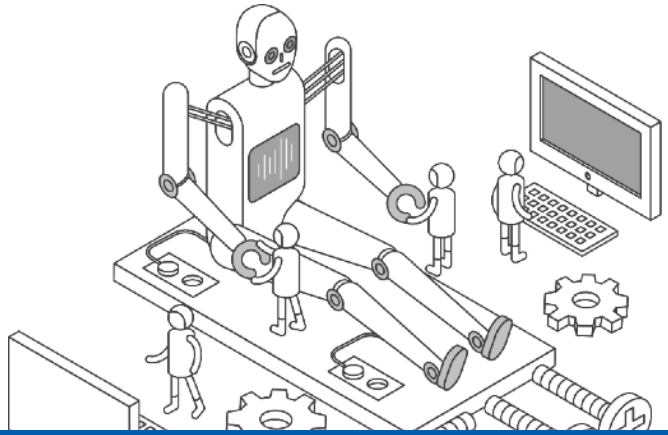
2021 직업계고 AI 전문교육

ARTIFICIAL INTELLIGENCE
BIG DATA
SMART FACTORY

AI·빅데이터 심화과정

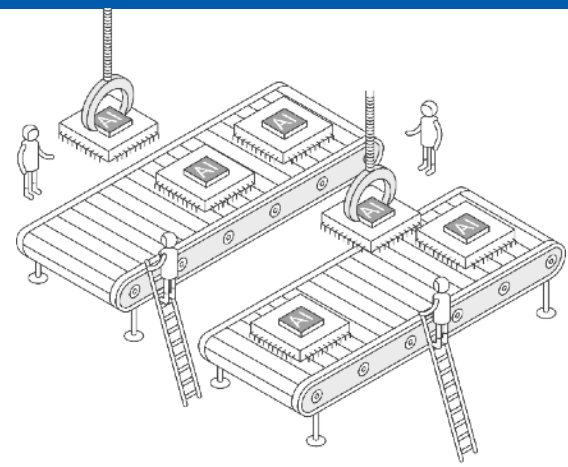
RNN으로 음성 인식하기

박수철
github.com/scpark20
GaudioLab, 모두의연구소



서울특별시교육청 kocoafab (주)네패스 코코아랩 모두의 연구소

RNN으로 음성 인식하기



Seq2Seq
9/24 - RNN으로 소설쓰기 (Aiffel 외)
9/29, 10/1 - 26. 뉴스 요약봇 만들기
10/6, 10/8, 10/15 - 27. 트랜스포머로 만드는 대화형 챗봇

CNN/GAN
10/20, 10/22 - 22. 난 스케치를 할테니 너는 채색을 하거라
10/27, 11/3 - 21. 흐린 사진을 선명하게

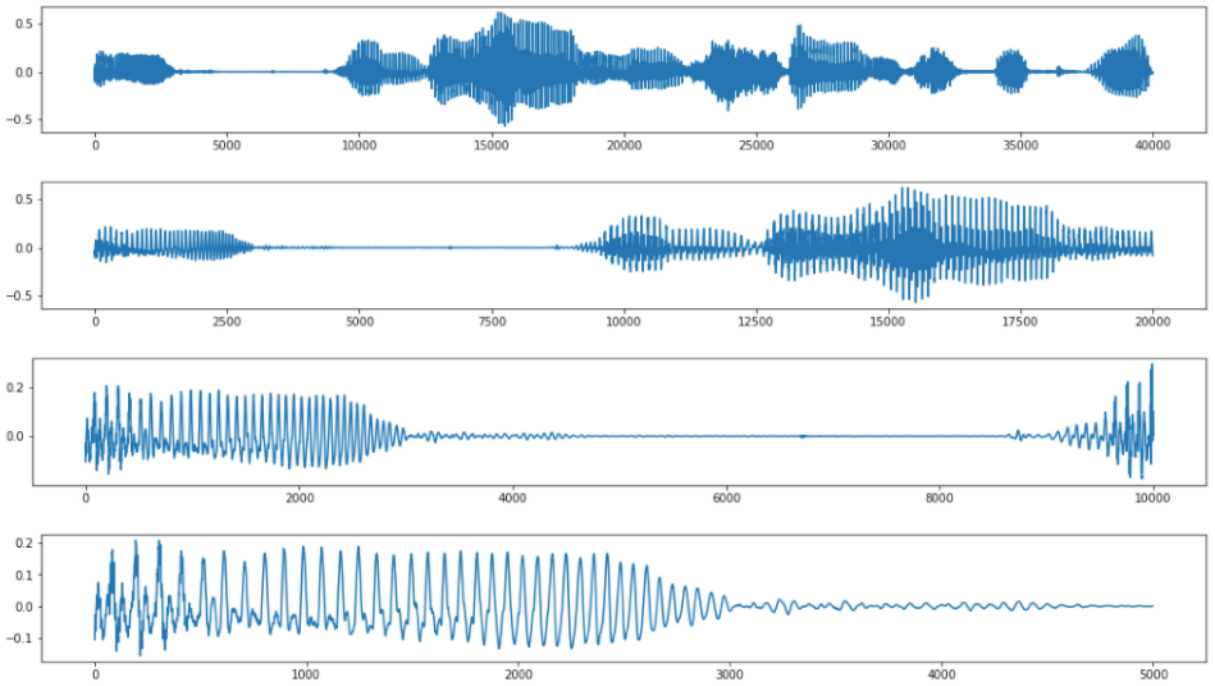
RNN+CNN
11/5 - RNN으로 음성인식하기 (Aiffel 외)
11.10, 11/12 - 19. 직접 만들어보는 OCR

Ablation study
11/17 - 17. 없다면 어떻게 될까?

- 소리는 매질 (공기, 물)의 진동으로 전해지며, 귀는 이러한 진동을 감지하여 뇌로 전달합니다.

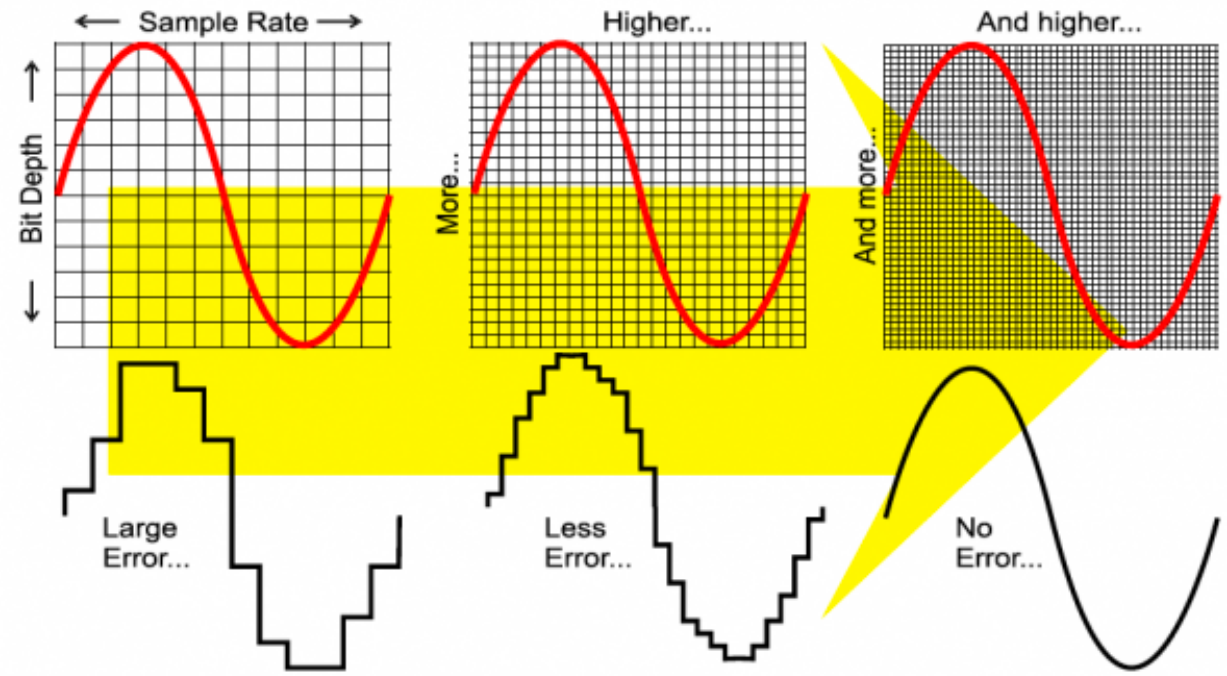
- 이 진동을 디지털화하여 수치로 표현할 것을 wave data라고 합니다.

