COMS 631 - Approximate Matrix Multiplication

Scribe: Raj Gaurav Ballabh Kumar

25th March 2019

1 Introduction

Running times for Matrix Multiplication algorithms that we know are:
$O(n^3)$
$O(n^{2.7})$ //Strassen's Algorithm
$O(n^{2.376})$
$O(n^{2.374})$
$O(n^{2.3728642})$
$O(n^{2.3728639})$

With the current known techniques, we cannot hope to perform this in $O(n^2)$. But can we do an *approximate* matrix multiplication in $O(n^2)$ time? This will be the question we will be answering.

Given: 2 Matrices A and B, where A is a $r \times n$ matrix and B is a $n \times c$ matrix.
Goal: Approximate A $\cdot$ B

Definition 1.1. Frobenius Norm: For the purpose of this approximation we will require to calculate the norm of a Matrix. Here we are going to use the Frobenius Norm of a matrix. For a Matrix M, it's Frobenius norm is represented by $||M||_F$.

$$||M||_F = \sqrt{\sum_i \sum_j M_{ij}^2} \tag{1}$$

So, now the goal is to compute a Matrix C such that $||AB - C||_F$ is small.
We introduce some notation that we will be using along the way,
Given a Matrix A:
A_r^l: l^{th} row of Matrix A
A_c^l: l^{th} column of Matrix A

Then we can talk about the norm of *only* this row or column.
$||A_r^l||_2 = L_2$ Norm of the vector with entries from the l^{th} row $= ||A_r^l||_F$
$||A_c^l||_2 = L_2$ Norm of the vector with entries from the l^{th} column $= ||A_c^l||_F$

L_2 norm is another norm of a vector that we saw in one of the previous lectures.

1.1 Alternative way to view Matrix Multiplication:

So, how do we actually perform Matrix Multiplication?
If we were multiplying A and B, the ij^{th} entry of the resulting matrix would be given by:
$AB_{ij} = A_r^i \cdot B_c^j$
So basically what we are doing is we are multiplying the i^{th} row of A with the j^{th} column of B and then we sum it up,
We instead use a different way of multiplying 2 matrices. Instead of the above method, we multiply the i^{th} *column* of A with the j^{th} *row* of B and then sum it up. Notice that in this case we would be multiplying a $r \times 1$ column vector of A with a $1 \times c$ row vector of B.
Therefore each one of the n multiplications will give us an $r \times c$ matrix.
All of these n matrices will then have to be added together to get the final resultant matrix $A \cdot B$.

Let's show using an example that the method we are using is actually correct. We use the same 2 matrices A and B we defined earlier.

$A \cdot B =$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & a_{r3} & \dots & a_{rn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1c} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2c} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & b_{n3} & \dots & b_{nc} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & \dots & a_{11}b_{1c} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & \dots & a_{21}b_{1c} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{r1}b_{11} & a_{r1}b_{12} & a_{r1}b_{13} & \dots & a_{r1}b_{1c} \end{bmatrix} +$$

$$\begin{bmatrix} a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} & \dots & a_{12}b_{2c} \\ a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} & \dots & a_{22}b_{2c} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{r2}b_{21} & a_{r2}b_{22} & a_{r2}b_{23} & \dots & a_{r2}b_{2c} \end{bmatrix} + \dots +$$

$$\begin{bmatrix} a_{1n}b_{n1} & a_{1n}b_{n2} & a_{1n}b_{n3} & \dots & a_{1n}b_{nc} \\ a_{2n}b_{n1} & a_{2n}b_{n2} & a_{2n}b_{n3} & \dots & a_{2n}b_{nc} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{rn}b_{n1} & a_{rn}b_{n2} & a_{rn}b_{n3} & \dots & a_{rn}b_{nc} \end{bmatrix}$$

If we sum up all of the above n matrices we will in fact see that each ij^{th} entry of the resulting matrix will be equal to the corresponding ij^{th} entry in the matrix obtained by performing $A \cdot B$ the normal way.
Hence our revised method for calculating matrix product is correct as well. This is the key idea in this algorithm which we present below.

