Algorithm Design and Analysis(H)
Southern University of Science and Technology
Mengxuan Wu
12212006

Assignment 3: Applications of Fast Fourier Transform

Mengxuan Wu

1 Convolution

Convolution is a fundamental arithmetic operation on two functions, which produces a third function. Suppose we have two discrete functions f and g, and we want to calculate the convolution of the two functions. The convolution of f and g is defined as

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau)$$

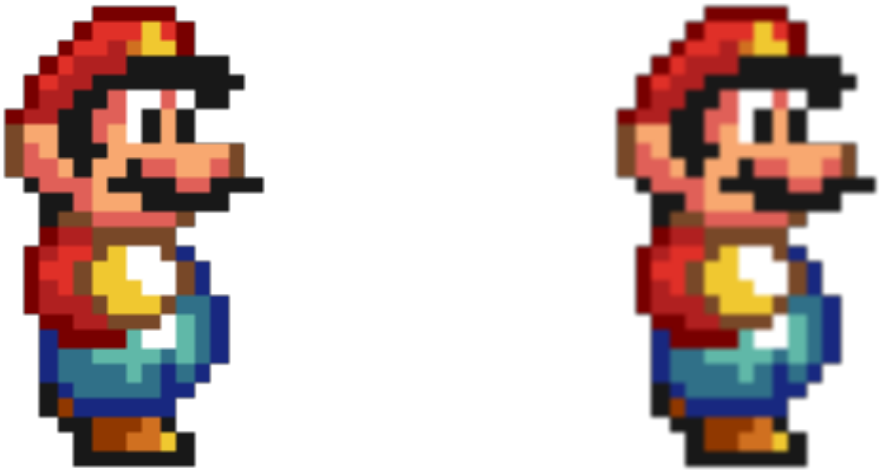
An easy way to convert a convolution operation to a polynomial multiplication problem is by treating the value of the function as the coefficient of a polynomial. For example, suppose we have two functions $f = [1, 2, 3]$ and $g = [4, 5, 6]$. We can convert the two functions to two polynomials $f(x) = 1 + 2x + 3x^2$ and $g(x) = 4 + 5x + 6x^2$. Then the convolution of f and g is equivalent to the multiplication of the two polynomials $f(x)g(x) = 4 + 13x + 28x^2 + 27x^3 + 18x^4$. And the coefficients of the result polynomial are the values of the convolution of the two functions, which is $[4, 13, 28, 27, 18]$.

In my opinion, the superiority of the FFT is that it provides a fast way to calculate the convolution of two functions. Since the convolution operation is essential in multiple fields, combined with the well-known optimization trick of "making common things fast", the FFT algorithm has been widely used in many applications. In most applications listed below, we can find that the FFT algorithm is used to speed up the convolution operation, and further improve the efficiency of the algorithm.

2 Image Processing and Feature Extraction

Artificial intelligence and machine learning have been widely used in image processing and feature extraction. And one core technique in image processing is using a convolutional neural network (CNN) to extract features from images, which is based on the convolution operation and can be implemented by the Fast Fourier Transform (FFT).

We take the image in Figure 1a as an example. The image can be represented as a 2D matrix, where each element represents the pixel value of the image. Suppose we want to blur the image, a natural way is to mix each pixel with its neighbors. For example, we can let each pixel in the new image be the average of itself and its closest 8 neighbors on each side.



(a) Original Image (b) Blurred Image

Figure 1: Blurring an Image

In this case, we define a kernel, which is a 3×3 matrix with all elements equal to $\frac{1}{9}$. Then we slide the kernel over the image. At each position, we multiply the kernel with the image, element-wise, and sum up the results. The result is the pixel value of the new image at that position.

$$a'_{i,j} = \sum_{m=-1}^{1} \sum_{n=-1}^{1} a_{i+m,j+n} \cdot \frac{1}{9}$$

Or more intuitively:

$$a'_{i,j} = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \times_{\text{element-wise}} \begin{bmatrix} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{bmatrix}$$

We can do more with the kernel by changing the values of the kernel. For example, we usually use a kernel with a Gaussian distribution to blur the image, which samples the pixels in a 2D Gaussian distribution. That means a kernel that takes greater weight in the center and less weight on the edges.

