ASR 기초 공부

2026.01.25.
HyorinJung

Index

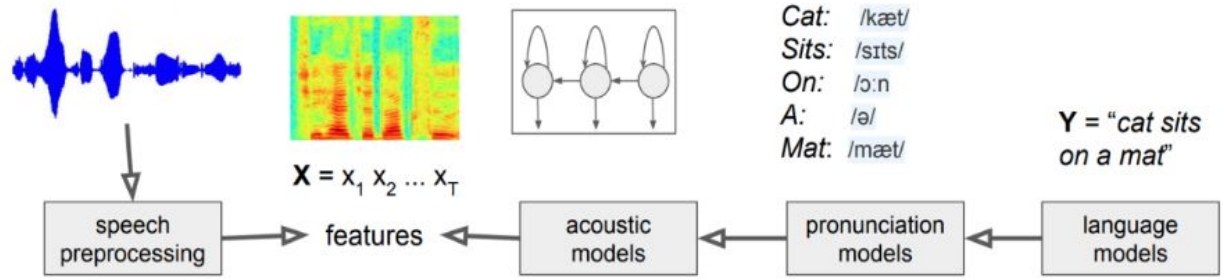
- ASR 발전
- 음성학 이론
- Wav2vec 2.0 모델 공부

ASR 구성요소

- 우변의 첫번째 항 P(X|Y) 는 음향 모델(Acoustic Model),P(Y) 는 언어 모델(Langu

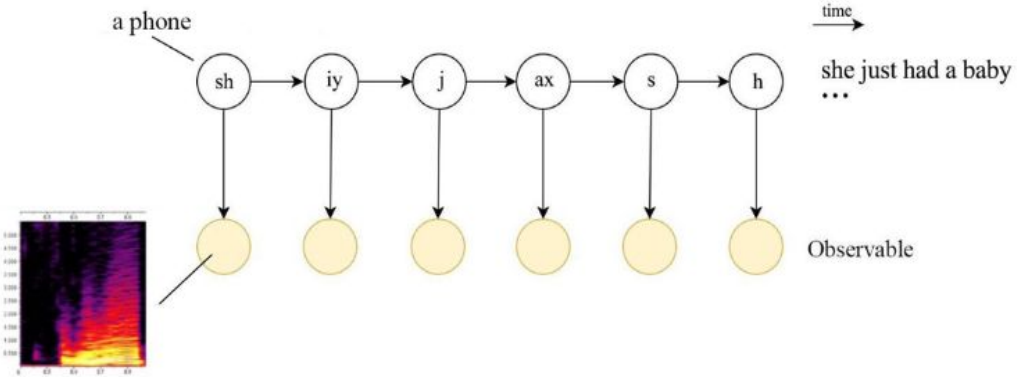
$$\hat{Y} = \underset{Y}{\operatorname{argmax}} P(X|Y)P(Y)$$