- 진동의 속성으로는 세기를 나타내는 진폭(amplitude)과 진동의 빠르기를 나타내는 주파수(frequency) 등이 있습니다.



waveform의 예

- Wave data의 속성으로 sampling rate, bit depth가 있습니다.
- Sampling rate는 진동을 1초에 몇번 측정한 것인지를 나타냅니다. 음악은 대개 44.1k, 48k sample rate를 가지고, speech 데이터는 대개 8k, 16k, 22k 등을 가집니다.
- Bit depth는 sampling된 값을 digital로 기록하기 위해 필요한 bit의 수를 나타냅니다. 예를 들어 한 sample을 기록하는데 16bit를 사용한다하면 가장 작은 값을 -32768로 두고, 가장 높은 값을 32767로 두어 -32768-32767의 수로 sample들을 기록합니다.
- 8bit를 사용한다면 -256-255의 값을 사용합니다. 32bit를 써서 float값을 사용하는 경우도 있습니다.



https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html

- 인간은 소리를 들을 때, amplitude정보와 frequency정보를 받아들입니다.
- frequency정보는 wave를 fourier transform하여 얻을 수 있습니다.
- 컴퓨터에서는 discrete정보를 다루므로 discrete fourier transform (DFT)을 이용합니다.
- DFT는 linear transform이고 invertible합니다.

DFT

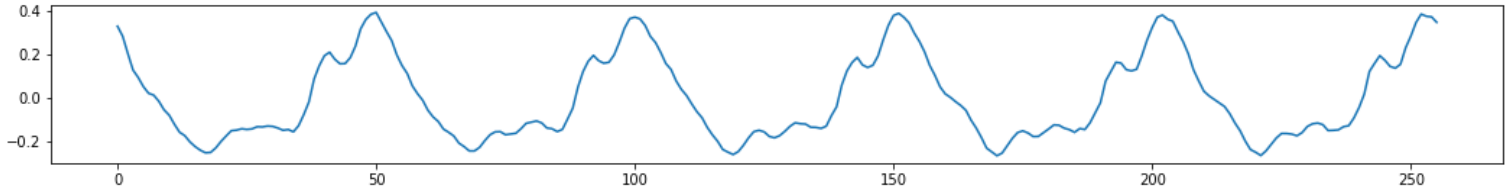
$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn}$$

$$= \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N}kn\right) - i \cdot \sin\left(\frac{2\pi}{N}kn\right)\right]$$