1.2 Key Idea:

$$A \cdot B = \sum_{l=1}^{n} A_c^l B_r^l \tag{2}$$

So now we have n matrices. We need to sum them up. Each individual product takes $O(rc)$ time. Summing them is going to take $O(nrc)$ time. This is the same time that it takes to exactly multiply 2 matrices. So we are not really doing any better.

However, what we can do is that, instead of adding all of the n matrices to get the final matrix, we randomly select k matrices and sum them up, where $k < n$, thus taking only $O(krc)$ time.

So now we have the following 3 questions that we have to answer:
- How do we decide k
- Once k is decided, how do we decide *which* k out of the n matrices do we add
- How good of an approximation does this give

1.3 A simpler problem:

Before we move onto the original problem, let us try and think about a simpler problem instead.

Given: Bunch of numbers $a_1, a_2, a_3, \ldots, a_n$.
Goal: Compute the sum of these numbers.

- Can we use the sampling approach that we discussed earlier in order to solve this problem as well?
- This is our initial algorithm

1.3.1 Algorithm 1:

1. `sum` $= 0$
2. for i $= 1$ to k
3. uniformly at random pick $s \in \{1, 2, 3, \ldots, n\}$
4. `sum` $=$ `sum` $+ a_s$
5. `output` $= \dfrac{\texttt{sum} \cdot n}{k}$
6. `return output`

1.3.2 Analyzing Algorithm 1:

Let X_i be a RV.
where X_i is the value of a_s in the i^{th} iteration.

$\therefore \texttt{sum} = \sum_{i=1}^{k} X_i$
$\therefore E[\texttt{sum}] = E[\sum_{i=1}^{k} X_i] = \sum_{i=1}^{k} E[X_i]$

$$E[X_i] = Pr[X_i = a_1] \cdot a_1 + Pr[X_i = a_2] \cdot a_2 + \ldots + Pr[X_i = a_n] \cdot a_n$$
$$= \frac{a_1}{n} + \frac{a_2}{n} + \ldots + \frac{a_n}{n}$$
$$= \frac{\sum_{i=1}^{n} a_i}{n}$$

$$\therefore E[\texttt{sum}] = k \cdot \frac{\sum_{i=1}^{n} a_i}{n}$$

$$\therefore E[\texttt{output}] = \frac{n}{k} \cdot k \cdot \frac{\sum_{i=1}^{n} a_i}{n} = \sum_{i=1}^{n} a_i$$

The way that we have been designing algorithms so far has been as follows:
- We want to bound this number:
  Pr(output of our algorithm is close to the true value)
- The way that we have been doing this is:
  - we set up an algorithm such that its expected value is the true value
  - then we try and bound the Pr(output of our algorithm is close to the expected value)

However this technique is not going to work in this case because the variance in this case is going to be very large. Consider the case where all elements are 0 except the last one which is 1 billion or something. If the algorithm never picks the last element, your output is going to be 0.

One way to get around this problem is by changing the sampling technique. Instead of doing uniform sampling, let us sample the numbers based on the magnitude of those numbers, i.e. the larger number gets picked with a higher probability and a smaller number gets picked with smaller probability.

1.3.3 Algorithm 2:

So suppose we have been somehow given a distribution $P =< p_1, p_2, p_3, \ldots, p_n >$ over {1,2,3, …, n}, this would imply that 1 is picked with probability p_1, 2 with probability p_2 and so on.

