Polynomial Multiplication using Fast Fourier Transform

Pritam Chandra

1 Preliminaries and notations

1. We will denote the set $\{0, \ldots, n-1\}$ by $[n]$.

2. The notation A for a set of vectors $\{a_1, \ldots, a_k\}$ will be interchangebly used for the matrix with k column vectors $a_1, \ldots, a_k$.

3. $\omega = e^{-\frac{2\pi}{n}i}$ is an n-th root of unity. For any $x \in \mathbb{C}^n$, $\bar{x}$ is its complex conjugate.

4. For any vector $x \in \mathbb{C}^n$, the components of x will be immediately understood as $x = [x_0, x_1, \ldots, x_{n-1}]$.

5. For $x, y \in \mathbb{C}^n$, the inner product $\langle x, y \rangle := \sum_{j \in [n]} x_j \bar{y}_j$. The norm used is the one arising from this inner product.

2 The Fourier basis of $\mathbb{C}^n$

Define the matrix F on $\mathbb{C}^n$ as

$$F = \left[\frac{\omega^{-jk}}{n}\right] = \frac{1}{n}[\bar{\omega}^{jk}] \text{ for } j, k \in [n]. \tag{1}$$

Let the columns of this matrix be $f_0, \ldots, f_{n-1}$. Then for $l, k \in [n]$ observe that[1]

$$\langle f_k, f_l \rangle = \frac{1}{n^2} \sum_{j \in [n]} \omega^{-jk} \overline{\omega^{-jl}} = \frac{1}{n^2} \sum_{j \in [n]} \omega^{j(l-k)} = \begin{cases} 0, & \text{if } k \neq l \\ 1/n, & \text{if } k = l \end{cases}. \tag{2}$$

Thus F is an orthogonal system and hence forms a basis of $\mathbb{C}^n$. We call F as the *Fourier Basis* of $\mathbb{C}^n$.

Notice that F is not an orthonormal matrix as, from (2), its columns f_k have norm $\|f_k\| = \frac{1}{\sqrt{n}}$. But, we can normalize F to obtain the orthonormal system $\sqrt{n}F$.

1. Using $\omega^j = \omega^{j \bmod n}$, $\overline{\omega^j} = \omega^{-j}$ and $1 + \omega + \cdots + \omega^{n-1} = 0$ for any $j \in \mathbb{N}$.

3 Discrete Fourier Transform (DFT)

For an $x \in \mathbb{C}^n$, the (discrete) *Fourier transform* of x, denoted by $\hat{x}$, is the representation of x in the Fourier basis.

If $x = [x_0, \ldots, x_{n-1}]$ in the standard basis, then by the change of basis formula we can compute $\hat{x} = Wx$, where $W = F^{-1}$. W is called the *DFT matrix* of dimension n. Now, using the orthonormality of $\sqrt{n}F$ we can see that

$$(\sqrt{n}F)^{-1} = (\sqrt{n}F)^{*} \Longrightarrow \frac{1}{\sqrt{n}} F^{-1} = \sqrt{n}\, F^{*} \Longrightarrow W = F^{-1} = nF^{*}. \tag{3}$$

Subsequently, using (1) we obtain the matrix representation of W as

$$W = nF^{*} = n\frac{1}{n}[\overline{\omega^{-kj}}] = [\omega^{jk}], \text{ for } j,k \in [n]. \tag{4}$$

Hence, for any $x \in \mathbb{C}^n$ its fourier transform $\hat{x} = Wx$ is computed as

$$\hat{x} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{(n-1)} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)^2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} = Wx. \tag{5}$$

4 A clever bypass in the computation of $\hat{x}$

For an arbitrary matrix A the multiplication Ax requires n^2 basic operations, where n is the dimensioin of A. However, in the special case of fourier transform, given n is an even number, one can exploit the structutre of W to obtain a significant improvement.

For $j \in \mathbb{N}$, let $\omega_j = e^{-\frac{2\pi}{j}i}$ be the j-th root of unity, and let W_j be the DFT matrix of dimension j. Let $n = 2m$. Then

$$(\omega_n)^2 = \omega_{_m} \text{ and } (\omega_n)^m = e^{-\pi i} = -1. \tag{6}$$

Now for an $x \in \mathbb{C}^n$, define $x_e, x_o \in \mathbb{C}^m$ as

$$x_e = [x_0, x_1, \ldots, x_{n-2}] \text{ and } x_o = [x_1, x_3, \ldots, x_{n-1}].$$

Notice x_e and x_o are the even and odd components of x. Now say $\hat{x}_e = W_m x_e$ and $\hat{x}_o = W_m x_o$ are the fourier transforms of x_e and x_o respectively. We denote their components using the usual subscripts, that is $x_e = [x_{e,j}]$ and $x_o = [x_{o,j}]$ for $j \in [m]$.

We claim that the fourier transform $\hat{x} = W_n x$ of x can be entirely recovered from $\hat{x}_e$ and $\hat{x}_o$.