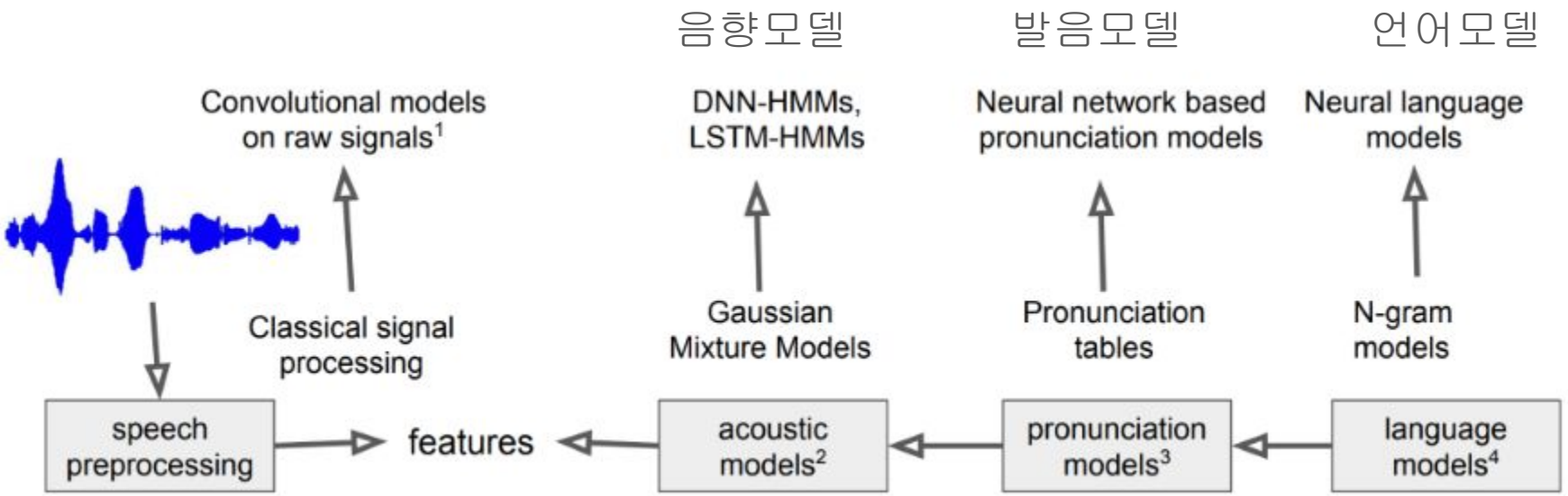
그림1 HMM, GMM 기반 음성인식 모델



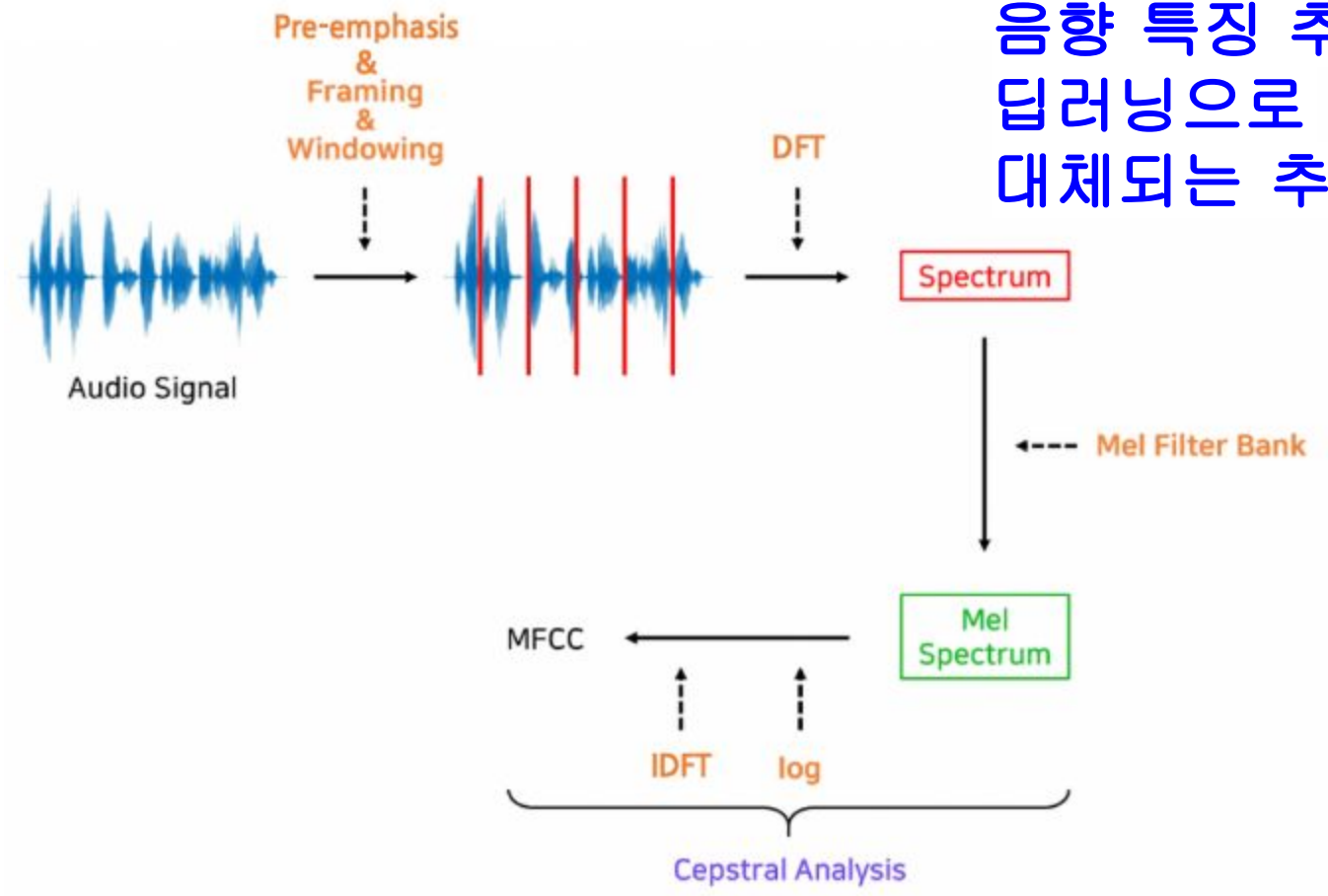
$$\mathbf{Y}^* = \arg\max_{\mathbf{Y}} p(\mathbf{X}|\mathbf{Y})\, p(\mathbf{Y})$$