Now we modify our algorithm as follows:

1. $\texttt{sum} = 0$
2. for i = 1 to k
3. pick $s \in \{1, 2, 3, \ldots, n\}$ as per the distribution P
4. $\texttt{sum} = \texttt{sum} + \frac{a_s}{p_s}$
5. $\texttt{output} = \frac{\texttt{sum}}{k}$
6. return output

1.3.4 Analyzing Algorithm 2:

Let X_i be the value of $\frac{a_s}{p_s}$ during the i^{th} iteration.
$\texttt{sum} = \sum_{i=1}^{k} X_i$
$E[\texttt{sum}] = E[\sum_{i=1}^{k} X_i] = \sum_{i=1}^{k} E[X_i]$

$$E[X_i] = Pr\left[X_i = \frac{a_1}{p_1}\right] \cdot \frac{a_1}{p_1} + Pr\left[X_i = \frac{a_2}{p_2}\right] \cdot \frac{a_2}{p_2} + \ldots + Pr\left[X_i = \frac{a_n}{p_n}\right] \cdot \frac{a_n}{p_n}$$
$$= p_1 \cdot \frac{a_1}{p_1} + p_2 \cdot \frac{a_2}{p_2} + \ldots + p_n \cdot \frac{a_n}{p_n}$$
$$= \sum_{i=1}^{n} a_i$$

$E[\texttt{sum}] = \sum_{i=1}^{k} \sum_{i=1}^{n} a_i = k \cdot \sum_{i=1}^{n} a_i$
$E[\texttt{output}] = \frac{\texttt{sum}}{k} = \frac{k \cdot \sum_{i=1}^{n} a_i}{k} = \sum_{i=1}^{n} a_i$

1.3.5 How to find P?

So now the question is how do we get hold of a good probability distribution P? Try this:

$$P = \left\langle \frac{a_1}{\sum_{i=1}^{n} a_i}, \frac{a_2}{\sum_{i=1}^{n} a_i}, \ldots, \frac{a_n}{\sum_{i=1}^{n} a_i} \right\rangle$$

The variance of this algorithm is going to be 0. This is because if you look at our algorithm, every time we pick an s, we are adding $\frac{a_s}{p_s}$ to the sum. The value of p_s in the distribution labelled P is just $\frac{a_s}{\sum_{i=1}^{n} a_i}$. Hence $\frac{a_s}{p_s}$ is just going to be $\sum_{i=1}^{n} a_i$, which is a fixed quantity. There is no randomness there.

So this distribution works perfectly! The only issue is that in order to get this distribution, we need to calculate $\sum_{i=1}^{n} a_i$, which is what we set out to do in the first place. But the idea that we can draw from here is that the kind of answer that our algorithm produces depends upon the type of distribution P that we have with us. Now we carry over the same concept to our original problem of Matrix addition.

The entire process can be described in brief as follows:
Sample k numbers $\rightarrow$ Scale them $\rightarrow$ Sum them up $\rightarrow$ Divide by k to take the average

Sampling the numbers in the first step is *the* main step in this algorithm. If we can come up with a good distribution, then our problem is solved. So let us now see how we can come up with some good distribution in the case of Matrix addition.

2 Carrying over the idea to Matrices:

Given 2 Matrices A and B, where A is a $r \times n$ matrix and B is a $n \times c$ matrix. From equation 2 we have:

$$A \cdot B = \sum_{l=1}^{n} A_c^l B_r^l = A_c^1 B_r^1 + A_c^2 B_r^2 + A_c^3 B_r^3 + \ldots + A_c^n B_r^n \tag{3}$$

So now we have to sum up n matrices instead of n numbers. Using the same analogy described in the previous section, we want to sample some k matrices from these n matrices so that we can add and then scale the sum to get our final sum.

We need to figure out a distribution and for this we will be using the Frobenius norm that we described in equation 1.
Let F_i represent the Frobenius norm of the i^{th} sum in equation 3.

$$F_i = ||A_c^i B_r^i||_F \tag{4}$$

Then the probability distribution that we use is:

$$P_i = \frac{F_i}{\sum_{i=1}^{n} F_i} \tag{5}$$

Now how much time does computing this distribution P take?
The numerator can be calculated in $O(rc)$ time. The denominator takes $O(nrc)$ time. But recall that we can perform the exact matrix multiplication in $O(nrc)$ time. So this doesn't really help us much. Let us look at the summation of denominator and see if we can come up with a better bound on it.

