Lecture 14

Applications of the DFT: Convolution

14.1 Introduction

In this lecture, we will review the application of the DFT to perform circular and linear convolutions. Note that in order to perform linear convolutions based on DFTs, we need to be careful to zero-pad our sequences appropriately in order to avoid the 'circular' effects. Please see below for details, in a 2D application.

14.2 Computational Complexity

Remember that we typically express computational complexity of a certain operation on a length-M signal using the notation $O(g(M))$, where $g(M)$ is some function of M. This 'O' notation means that asymptotically (as $M \to \infty$), the number of calculations (say: scalar multiplications) needed to compute the operation is $< Cg(M)$ for some constant C. For instance, the computational complexity of pointwise multiplication of two vectors is $O(M)$ (ie: $g(M) = M$ in this case), whereas the complexity of matrix-vector multiplication of an arbitrary $M \times M$ matrix times a length-M vector is $O(M^2)$ (note that this is true for general matrix-vector multiplication, but can be done faster for specific choices of matrices, as shown below).

As we have mentioned in previous lectures, the computational complexity (say, in terms of the number of multiplications required, up to a constant factor) of naïvely computing the length-M DFT based on its definition (Equation 11.1) is $O(M^2)$. In contrast, the complexity of calculating the DFT using an FFT algorithm is $M \log M$.

Similarly, the computational complexity of naïvely computing a circular convolution between two length-M sequences is M^2:

$$f_3[m] = (f_1 \circledast f_2)[m] = \sum_{n=0}^{M-1} f_1[n] f_2[(m-n)_M] \tag{14.1}$$

where a periodic extension is assumed in order to implement the circular convolution, as indicated by the modulo operation $(\cdot)_M$. Note that a direct implementation of Equation 14.1 requires calculating M entries of $f_3[n]$, each of which requires the multiplication (and subsequent summation) of M entries of $f_1[n]$ and $f_2[n]$: this leads to the naïve computational complexity of M^2, which is relatively very slow for large vectors. There has to be a better way![1].

However, since we can rely on the (circular) convolution property of the DFT to perform our convolution, we can take a 'shortcut':

1. Calculate the DFT of the input sequences, is: $\hat{f}_1[m]$ and $\hat{f}_2[m]$ using FFT (which, as its name indicates, is fast, $O(M \log M)$).

2. Calculate the DFT of the output sequence by simple (and very fast, $O(M)$) pointwise multiplication, $\hat{f}_3[m] = \hat{f}_1[m]\hat{f}_2[m]$.

3. Calculate our desired output $f_3[n]$ by inverse DFT of $\hat{f}_3[m]$, which again can be implemented using the inverse FFT algorithm, $\sim O(M \log M)$).

In other words, because the computational complexity of implementing a DFT via an FFT algorithm is $\sim O(M \log M)$, we can perform a convolution via several FFTs, with overall $O(M \log M)$ complexity.

Importantly, this ability to implement convolutions rapidly using the FFT can be extended to multiple dimensions, as discussed next.

14.3 Convolution in 2D

Figure 14.1 illustrates the ability to perform a circular convolution in 2D using DFTs (ie: computed rapidly using FFTs). Note that this operation will generally result in a circular convolution, not a linear convolution, as will be explored further in the next section.

14.4 Convolution with Zero-Padding

In order to calculate linear (not circular) convolutions using DFTs, we need to zero-pad our sequences prior to convolution/DFT, such that we avoid overlap between the non-zero

[1] A bit of history from http://www.dspguide.com/ch18/2.htm: "FFT convolution uses the principle that multiplication in the frequency domain corresponds to convolution in the time domain. The input signal is transformed into the frequency domain using the DFT, multiplied by the frequency response of the filter, and then transformed back into the time domain using the Inverse DFT. This basic technique was known since the days of Fourier; however, no one really cared. This is because the time required to calculate the DFT was longer than the time to directly calculate the convolution. This changed in 1965 with the development of the Fast Fourier Transform (FFT). By using the FFT algorithm to calculate the DFT, convolution via the frequency domain can be faster than directly convolving the time domain signals. The final result is the same; only the number of calculations has been changed by a more efficient algorithm."