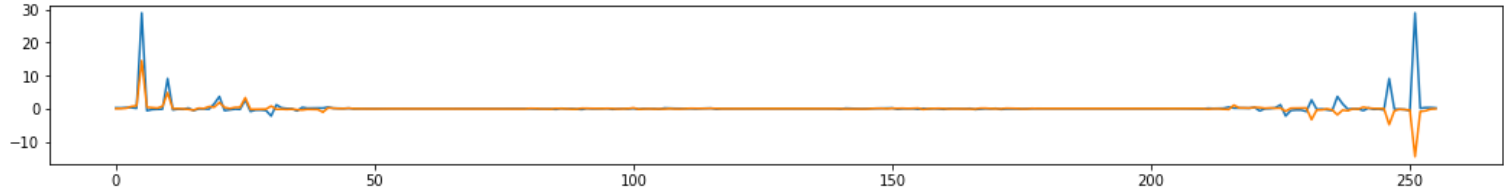
Inverse-DFT

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{i\frac{2\pi}{N}kn}$$

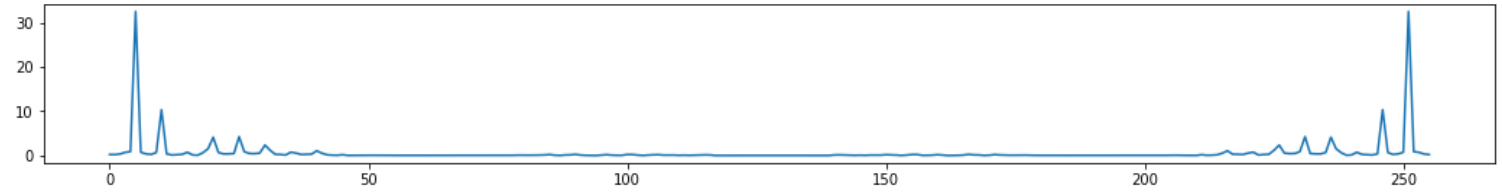
Wave



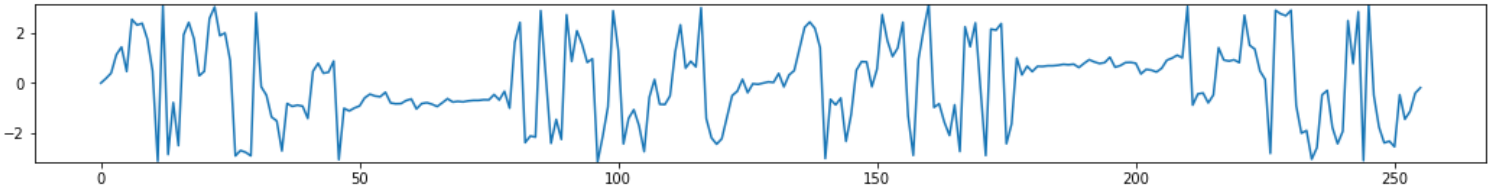
DFT
Real/Imaginary

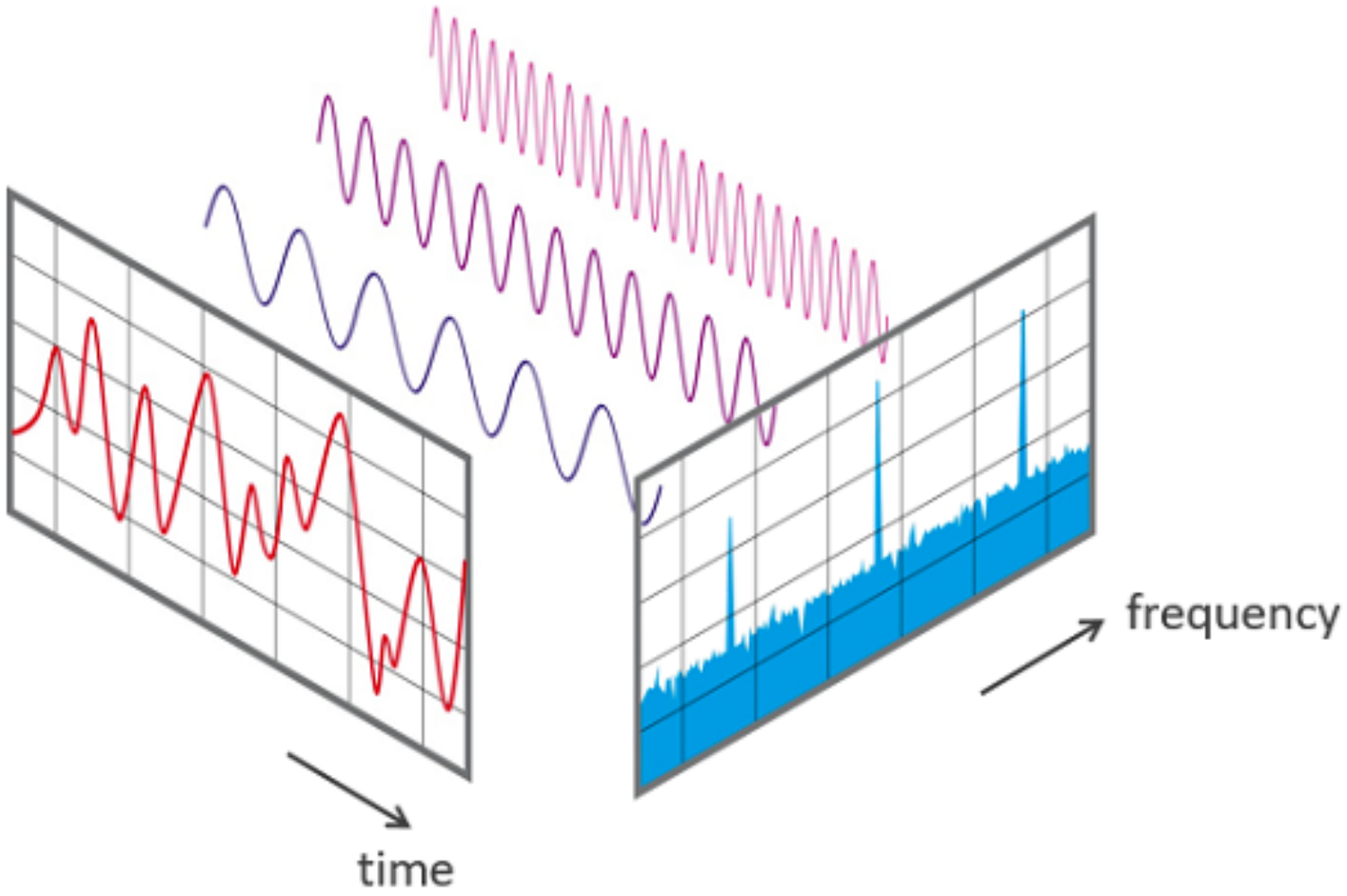


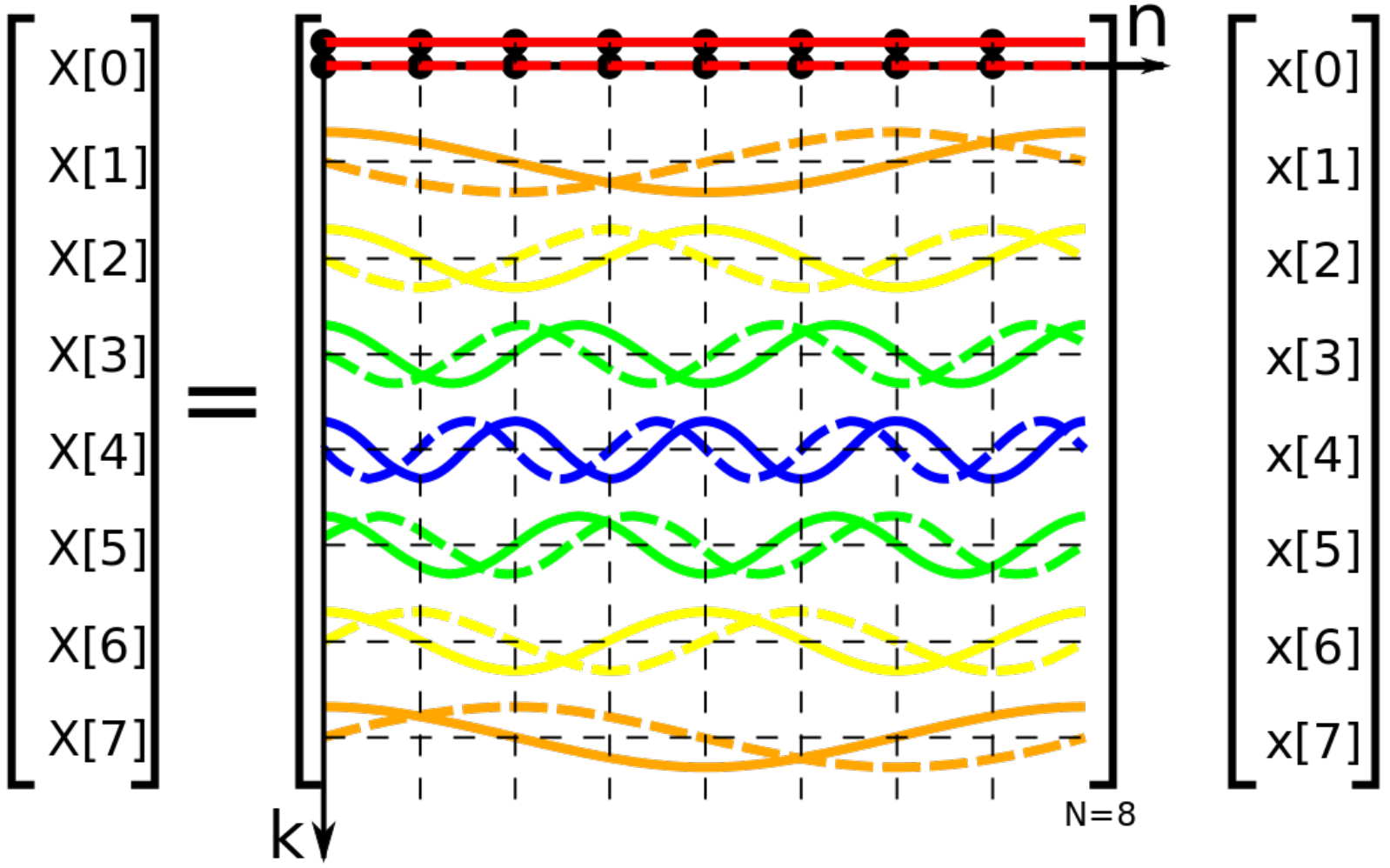
Magnitude



Phase




frequency
time



https://en.wikipedia.org/wiki/DFT_matrix

- DFT는 수행시간이 n개의 sample에 대해 $O(n^2)$이고, 중복된 계산을 줄이는 Fast Fourier transform (FFT)를 사용하면 $O(n \log n)$으로 줄일 수 있어 보통 FFT를 사용합니다.