Consider 1 particular matrix.

$$A_c^1 B_r^1 = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{r1} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & \ldots & b_{1c} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & \ldots & a_{11}b_{1c} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & \ldots & a_{21}b_{1c} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{r1}b_{11} & a_{r1}b_{12} & a_{r1}b_{13} & \ldots & a_{r1}b_{1c} \end{bmatrix}$$

$||A_c^1 B_r^1||_F$

$= (a_{11}b_{11})^2 + (a_{11}b_{12})^2 + (a_{11}b_{13})^2 + (a_{11}b_{14})^2 + \ldots + (a_{11}b_{1c})^2 +$ //First row
$(a_{21}b_{11})^2 + (a_{21}b_{12})^2 + (a_{21}b_{13})^2 + (a_{21}b_{14})^2 + \ldots + (a_{21}b_{1c})^2 +$ //Second row
$\vdots$
$(a_{r1}b_{11})^2 + (a_{r1}b_{12})^2 + (a_{r1}b_{13})^2 + (a_{r1}b_{14})^2 + \ldots + (a_{r1}b_{1c})^2 +$ //r^{th} row

$= a_{11}^2(b_{11}^2 + b_{12}^2 + b_{13}^2 + \ldots + b_{1c}^2) +$
$a_{21}^2(b_{11}^2 + b_{12}^2 + b_{13}^2 + \ldots + b_{1c}^2) +$
$\vdots$
$a_{r1}^2(b_{11}^2 + b_{12}^2 + b_{13}^2 + \ldots + b_{1c}^2)$

$= (a_{11}^2 + a_{21}^2 + \ldots + a_{r1}^2)(b_{11}^2 + b_{12}^2 + b_{13}^2 + \ldots + b_{1c}^2)$

$= ||A_c^1||_2^2 \; ||B_r^1||_2^2$

where $||A_c^1||_2^2$ is the square of the L_2 norm of A_c^1. Same for B_r^1.

Note that now we can calculate the L_2 norm for A_c^1 in time $O(r)$ and for B_r^1 in $O(c)$, bringing the total time to calculate $||A_c^1 B_r^1||_F$ to $O(r+c)$. For n such matrices, the time now becomes $O(n(r+c))$. This is a better estimate than our earlier bound of $O(nrc)$.

So this is how we define our probability distribution $P =< p_1, p_2, \ldots, p_n >$ over $\{1, 2, 3, \ldots, n\}$ now:

$$p_l = \frac{||A_c^l||_2 \; ||B_r^l||_2}{\sum_{i=1}^{n} ||A_c^i||_2 \; ||B_r^i||_2} \tag{6}$$

Hence now we have a method to compute our probability distribution P specified in equation 5 and 6 in time $O(n(r+c))$.

3 Algorithm

Now we state our algorithm that we will be using to compute the approximate matrix multiplication:

3.1 Algorithm a:

Input: 2 Matrices A and B, where A is a $r \times n$ matrix and B is a $n \times c$ matrix. A probability distribution $P =< p_1, p_2, p_3, \ldots, p_n >$ computed as in equation 6.

1. Pick l as per distribution P
2. Output $G = (A_c^l \cdot B_r^l) \cdot \dfrac{1}{p_l}$

Note: A_c^l is a $r \times 1$ matrix and B_r^l is a $1 \times c$ matrix. Their product is a $r \times c$ matrix, every entry of which is scaled by $\frac{1}{p_l}$. This gives us the matrix G. The expectation is that this matrix is going to be the same or atleast kind of the same as the matrix obtained by $A \cdot B$.

3.2 Claim:

If we fix i, j, then $E[G_{ij}] = AB_{ij}$. What this means is that the expected value of each ij^{th} entry in the matrix G is going to be the same as the corresponding ij^{th} entry in the matrix obtained by performing $A \cdot B$.

