Filtering in Streaming Applications

We will start at 8:05am Eastern.

If you have them, headphones are a good idea.

Next Tuesday (election day in the USA):
  → We will still hold all live sessions
  → Participation will not be monitored
  → Recording will be made available afterward
  → You are still encouraged to attend in person if possible

29 October 2020
Filtering Music

Consider a song (contained in `am_synth.wav`), consisting of three separate “voices,” each of which is band-limited:

- “bass”: 40-170 Hz
- “melody”: 170-370 Hz
- “harmony”: 370-750 Hz

How can we do this?
Now consider the same task, but with a recording of the same song played on guitars rather than on synthesized cosine waves (am.wav).

Predict how this same approach will perform on this recording.

And try it!
Filtering in a Streaming Application

In many applications, we don’t have the entire signal we want to process available to us at the start (we receive it a little bit at a time). Examples:

- a live speaker at an event
- streaming music online

How can we process these signals in a similar way, without access to the entire signal?
Short-time Fourier transforms are based on the analysis of a sequence of finite-length portions of an input signal.
Algorithm 1

Chop the input signal into pieces that are each of length $N$.

Filter each piece by zeroing FFT components outside passband.

Compare original to this new result.

How effective is this algorithm? How can it be improved?
Algorithm 1

Chop the input signal into pieces that are each of length $N$.
Filter each piece by zeroing FFT components outside passband.

Q: How effective is this algorithm?
A: Not very.

One major problem with this algorithm is that if you convolve window 0 with a filter, part of the result should fall into window 1. This is not possible with algorithm 1.
Overlap-Add Method

Algorithm 1’s big problem can be fixed with overlapping windows.

How does overlapping help? How would you choose $s$ and $N$?
Overlap-Add Method

How does overlapping help? How would you choose $s$ and $N$?

Fill each window with $s$ samples of the input and $N-s$ zeros.

Then convolve each window with the filter and sum the windows.

Notice that the convolution of the filter with $x[0:s]$ must not fall outside $0 \leq n < N$. If it did, it would wrap around to the beginning of window 0.
Overlap-Add Method

Create a filter, but limit its unit sample response to some length $L$. Pad this unit sample response with some number $s$ of zeros to create a unit sample response of length $N = L + s$.

$$N - L$$

Divide input signal into blocks of length $s$, which we pad with $L$ zeros to produce a new window of length $N = s + L$.

Convert each length-$N$ block to the frequency domain and multiply by the frequency-domain representation of the filter.

Convert this result back to the time domain. $L$ partial values at the end of each block are added to $L$ partial values at the beginning of the next block.
Overlap-Add: Graphical Depiction

\[ N = 8192 \]
\[ S = 8192 - 2048 \]
\[ L = 2048 \]
Filter Design

Design a filter for the overlap-add method: \( s = 6144 \) and \( N = 8192 \). The filter should pass frequencies in the range \( \Omega_l < \Omega < \Omega_h \).

Method 1: \( N = 8192 \)

\[
X[k] = \begin{cases} 
1 & \text{if } N \frac{\Omega_l}{2\pi} \leq |k| \leq N \frac{\Omega_h}{2\pi} \\
0 & \text{otherwise}
\end{cases}
\]

Method 2: \( N = 2048 \)

\[
X[k] = \begin{cases} 
1 & \text{if } N \frac{\Omega_l}{2\pi} \leq |k| \leq N \frac{\Omega_h}{2\pi} \\
0 & \text{otherwise}
\end{cases}
\]

Method 3: Start with method 2.
Then take inverse FFT; zero-pad to \( N=8192 \), and take FFT.

Method 4: Start with method 1.
Then take inverse FFT, apply rectangular window with width 2048, and take FFT.
Design a filter for the overlap-add method: $s = 6144$ and $N = 8192$. The filter should pass frequencies in the range $\Omega_l < \Omega < \Omega_h$.

Ultimately we need a filter $H[k]$ of length $N = 8192$ (window size). However, $h[n]$ must be no longer than $N = 2048$ samples. Therefore, design a filter using $N = 2048$. Take the inverse transform. Pad to $N = 8192$ samples. Take the transform.

→ Could use method 3 or 4 (they are equivalent).
Filter Design

Design a bandpass filter to extract 170-340 Hz frequency region from signal sampled with $f_s = 44,100$ Hz with $N_f = 2048$. 

![Diagram of filter coefficients $H_1[k]$ and impulse response $h_1[n]$]
Filter Design

Zero-pad to make filter length equal to window length.

\[ h_2[n] \]

\[ H_2[k] \]

Listen to result.
Filter Design

What was wrong with the previous method? How can we fix it?
Filter Design

Apply a triangular window $w[n]$.

Notice that $H_2[k]$ is now a smoother function of $k$. 

---

$w[n]$

$h_2[n]$

$H_2[k]$

$H_2[k]$
Filter Design

Better yet, try a Hann window.

\[ h_2[n] \]

0 \hspace{1cm} N_f \hspace{1cm} n \hspace{1cm} N_w

\[ H_2[k] \]

- \frac{N_w}{2} \hspace{1cm} 0 \hspace{1cm} \frac{N_w}{2} \hspace{1cm} k

\[ H_2[k] \]

-128 \hspace{1cm} -64 \hspace{1cm} -32 \hspace{1cm} 0 \hspace{1cm} 32 \hspace{1cm} 64 \hspace{1cm} 128 \hspace{1cm} k

\( H_2[k] \) is now even smoother.
Let’s try it!
Overlap-Add Method

Importantly, we can process the first window without waiting for the entire song to be transmitted – very important for streaming applications.

But, it turns out that this method also tends to be more efficient in normal applications as well!
Each FFT of length $N$ contributes $s$ samples to the output.

Number of windows $= N_x / s$.

Number of multiplies per window $\approx 2N \log_2(N)$
(only need to calculate frequency response once)

Total number of multiplies $\approx 2N_x \frac{N}{s} \log_2(N)$.

Typically $\frac{N}{s}$ is near 1 (it was $\frac{3}{4}$ in today’s example).

Total $\approx 2N_x \log_2(N)$.

Compared to $\approx 3N_x \log_2(N_x)$ for full-length FFTs.