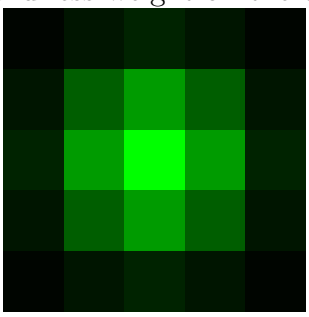
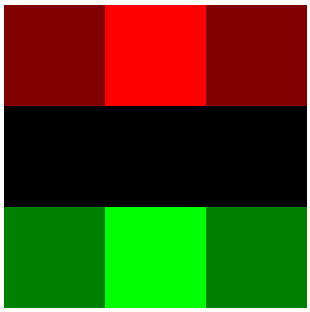


Figure 2: Gaussian Blur Kernel Visualization

0.00296902	0.0133062	0.0219382	0.0133062	0.00296902
0.0133062	0.0596343	0.0983203	0.0596343	0.0133062
0.0219382	0.0983203	0.162103	0.0983203	0.0219382
0.0133062	0.0596343	0.0983203	0.0596343	0.0133062
0.00296902	0.0133062	0.0219382	0.0133062	0.00296902

Table 1: Gaussian Blur Kernel Matrix

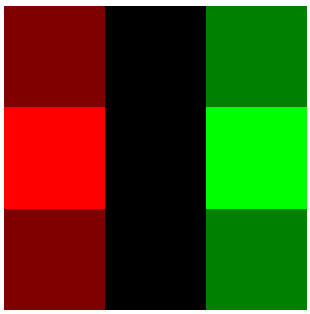
Now we can talk about feature extraction. A feature, such as an edge, a corner, or a texture, is a pattern that can be found in an image. It is common practice to use a CNN to extract features from images. For example, a `sobel` kernel, which is a 3×3 matrix that apply a positive weight to the pixels on one side of the center pixel and a negative weight to the pixels on the other side, can be used to extract edges from an image.



(a) Horizontal Sobel Kernel (green is positive, red is negative)

(b) Horizontal edge extracted

Figure 3: Horizontal Edge Extraction



(a) Vertical Sobel Kernel

(b) Vertical edge extracted

Figure 4: Vertical Edge Extraction

We can notice that this kernel multiplication is closely related to the convolution operation. What a kernel do is to multiply the kernel matrix with the image, element-wise, and sum up the results. If we treat the value of the image as coefficients of a polynomial, and the kernel as another polynomial, then the kernel multiplication is equivalent to the convolution of the two polynomials. This is where the FFT comes in.

Since CNNs are widely used in image processing, the FFT algorithm can be said to play a crucial role in speeding up the neural network operation.

3 Signal Processing

3.1 Fourier Transform

The most common application of the FFT is in signal processing, where we use the FFT to analyze the frequency components of a signal. Given any signal, we can use the FFT to efficiently decompose the signal into multiple frequency components (particularly sine and cosine waves).

Let $f(t)$ be a signal, where t is the time and $f(t)$ is the amplitude of the signal at time t. The Fourier Transform of $f(t)$ is defined as

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt$$

Since $f(t)$ is always a real number, the final result of the Fourier Transform is a complex number. The magnitude of the complex number $|F(\omega)|$ represents the amplitude

of the frequency component ω in the signal, or simply to say how strong the frequency component is in the signal. The phase of the complex number $\angle(F(\omega))$, which corresponds to the angle of the complex number with respect to the real axis, represents the phase shift of the frequency component ω in the signal. With the two pieces of information, we can locate each frequency components in the signal.

The Inverse Fourier Transform is defined as

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$$

which is the reverse operation of the Fourier Transform.

3.2 Discrete Fourier Transform

In practice, we usually deal with signals that are samples at discrete time points. The Discrete Fourier Transform (DFT) is a discrete version of the Fourier Transform, which is defined as

$$F(k) = \sum_{n=0}^{N-1} f(n) e^{-i2\pi kn/N}$$

We can notice that this formula is very similar to the equation we used to calculate the polynomial multiplication, where we use FFT to convert coefficient representation to point-value representation:

$$b_i = \sum_{j=0}^{N-1} a_j e^{-i2\pi j/N}$$

where a_j is the coefficient of the polynomial $a(x)$, and b_i is the point-value representation of the polynomial $b(x)$. This means that we can use the FFT algorithm to calculate the DFT in $O(N \log N)$ time complexity.

3.3 Applications

The applications of the decomposition of signals into frequency components are numerous. We can take the compression of audio files as an example. If we see the audio signal as a combination of multiple frequency components, then we don't need to store the entire signal. Instead, we can store the frequency components with the highest amplitude, and the phase shift of each frequency component. This is the idea behind the MP3 format, which compresses the audio signal by removing the frequency components with low amplitude.

4 Image Compression

One common application of the FFT in image processing is image compression. A variant of FFT, the Discrete Cosine Transform (DCT), is used in image compression in format like JPEG. Its core idea is similar to audio compression, where we transform the image matrix into another matrix that represents the frequency components of the image, and then we wipe out the frequency components with lower amplitude.

