithm for fast matrix multiplication. It is the most commonly implemented algorithm in BLAS libraries because it has both good constant factors and good asymptotic performance. Lookup this algorithm and understand why it works. This is a problem that every computer science undergrad should do at some point in their undergrad career.

4.7 Matrix Chain Ordering Problem (MCOP)

Matrix multiplication is famously not commutative, but it is associative. That is, no matter where you place the parentheses, you will get the same answer. Computationally, however, some choices of parentheses can be much better than others.

Problem 12. Let $A : \mathbb{R}^{m \times n}$, $B : \mathbb{R}^{n \times o}$, $C : \mathbb{R}^{o \times p}$.

1. What is the runtime of computing $(AB)C$?

2. What is the runtime of computing $A(BC)$?

3. Under what conditions would you choose the former parenthesization over the latter?

Problem 13. Recall that in PyTorch, only the first matrix in a matrix multiplication can be sparse, and the second must be dense. It is still possible to compute the product ABC if any one of the matrices is dense and the other two are sparse. For example, if A and B are sparse, then the parenthesization $A(BC)$ can be computed in PyTorch. How should you rewrite the multiplication ABC so that it can be computed when both A and C are sparse?

Hint: You can swap the order of matrix multiplications by using the fact that for any two matrices X and Y, $XY = (Y^T X^T)^T$.

The *Matrix Chain Ordering Problem* (MCOP) is the problem of finding the optimal ordering of parentheses for a given sequence of n matrices. There is a classic dynamic programming solution to this problem that takes time $\Theta(n^3)$.[2] Many more complicated algorithms exist for solving MCOP, the best of which currently takes time $\Theta(n \log n)$. The best lower bound for MCOP is $\Omega(n)$, and it is therefore an open problem whether MCOP can be solved in linear time.

Note: The Θ and Ω notation above is correct. It should be obvious why these values are consistent with each other and do not contradict the definitions of Θ and Ω. If it's not obvious to you, you should get this clarified.

Problem 14. (Optional) Give the dynamic programming solution for MCOP. This is a problem that every computer science undergrad should do at some point in their undergrad career.

Problem 15. Parenthesization becomes critical when the matrices are vectors. What is the optimal way to parenthesize the expressions:

1. $\mathbf{x}^T\mathbf{x}\mathbf{x}^T\mathbf{x}$?

2. $\mathbf{x}\mathbf{x}^T\mathbf{x}\mathbf{x}^T$?

[2]It's important to note that the n here refers to the number of matrices, not the dimensions of any given matrices.

Problem 16. Based on your solution to Problem 15 above, what is the asymptotically fastest way to compute the following expression

$$\prod_{i=1}^{n}(\mathbf{x}\mathbf{x}^T) \tag{3}$$

4.8 LAPACK

LAPACK provides more complex matrix operations than BLAS, and LAPACK functions are typically internally written in terms of BLAS operations. PyTorch has full support for dense matrices, and no support for sparse matrices.

Let $A : \mathbb{R}^{n \times n}$. Then the matrix inverse A^{-1} and matrix determinant $\det(A)$ can both be reduced to matrix multiplication problems. The runtime is therefore the same as matrix multiplication, which we talk about as $O(n^3)$ even though $O(n^{2.373})$ algorithms exist. The runtime of computing the top-k eigenvectors and eigenvalues is $O(n^2 k)$.

NOTE: In the next set of notes, we will see that the pagerank technique is equivalent to computing the top eigenvalue of a matrix. We will then explore several algorithms for computing eigenvalues that have faster but more complex runtime expressions. For this set of notes, however, just assume that computing the top-k eigenvectors takes time $O(n^2 k)$ as stated above.

Problem 17. Let $\mathbf{x} : \mathbb{R}^n$ and $A : \mathbb{R}^{n \times n}$. What is the formula for the following matrix norms? (Note that the $\|\cdot\|_2$ and $\|\cdot\|_p$ norms are defined differently for matrices than vectors.)

1. $\|A\|_2$

2. $\|A\|_p$

3. $\|A\|_F$

Problem 18. Use the formulas above to calculate the runtime of computing the following matrix norms.

1. $\|A\|_2$

2. $\|A\|_F$

We do not have enough tools yet for calculating the runtime of generic $\|\cdot\|_p$ norms, and that's why I didn't ask you to provide that runtime.

Problem 19. Let $A : \mathbb{R}^{m \times n}$, $\mathbf{x} : \mathbb{R}^n$, and $\lambda : \mathbb{R}$. Assume that A and $\mathbf{x}$ are dense. What is the runtime for computing the following expressions?

1. $(AA^T)^{-1}A\mathbf{x}$

2. $A(A^TA)^{-1}$

3. $\|AA^TA\mathbf{x}\|_F^2$

4. $\|(\lambda I + A)\mathbf{x}\mathbf{x}^T\|_F$

   Note that I is the identity matrix. For the expression above to be well defined, m must be equal to n.