Proposition 1. *Given the quantities as defined above*

$$\begin{aligned} \hat{x}_j &= \hat{x}_{e,j} + \omega_n^j \hat{x}_{o,j} \\ \hat{x}_{j+m} &= \hat{x}_{e,j} - \omega_n^j \hat{x}_{o,j} \quad \textit{for } j \in [m] \end{aligned} \tag{7}$$

Proof of the proposition is provided towards the end of the notes. Notice that computing $\hat{x}$ in this manner requires $2m^2 + 2m$ operations, where m^2 operations are required for computing the Fourier transform of each of x_e and x_o, and $2m$ more operations are required for evaluating (7). Compared to $4m^2$ operations of the original method this provides an improvement of nearly 2 folds.

5 Fast Fourier Transform (FFT)

The construction in **4** with the result in (7) sets the ground for a recursive algorithm. In particular, x_e and x_o are vectors in a lower dimension. So, one can apply on them the same process applied on x to calculate $\hat{x}_e$ and $\hat{x}_o$. The only requirement is that m be even.

This process can be repeated easily if $n = 2^l$ for some l; which would ensure that the smaller vectors obtained at each step are from a space of even dimension, until the base case $n = 1$ is reached. In the base case the fourier transform of a number x is x itself. Summing these up, we can easily construct the algorithm mentioned for FFT as showed below.

$$\begin{array}{l}
\text{def FFT}(x)\text{:} \\
\quad \text{let } n = \dim x \\
\quad \#\, \textit{n must be a power of } 2 \\
\quad \text{if } n = 1 : \text{return } x \\
\quad \omega = e^{-\frac{2\pi}{n} i} \\
\quad \text{calculate } x_e,\ x_o \\
\quad \hat{x}_e = \text{FFT}(x_e),\ \hat{x}_o = \text{FFT}(x_o) \\
\quad \text{for } j \in [n/2]\text{:} \\
\qquad \hat{x}_j = \hat{x}_{e,j} + \omega_n^j \hat{x}_{o,j} \\
\qquad \hat{x}_{j+n/2} = \hat{x}_{e,j} - \omega_n^j \hat{x}_{o,j} \\
\quad \text{return } \hat{x}
\end{array} \tag{8}$$

This is a standard divide and conquer algorithm with the divide step in 6, and the merge step in 8,9,10 of cost $\mathcal{O}(n)$ (similar to merge sort). Hence the run time $T(n)$ of the algorithm satisfies the recurrence relation

$$T(n) = 2T(n/2) + \mathcal{O}(n).$$

Solving this we get $T(n) = \mathcal{O}(n \log n)$. This is a significant improvement compared to the $\mathcal{O}(n^2)$ computations required by the naive method.

6 Inverse Fourier Transform

We saw in **3** that the Fourier transform of x was given by $\hat{x} = Wx$. In the discussion in **2** we established that W is an invertible matrix with $F = W^{-1}$. Using this we naturally define the inverse Fourier transform (IFT). For $x \in \mathbb{C}^n$, its inverse Fourier transform is denoted by $\check{x}$, and is defined as

$$\check{x} = Fx. \tag{9}$$

It immediately follows that $(\hat{x})\check{} = (\check{x})\hat{} = x$. Additionally, from (1) notice that the matrix representation of $nF = [\bar{\omega}^{jk}] = \bar{W}$. So,

$$\check{x} = Fx = \frac{1}{n}(nF)x = \frac{1}{n}\bar{W}x.$$

Now if ω is an n-th root of unity, so is $\bar{\omega} = e^{\frac{2\pi}{n}i}$. It is also easy to verify that if we re-define $\hat{x} = \bar{W}x$, then replacing ω by $\bar{\omega}$ satisfies (6) and hence satisfies the equations in (7). Thus one can compute $n\check{x}$ by merely replacing ω by $\bar{\omega}$ in the FFT algorithm (8). Finally, $\check{x}$ is obtained by scaling the result by a factor of $1/n$. This is called the IFT algorithm.

7 Convolution Rings

Every vector space V forms an abelian group under the addition operation. However, it does not demand a binary multiplication to be defined on V. In fact, in many cases natural or useful multiplications can be difficult to formulate. A vector space equipped with a multiplication forms a ring (maybe without a multiplicative identity). In this section we define two such multiplications.

Convolution. We denote the convolution operator by $*$, and for $x, y \in \mathbb{C}^n$ we define

$$x * y = h \text{ where } h_j = \sum_{k \in [n]} x_k \, y_{(j-k) \bmod n} \text{ for } j \in [n].$$

The set $\mathbb{C}^n$ equipped with $*$ is forms a convolution ring without an identity.

