Speech Recogniton 첫걸음

조희철

2018년 8월 15일

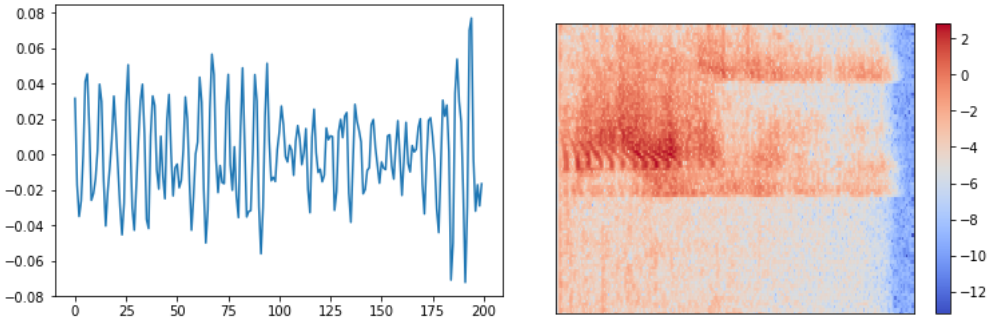
1 From Speech Waveform To MFCC

- 음파를 다루는 python library로 librosa가 있다. wav파일을 가공하기 위해서는 librosa를 이용하거나 tensorflow 내에 있는 api를 이용하면 된다. data는 $[-1, 1]$의 값을 가진다.

```
audio_file = '00b01445_nohash_0.wav'
y, sr = librosa.load(audio_file,sr=16000) 또는
from tensorflow.python.ops import io_ops
from tensorflow.contrib.framework.python.ops import audio_ops

wav_loader = io_ops.read_file(audio_path)
wav_decoder = audio_ops.decode_wav(wav_loader, desired_channels=1,desired_samples = sr)
```

- wav파일이나 mp3파일을 load하면 그림(1a)과 같은 1차원 data가 얻어진다. load할 때는 초당 몇개의 sampling을 할 것인지 sampling rate을 지정하여야 한다. 보통 16,000정도를 사용한다. 16,000은 사람의 가청 주파수를 고려하여 선택된 수이다.



(a) 음파의 일부 (b) log magnitude spectrogram

그림 1: 음파와 spectrogram

- 음파 data는 분석하기 위해서는 data를 window size(frame length) 크기로 stride(frame step) 만큼씩 이동하면서 자르고, 자른 data를 discrete Fourier Transformation 한다. 이렇게 하는 것을 stft(short time Fourier transform) 이라고 한다. 이때 output의 크기를 결정할 fft-length를 정해주어야 한다(함수에 따라서는 fft-length를 지정하지 않고, window size보다 같거나 큰 2^n(fft-length)을 찾고 $2^{n-1}+1$로 정하기도 한다. 같거나 큰 2^n을 찾는 것은 bit_length()를 이용하면 된다.).

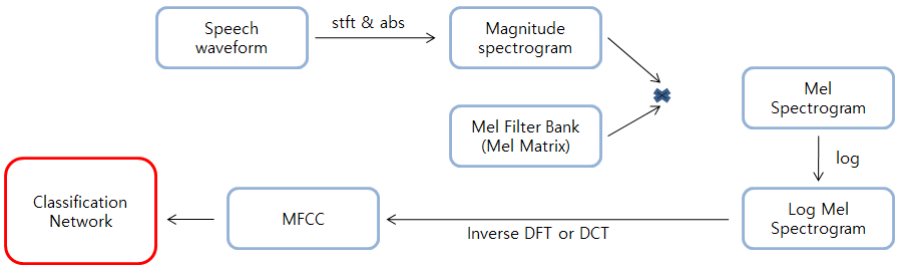


그림 2: 전체적인 모델 구조. wave data를 MFCC형태로 변환 후, Neural Network에 넣어서 분류작업을 수행한다. MFCC는 이미지와 동일한 2차원 array이므로 convolution layer를 가지는 분류 모델을 사용할 수 있다.