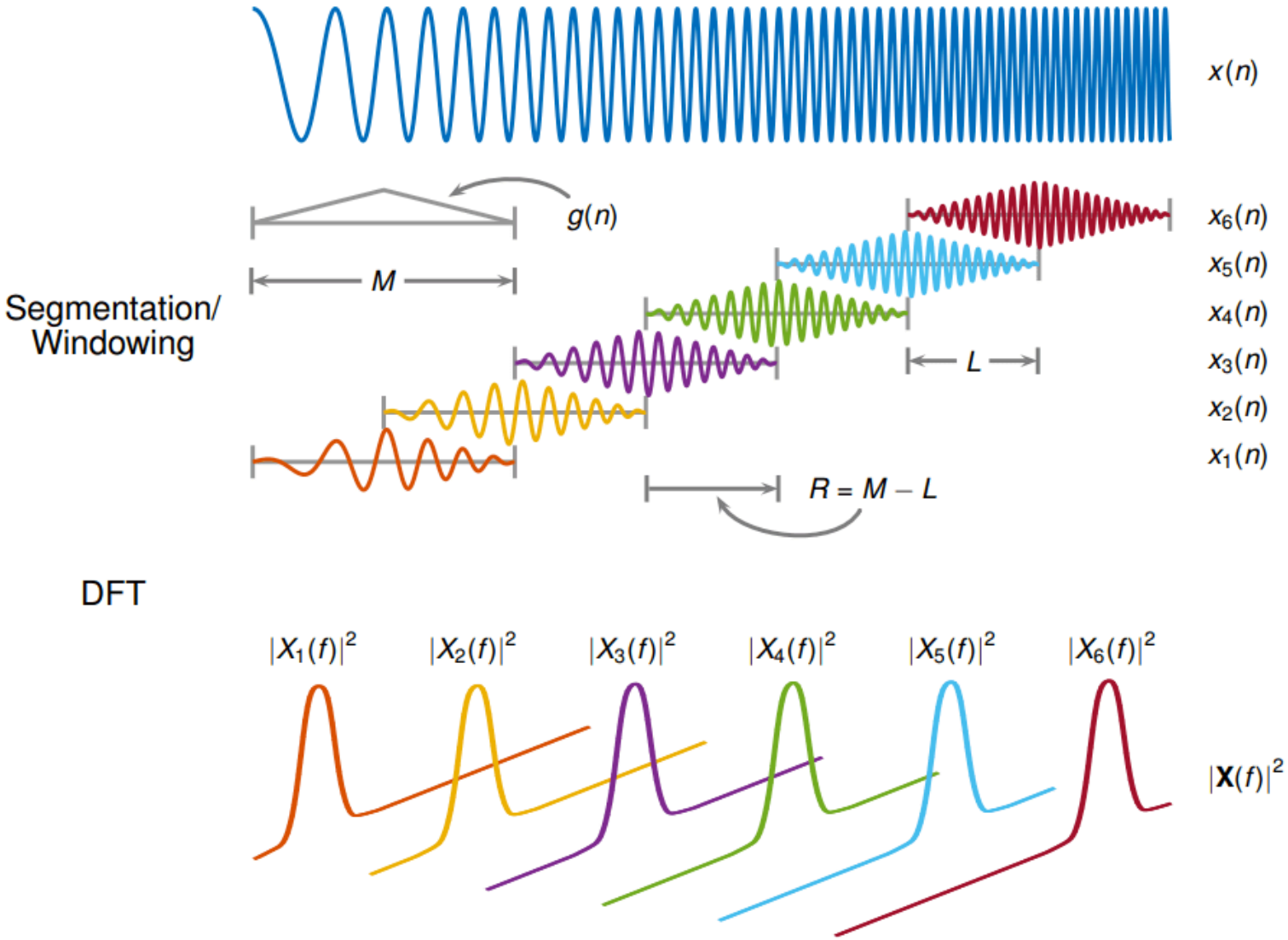
- FFT는 2의 거듭제곱 갯수 (2, 4, 8, 16, 32, 64, 128…)의 sample을 필요로 합니다.

- 음악이나 speech는 매우 긴 wave data이며, 따라서 짧은 시간 단위로 잘라서 FFT로 분석합니다. 이를 Short Time Fourier transform (STFT)라 합니다.

- Wave에 STFT를 적용하여 얻은 결과를 spectrogram이라고 부릅니다. Spectrogram의 결과는 2D complex array인데, 대개 magnitude 값만 취해 visualization하거나 analysis의 대상으로 삼습니다.

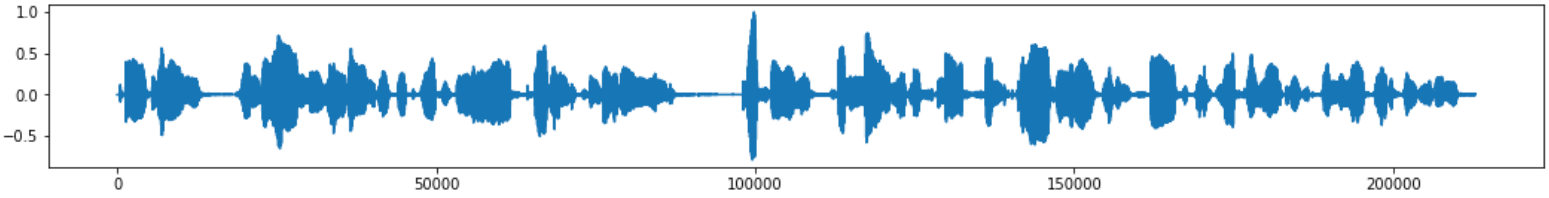
- STFT를 수행하는데는 FFT size, hop size, window function등의 parameter가 있습니다.

- FFT size: FFT할 sample의 갯수
Hop size: FFT window를 움직일 크기 (convolution에서의 stride와 같음)
window function: FFT 수행전에 곱해줄 function



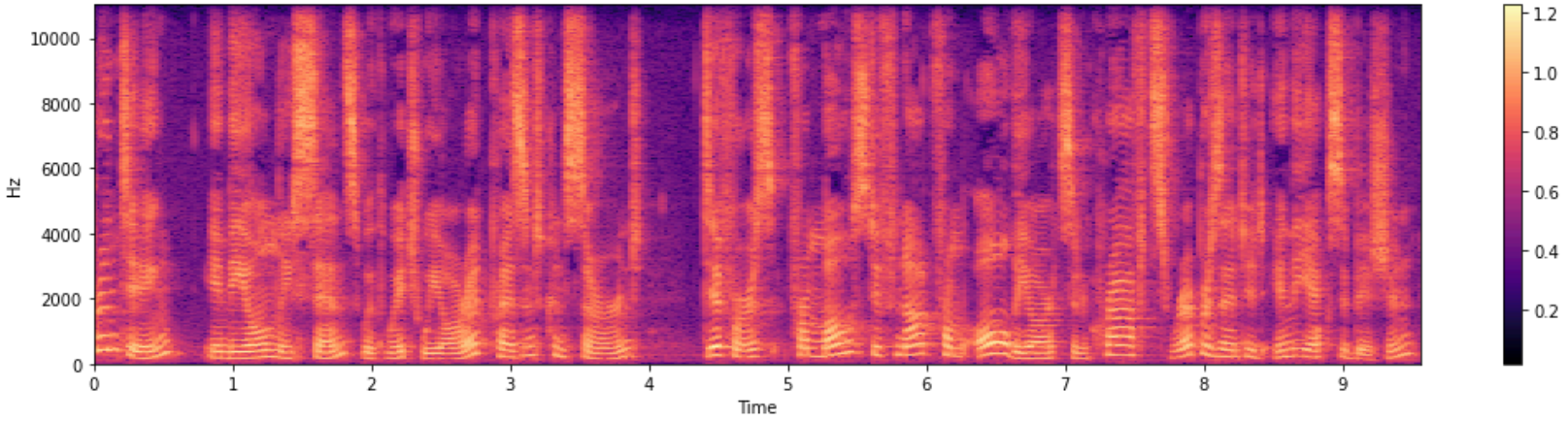
https://www.mathworks.com/help/signal/ref/stft.html