Hadamard Product. We denote the Hadamard operator by $\cdot$. For $x, y \in \mathbb{C}^n$

$$x \cdot y = g \ \text{ where } g_j = x_j y_j \text{ for } j \in [n] .$$

While the hadamard product is easy to compute, convolution looks decently complicated. Convolution, however, is the natural product that defines the multiplication of polynomials. In particular, if polynomials $x(t)$ and $y(t)$ with degree at most $n-1$ are represented with their coefficient vectors x and y, then $x * y$ is the coeffcient vector of the polynomial $x(t)y(t) \bmod (t^n - 1)$. Thus the costly operation $*$ comes with the tradeoff of being the natural product for polynomials.

The primary motivation behind FFT is to reduce the cost of computing this crucial operation $*$. This is achieved by the following result which offers a very suitable interplay between the convolution and the hadamard product. For $x, y \in \mathbb{C}^n$

$$\widehat{x * y} = (\hat{x} \cdot \hat{y}). \tag{10}$$

This can be naturally understood in the context of polynomial multiplication as discussed in the following section.

8 Polynomial Multiplication using FFT

Let $p(t) = p_0 + p_1 t + \cdots + p_{n-1} t^{n-1}$ be a polynomial of degree at most $n-1$, where n is a power of 2. Then p can be uniquely represented by its coefficients as a vector in $\mathbb{C}^n$. That is $p = [p_0, p_1, \ldots, p_{n-1}]$. Then, using (5) observe that for $j \in [n]$

$$\hat{p}_j = \sum_{k \in [n]} p_k \omega^{kj} = p(\omega^j). \tag{11}$$

This is a crucial observation. It says that given a polynomial $p(t)$, the fourier transform of its coefficient vector is its evaluation at the points $[1, \omega, \ldots, \omega^{n-1}]$, the roots of unity.

It is easy to see (10) using this understanding. Let $x(t)$ and $y(t)$ be two polynomials of degree m and k respectively. Choose $n \geq m + k + 1$, a power of 2. Say $h(t) = x(t)\, y(t)$. Then the coefficient vectors h, x, y of these polynomials are related as $h = x * y$.

Note that the restriction on n to be larger than $m + k + 1$ ensures that $h(t) \bmod (t^n - 1) = h(t)$, or the process computes the actual product of x and y without any modular reduction.

From (11), we have $\hat{h}_j = h(\omega^j) = x(\omega^j)\, y(\omega^j) = \hat{x}_j\, \hat{y}_j$. Therefore, from the definition of Hadamard product it is clear that $\hat{h} = \hat{x} \cdot \hat{y}$, thereby proving the identity in (11). Finally, using this identity, we can define the polynomial multiplication algorithm as follows.

```
# input two polynomials in vector form
def multiply(x, y):
        m = deg x, k = deg y
        n = 2^⌈log2(m+k+1)⌉
        x̂ = FFT(x), ŷ = FFT(y)
        return IFT(x̂ · ŷ)
```

This algorithm runs in $\mathcal{O}(n \log n)$ time again, which is a great improvement to the $\mathcal{O}(n^2)$ run time of the naive approach.

9 Proof of *Proposition 1*

Extracting $\hat{x}_j$ from (5) for $j \in [m]$ we have,

$$\begin{aligned}
\hat{x}_j &= \sum_{k \in [n]} \omega_n^{jk}\, x_k \\
&= \sum_{k=0}^{m-1} \omega_n^{j\,(2k)}\, x_{2k} + \sum_{k=0}^{m-1} \omega_n^{j\,(2k+1)}\, x_{2k+1} \\
&= \sum_{k=0}^{m-1} (\omega_n^2)^{jk}\, x_{2k} + \omega_n^j \cdot \sum_{k=0}^{m-1} (\omega_n^2)^{jk}\, x_{2k+1} \\
&= \sum_{k=0}^{m-1} \omega_m^{jk}\, x_{e,k} + \omega_n^j \cdot \sum_{k=0}^{m-1} \omega_m^{jk}\, x_{o,k}, \qquad \ldots\text{[using (6)]} \\
&= \hat{x}_{e,j} + \omega_n^j\, \hat{x}_{o,j}
\end{aligned}$$

Continuing similarly with the remaining terms,

$$\begin{aligned}
\hat{x}_{j+m} &= \sum_{k \in [n]} \omega_n^{(j+m)k}\, x_k \\
&= \sum_{k=0}^{m-1} \omega_n^{(j+m)\,(2k)}\, x_{2k} + \sum_{k=0}^{m-1} \omega_n^{(j+m)(2k+1)}\, x_{2k+1} \\
&= \sum_{k=0}^{m-1} (-\omega_n^j)^{2k}\, x_{2k} + \sum_{k=0}^{m-1} (-\omega_n^j)^{(2k+1)} x_{2k+1} \qquad \ldots\text{[using (6)]} \\
&= \sum_{k=0}^{m-1} (\omega_m)^{jk}\, x_{e,k} - \omega_n^j \cdot \sum_{k=0}^{m-1} (\omega_m)^{jk}\, x_{o,k}, \qquad \ldots\text{[using (6)]} \\
&= \hat{x}_{e,j} - \omega_n^j\, \hat{x}_{o,j} \qquad \square
\end{aligned}$$