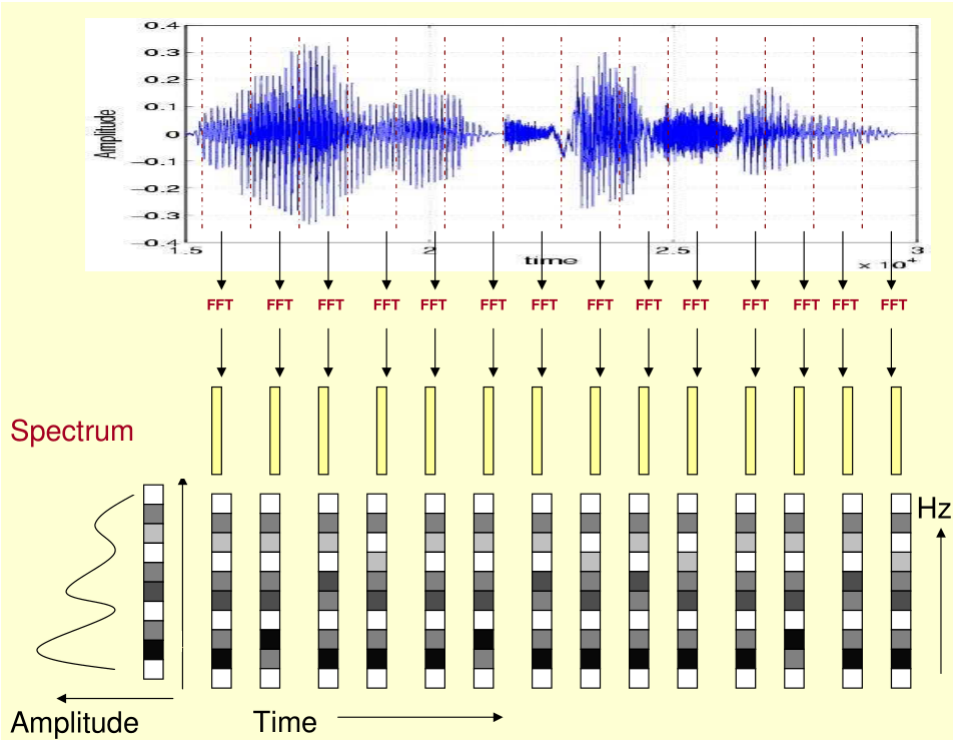


그림 3: 음파를 Discrete Fourier Transform하여 Spectrogram을 만드는 과정. 그림에서는 window size와 stride가 동일하여 겹치지 않게 Fourier Transform하는 것처럼 그려졌지만, window size보다 stride를 작게하여 Fourier Transform한다.

```
y: data, length=16,000
--> stft(window size = 480, stride = 160, fft length = 512)
--> (98, $\frac{\text{fft length}}{2}+1$)
```

- stft를 하면, complex number로 된 2차원 array가 구해진다. 이렇게 구해진 frequency spectrum을 시간 축과 함께 시각적으로 표현한 것을 spectrogram이라 한다. complex number로 된 data를 spectrogram으로 나타내기 위해서 absolute를 취한다. absolute를 취한 것을 magnitude spectrogram, 제곱한 것은 power spectrogram이라 한다.
- stft를 계산하는 함수로 전달되는 input signal array는 시간 정보를 내부적으로 가지고 있다고 가정한다.

sampling 간격이 시간이라고 보면된다. 예를 들어 다음 두 함수를 보자.

$$\begin{aligned} y_1 &= \sin(1000 \times 2\pi x) + 0.5\sin(3500 \times 2\pi x), \\ y_2 &= \sin(2000 \times 2\pi x) + 0.5\sin(7000 \times 2\pi x) \end{aligned}$$

y_2에서 y_1보다 2배 조밀하게 sampling 한다면, sampling된 결과 array는 동일하게 된다. 이렇게 되면, sampling된 결과만으로 y_1, y_2를 구별할 수가 없다. stft는 sampling된 결과 array를 입력으로 받기 때문에, 이 array 속에 시간 정보가 내재되어 있다고 보는 것이다[1].

