Freivald's Algorithm

Freivald's algorithm is a simple algorithm that beautifully demonstrates the motivation behind using probabilistic algorithms. It comes under the umbrella of **Monte Carlo** algorithms, as it outputs an answer, which is correct with a certain probability, but has a fixed running time.

Problem

In this problem, we deal with 3 $n \times n$ matrices, which we can refer to as A, B, C.

Given these three matrices, we are required to find out if

$$A \times B = C$$

Simple Method

The simplest way to verify this condition is to multiply the two matrices A, B and then verifying if their product matches with the matrix C.

- If we use the simple matrix multiplication algorithm, the time complexity of this method will be $O(n^3)$.
- However, we can make use of faster matrix multiplication algorithms such as Strassen's matrix multiplication algorithm which has time complexity $O(n^{2.81})$, reducing the time complexity. (There have been breakthroughs, and the best known matrix multiplication algorithm has a time complexity of $O(n^{2.3728})$

However, this time complexity might be too slow in certain applications, and we must figure out methods that can do the verification with much better time complexity.

Probabilistic Algorithm

Instead of ensuring the output we get is 100% correct, we end up going for an algorithm which has a much better time complexity $O(n^2)$, at the cost of the risk of the algorithm failing with a non-zero probability. Hence, we'll go over how Freivald's algorithm works

The algorithm we'll use has two main properties

- If $A \times B = C$, then the probability with which we'll say that the condition satisfies is 1
- If $A \times B \neq C$, then the probability with which we'll say that the condition satisfies is $\leq \frac{1}{2}$

Freivald's Algorithm

For demonstration, we'll make use of matrices with only binary numbers $\{0, 1\}$.

The algorithm is as follows

- First, we pick a random binary vector r

$$r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}$$

- The probability that $r_i = 1$ is $\frac{1}{2}$, for all values of i. Therefore, every binary column vector of size n is equiprobable of being selected/generated.
- Next we'll use this matrix to perform a verification whether the condition $A \times B = C$ is satisfied.

  We know that if $AB = C$, this implies that $ABr = Cr$.

  Therefore, if we verify if $ABr = Cr$, we are indirectly verifying if $AB = C$ with some probability. If we find out that $ABr \neq Cr$, then we can output that the condition isn't satisfied with 100% confidence. However, if the condition $ABr = Cr$ is satisfied, it increases the probability of the initial condition being true, although it doesn't guarantee it.

  One may feel that computing ABr and Cr provides no improvement over just performing normal matrix multiplication. However, one can easily see that multiplying a $n \times n$ matrix with $1 \times n$ matrix can be done in $O(n^2)$ time.

$$Br = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}$$

  Computing Br takes $O(n^2)$ time as it involves computing an $n \times n$ matrix by $1 \times n$ matrix. Similarly, computing $A(Br)$ also takes $O(n^2)$ time, and Cr has the same time complexity to compute too.

  Therefore, checking if $A(Br) = Cr$ takes $O(n^2)$ time.

  Hence, we end up doing 3 matrix multiplications with a column vector which take $O(n^2)$ time each, and performing an equality check of two column vectors, which takes $O(n)$ time.

Proving that if $AB \neq C$, then $Pr[ABr \neq C] \geq \frac{1}{2}$

We'll define the difference matrix D as

$$D = AB - C$$

If $D \neq 0$,(Null matrix) then we know the condition is not satisfied. Therefore, finding the probability that $AB \neq C \implies ABr \neq Cr$ is equivalent to finding the probability of $Dr = 0$

If $D \neq 0$, we know that $\exists\ i, j$ such that $d_{ij} \neq 0$.

We'll find a column vector v such that $Dv \neq 0$

$$Dv = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & \dots & d_{ij} & \dots & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ \vdots \\ v_j \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ (Dv)_i \\ \vdots \\ 0 \end{bmatrix} \neq 0$$

Now, we find a "bad" r, which essentially gives $Dr = 0$, even though $D \neq 0$.

We'll make use of v to get a "good" $r' = r + v$.

By using this r', we'll get $Dr' \neq 0$.

This can be easily proven

$$Dr' = D(r + v) = Dr + Dv = 0 + Dv \neq 0$$

We can show that the mapping between r and r' is one-to-one i.e. injective.

To show that a mapping is injective, we need to show that $f(x_1) = f(x_2) \implies x_1 = x_2$

We define the function $f(r) = r + v = r'$

If $r_1 + v = r_2 + v = r'$, we could subtract v from both sides, to show that $r_1 = r_2$

Hence, we've shown that the mapping of r to r' is one to one. Hence the set of "bad" r is injective on the set of "good" r.

Since the mapping is injective, we can say that the range of the function is $\geq$ the domain of the function.

Therefore, we know that |good r| $\geq$ |bad r| .

Hence, we can say that the number of r for which $Dr \neq 0$ is greater than the number of r for which $Dr = 0$, if it's given that $D = 0$.

Hence, we've proven that the probability that $ABr \neq Cr$ given $AB \neq C$ is $\geq \frac{1}{2}$.

Now, a 0.5 probability of success might not seem that great, but we can make use of this property to take multiple trials by choosing different values for r.

Say the probability of failure is 1/2. If we pick k different vectors, then the probability of failing to identify that the condition doesn't satisfy will be $\frac{1}{2^k}$.

Assuming that the trials are independent, and we pick the vectors randomly, the probability of the test failing after taking 10 different vectors is $1/2^{10} = 0.0009$, which is very low.

Therefore, we have an algorithm which instead of performing the entire matrix multiplication to verify the product of 2 matrices, uses a probabilistic method to verify the product with very low chance of failure, which can be adjusted to suit the application's need, at the cost of running time.