그림2 ACOUSTIC MODEL

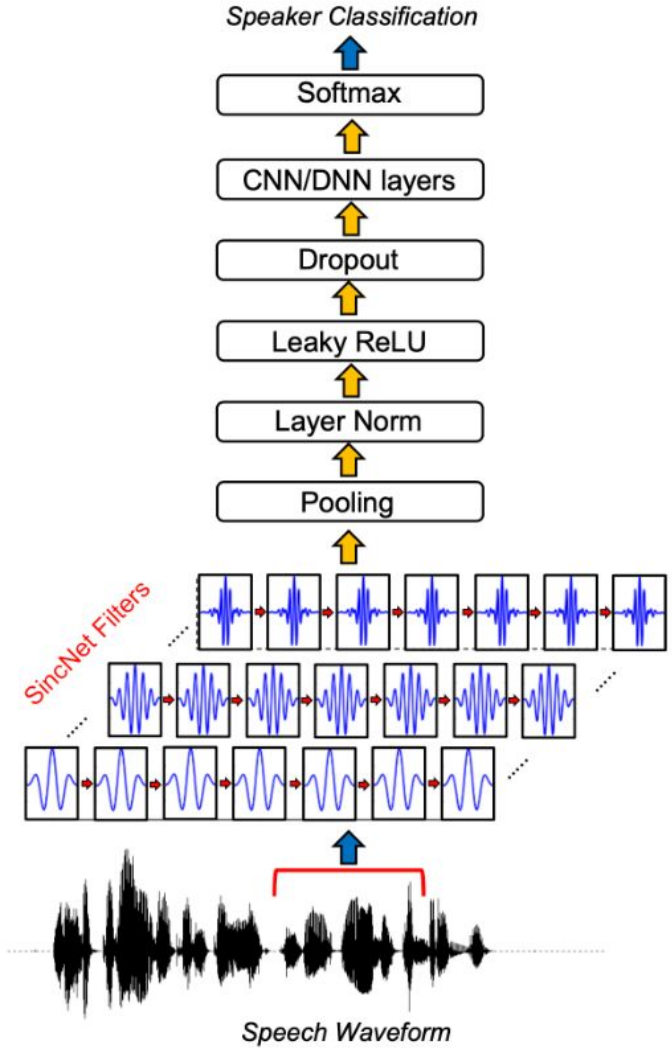



음향모델
발음모델
언어모델
Convolutional models on raw signals[1]
DNN-HMMs, LSTM-HMMs
Neural network based pronunciation models
Neural language models
Classical signal processing
Gaussian Mixture Models
Pronunciation tables
N-gram models
speech preprocessing
features
acoustic models[2]
pronunciation models[3]
language models[4]

음향 특징 추출도
딥러닝으로
대체되는 추세



Mel-Frequency Cepstral Coefficients(MFCC)를 만드는 전체 과정을 도식화한 그림


Speaker Classification
Softmax
CNN/DNN layers
Dropout
Leaky ReLU
Layer Norm
Pooling
SincNet Filters
Speech Waveform