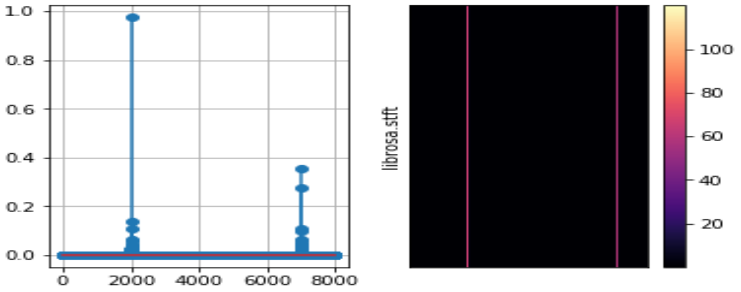


그림 4: $y = \sin(2000 \times 2\pi x) + 0.5\sin(7000 \times 2\pi x)$에 대하여 DFT, librosa.stft를 적용한 결과. stft의 frequency 정보는 상대적인 frequency이지 절대적인 것이 아니다.

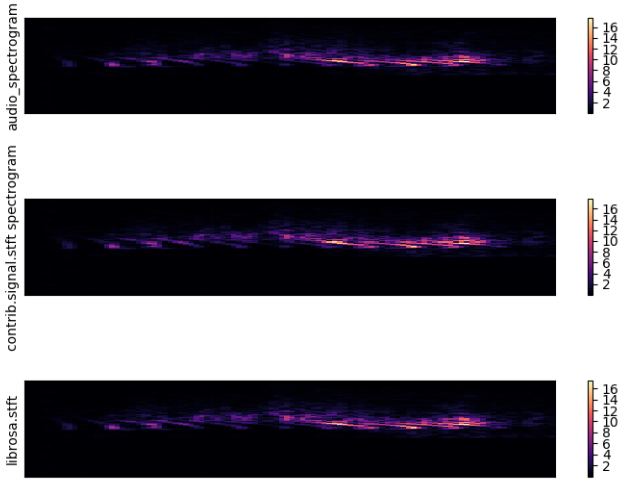


그림 5: stft를 구하는 3가지 방식(가로축이 frequency, 세로축이 time). 위에서 부터 audio_ops.audio_spectrogram, tf.contrib.signal.stft, librosa.stft를 이용한 결과임. 위 2가지는 수치적인 차이를 제외하면 완전히 일치하고, librosa를 이용한 것은 return 되는 array size도 약간 차이나고 결과값도 완전 일치하지는 않고 거의 유사하다.

- stft의 결과로 얻어지는 magnitude spectrum의 shape이 (N, M)이라고 하면, N은 input speech wave data의 길이, window size, stride에 의해 결정되고, M(fft unique bins)은 fft length에 의해

  $$M = \frac{\text{fft length}}{2} + 1$$

  로 계산된다. fft length를 입력받지 않고 stft를 계산하는 함수는 window size에 의해 내부적인 공식으로 결정한다.

- 주파수별 energy를 잘 나타내기 위해서는 log를 취해서 scale을 변경하기도 하고, mel scale이라는 방식으로 scale을 바꾸기도 한다.

[1] scipy.fftpack.fft는 2번째 argument로 넘어가는 n이 시간축을 조절하는 역할을 한다.

- 인간의 귀는 낮은 주파수대에서 높은 resolution을 가지고, 높은 주파수대에서 낮은 resolution을 갖는다. 이러한 이유로 주파수를 log scale 변환이 필요하다. mel scale은 frequency f를 다음과 같은 공식을 이용해서 m으로 바꾼다. 그런데, mel scale formula는 여러 가지가 있다.

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

- 이제 mel scale로 변화하는 구체적인 방법을 살펴보자. magnitude spectrogram$(N \times M)$을 mel filter bank(mel matrix, $M \times K$)와 곱하여 mel spectrogram$(N \times K)$을 만든다. 여기서 K는 filter bank를 생성할 때 지정하는 mfcc의 갯수이다.

```
sr = 16000
n_mfcc = 40
librosa_mel_matrix =
    librosa.filters.mel(sr=sr,n_fft=512,n_mels=n_mfcc,fmin=80.0,fmax=7600.0,norm=1) #
    (n_mfcc,n_fft//2 +1)

num_spectrogram_bins = 257
tf_mel_matrix = tf.contrib.signal.linear_to_mel_weight_matrix(num_mel_bins=n_mfcc,
    num_spectrogram_bins=num_spectrogram_bins, sample_rate=sr, lower_edge_hertz=80.0,
    upper_edge_hertz=7600.0) # (257, 40)
```

- 이제, log를 취하면 log mel spectrogram이 된다.