Wave



STFT

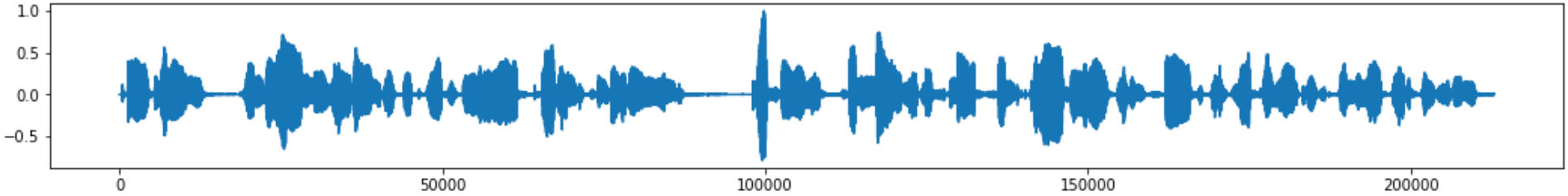
Spectrogram
(magnitude)



Spectrogram Librosa 실습

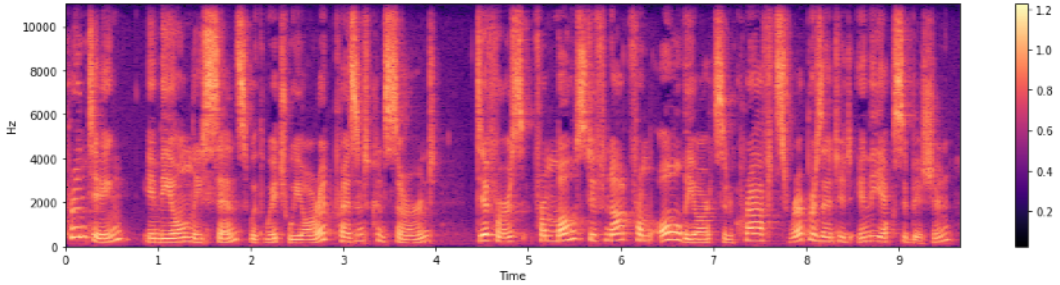
```
# 첫번째 파일 로드
wav, _ = librosa.core.load('LJ001-0001.wav')
# normalizing
wav /= max(abs(wav))
```

librosa.core.load함수로 wav파일을 로드합니다. sample들 중 가장 크기가 큰 값으로 나누어 normalization을 할 수 있습니다.




```
# spectrogram 구하기
spec = librosa.core.stft(wav, n_fft=2048, hop_length=512)
spec = np.abs(spec)
```

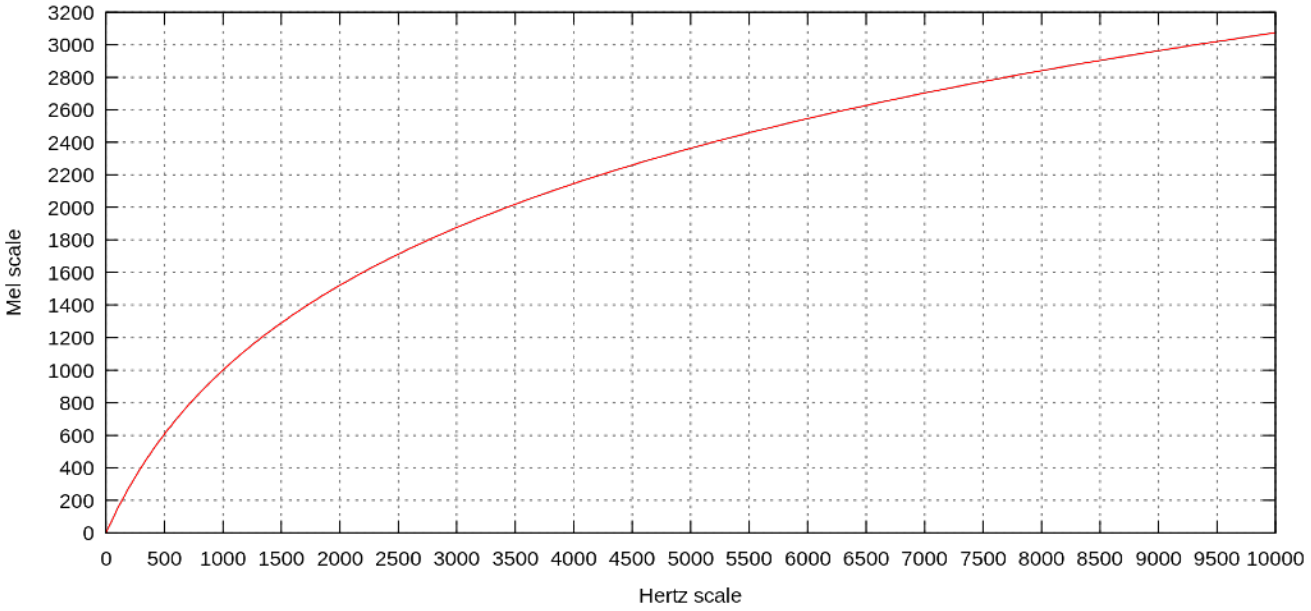
librosa.core.stft함수로 wav를 spectrogram으로 변환합니다. n_fft와 hop_length argument로 FFT size와 hop size를 설정할 수 있습니다. Magnitude값을 얻기 위해 abs function을 취합니다.



- 인간의 귀는 낮은 주파수대역의 소리를 듣는데 치우쳐 있습니다.
- Spectrogram은 낮은 주파수부터 높은 주파수까지 고르게 정보를 가지고 있기 때문에, mel-spectrogram이라는 표현 방식을 사용하기도 합니다.
- Spectrogram으로부터 mel-spectrogram을 얻으려면 mel scale을 표현한 matrix를 곱해 주면 됩니다.

