Block Convolution

Problem:
- An input signal $x[n]$, has very long length (could be considered infinite)
- An impulse response $h[n]$ has length $P$
- We want to take advantage of DFT/FFT and compute convolutions in blocks that are shorter than the signal

Approach:
- Break the signal into small blocks
- Compute convolutions
- Combine the results

Overlap-Add Method

We decompose the input signal $x[n]$ into non-overlapping segments $x_r[n]$ of length $L$:

$$x_r[n] = \begin{cases} x[n] & nL \leq n < (r+1)L - 1 \\ 0 & \text{otherwise} \end{cases}$$

The input signal is the sum of these input segments:

$$x[n] = \sum_{r=0}^{N-1} x_r[n]$$

The output signal is the sum of the output segments $x_r[n] * h[n]$:

$$y[n] = x[n] * h[n] = \sum_{r=0}^{\infty} x_r[n] * h[n]$$

(1)

Each of the output segments $x_r[n] * h[n]$ is of length $M = L + P - 1$.

Last Time

- Discrete Fourier Transform
  - Properties of the DFT
  - Linear convolution through circular

Today

- Linear convolution with DFT
  - Overlap and add
  - Overlap and save
  - Fast Fourier Transform (start)

Overlapping Method

We can compute each output segment $x_r[n] * h[n]$ with linear convolution.

DFT-based circular convolution is usually more efficient:
- Zero-pad input segment $x_r[n]$ to obtain $x_{zp}[n]$, of length $M$
- Zero-pad the impulse response $h[n]$ to obtain $h_{zp}[n]$, of length $N$ (this needs to be done only once).
- Compute each output segment using:

$$x_r[n] * h[n] = DFT^{-1}\{DFT\{x_{zp}[n]\} \cdot DFT\{h_{zp}[n]\}\}$$

Since output segment $x_r[n] * h[n]$ starts offset from its neighbor $x_{r-1}[n] * h[n]$ by $L$, neighboring output segments overlap at $P - 1$ points.

Finally, we just add up the output segments using (1) to obtain the output.
**Example of overlap and add:**

\[ x[n] = x_0[n] + x_1[n] + x_2[n] \]

\[ y[n] = y_0[n] + y_1[n] + y_2[n] \]

**Recall:**

**Example of overlap and save:**

**DFT and Sampling the DTFT**

\[ X(e^{j\omega}) = e^{-j2\omega \sin \left( \frac{5\omega}{2} \right)} \]

**Basic Idea**

We split the input signal \( x[n] \) into overlapping segments \( x_i[n] \) of length \( L + P - 1 \).

Perform a circular convolution of each input segment \( x_i[n] \) with the impulse response \( h[n] \), which is of length \( P \) using the DFT. Identify the \( L \)-sample portion of each circular convolution that corresponds to a linear convolution, and save it.

This is illustrated below where we have a block of \( L \) samples circularly convolved with a \( P \) sample filter.
Circular Convolution as Matrix Operation

Circular convolution:

\[ h[n] \odot x[n] = \begin{pmatrix} h[0] & h[N-1] & \cdots & h[1] \\ h[1] & h[0] & \cdots & h[2] \\ \vdots & \vdots & \ddots & \vdots \\ h[N-1] & h[N-2] & \cdots & h[0] \end{pmatrix} \begin{pmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{pmatrix} = H_N \cdot x \]

- \( H_N \) is a circulant matrix.
- The columns of the DFT matrix are Eigen vectors of circulant matrices.
- Eigen vectors are DFT coefficients. How can you show? Proof in HW

Fast Fourier Transform Algorithms

- We are interested in efficient computing methods for the DFT and inverse DFT:
  \[ X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad k = 0, \ldots, N - 1 \]
  \[ x[n] = \sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad n = 0, \ldots, N - 1 \]
  where \( W_N = e^{-j(2\pi/N)} \).

Fast Fourier transform algorithms enable computation of an \( N \)-point DFT (or inverse DFT) with the order of just \( N \cdot \log_2 N \) complex multiplications. This can represent a huge reduction in computational load, especially for large \( N \).

<table>
<thead>
<tr>
<th>( N )</th>
<th>( N^2 )</th>
<th>( N \cdot \log_2 N )</th>
<th>( \frac{N^2}{W_{\log_2 N}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>256</td>
<td>64</td>
<td>4.0</td>
</tr>
<tr>
<td>128</td>
<td>16,384</td>
<td>986</td>
<td>18.3</td>
</tr>
<tr>
<td>1,024</td>
<td>1,048,576</td>
<td>10,240</td>
<td>102.4</td>
</tr>
<tr>
<td>8,192</td>
<td>67,108,864</td>
<td>106,496</td>
<td>630.2</td>
</tr>
<tr>
<td>6 \times 10^6</td>
<td>36 \times 10^{12}</td>
<td>135 \times 10^6</td>
<td>2.67 \times 10^8</td>
</tr>
</tbody>
</table>
* 6Mp image size

