ECE 310 Fall 2023

Lecture 27

Fast convolution via the DFT

Corey Snyder

Learning Objectives

After this lecture, you should be able to:

- Explain the computational complexity of performing convolution in the time-domain.
- Understand how we may compute linear convolution by using the circular convolution and the DFT.

Recap from previous lecture

We introduced the fast Fourier transform in the previous lecture as an improvement on the computational complexity of calculating the DFT of a finite-length signal. Specifically, we showed that direct computation of the DFT incurs $\mathcal{O}(N^2)$ complexity while the fast Fourier transform has $\mathcal{O}(N \log N)$ complexity. In this lecture, we will apply the computational advantages of the fast Fourier transform to develop an algorithm for fast convolution via the DFT.

1 Computational complexity of convolution sum

First, we would like to consider how we may implement linear convolution in practice. Recall that the convolution of two signals $x[n]$ and $h[n]$ is given by

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \tag{1}$$

If we assume both signals are length N, the summation will have at most N non-zero terms. Thus, the cost of computing each value of $y[n] = x[n] * h[n]$ at a particular index n is $\mathcal{O}(N)$, i.e. has "on the order of N operations". We also know that the resulting length of linear convolution is $N + L - 1$ where N and L are the respective lengths of the two signals. Without loss of generality, we assume $N = L$, thus we must compute this summation for $2N - 1 = \mathcal{O}(N)$ values of n. Therefore, we have:

$$\text{Computational complexity of convolution sum} = \mathcal{O}(\underbrace{N}_{\text{cost/summation}} \cdot \underbrace{N}_{\#\text{ summations}}) = \mathcal{O}(N^2). \tag{2}$$

We have discussed previously how implementations of fundamental operations like convolution and the DFT are faster with a matrix-based implementation. However, a matrix-vector multiplication implementation of the convolution sum will have the same $\mathcal{O}(N^2)$. The improvement will simply be a constant-factor speed-up but not an asymptotic, i.e. as $N \to \infty$, improvement.

2 Linear convolution via the DFT

We now would like to improve on the $\mathcal{O}(N^2)$ computational complexity of linear convolution. We showed in the next lecture that the fast Fourier transform (FFT) algorithm for computing the DFT has $\mathcal{O}(N \log N)$ complexity (where log denotes log base-2: $\log_2$). Thus, we may want to construct an algorithm that exploits this improved complexity to speed up our calculation of convolution.

First, remember that we have the following circular convolution property of the DFT for length-N signals $x[n]$ and $h[n]$:

$$x[n] \circledast h[n] \overset{\text{DFT}}{\longleftrightarrow} X[k]H[k]. \tag{3}$$

We do not want to compute the circular convolution between $x[n]$ and $h[n]$ since this is not equivalent to linear convolution. However, we do want the computational advantages of working in the DFT-domain via the FFT. Thus, we would like to develop an algorithm that makes the circular convolution of $x[n]$ and $h[n]$ become equivalent to the linear convolution of the two signals.

2.1 Fast convolution algorithm

We start by considering a concrete example. Suppose we have the following two length-4 signals:

$$x[n] = \{\underset{\uparrow}{3}, 1, 0, -1\} \tag{4}$$

$$h[n] = \{\underset{\uparrow}{5}, 1, -1, 4\} \tag{5}$$

For both linear and circular convolution, we will use the matrix-vector method for concise presentation, though the same takeaways would be clear with the corresponding table method. The linear convolution result $y[n]$ and circular convolution result $\tilde{y}[n]$ would then be

$$y[n] = x[n] * h[n] = \mathbf{H}x = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 1 & 5 & 0 & 0 \\ -1 & 1 & 5 & 0 \\ 4 & -1 & 1 & 5 \\ 0 & 4 & -1 & 1 \\ 0 & 0 & 4 & -1 \\ 0 & 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 15 \\ 8 \\ -2 \\ 6 \\ 3 \\ 1 \\ -4 \end{bmatrix} \tag{6}$$

$$\tilde{y}[n] = x[n] \circledast h[n] = \tilde{\mathbf{H}}x = \begin{bmatrix} 5 & 4 & -1 & 1 \\ 1 & 5 & 4 & -1 \\ -1 & 1 & 5 & 4 \\ 4 & -1 & 1 & 5 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 18 \\ 9 \\ -6 \\ 6 \end{bmatrix}. \tag{7}$$