Frequency f to mel-frequency m

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



https://en.wikipedia.org/wiki/Mel_scale

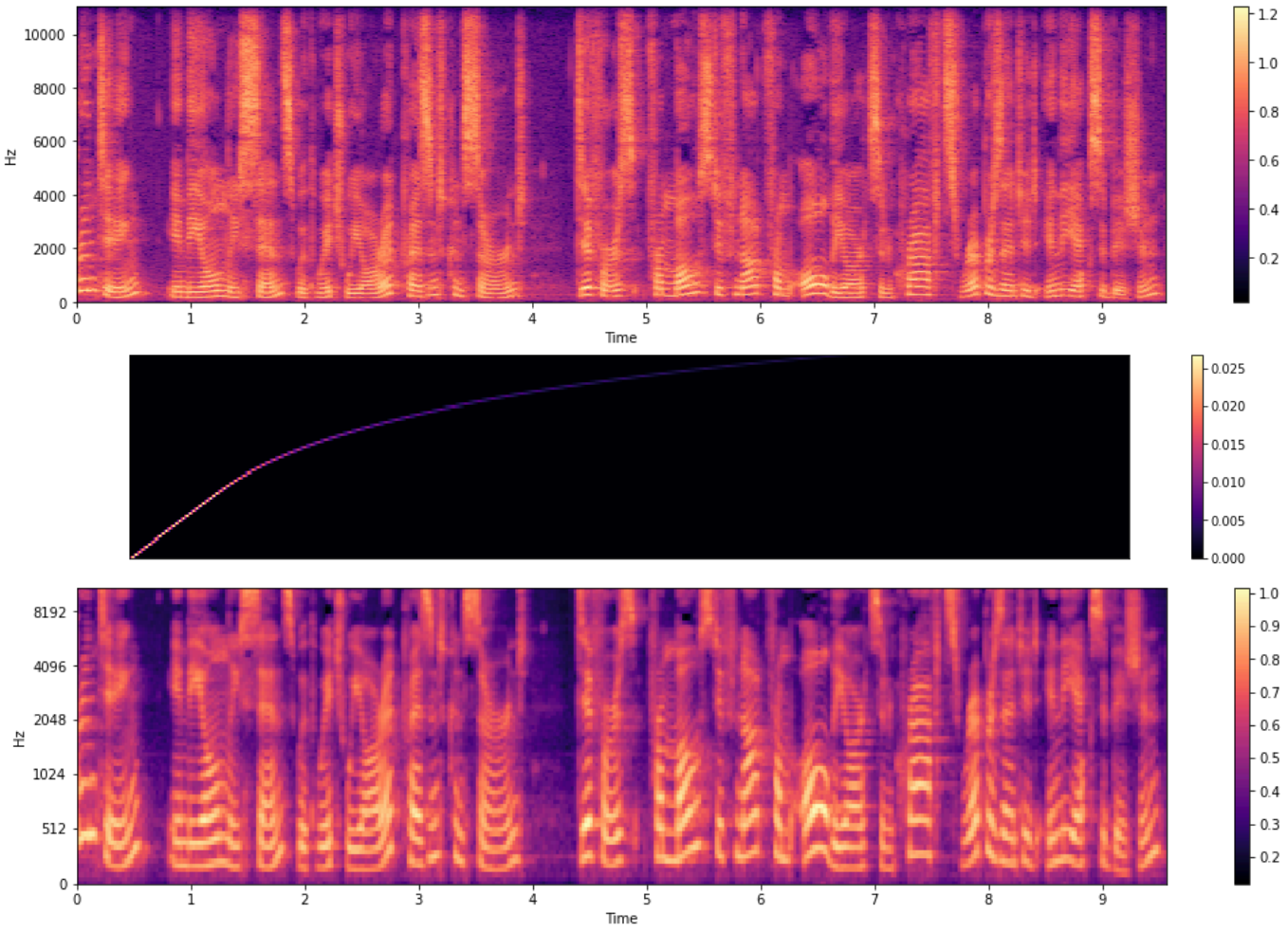
Spectrogram

•

Mel-scale matrix

=

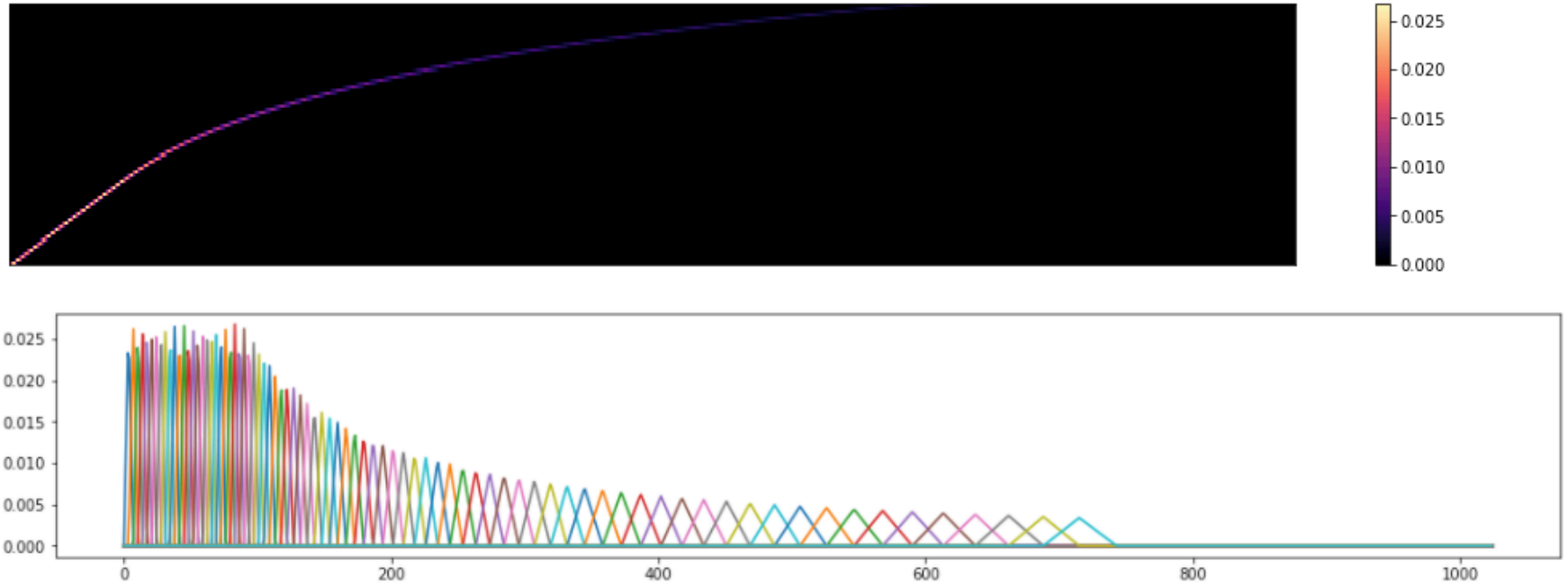
Mel-spectrogram



Mel-spectrogram Librosa 실습

```
# mel-spectrogram을 만들기 위한 matrix 구하기
mel_matrix = librosa.filters.mel(sr=22050, n_fft=2048, fmin=0.0, fmax=8000, n_mels=80)
```

```
mel matrix shape : (80, 1025)
```



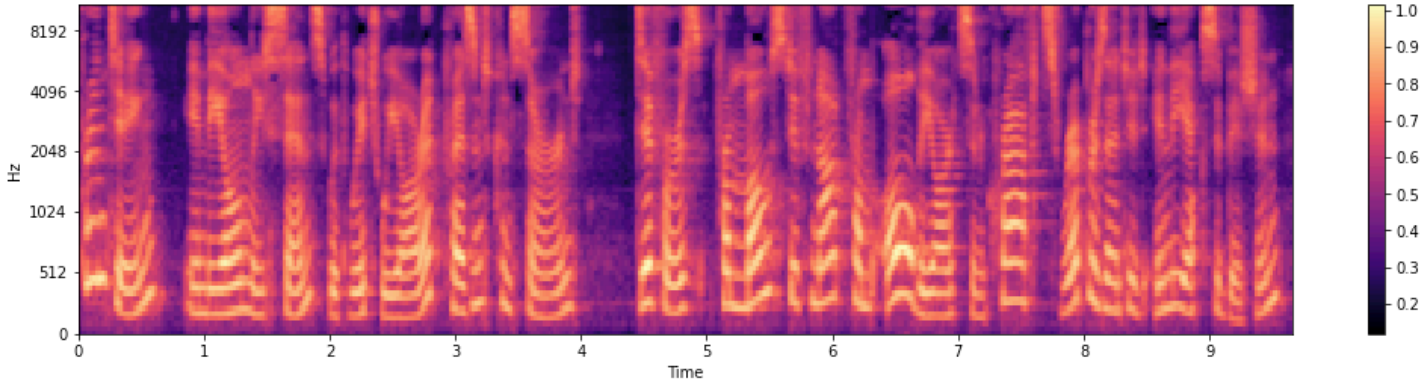
Mel-spectrogram을 만들기 위한 matrix를 만듭니다. n_fft를 2048, n_mels를 80으로 지정했으므로 (80, 1025) shape을 가진 matrix가 만들어집니다. 시각화를 통해 저음역의 성분들은 간격이 좁게 반영되고, 고음역의 성분들은 간격이 넓게 반영되는 것을 알 수 있습니다.