Proof:

$$E[G_{ij}] = Pr[l=1] \cdot \left[\frac{A_c^1 \cdot B_r^1}{p_1}\right]_{ij} + Pr[l=2] \cdot \left[\frac{A_c^2 \cdot B_r^2}{p_2}\right]_{ij} + \ldots + Pr[l=n] \cdot \left[\frac{A_c^n \cdot B_r^n}{p_n}\right]_{ij}$$

$$= p_1 \cdot \left[\frac{A_c^1 \cdot B_r^1}{p_1}\right]_{ij} + p_2 \cdot \left[\frac{A_c^2 \cdot B_r^2}{p_2}\right]_{ij} + \ldots + p_n \cdot \left[\frac{A_c^n \cdot B_r^n}{p_n}\right]_{ij}$$

$$= [A_c^1 \cdot B_r^1]_{ij} + [A_c^2 \cdot B_r^2]_{ij} + \ldots + [A_c^n \cdot B_r^n]_{ij}$$

$$= [A \cdot B]_{ij}$$

The notation probably need some explaining.
G_{ij} - ij^{th} entry in the matrix G
$\left[\frac{A_c^1 \cdot B_r^1}{p_1}\right]_{ij}$ - ij^{th} entry in the matrix obtained by multiplying $A_c^1 \cdot B_r^1$ and multiplying each entry by p_1

3.3 Calculating Variance:

Now let us try and calculate the variance. In Section 1.3.2 we saw that it was not possible to bound the variance. but can we do it here?

$$Var[G_{ij}] = E[(G_{ij})^2] - E^2[G_{ij}] \leq E[(G_{ij})^2] \tag{7}$$

$$E[(G_{ij})^2] = Pr[l=1] \cdot \left[\left[\frac{A_c^1 \cdot B_r^1}{p_1}\right]_{ij}\right]^2 + Pr[l=2] \cdot \left[\left[\frac{A_c^2 \cdot B_r^2}{p_2}\right]_{ij}\right]^2 + \ldots + Pr[l=n] \cdot \left[\left[\frac{A_c^n \cdot B_r^n}{p_n}\right]_{ij}\right]^2$$

$$= p_1 \cdot \left[\left[\frac{A_c^1 \cdot B_r^1}{p_1}\right]_{ij}\right]^2 + p_2 \cdot \left[\left[\frac{A_c^2 \cdot B_r^2}{p_2}\right]_{ij}\right]^2 + \ldots + p_n \cdot \left[\left[\frac{A_c^n \cdot B_r^n}{p_n}\right]_{ij}\right]^2$$

$$= \frac{1}{p_1} \cdot [[A_c^1 \cdot B_r^1]_{ij}]^2 + \frac{1}{p_2} \cdot [[A_c^2 \cdot B_r^2]_{ij}]^2 + \ldots + \frac{1}{p_n} \cdot [[A_c^n \cdot B_r^n]_{ij}]^2$$

$$= \frac{1}{p_1} \cdot [A_{i1} \cdot B_{1j}]^2 + \frac{1}{p_2} \cdot [A_{i2} \cdot B_{2j}]^2 + \ldots + \frac{1}{p_n} \cdot [A_{in} \cdot B_{1n}]^2$$

$$= \sum_{l=1}^{n} \frac{A_{il}^2 B_{lj}^2}{p_l}$$

$$\therefore E[(G_{ij})^2] = \sum_{l=1}^{n} \frac{A_{il}^2 B_{lj}^2}{p_l} \tag{8}$$

Here A_{il} is the element at the i^{th} row and l^{th} column in the matrix A.

