Audio Pre-Processing For Deep Learning

Article · December 2020

CITATION
1

READS
5,087

1 author:

Saman Arzaghi
University of Tehran
2 PUBLICATIONS **1** CITATION

SEE PROFILE

Audio Pre-Processing For Deep Learning

Saman Arzaghi

School of Mathematics, Statistics and Computer Science

College of Science, University of Tehran

samanarzaghi@ut.ac.ir

Dec 2020

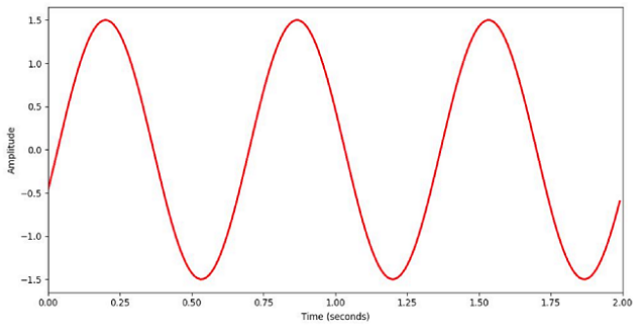
Abstract

Audio signals are continuous (analog) signals that gradually decrease in amplitude as the sound source decreases. Computers, on the other hand, store their data digitally stream strings of bits zero and one. Digital data is naturally discrete because the value of zero or one digital data is only valid at a given moment. Therefore, the continuous analog audio signal must be converted to a discontinuous digital form so that the computer can store or process audio. Of course, the digital data must be converted back to analog form so that it can be heard through an audio system. Two-way conversion between analog and digital signals is the primary operation of all adapter cards and sound cards.

In this article, we will discuss different ways to represent audio (like Waveform, FFT, STFT, and MFCC), the difference between them, How to turn each into another, and some codes for each representation.

1 What is Sound?

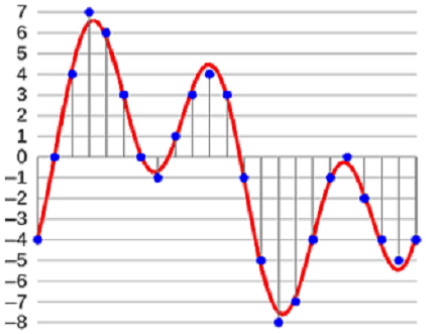
Well, sound produce when an object vibrates and those vibrations determine the oscillation of air molecules which basically creates an alternation of air pressure and this high pressure alternated with low pressure causes a wave and we can represent this wave using wave form.



Simple waveform sound

When we talk about sounds around us like our own voice or the sound of the whistle, we are talking continuous waveforms and they are analog, but obviously, we can't really store analog waveforms we need an away of digitalizing them, and for doing that we can use analog-digital conversion (ADC) process and for that we must perform two steps:

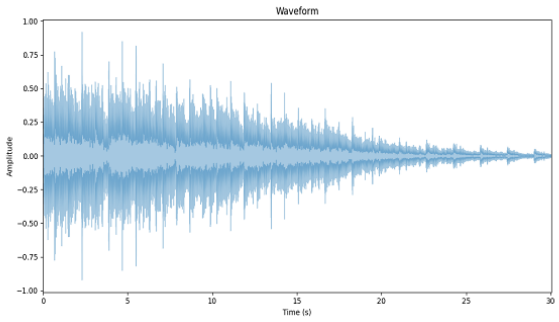
- **Sampling** $\longrightarrow$ Sample the signal at specific time intervals
- **Quantization** $\longrightarrow$ Quantize the amplitude given and represent with a limited number of bits (note that more bits we store the amplitude and-the better quality of sound will be)



A sound wave, in red, represented digitally, in blue
(after sampling and 4-bit quantisation)