Mel-spectrogram Librosa 실습

```
spec = librosa.core.stft(wav, n_fft=2048, hop_length=512)
spec = np.abs(spec)

# spectrogram에 mel-matrix 적용
mel = mel_matrix @ spec
```

```
mel-spectrogram shape: (80, 416)
```



- Spectrogram에 mel matrix를 곱하여 mel-spectrogram을 구합니다.
- Matrix multiplication은 python의 내장 operator인 @로 수행할 수 있습니다.
- Spectrogram의 shape이 (n_fft/2+1, time)이고, mel matrix의 shape이 (n_mels, n_fft/2+1)이므로, mel-spectrogram의 shape은 (n_mels, time)이 됩니다.

- Automatic Speech Recognition은 audio feature vector를 읽어들여 grapheme tokens을 출력하는 모델입니다.

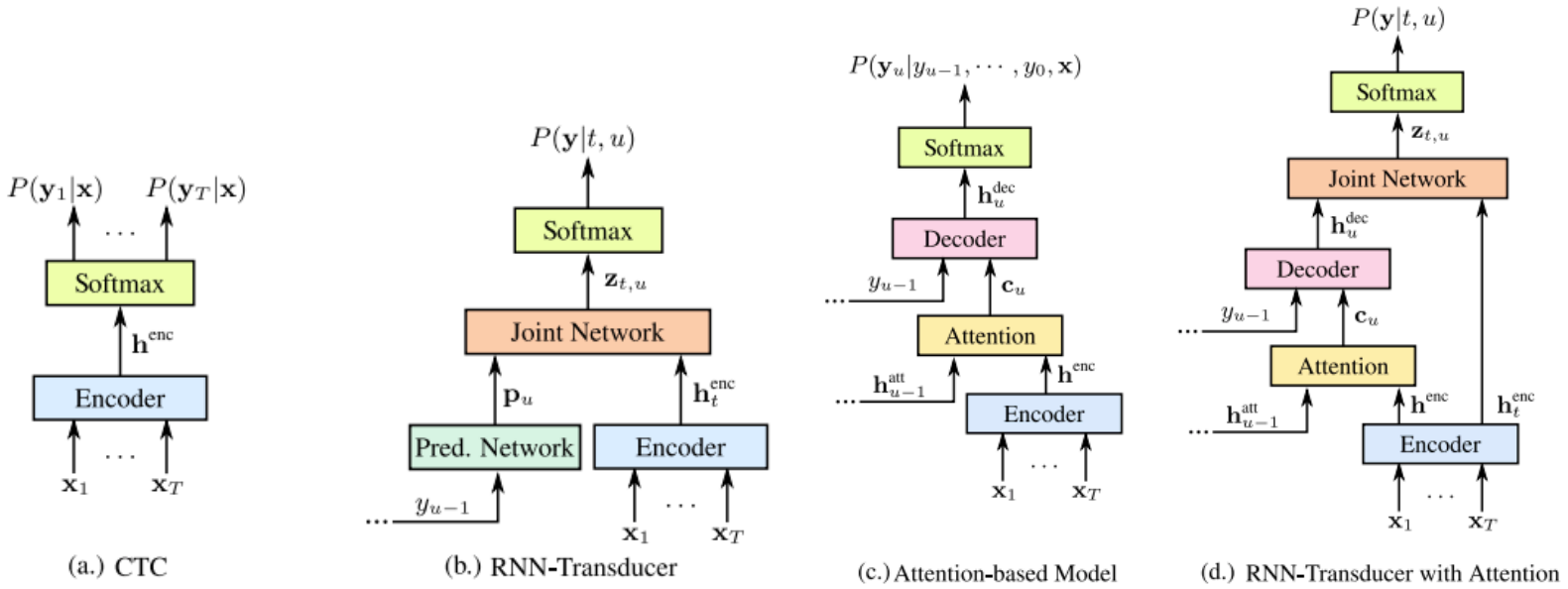
- Audio feature vector는 waveform이나 spectrogram, mel-spectrogram, MFCC등이 사용됩니다.

- 최근에는 mel-spectrogram을 사용하거나 wav2vec과 같은 neural encoder를 통해 encoding한 결과를 input으로 사용합니다.

- Audio feature들은 다음과 같이 d-dimensional vector들의 sequence로 표기할 수 있습니다.
  $\mathbf{x} = \left(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\right)$, where $\mathbf{x}_t \in \mathbf{R}^d$

- 출력이 되는 token들은 다음과 같이 integer들의 sequence로 표기할 수 있습니다.
  $\mathbf{y} = \left(y_1, y_2, \ldots, y_L\right)$, where $y_l \in \mathbf{N}$

- 문자에 사용하기 위한 token들 외에 알고리즘 내부에서 Start Of Sentence (SOS)나 blank를 표시하기 위해 주로 0값을 사용합니다.



- 현대의 end-to-end ASR 모델은 주로 3가지 framework을 사용합니다.
- Connectionist Temporal Classification (CTC) 는 audio sequence를 인코딩하는 encoder만으로 구성됩니다.
- Transducer는 audio sequence에 대한 encoder와 token sequence에 대한 prediction network, 그리고 두 network의 결과를 합치는 joint network로 구성됩니다.
- Attention-based 모델은 audio sequence에 대한 encoder, token sequence에 대한 decoder가 있으며, decoding시에 attention 모듈을 통해 encoder에서 인코딩된 정보를 참조합니다.
- 이밖에 Transducer에 attention개념을 적용한 hybrid 모델이 있습니다.

Prabhavalkar, Rohit, et al. "A Comparison of Sequence-to-Sequence Models for Speech Recognition." *Interspeech*. 2017.

- CTC는 2006년에 Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks 논문에서 제안된 알고리즘입니다.

- CTC는 segment되지 않은 sequence data에 labelling을 하기 위한 목적에서 설계되었습니다.

- ASR에 사용되는 audio sequence, label sequence 쌍과 같이 길이가 서로 다르고 alignment가 주어지지 않은 조건에 사용될 수 있습니다.

- CTC는 input sequence가 output sequence보다 길어야한다는 제약이 있습니다. (Transducer는 제약 없음) 대체로 audio sequence는 label sequence보다 길므로 이에 적합합니다.

- CTC는 loss를 구하고 트레이닝을 하도록 하는 framework이고, audio sequence를 다루기 위한 encoder는 RNN, CRNN, 또는 Transformer등 sequential data를 다룰 수 있는 모델이면 무엇이든 적용 가능합니다.

- CTC algorithm에서는 path라는 개념을 도입합니다.

- Audio sequence 와 label sequence 의 길이가 다르므로 각 label을 audio feature에 대응시키는 방법이 필요합니다.