- Mel-frequency cepstral coefficients(MFCCs): log mel spectrogram에 Discrete Cosine Transform(DCT)이나 Inverse DFT를 취하여 MFCC를 구한다. DCT는 실수값으로 계산되기 때문에 더 편리하다. mfcc의 계산은 `tf.contrib.signal.mfccs_from_log_mel_spectrograms(log mel spectrogram)` 또는 `audio_ops.mfcc(squared magnitude spectrogram)`으로 하면된다. 최종적으로 MFCC array는 $N \times K$가 된다.

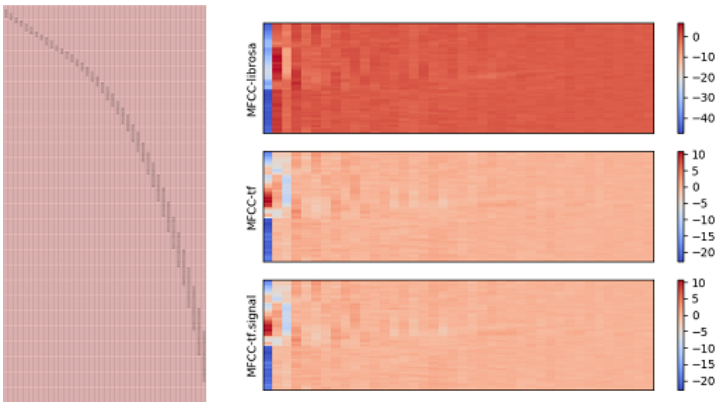


그림 6: Mel Filter Bank: sparse matrix인데, tensorflow의 tf.contrib.signal.linear_to_mel_weight_matrix와 librosa.filters.mel은 scale이 다른 결과를 준다. Filter Bank가 다르기 때문에, Spectrogram도 달라지고 계속해서 MFCC까지 다른 결과를 낸다.

- (참고) 지금까지 stft을 이용하여 spectrogram을 구하고 mfcc까지 계산하는 방식을 살펴보았다. mfcc로 feature를 구하지 않고, 단순 average로 feature를 구하는 방법도 있다. 먼저 구해진 squared spectrogram의 shape이 $(N, 98, 257)$이라고 하자. 앞에서 예로 mfcc를 40로 설정했는데, 비슷한 정도의 averaging data를 구하려면 $int(257/40) = 6$개씩 average하면 된다. 실제 6개씩 average pooling하면 output의 shape은 $(98, 43)$이 된다.

```
# to use tf.nn.pool, we have to make spectrogram 4 dimensional. (N,98,257,1)
output = tf.nn.pool(tf.expand_dims(spectrogram, -1), window_shape=[1,6], strides=[1,6],
    pooling_type='AVG', padding='SAME')
```

2 Model Training

- tensorflow의 Simple Audio Recognition[2]을 바탕으로 살펴보자. 먼저 data를 준비하는 단계부터 보자.
- 지금까지 audio 파일로 부터 mfcc형태로 data를 변환하는 과정을 살펴보았다. 실제 training에서는 audio data에 volume 조절, time shift, backgroud sound 추가를 통해, audio data를 변형한 후 mfcc로 변환한다. 이런 변형 과정에서 time shift 정도, background sound(noise) 추가 등이 random하게 이루어지기 때문에 data를 미리 만들어 놓을 수는 없다.
- 이 모델은 35개의 단어에 대한 1초짜리 wav파일을 분류하는 모델이다. 각 단어는 평균 3000개 정도의 파일을 가지고 있다. 모델은 35개의 단어 중 10개 정도를 골라 분류하는 방식인데, 나머지 단어들은 'unknown'으로 분류한다. 'unknown'으로 분류된 것들은 그 중 일부만(e.g. 10%) training에 사용된다.
- `'_silence_'`: 모델이 관심을 두지 않을 수 있는 항목을 추가한다. silence label data는 완전히 소리를 없애는 것은 아니고, background noise만 더해서 생성한다.
- 지정한 단어 10개에 unknown, silence를 더해 모두 12개 항목을 분류대상으로 한다.

	training(80%)	validation(10%)	test(10%)
yes(2) no(3) up(4) ⋮ go(11)	30,769개	3,703개	4,074개
`'_silence_'`(0)	3,077개	371개	408개
`'unknown'`(1)	3,077개	371개	408개