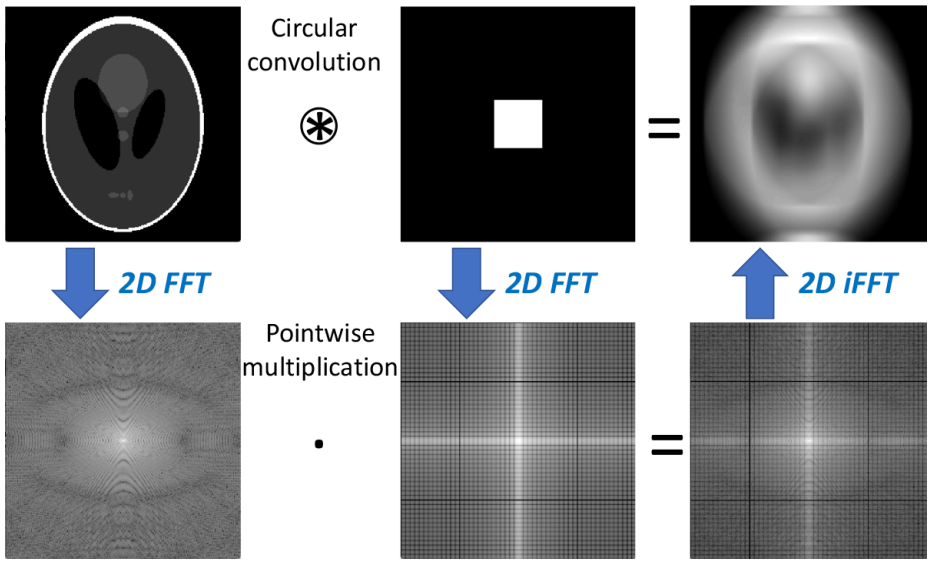

Figure 14.1: Circular convolution in 2D, performed either directly or through the FFT.

portion of the periodic extension of our sequences. This process is demonstrated in figure 14.2.

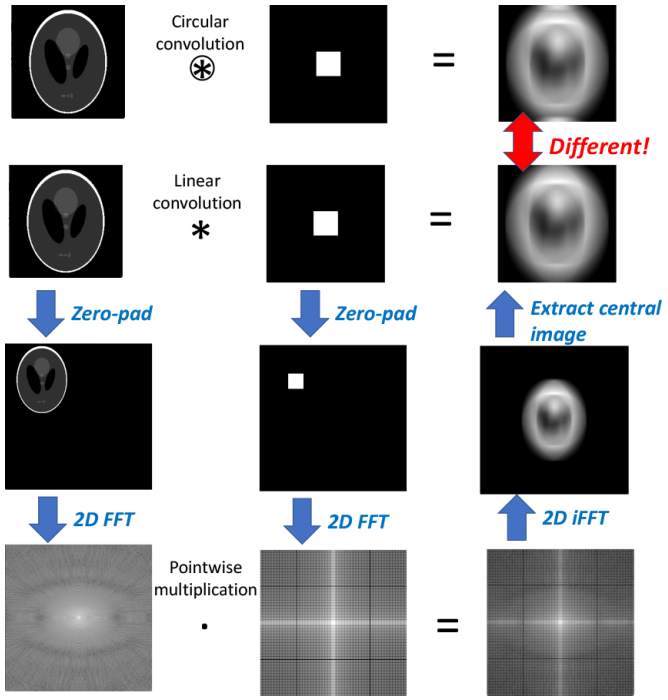

Figure 14.2: Linear convolution in 2D, performed either directly or through a zero-padded FFT. Note that the linear convolution and circular convolution produce different results (as can be observed near the top and bottom of the images).