From equation 7 and 8 we get,

$$Var[G_{ij}] \leq \sum_{l=1}^{n} \frac{A_{il}^2 B_{lj}^2}{p_l} \tag{9}$$

3.4 Algorithm b:

1. Run algorithm a specified in Section 3.1 k times
2. Let $G_1, G_2, \ldots, G_k$ be the k matrices returned
3. $\texttt{H} = \frac{\sum_{i=1}^{k} G_i}{k}$
4. return `H`

From equation 9 and algorithm b step 3, we get,

$$Var[H_{ij}] \leq \frac{1}{k} \cdot \sum_{l=1}^{n} \frac{A_{il}^2 B_{lj}^2}{p_l} \tag{10}$$

3.4.1 Analyzing Algorithm b:

Let us look at $E[||H - AB||_F^2]$. We are representing the matrix resulting from $A \cdot B$ by AB. So AB_{ij} means the ij^{th} entry in $A \cdot B$.

$E[||H - AB||_F^2]$

$= E[\sum_{i=1}^{r} \sum_{j=1}^{c} [(H - AB)_{ij}]^2]$ //From equation 1

$= E[\sum_{i=1}^{r} \sum_{j=1}^{c} [(H_{ij} - AB_{ij})]^2]$

$= \sum_{i=1}^{r} \sum_{j=1}^{c} E[(H_{ij} - AB_{ij})^2]$ //By linearity of expectations

$= \sum_{i=1}^{r} \sum_{j=1}^{c} E[(H_{ij} - E[H_{ij}])^2]$ //From claim 3.2 and our definition of H in //Algorithm b, step 3.

$= \sum_{i=1}^{r} \sum_{j=1}^{c} Var(H_{ij})$ //From definition of Variance

$$\therefore E[||H - AB||_F^2] = \sum_{i=1}^{r} \sum_{j=1}^{c} Var(H_{ij}) \tag{11}$$

From equations 10 and 11 we can now write:

$E[||H - AB||_F^2]$

$\leq \frac{1}{k} \cdot \sum_{i=1}^{r} \sum_{j=1}^{c} \sum_{l=1}^{n} \frac{A_{il}^2 B_{lj}^2}{p_l}$

$= \frac{1}{k} \cdot \sum_{l=1}^{n} \frac{1}{p_l} \sum_{i=1}^{r} \sum_{j=1}^{c} A_{il}^2 B_{lj}^2$

$= \frac{1}{k} \cdot \sum_{l=1}^{n} \frac{1}{p_l} \sum_{i=1}^{r} A_{il}^2 \sum_{j=1}^{c} B_{lj}^2$

$= \frac{1}{k} \cdot \sum_{l=1}^{n} \frac{1}{p_l} \sum_{i=1}^{r} A_{il}^2 \; ||B_r^l||_2^2$ //From our definition of $||B_r^l||_2$ on page 1

$= \frac{1}{k} \cdot \sum_{l=1}^{n} \frac{1}{p_l} \; ||B_r^l||_2^2 \sum_{i=1}^{r} A_{il}^2$

$= \frac{1}{k} \cdot \sum_{l=1}^{n} \frac{1}{p_l} \; ||B_r^l||_2^2 \; ||A_c^l||_2^2$ //From our definition of $||A_c^l||_2$ on page 1

$= \frac{1}{k} \cdot \sum_{l=1}^{n} \frac{\sum_{i=1}^{n} ||A_c^i||_2 \; ||B_r^i||_2}{||A_c^l||_2 \; ||B_r^l||_2} \; ||B_r^l||_2^2 \; ||A_c^l||_2^2$ //From 6

$= \frac{1}{k} \cdot \sum_{l=1}^{n} \left(\sum_{i=1}^{n} ||A_c^i||_2 \; ||B_r^i||_2 \right) \; ||A_c^l||_2 \; ||B_r^l||_2$

$= \frac{1}{k} \cdot \left(\sum_{i=1}^{n} ||A_c^i||_2 \; ||B_r^i||_2 \right) \sum_{l=1}^{n} \; ||A_c^l||_2 \; ||B_r^l||_2$