Clearly, $y[n] \neq \tilde{y}[n]$. The values are not the same, nor are the lengths of the two convolution results. A simple solution would then be to change $x[n]$ and $h[n]$ such that the result of their circular convolution will have the same length as their ordinary linear convolution. We do not want this adjustment to change the frequency content or information in $x[n]$ or $h[n]$. Thus, we can just pad them both with zeros! Let $x_{\text{zp}}[n]$ and $h_{\text{zp}}[n]$ be $x[n]$ and $h[n]$ zero-padded to length-7, i.e. the sum of their respective lengths minus one.

$$x_{\text{zp}}[n] = \{\underset{\uparrow}{3}, 1, 0, -1, 0, 0, 0\} \tag{8}$$

$$h[n]_{\text{zp}}[n] = \{\underset{\uparrow}{5}, 1, -1, 4, 0, 0, 0\} \tag{9}$$

Now, we can compute the circular convolution result for $x_{\text{zp}}[n]$ and $h_{\text{zp}}[n]$ to see how this will impact our new output $\hat{y}[n]$:

$$\hat{y}[n] = x_{\text{zp}}[n] \circledast h_{\text{zp}}[n] = \tilde{\mathbf{H}}_{\text{zp}} x_{\text{zp}} = \begin{bmatrix} 5 & 0 & 0 & 0 & 4 & -1 & 1 \\ 1 & 5 & 0 & 0 & 0 & 4 & -1 \\ -1 & 1 & 5 & 0 & 0 & 0 & 4 \\ 4 & -1 & 1 & 5 & 0 & 0 & 0 \\ 0 & 4 & -1 & 1 & 5 & 0 & 0 \\ 0 & 0 & 4 & -1 & 1 & 5 & 0 \\ 0 & 0 & 0 & 4 & -1 & 1 & 5 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 15 \\ 8 \\ -2 \\ 6 \\ 3 \\ 1 \\ -4 \end{bmatrix} \tag{10}$$

Now we have $y[n] = \hat{y}[n]$! We see above that padding zeros not only gives us the appropriate length for the linear convolution, but also that these zeros prevent the circular modulation from inducing incorrect output values. In other words, these padded zeros line up with the periodically extended values in $h[n]$ that would otherwise cause incorrect output values like when we saw $\tilde{y}[n] \neq y[n]$ for the first few indices of n. We would also like to point out that padding zeros to $x[n]$ and $h[n]$ would not change the linear convolution result as we will simply have additional zeros at the end of the result. This is consistent with our normal zero-extension assumption with linear convolution and the DTFT.

Fast convolution algorithm. We may now use this zero-padding trick to design a convolution algorithm with improved computational complexity. Let $x[n]$ and $h[n]$ be length-N and length-L signals, respectively. We may compute $y[n] = x[n] * h[n]$ in the following steps:

1. Zero-pad $x[n]$ and $h[n]$ each to length $N + L - 1$. Let $x_{\text{zp}}[n]$ and $h_{\text{zp}}[n]$ denote these zero-padded signals. [**Cost** $= \mathcal{O}(1)$]
2. Take the DFT of $x_{\text{zp}}[n]$ and $h_{\text{zp}}[n]$ using the FFT to obtain $X_{\text{zp}}[k]$ and $H_{\text{zp}}[k]$. [**Cost** $= \mathcal{O}(N \log N)$]
3. Multiply $X_{\text{zp}}[k]$ and $H_{\text{zp}}[k]$ element-wise to compute $Y_{\text{zp}}[k]$. [**Cost** $= \mathcal{O}(N)$]
4. Compute the inverse DFT of $Y_{\text{zp}}[k]$ using the FFT to obtain the output $y[n]$. [**Cost** $= \mathcal{O}(N \log N)$]

Altogether, we have

$$\text{Comp. complexity of fast convolution} = \mathcal{O}(1) + \mathcal{O}(N \log N) + \mathcal{O}(N) + \mathcal{O}(N \log N) = \mathcal{O}(N \log N). \tag{11}$$

Thus, we have designed a linear convolution algorithm with computational complexity of $\mathcal{O}(N \log N) < \mathcal{O}(N^2)$!

There are a couple important things to note here. First, we assume that zero-padding is a *constant-time operation*, which means its computational cost does not depend on the length of the signal N. Second, we have collapsed the cost of fast linear convolution using the asymptotic big-O notation. Note that in practice step (2) above computes two FFTs while step (4) computes an inverse FFT (also with cost $\mathcal{O}(N \log N)$). Thus, we do have a factor of three embedded in this algorithm. As $N \to \infty$, like we consider with big-O notation, this constant factor is not important. However, for smaller values of N, this constant factor may be impactful, e.g. $3N \log_2 N + N > N^2$ for some values of N. The key takeaway from this lecture is that our convolution via the FFT algorithm becomes more advantageous as N becomes larger. Lab 5 of ECE 311 explores this relationship across increasing lengths for the signals we convolve.