Most FFT algorithms exploit the following properties of \( W_N^{kn} \):

- Conjugate Symmetry
  \[ W_N^{k-N} = W_N^{-kn} = (W_N^{kn})^* \]
- Periodicity in \( n \) and \( k \):
  \[ W_N^{kn} = W_N^k W_N^{n N} \]
- Power:
  \[ W_N^N = W_N^{N/2} \]
Most FFT algorithms decompose the computation of a DFT into successively smaller DFT computations.
- Decimation-in-time algorithms decompose $x[n]$ into successively smaller subsequences.
- Decimation-in-frequency algorithms decompose $X[k]$ into successively smaller subsequences.
- We mostly discuss decimation-in-time algorithms here.

Assume length of $x[n]$ is power of 2 ( $N = 2^n$). If smaller zero-pad to closest power.

Decimation-in-Time Fast Fourier Transform

Let $n = 2r$ (n even) and $n = 2r + 1$ (n odd):

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r]W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1]W_N^{2(r+1)k}$$

$$= \sum_{r=0}^{(N/2)-1} x[2r]W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1]W_N^{2rk}$$

Note that:

$$W_N^{2rk} = e^{-j(\frac{2\pi}{N})2rk} = e^{-j(\frac{2\pi}{N})r} = W_N^r$$

Remember this trick, it will turn up often.

Decimation-in-Time Fast Fourier Transform

We start with the DFT

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}, \quad k = 0, \ldots, N - 1$$

Separate the sum into even and odd terms:

$$X[k] = \sum_{n \text{ even}} x[n]W_N^{kn} + \sum_{n \text{ odd}} x[n]W_N^{kn}$$

These are two DFT’s, each with half of the samples.

Decimation-in-Time Fast Fourier Transform

Hence:

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r]W_N^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1]W_N^{rk}$$

$$\Delta G[k] + W_N^k H[k], \quad k = 0, \ldots, N - 1$$

where we have defined:

$$G[k] \Delta \sum_{r=0}^{(N/2)-1} x[2r]W_N^{rk} \quad \Rightarrow \text{DFT of even idx}$$

$$H[k] \Delta \sum_{r=0}^{(N/2)-1} x[2r+1]W_N^{rk} \quad \Rightarrow \text{DFT of odd idx}$$

Decimation-in-Time Fast Fourier Transform

An 8 sample DFT can then be diagrammed as

Both $G[k]$ and $H[k]$ are periodic, with period $N/2$. For example

$$G[k + N/2] = \sum_{r=0}^{(N/2)-1} x[2r]W_N^{rk+N/2}$$

$$= \sum_{r=0}^{(N/2)-1} x[2r]W_N^{kr+N/2}W_N^{N/2}$$

$$= \sum_{r=0}^{(N/2)-1} x[2r]W_N^{rk}N/2$$

$$= G[k]$$

so

$$G[k + (N/2)] = G[k]$$

$$H[k + (N/2)] = H[k]$$
The periodicity of $G[k]$ and $H[k]$ allows us to further simplify.
- For the first $N/2$ points we calculate $G[k]$ and $W_N^k H[k]$, and then compute the sum

$$X[k] = G[k] + W_N^k H[k] \quad \forall \{k : 0 \leq k < \frac{N}{2}\}.$$ 

How does periodicity help for $\frac{N}{2} \leq k < N$?

Note that the inputs have been reordered so that the outputs come out in their proper sequence.
- We can define a butterfly operation, e.g., the computation of $X[0]$ and $X[4]$ from $G[0]$ and $H[0]$:

$$G[0] = G[0] + W_N^0 H[0]$$

$$H[0] = H[0] - W_N^0 H[0]$$

This is an important operation in DSP.
Decimation-in-Time Fast Fourier Transform

- We can use the same approach for each of the \( N/2 \) point DFT’s. For the \( N = 8 \) case, the \( N/2 \) DFTs look like

\[
x[0] \quad N/4 - Point \quad DFT \quad x[2] \quad W^0_{N/4} \quad x[4] \quad W^1_{N/4} \quad x[6] \\
x[1] \quad N/4 - Point \quad DFT \quad x[3] \quad W^0_{N/4} \quad x[5] \quad W^1_{N/4} 
\]

*Note that the inputs have been reordered again.

Decimation-in-Time Fast Fourier Transform

At this point for the 8 sample DFT, we can replace the \( N/4 = 2 \) sample DFT’s with a single butterfly.

The coefficient is

\[
W_{N/4} = W_{8/4} = W_2 = e^{-j\pi} = -1
\]

The diagram of this stage is then

\[
x[0] \quad \rightarrow \quad x[0] + x[4] \\
x[4] \quad \rightarrow \quad x[0] - x[4]
\]

Decimation-in-Time Fast Fourier Transform

In general, there are \( \log_2 N \) stages of decimation-in-time.