2 Start Working With an Audio File

Ok, now let's take a look at the ending solo on High Hopes by Pink Floyd, which is indeed my favorite. This shows us the loudness (amplitude) of sound wave changing with time. Here amplitude = 0 represents silence. (From now on, we will analyze this audio):



Plot audio wave in time domain
(This amplitude is actually the amplitude of air particles which are oscillating
because of the pressure change in the atmosphere due to sound)

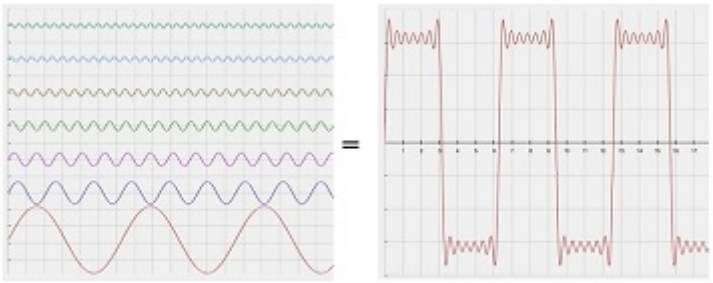
You can use the Python code below to extract waveform from a raw file.wav and then show the plot:

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)

file = "highhopes.wav"

# Load audio file with Librosa
```

```
signal, sample_rate = librosa.load(file, sr=22050)

# DISPLAY WAVEFORM
def waveform():
    plt.figure(figsize=FIG_SIZE)
    librosa.display.waveplot(signal, sample_rate, alpha=0.4)
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.title("Waveform")
    plt.show()
```

3 Fourier Transform

If you take look at the plot audio wave in time domain, it looks so complex to understand, but nature has given us an incredible way of knowing quite a lot about complex sounds, and that's given through a Fourier transform (FT) . A Fourier transform is decomposing complex periodic sound into a sum of sine waves oscillating at different frequencies.

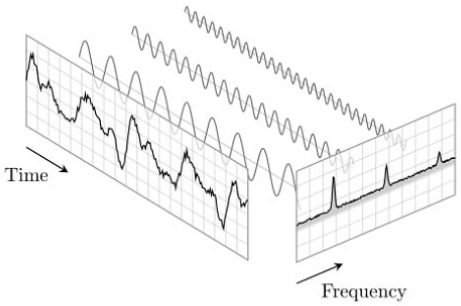


FT decomposes complex audio to simpler ones

It is great because now we can decomposed complex sounds into simpler ones and analyze them.

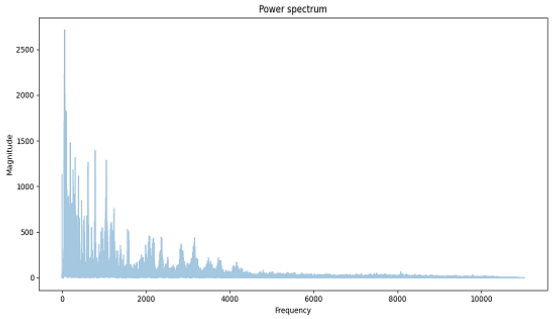
4 Fast Foureir Transform

The "Fast Fourier Transform" (FFT) is an important measurement method in the science of audio and acoustics measurement. It converts a signal into individual spectral components and thereby provides frequency information about the signal. FFTs are used for fault analysis, quality control, and condition monitoring of machines or systems. you might be wondering what is difference between fft and ft, so the answer is that The Fast Fourier Transform is a particularly efficient way of computing a DFT (**Discrete Fourier Transform (DFT)** is the discrete version of the Fourier Transform (FT) that transforms a signal (or discrete sequence) from the time domain representation to its representation in the frequency domain) Whereas, FFT is any efficient algorithm for calculating the DFT and its inverse by factorization into sparse matrices). it seems a little confusing because it is , but you don't need to get that much deep into details although if you need more details I suggest you take a look at Wikipedia.



How FFT works

FFT analysis for the same song that we analyzed earlier is shown below:



Plot FFT of the song

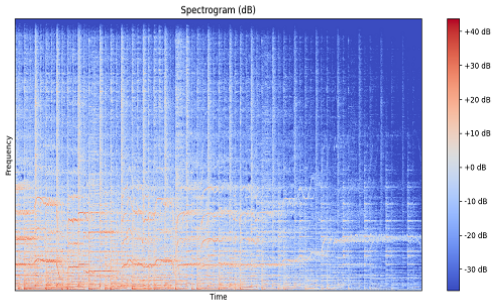
You can use the Python code below to extract FFT from a raw file.wav and then show the plot.

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)

file = "highhopes.wav"

# Load audio file with Librosa
signal, sample_rate = librosa.load(file, sr=22050)

# DISPLAY FFT TO POWER SPECTRUM
def fft():
    # Lerform Fourier transform
    fft = np.fft.fft(signal)
    # Lalculate abs values on complex numbers to get magnitude
    spectrum = np.abs(fft)
    # Create frequency variable
    f = np.linspace(0, sample_rate, len(spectrum))
    # Take half of the spectrum and frequency
    left_spectrum = spectrum[:int(len(spectrum)/2)]
    left_f = f[:int(len(spectrum)/2)]
    # Plot spectrum
    plt.figure(figsize=FIG_SIZE)
    plt.plot(left_f, left_spectrum, alpha=0.4)
```

```
    plt.xlabel("Frequency")
    plt.ylabel("Magnitude")
    plt.title("Power spectrum")
    plt.show()
```