표 1: training에 사용하는 audion data 구분

- audio data 가공
  - audio 파일로 부터 sampling rate/sec(=16000)를 지정하여 data를 읽어온다. 우리가 사용하는 audio file은 1초짜리이므로 data의 shape은 $(16000,)$이 된다.
  - `` `_silence_' `` flag로 지정된 data는 0을 곱해서 volume을 0으로 만든다. 다음 단계에서 backgroud noise만 더해진다.
  - time shift: $(-100, 100)$ ms $= (-1600, 1600)$ sampling point 사이에서 random하게 time shift amount를 정해서 이동시킨다. 이동 후 빈공간은 0으로 padding한다.
  - background noise 추가: `background_frequency(0.8)` 확률로 background noise를 추가할지 결정

[2] `https://www.tensorflow.org/tutorials/sequences/audio_recognition`

한 후, `0~background_volume(0.1)` 사이의 값으로 volume을 조절하여 background noise를 생성한다. background noise는 data길이가 긴데, 16,000 sampling point(=1초)만큼만 random하게 선택한다. 생성된 background noise를 time shift 처리된 audio data에 더한 후, $[-1, 1]$ 사이의 값으로 clip한다. 여기까지 처리하면, mfcc 형태로 변환하기 전 단계의 data가 된다.

- Model: audio 파일에 대한 분류 모델. audio data를 2 dimension data $(98, 40)$으로 변형했기 때문에 이미지와 동일하게 취급할 수 있다. 여기서는 5개의 모델을 제시하고 있다.
  - Single FC: 1 layer FC 모델로 sanity check 목적.
  - Standard Convolution: 가장 정확도가 높은 모델로 conv2d-relu-dropout-maxpooling-conv2d-relu-dropout-fc(12)-softmax로 되어 있다[3].

```
Standard Convolution:
mfcc data: $(N,98,40)$ -> reshape $(N,98,40,1)$
-> conv2d(f=$64$,k=(20,8),s=1,p='s') $(N,98,40,64)$
-> relu -> dropout(0.5) -> max pooling(k=[1, 2, 2, 1], s=[1, 2, 2, 1]) $(N,49,20,64)$
-> conv2d(f=$64$,k=(10,4),s=1,p='s') $(N,49,20,64)$
-> relu -> dropout(0.5)
-> flatten $(N,62720)$
-> FC(12) -> softmax
```

  - Tiny Convolution: micro controllers같은 소형 device에 사용할 수 있는 모델.

```
Tiny Convolution:
mfcc data: $(N,98,40)$ -> reshape $(N,98,40,1)$
-> conv2d(f=$8$,k=(10,8),s=2,p='s') $(N,49,20,8)$
-> relu -> dropout(0.5)
-> flatten $(N,7840)$
-> FC(12) -> softmax
```

  - Low Latency Convolution[4]: multiplication 횟수와 parameter 갯수를 줄이는 CNN architecture를 찾는 것이 이 방법의 핵심이다. convolution layer를 1개만 사용한 모델이다. time에 대한 kernel size를 크게 하는 것이 특징이다.

```
Tiny Convolution:
mfcc data: $(N,98,40)$ -> reshape $(N,98,40,1)$
-> conv2d(f=$186$,k=(98,8),s=1,p='v') $(N,1,33,186)$
-> relu -> dropout(0.5)
-> flatten $(N,6138)$
-> FC(128) -> dropout (-> relu) * 구현코드에 relu 가빠져있다 .
-> FC(128) -> dropout (-> relu)
-> FC(12) -> softmax
```

[3] 코드의 comment에는 두번째 max pooling이 적혀 있는데, 구현에서는 빠져 있음.

[4] http://www.isca-speech.org/archive/interspeech_2015/papers/i15_1478.pdf

– Low Latency SVDF[5]: We propose a scheme to compress an existing fully-trained DNN using a low-rank approximation of the weights associated with individual nodes in the first hidden layer by means of a rank-constrained DNN layer topology. This allows us to significantly reduce the number of independent parameters in the model, without any loss in performance. SVD는 singular value decomposition 인것 같은데, F가 뭔지는 논문에 나오지도 않는다.

[5] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43813.pdf