결정적 X 확률적 O

음운(Phoneme)

- 단어의 뜻을 구별하는 말소리의 가장 작은 단위
- 분절음 : 음소(자음, 모음) / 비분절음 : 운소(음의 장단, 음의 고저, 강세, 억양 등)

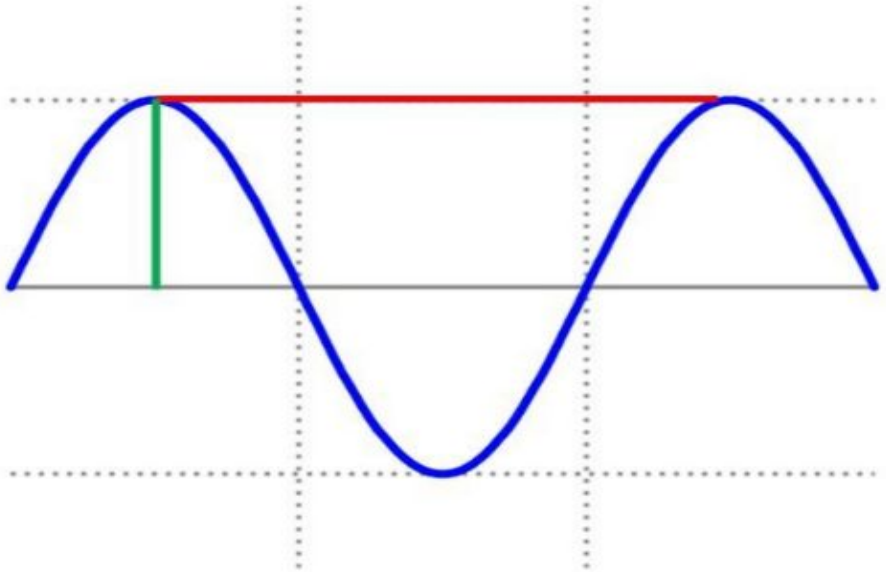
음절 (Syllable)

- 한번에 발음할 수 있는 소리의 덩어리
- 반드시 모음이 있어야함(모음 단독 또는 자음+모음)

파동 (Wave)

수식1 주기와 주파수 사이의 관계

$$T = \frac{1}{f}$$



Wavelength (λ)

Distance between identical points on consecutive waves

Amplitude

Distance between origin and crest (or trough)

Frequency (ν)

Number of waves that pass a point per unit time

Speed

= wavelength x frequency

그림1 x 축은 시간(time) - 초(second) / y 축은 음압(sound pressure)

Digitalization의 과정

Step 1. Sampling

Step 2. Quantization

Step 3. Encoding

결과 : Raw Wave Signal (디지털 파형)

Step 1. Sampling

- 일정한 시간 간격마다 음성 신호를 샘플해서 연속 신호(continous signal)를 이산 신호(discrete signal)로 변환
- Sampling rate : 1초에 몇 번 샘플하는지 나타내는 지표
  - 1초에 4만4100번 샘플한다면 sampling rate fs는 44100, 또는 44.1KHz
  - 44.1KHz의 샘플링된 신호는 1초에 44100개의 실수(real number)로 구성

Step 2. Quantization

- 샘플링된 신호에 양자화(quantization)을 실시하는 과정
- 양자화란 실수 범위의 이산 신호를 정수(integer) 이산 신호로 바꾸는 걸 의미
  - 8비트 양자화 >> 실수 범위의 이산 신호가 -128~127의 정수로 변환
  - 16비트 양자화 >> 실수 범위의 이산 신호가 -32768~32767 정수로 변환
- 압신(companding) 기법(압축 + 신장)