5 Short-Time Fourier Transform

Note that when we do Fourier transform basically we move from the time domain to the frequency domain and because of it we lose information about time. at first, it seems we lost a lot of information but there is a solution to that, and it's called short-time Fourier transform (STFT) given through a Fourier transform. if you're interested how does the plot looks like, take a look at the below figure.



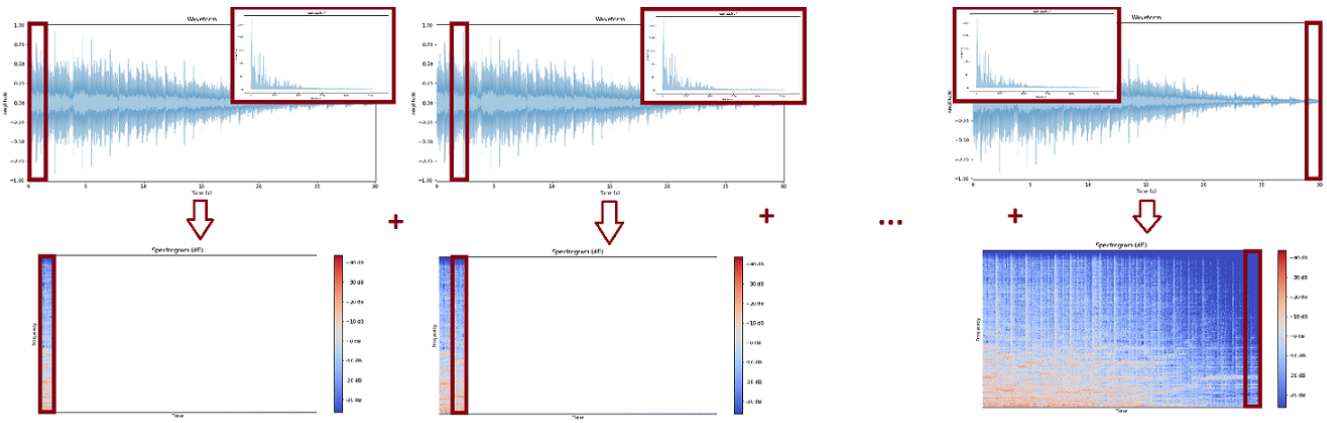
Plot FTFT of the song

You can use the Python code below to extract STFT from a raw file.wav and then show the plot.