- Each stage requires \( N/2 \) complex multiplications, some of which are trivial.
- The total number of complex multiplications is \( (N/2) \log_2 N \).

- The order of the input to the decimation-in-time FFT algorithm must be permuted.
  - First stage: split into odd and even. Zero low-order bit first
  - Next stage repeats with next zero-lower bit first
  - Net effect is reversing the bit order of indexes

Decimation-in-Time Fast Fourier Transform

This is illustrated in the following table for \( N = 8 \).

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Bit-Reversed Binary</th>
<th>Bit-Reversed Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>

Decimation-in-Frequency Fast Fourier Transform

The DFT is

\[
X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}
\]

If we only look at the even samples of \( X[k] \), we can write \( k = 2r \).

\[
X[2r] = \sum_{n=0}^{N-1} x[n]W_N^{2rn}
\]

We split this into two sums, one over the first \( N/2 \) samples, and the second of the last \( N/2 \) samples.

\[
X[2r] = \sum_{n=0}^{(N/2)-1} x[n]W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2]W_N^{2rn(n+N/2)}
\]
Non-Power-of-2 FFT’s

But \( W_{N/2}^{(n+N/2)} = W_N^{2m} W_N^{n} = W_N^{2m} W_N^{n/2} \)

We can then write

\[
X[2r] = \sum_{n=0}^{(N/2)-1} x[n] W_N^{2m} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{n}\] 

\[
X[2r+1] = \sum_{n=0}^{(N/2)-1} x[n] W_N^{2m} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{n}\] 

This is the N/2-length DFT of first and second half of x[n] summed.

Decimation-in-Frequency Fast Fourier Transform

The diagram for and 8-point decimation-in-frequency DFT is as follows

This is just the decimation-in-time algorithm reversed!
The inputs are in normal order, and the outputs are bit reversed.

Decimation-in-Frequency Fast Fourier Transform

A similar argument applies for any length DFT, where the length \( N \) is a composite number.

For example, if \( N = 6 \), a decimation-in-time FFT could compute three 2-point DFT’s followed by two 3-point DFT’s

Non-Power-of-2 FFT’s

Good component DFT’s are available for lengths up to 20 or so.
Many of these exploit the structure for that specific length. For example, a factor of

\[
W_N^{N/4} = e^{-j\pi/4} = e^{-j\pi/2} = j
\]

Why?

just swaps the real and imaginary components of a complex number, and doesn’t actually require any multiplies.
Hence a DFT of length 4 doesn’t require any complex multiplies.
Half of the multiplies of an 8-point DFT also don’t require multiplication.
Composite length FFT’s can be very efficient for any length that factors into terms of this order.

For example \( N = 693 \) factors into

\[
N = (7)(9)(11)
\]
each of which can be implemented efficiently. We would perform

- 9 \times 11 DFT’s of length 7
- 7 \times 11 DFT’s of length 9, and
- 7 \times 9 DFT’s of length 11
- Historically, the power-of-two FFTs were much faster (better written and implemented).
- For non-power-of-two length, it was faster to zero pad to power of two.
- Recently this has changed. The free FFTW package implements very efficient algorithms for almost any filter length. Matlab has used FFTW since version 6.

### FFT as Matrix Operation

$$\begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} = \begin{bmatrix} W_N^0 & W_N^1 & \cdots & W_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ W_N^{N-1} & W_N^{N-2} & \cdots & W_N^0 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix}$$

- $W_N$ is fully populated $\Rightarrow N^2$ entries.

### FFT as Matrix Operation

$$\begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} = \begin{bmatrix} W_N^0 & W_N^1 & \cdots & W_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ W_N^{N-1} & W_N^{N-2} & \cdots & W_N^0 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix}$$

- $W_N$ is fully populated $\Rightarrow N^2$ entries.
- FFT is a decomposition of $W_N$ into a more sparse form:

$$F_N = \begin{bmatrix} I_{N/2} & D_{N/2} \\ -D_{N/2} & I_{N/2} \end{bmatrix} \begin{bmatrix} W_N/2 & 0 \\ 0 & W_N/2 \end{bmatrix} \begin{bmatrix} W_N & \cdots & W_N^{N/2-1} \\ \cdots & \cdots & \cdots \\ W_N^{N/2-1} & \cdots & W_N \end{bmatrix}$$

- $I_{N/2}$ is an identity matrix. $D_{N/2}$ is a diagonal with entries $1, W_N, \ldots, W_N^{N/2-1}$.

### Beyond NlogN

- What if the signal $x[n]$ has a k sparse frequency
  - A. Gilbert et. al, "Near-optimal sparse Fourier representations via sampling"
  - H. Hassanieh et. al, "Nearly Optimal Sparse Fourier Transform"
  - Others.....
- $O(k \log N)$ instead of $O(N \log N)$

From: http://groups.csail.mit.edu/netmit/sFFT/