$= \frac{1}{k} \cdot \left(\sum_{i=1}^{n} ||A_c^i||_2 \; ||B_r^i||_2 \right)^2$

$\leq \frac{1}{k} \sum_{i=1}^{n} ||A_c^i||_2^2 \cdot \sum_{i=1}^{n} ||B_r^i||_2^2$ //Cauchy Schwartz Ineq: $(\sum a_i b_i)^2 \leq \sum a_i^2 \sum b_i^2$

$$= \frac{1}{k} ||A||_F^2 \, ||B||_F^2$$

$$\therefore E[||H - AB||_F^2] \leq \frac{1}{k} ||A||_F^2 \, ||B||_F^2 \tag{12}$$

3.4.2 Bounding the error probability:

Now let us try and bound the probability that the result obtained by the Algorithm b is greater than some fixed quantity i.e. we want to bound the probability $Pr[\ ||H-AB||_F > \epsilon \cdot ||A||_F \cdot ||B||_F]$. Ideally we would have wanted to bound this probability instead: $Pr[\ ||H - AB||_F > \epsilon \cdot ||A \cdot B||_F]$. But we do not know how to do this. Hence we will have to settle for the first one instead. $||A||_F \cdot ||B||_F$ and $||A \cdot B||_F$ can be related by using the following inequality:

$$||A \cdot B||_F \leq ||A||_F \cdot ||B||_F$$

Hence the first bound would work, it's just that it won't give us the tight bound we would have hoped to get.

Using Markov's Inequality we get:

$$Pr\Big[\ ||H - AB||_F > \epsilon \cdot ||A||_F \cdot ||B||_F\Big] \leq \frac{E[\ ||H - AB||_F]}{\epsilon \cdot ||A||_F \cdot ||B||_F} \tag{13}$$

Now we want to use equation 12 and 13 to get some bound. But notice that in equation 12 we have $E[||H - AB||_F^2]$ whereas in equation 13 we have $E[\ ||H - AB||_F]$.

To find the relation between the expected value of a RV and the expected value of the square of the RV, we use Jensen's inequality:

$$E^2[X] \leq E[X^2] \tag{14}$$

From 12 and 14, we get

$$\Big(E[||H - AB||_F]\Big)^2 \leq \frac{1}{k} ||A||_F^2 \, ||B||_F^2$$

$$\therefore E[||H - AB||_F] \leq \frac{1}{\sqrt{k}} ||A||_F \, ||B||_F$$

From equation 13, we get

$$Pr\Big[\ ||H - AB||_F > \epsilon \cdot ||A||_F \cdot ||B||_F\Big] \leq \frac{1}{\epsilon \cdot ||A||_F \cdot ||B||_F} \cdot \frac{1}{\sqrt{k}} ||A||_F \, ||B||_F$$

$$\therefore Pr\Big[\ ||H - AB||_F > \epsilon \cdot ||A||_F \cdot ||B||_F\Big] \leq \frac{1}{\epsilon} \cdot \frac{1}{\sqrt{k}}$$

$$\therefore Pr\Big[\ ||H - AB||_F > \epsilon \cdot ||A||_F \cdot ||B||_F\Big] \leq \frac{1}{\epsilon\sqrt{k}} \tag{15}$$

In order for this probability to be small, we want:
$Pr\Big[\ ||H - AB||_F > \epsilon \cdot ||A||_F \cdot ||B||_F\Big] \leq \delta$
$\implies \dfrac{1}{\epsilon\sqrt{k}} \leq \delta$
$\implies k \geq \dfrac{1}{\epsilon^2\delta^2}$

$$\therefore k \geq \frac{1}{\epsilon^2\delta^2} \tag{16}$$

where
- k was the number specified in step 1 of Algo b specified in Section 3.4
- δ is something small like 0.0001.

Time taken to run algorithm a: $O(rc)$
Repeating k times.
$\therefore$ Time taken to run algorithm b: $O(krc)$
Total time taken = Time taken to compute probability distribution P + Time taken to run algorithm $b = O(krc) + O(n(r + c))$

3.4.3 Tuning the (ϵ, δ) parameters:

ϵ is the approximation error and δ is the probability error. Typically we want δ to be very small (think 10^{-6}) since it represents the probability that our algorithm will fail to produce a result such that $[\ ||H - AB||_F < \epsilon \cdot ||A||_F \cdot ||B||_F]$. So if the denominator contains δ^2, the resulting value of k computed as per equation 16 might be very large, in fact probably larger than n. Recall that the exact matrix multiplication took $O(nrc)$ time and the idea was to do *approximate* matrix multiplication in $O(krc)$ time where $k < n$. So if k becomes greater than n, then there is no point of doing whatever it is we are doing.