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)

file = "highhopes.wav"

# load audio file with Librosa
signal, sample_rate = librosa.load(file, sr=22050)
# DISPLAY STFT TO SPECTROGRAM
def stft():
    hop_length = 512 # In num. of samples
    n_fft = 2048 # Window in num. of samples
    # Calculate duration hop length and window in seconds
    hop_length_duration = float(hop_length)/sample_rate
    n_fft_duration = float(n_fft)/sample_rate
    print("STFT hop length duration is: {}s".format(hop_length_duration))
    print("STFT window duration is: {}s".format(n_fft_duration))
    # Perform stft
    stft = librosa.stft(signal, n_fft=n_fft, hop_length=hop_length)
    # Calculate abs values on complex numbers to get magnitude
    spectrogram = np.abs(stft)
    # Display spectrogram
    plt.figure(figsize=FIG_SIZE)
```

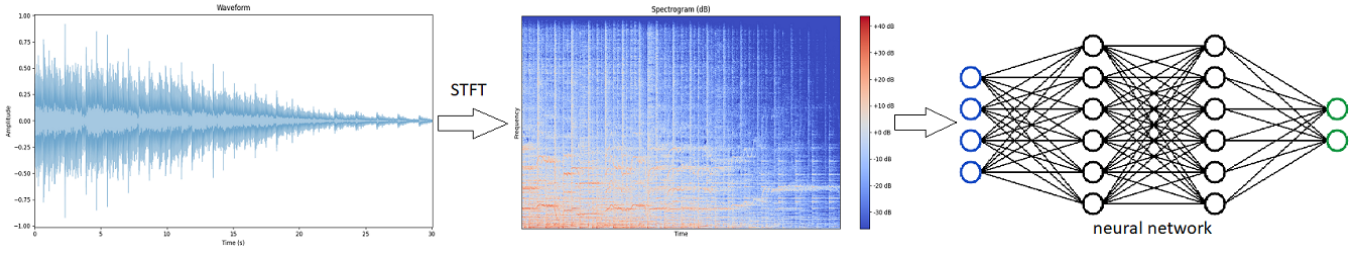
```
    librosa.display.specshow(spectrogram, sr=sample_rate, hop_length=hop_length)
    plt.xlabel("Time")
    plt.ylabel("Frequency")
    plt.colorbar()
    plt.title("Spectrogram")
    plt.show()
    # Apply logarithm to cast amplitude to Decibels
    log_spectrogram = librosa.amplitude_to_db(spectrogram)
    plt.figure(figsize=FIG_SIZE)
    librosa.display.specshow(log_spectrogram, sr=sample_rate, hop_length=
hop_length)
    plt.xlabel("Time")
    plt.ylabel("Frequency")
    plt.colorbar(format="%+2.0f dB")
    plt.title("Spectrogram (dB)")
    plt.show()
```

But let's say how does it work and how we can build STFT. It computes several Fourier transforms at different intervals and in doing so it preserves information about time and the way sounds evolved it's like over time. So the different intervals at which we perform the Fourier transform is given by the frame size and so a frame is a bunch of samples and so we fix he number of samples and we are giver spectrogram.



How does STFT works

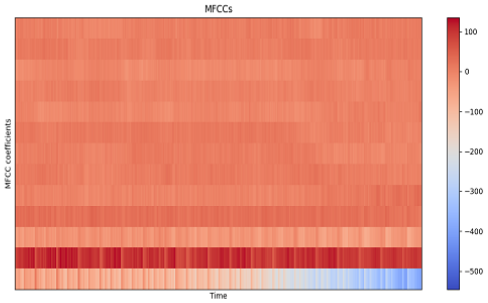
Now we may be wondering why did we have to learn about spectrogram because spectrograms are fundamental for performing like deep learning like applications like on audio data the whole pre-processing pipeline for audio data for deep learning is based on spectrograms. So we can pass wave form audios into state and we get spectrogram and we use spectrogram as an input for our deep learning model.



Data conversion steps (using STFT)

6 Mel Frequency Cepstral Coefficient

