Notes on the Fast Fourier Transform

Olivier Verdier

1 Fast Fourier Transform

1.1 Algorithm

The discrete Fourier transform may be expressed as a matrix-vector multiplication. The matrices F_n have size n and their entries are

$$[F_n]_{ij} = (\omega_n)^{ij} = \mathrm{e}^{\mathrm{i}\frac{2\pi}{n}ij} \qquad i,j = 0,\ldots,n-1.$$

So it means that the matrices have the form:

$$F_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

where $\omega = \omega_n$, and the matrix F_n has n rows and n columns.

The key observation is that if n is even, and if one groups together the even and odd elements of the multiplied vector, the calculation simplifies significantly. We are trying to compute

$$y_j = \sum_{k=0}^{2n-1} (\omega_{2n})^{kj} z_k.$$

By grouping together the even and odd terms, we obtain

$$y_j = \sum_{k=0}^{n-1} (\omega_{2n})^{2kj} z_{2k} + \sum_{k=0}^{n-1} (\omega_{2n})^{(2k+1)j} z_{2k+1}$$

Notice that

$$(\omega_{2n})^{2kj} = \mathrm{e}^{\mathrm{i}\frac{2\pi}{2n}2kj} = \mathrm{e}^{\mathrm{i}\frac{2\pi}{n}kj} = (\omega_n)^{kj}.$$

We therefore also obtain

$$(\omega_{2n})^{(2k+1)j} = \omega_{2n}^j (\omega_n)^{kj}.$$

Now let us define the two vectors

$$(z'')_k := z_{2k} \qquad (z')_k := z_{2k+1} \qquad k = 0, \dots, n-1$$

These vectors contain the even and odd components of the vector z (*starting from zero*).

Together with the observations above, this yields:

$$y_j = \sum_{k=0}^{n-1} \omega_n^{kj} (z'')_k + \omega_{2n}^j \sum_{k=0}^{n-1} \omega_n^{kj} (z')_k \tag{1}$$

If we introduce $j+n$ in formula (1), we obtain:

$$y_{j+n} = \sum_{k=0}^{n-1} \omega_n^{k(j+n)} (z'')_k + \omega_{2n}^{j+n} \sum_{k=0}^{n-1} \omega_n^{k(j+n)} (z')_k$$

Now, notice that

$$\omega_n^{k(j+n)} = \omega_n^{kj} \underbrace{\omega_n^{kn}}_{=(\omega_n^n)^k = 1^k = 1} = \omega_n^{kj}$$

$$\omega_{2n}^{j+n} = \omega_{2n}^j \omega_{2n}^n = \omega_{2n}^j (-1) = -\omega_{2n}^j$$

So the final computation is that for $j \leq n$ we have:

$$\begin{aligned} y_j &= \sum_{k=0}^{n-1} (\omega_n)^{kj} (z'')_k \quad + (\omega_{2n})^j \sum_{k=0}^{n-1} (\omega_n)^{kj} (z')_k \qquad j = 0, \dots, n-1 \\ y_{j+n} &= \sum_{k=0}^{n-1} (\omega_n)^{kj} (z'')_k \quad - (\omega_{2n})^j \sum_{k=0}^{n-1} (\omega_n)^{kj} (z')_k \qquad j = 0, \dots, n-1 \end{aligned}$$

In order to account for the second terms in those equations, let us define the matrix

$$\Omega_n := \begin{bmatrix} 1 & & & & \\ & \omega_{2n} & & & \\ & & (\omega_{2n})^2 & & \\ & & & \ddots & \\ & & & & (\omega_{2n})^{n-1} \end{bmatrix}$$

that is, the diagonal matrix of size n which contains the values

$$(1, \omega_{2n}, \omega_{2n}^2, \dots, \omega_{2n}^{n-1})$$

on the diagonal.

If we define

$$Y_0 := F_n z'' \qquad Y_1 := \Omega_n F_n z'$$

the algorithm may be reformulated as

$$\begin{aligned} a &= Y_0 + Y_1 \\ b &= Y_0 - Y_1 \end{aligned}$$

and the final value y is the concatenation of the vectors a and b, namely

$$y = (a_0, a_1, \dots, a_{n-1}, b_0, b_1, \dots, b_{n-1})$$

or, more explicitly, the vector y has components $(y_0, \dots, y_{2n-1})$, and

$$\begin{aligned} y_j &= a_j \qquad \text{for} \qquad j = 0, \dots, n-1 \\ y_{n+j} &= b_j \qquad \text{for} \qquad j = 0, \dots, n-1 \end{aligned}$$

So, let us describe the algorithm once more:

1. Decompose z into even components (z'') and odd components (z')
2. Compute $Y_0 = F_n z''$ and $\widetilde{Y} = F_n z'$.
3. Compute $Y_1 \coloneqq \Omega_n \widetilde{Y}$
4. The result is the vector $y = (Y_0 + Y_1, Y_0 - Y_1)$

1.2 Operation Counting

Note that the only step which requires multiplication is item 3. That is exactly n multiplications. If κ_n is the cost of the fast Fourier transform of size n, then we have

$$\kappa_{2n} = n + 2\kappa_n.$$

That reflects the fact that to compute the Fourier transform of size $2n$ for the vector z, we need to compute

1. one Fourier transforms of size n for z' (cost κ_n);
2. one Fourier transforms of size n for z'' (cost κ_n);
3. n multiplications corresponding to the multiplication with Ω_n, which is a diagonal matrix of size n (cost n);

Suppose now that $n = 2^k$. Then we can use the formula recursively and obtain

$$\begin{aligned} \kappa_{2^k} &= 2^k + 2\kappa_{2^{k-1}} \\ &= 2^k + 2(2^{k-1} + 2\kappa_{k-2}) \\ &= 2 \times 2^k + 2\kappa_{n-2} \\ &= 3 \times 2^k + 2\kappa_{n-3} \\ &= \cdots \\ &= k \times 2^k \end{aligned}$$

so we get

$$\kappa_{2^k} = k2^k.$$

This is remarkable! Compare with the size of the standard matrix vector multiplication, which would cost

$$2^k 2^k.$$

It means that we have reduce the number of multiplications by a factor of $2^k/k$, which gets very big very quickly.