We have seen some similar kinds of problems before, where in order to make error probability arbitrarily small, we first get a bound with a fixed error probability, like $\delta = 1/3$, and then repeat the algorithm multiple times. If we fix $\delta = 1/3$, from equation 16 we get $k \geq O(1/\epsilon^2)$. We cannot really hope to avoid the dependency on ϵ because as we reduce the approximation error, we should be prepared to spend more time. In the earlier problems, we had to compute the output of a function. We repeated the algorithm l times and selected the output that appeared the majority of the times as our final answer. By taking the value of l to be $1/\log\delta$, we were able to reduce the error probability to some arbitrary value of δ.

If we were to carry over the same technique to our case, we would run into issues. Suppose we ran Algorithm b l times. Let the output of each of those l runs be $H_1, H_2, \ldots H_l$. What do we even mean by taking the *majority* of these l matrices? It does not make any sense. Recall that when say that we are going

to *approximate* a matrix, what we really mean is that we are creating a new matrix whose *Frobenius Norm* is close to the original matrix.

3.5 Algorithm c:

Run algorithm b l times
Let each of the l outputs be $H_1, H_2, H_3, \ldots, H_l$

1. for i = 1 to l
2. for j = 1 to l
3. compute $||H_i - H_j||_F$
4. check if $||H_i - H_j||_F < \epsilon \cdot ||A||_F \cdot ||B||_F$
5. pick H_s for which this check succeeds $\geq l/2$ times
6. output H_s

3.5.1 Why does this work:

What is δ? It says that if I run my algorithm b, the probability that for the output matrix H, [$||H - AB||_F > \epsilon \cdot ||A||_F \cdot ||B||_F$], is δ. In algorithm c, I have fixed my δ to be 1/3 (by choosing a fixed value of k in equation 16). This means, on an average, $l \cdot (2/3)$ of the output H_i's are going to have an error that is lesser than $\epsilon \cdot ||A||_F \cdot ||B||_F$]. If 2/3 of H_i's satisfy the property, then surely 1/2 of the H_i's are also going to satisfy it. So at least half of the matrices from H_1 to H_l are going to be good (error lesser than $\epsilon \cdot ||A||_F \cdot ||B||_F$]).

If the check in step 4 is satisfied for some particular H_i for greater than $l/2$ times, it means that we have succeeded in finding one of those "good" matrices. That is enough for us and we can return that matrix, H_s as the final answer to our algorithm c.

3.5.2 Time taken:

Algorithm b takes $O(krc + (r + c)n)$ time where $k = O(1/\epsilon^2)$
In algorithm c, we are repeating this $l = \log 1/\delta$ times. The pair wise comparisons in steps 1-6 take $O(\log^2 1/\delta)$ time.

$\therefore$ Total Time Taken $= O\Big(\Big(\frac{1}{\epsilon^2} \cdot rc + (r + c)n\Big) \log \frac{1}{\delta} + rc \log^2 \frac{1}{\delta}\Big)$
If $r = c = n$
Time $= O\Big(\Big(\frac{1}{\epsilon^2} \cdot n^2 + n^2\Big) \log \frac{1}{\delta} + n^2 \log^2 \frac{1}{\delta}\Big) = O\Big(\frac{1}{\epsilon^2} \cdot n^2 \log^2 \frac{1}{\delta}\Big)$
This was assuming failure probability $\delta \leq 1/3$.
If you want to generalize it to δ, $k = O\Big(\frac{1}{\epsilon^2} \log \frac{1}{\delta}\Big)$