Let's introduce another feature that is fundamental and as important as spectrogram for deep learning, it called Mel Frequency Cepstral Coefficient (MFCC). MFCCs capture of timbral/textural aspects of sound. it means if you have for example a piano and a violin playing the same melody you would have potentially like the same peach context and the same frequency but what would change is the quality of sound but MFCCs are capable of capturing the information and the differences. the below figure show us how does plot MFCC looks.



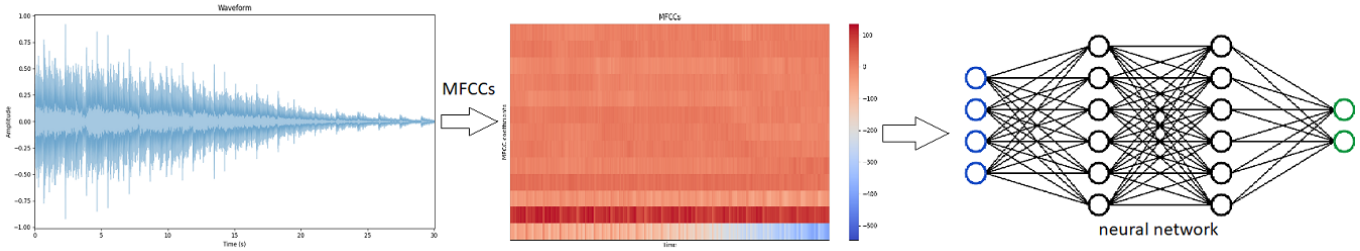
Plot MFCC of the song

You can use the Python code below to extract MFCC from a raw file.wav and then show the plot.

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)

file = "highhopes.wav"

# load audio file with Librosa
signal, sample_rate = librosa.load(file, sr=22050)
# DISPLAY MFCCs
def mfccs():
    hop_length = 512 # In num. of samples
    n_fft = 2048 # Window in num. of samples
    # Extract 13 MFCCs
    MFCCs = librosa.feature.mfcc(signal, sample_rate, n_fft=n_fft, hop_length=
   hop_length, n_mfcc=13)
    # Display MFCCs
```

```
    plt.figure(figsize=FIG_SIZE)
    librosa.display.specshow(MFCCs, sr=sample_rate, hop_length=hop_length)
    plt.xlabel("Time")
    plt.ylabel("MFCC coefficients")
    plt.colorbar()
    plt.title("MFCCs")
    plt.show()
```

And the way we use MFCCs is the same as how we use STFT.



Data conversion steps (using MFCC)

7 What Did We Use to Do?(optional)

In the past, traditional machine learning pre-processing for audio used to be so much different. let's take a look at that : we can take a lot of information from waveforms, for example, we could use waveforms extracting time-domain features and use spectrogram extracting frequency domain features. so we start from the waveform and extract those features we wanted and we combine these features and use it in machine learning algorithms like logistic regression or super vector machine. fortunately, with advanced deep learning, the whole process becomes a little bit more straight forward. After all, we don't need to see that much feature engineering because we use spectrogram. this is why deep learning models like in this case for audio is called end-to-end model cause you just use some basic information without worrying to much about extracting specific features

References

[1] Richard G. Lyons. Understanding Digital Signal Processing. 3rd ed. Pearson. 2010.

[2] The Sound of AI, https://www.youtube.com/channel/UCZPFjMe1uRSirmSpznqvJfQ?pbjreload=102

[3] Understanding Audio data, Fourier Transform, FFT and Spectrogram features for a Speech Recognition System, https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520

[4] Understanding the Fourier Transform by example, https://www.ritchievink.com/blog/2017/04/23/understanding-the-fourier-transform-by-example/