The formula we use to convert a 2D image matrix f into a 2D frequency matrix F is

$$\begin{cases} F_{1,1} = \frac{1}{n}\sum_{x=1}^{n}\sum_{y=1}^{n} f_{x,y} & \\ F_{1,v} = \frac{\sqrt{2}}{n}\sum_{x=1}^{n}\sum_{y=1}^{n} f_{x,y}\cdot\cos\left(\frac{(2y+1)v\pi}{2n}\right) & v = 2,3,\cdots,n \\ F_{u,1} = \frac{\sqrt{2}}{n}\sum_{x=1}^{n}\sum_{y=1}^{n} f_{x,y}\cdot\cos\left(\frac{(2x+1)u\pi}{2n}\right) & u = 2,3,\cdots,n \\ F_{u,v} = \frac{2}{n}\sum_{x=1}^{n}\sum_{y=1}^{n} f_{x,y}\cdot\cos\left(\frac{(2x+1)u\pi}{2n}\right)\cdot\cos\left(\frac{(2y+1)v\pi}{2n}\right) & \text{otherwise} \end{cases}$$

And the inverse formula is

$$f_{x,y} = \frac{1}{n}F_{1,1} + \frac{\sqrt{2}}{n}\sum_{v=2}^{n} F_{1,v}\cdot\cos\left(\frac{(2y+1)v\pi}{2n}\right) + \frac{\sqrt{2}}{n}\sum_{u=2}^{n} F_{u,1}\cdot\cos\left(\frac{(2x+1)u\pi}{2n}\right) + \frac{2}{n}\sum_{u=2}^{n}\sum_{v=2}^{n} F_{u,v}\cdot\cos\left(\frac{(2x+1)u\pi}{2n}\right)\cdot\cos\left(\frac{(2y+1)v\pi}{2n}\right)$$

A typical DCT matrix after the transformation is shown in Figure 5.

	1	2	3	4	5	6	7	8
1	1	2	6	7	15	16	28	29
2	3	5	8	14	17	27	30	43
3	4	9	13	18	26	31	42	44
4	10	12	19	25	32	41	45	54
5	11	20	24	33	40	46	53	55
6	21	23	34	39	47	52	56	61
7	22	35	38	48	51	57	60	62
8	36	37	49	50	58	59	63	64

Figure 5: Zigzag Order of Frequency Matrix

To be short, the values in the blue and green areas are the low-frequency components, and the values in the yellow and red areas are the high-frequency components. Since the human eye is more sensitive to low-frequency components, we can remove the high-frequency components to compress the image (setting the values in the yellow and red areas to zero). This is part of the idea used in the JPEG compression algorithm.

5 Probability and Statistics

FFT is widely used in calculating the frequency distribution of two or more random variables.

For example, suppose we have two random variables X and Y. And we want to calculate the probability distribution of the sum of the two random variables, $Z = X + Y$. The probability distribution of Z is the convolution of the probability distribution of X and Y.

$$P(Z = z) = \sum_x P(X = x)P(Y = z - x)$$

If the two random variables are discrete, we can treat the probability distribution as a polynomial, and the convolution of the two probability distributions is equivalent to the multiplication of the two polynomials. Take an example of two independent dice rolls, where X and Y are the random variables representing the results of the two dice rolls. The probability distribution of X and Y are:

$$P(X = x) = \begin{cases} \frac{1}{6} & x = 1, 2, 3, 4, 5, 6 \\ 0 & \text{otherwise} \end{cases}$$

The probability distribution of $Z = X + Y$ is the convolution of the two probability distributions. We can calculate this by calculating the convolution of the two polynomials.

$$\left(\sum_{i=1}^{6} \frac{1}{6}x^i\right)^2 = \frac{1}{36}x^2 + \frac{1}{18}x^3 + \frac{1}{12}x^4 + \frac{1}{9}x^5 + \frac{5}{36}x^6 + \frac{1}{6}x^7 + \frac{1}{9}x^8 + \frac{5}{36}x^9 + \frac{1}{12}x^{10} + \frac{1}{18}x^{11} + \frac{1}{36}x^{12}$$

The result is the probability distribution of the sum of two dice rolls.

$$P(Z = z) = \begin{cases} \frac{1}{36} & z = 2 \\ \frac{1}{18} & z = 3 \\ \frac{1}{12} & z = 4 \\ \frac{1}{9} & z = 5 \\ \frac{5}{36} & z = 6 \\ \frac{1}{6} & z = 7 \\ \frac{1}{9} & z = 8 \\ \frac{5}{36} & z = 9 \\ \frac{1}{12} & z = 10 \\ \frac{1}{18} & z = 11 \\ \frac{1}{36} & z = 12 \\ 0 & \text{otherwise} \end{cases}$$

Similar conversion can be done for the difference of two random variables, the product of two random variables, and the ratio of two random variables.