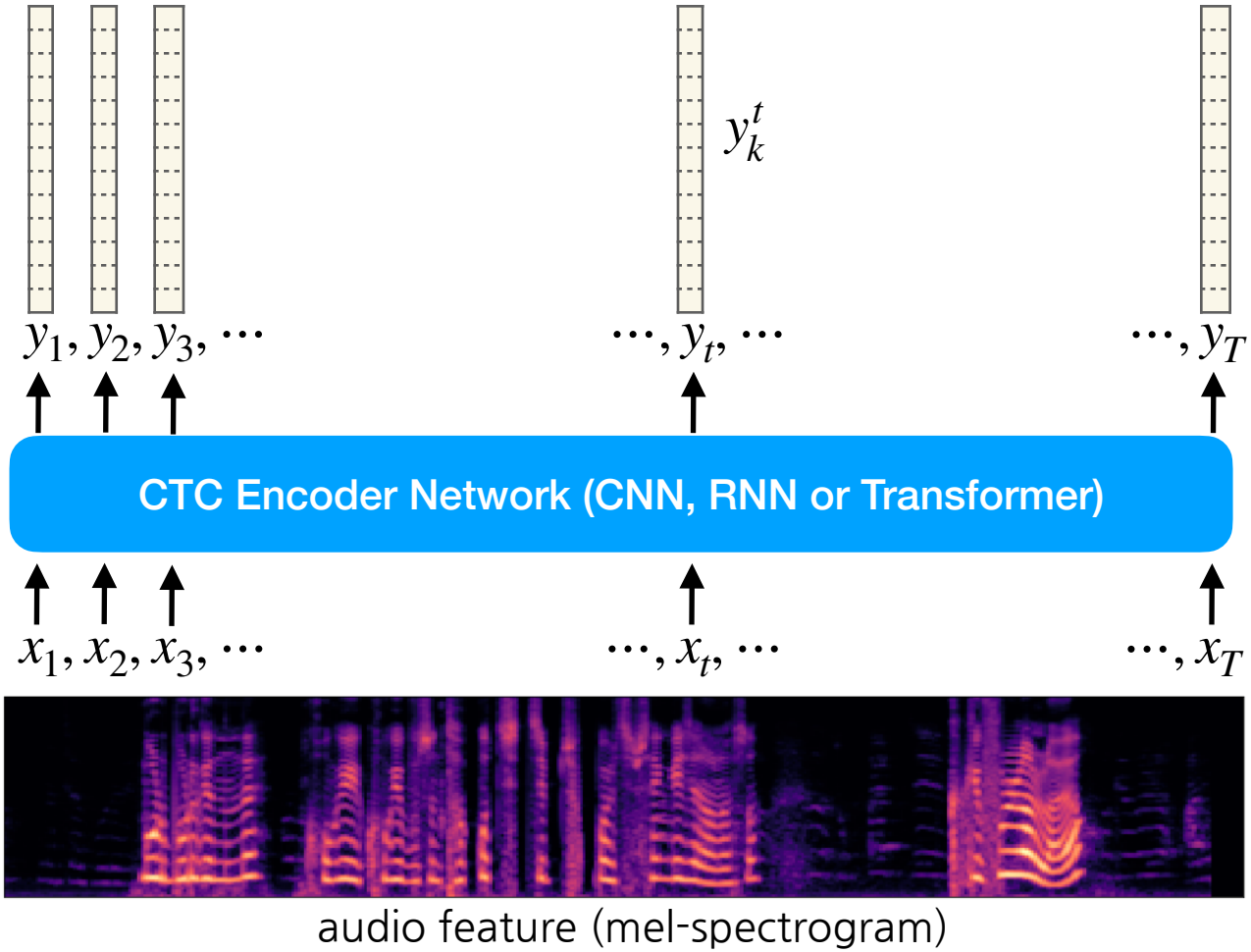
- 먼저, audio sequence에는 label sequence에 있는 label들 중 해당하지 않는 feature들도 있을 것 (말하지 않는 구간, 숨소리)이라는 가정하에 blank 을 추가합니다.

- 또한 label 하나가 여러 audio feature에 대응될 것이라는 가정하에 label이 반복될 수 있도록 합니다.

- label이 path상에서 반복된 것인지 원래 label sequence에서 연속해서 나타난 것인지 구별하기 위해 항상 구분되는 label 간에는 blank 을 넣는 것으로 정합니다. 즉, path 는 sequence 에 해당하지만, path 는 sequence 에 해당합니다.

-
  path는 항상 ‘blank’에서 시작하고 끝나는 것으로 정합니다.
- 이러한 식으로 sequence 라는 label sequence에 대해 path 또는 와 같은 path를 만들어 볼 수 있습니다.

- 이렇게 정의된 path는 audio sequence와 같은 길이로 만들 수 있습니다.



audio feature (mel-spectrogram)

- $x_1, x_2, x_3, \ldots, x_T$: 길이 T를 갖는 audio feature sequence, encoder의 입력으로 사용

- $y_1, y_2, y_3, \ldots, y_T$: encoder의 출력 벡터들

- y_k^t: t시점에 label k가 관측될 확률, label 0은 blank (softmax activation 사용)

- Audio sequence $\mathbf{x}$가 주어졌을 때, 경로 $\pi = \{\pi_1, \pi_2, \ldots, \pi_T\}$가 관측될 확률은

$$p(\pi \mid \mathbf{x}) = \prod_{t=1}^{T} y_{\pi_t}^t$$

CLASS `torch.nn.CTCLoss(blank=0, reduction='mean', zero_infinity=False)` [SOURCE]

The Connectionist Temporal Classification loss.

Calculates loss between a continuous (unsegmented) time series and a target sequence. CTCLoss sums over the probability of possible alignments of input to target, producing a loss value which is differentiable with respect to each input node. The alignment of input to target is assumed to be "many-to-one", which limits the length of the target sequence such that it must be $\leq$ the input length.

Parameters

- **blank** (*int, optional*) – blank label. Default 0 .
- **reduction** (*string, optional*) – Specifies the reduction to apply to the output: `'none'` | `'mean'` | `'sum'` . `'none'` : no reduction will be applied, `'mean'` : the output losses will be divided by the target lengths and then the mean over the batch is taken. Default: `'mean'`
- **zero_infinity** (*bool, optional*) – Whether to zero infinite losses and the associated gradients. Default: `False` Infinite losses mainly occur when the inputs are too short to be aligned to the targets.

Shape:

- Log_probs: Tensor of size (T, N, C), where $T = \text{input length}$, $N = \text{batch size}$, and $C = \text{number of classes (including blank)}$. The logarithmized probabilities of the outputs (e.g. obtained with `torch.nn.functional.log_softmax()`).
- Targets: Tensor of size (N, S) or $(\text{sum}(\text{target\_lengths}))$, where $N = \text{batch size}$ and $S = \text{max target length, if shape is } (N, S)$. It represent the target sequences. Each element in the target sequence is a class index. And the target index cannot be blank (default=0). In the (N, S) form, targets are padded to the length of the longest sequence, and stacked. In the $(\text{sum}(\text{target\_lengths}))$ form, the targets are assumed to be un-padded and concatenated within 1 dimension.
- Input_lengths: Tuple or tensor of size (N), where $N = \text{batch size}$. It represent the lengths of the inputs (must each be $\leq T$). And the lengths are specified for each sequence to achieve masking under the assumption that sequences are padded to equal lengths.
- Target_lengths: Tuple or tensor of size (N), where $N = \text{batch size}$. It represent lengths of the targets. Lengths are specified for each sequence to achieve masking under the assumption that sequences are padded to equal lengths. If target shape is (N, S), target_lengths are effectively the stop index s_n for each target sequence, such that `target_n = targets[n,0:s_n]` for each target in a batch. Lengths must each be $\leq S$ If the targets are given as a 1d tensor that is the concatenation of individual targets, the target_lengths must add up to the total length of the tensor.
- Output: scalar. If `reduction` is `'none'`, then (N), where $N = \text{batch size}$.


똑똑
파이팅!!