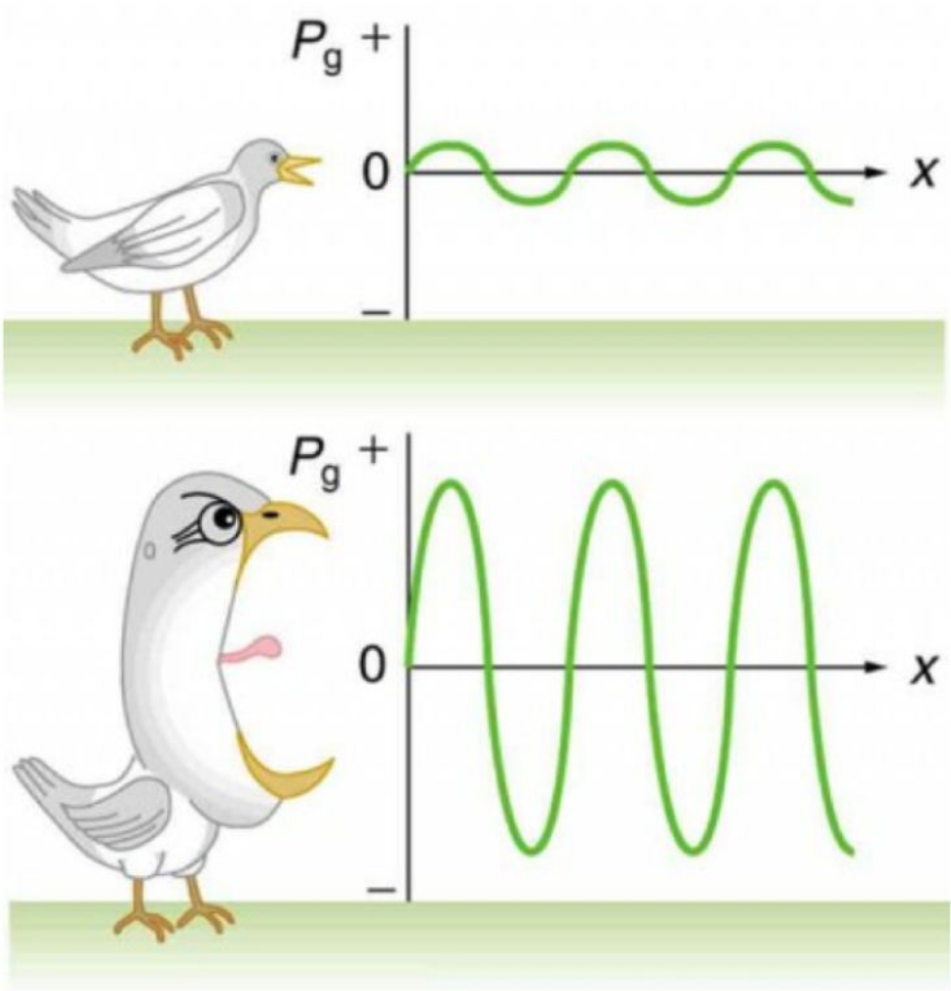
$$F(x) = \operatorname{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \qquad -1 \le x \le 1 \qquad \operatorname{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

μ-law , x축: 원래 진폭

Step 3. Encoding

- 정보 소스를 디지털 형식으로 변환, 압축, 저장하는 일련의 과정
  - wav, flac, mp3 등

소리크기 (Loudness)



LOUDNESS와 POWER

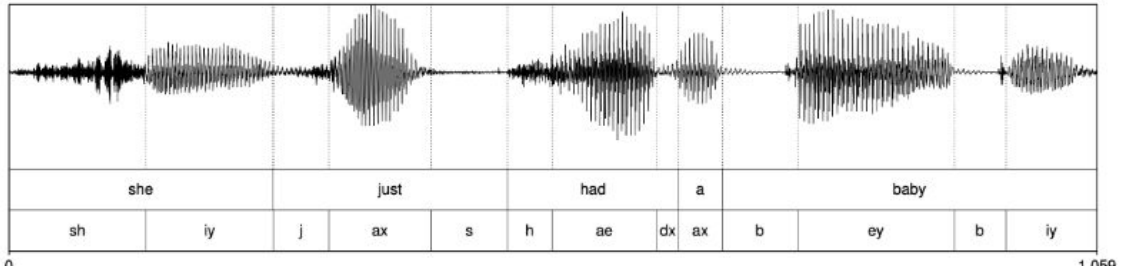
높낮이(Pitch)

- 인간은 1000Hz 이상의 고주파 소리에 대해서는 저주파 음성 대비 세밀하게 인식 X
- 100Hz ~ 1000Hz에 이르는 구간의 소리는 주파수 커질수록

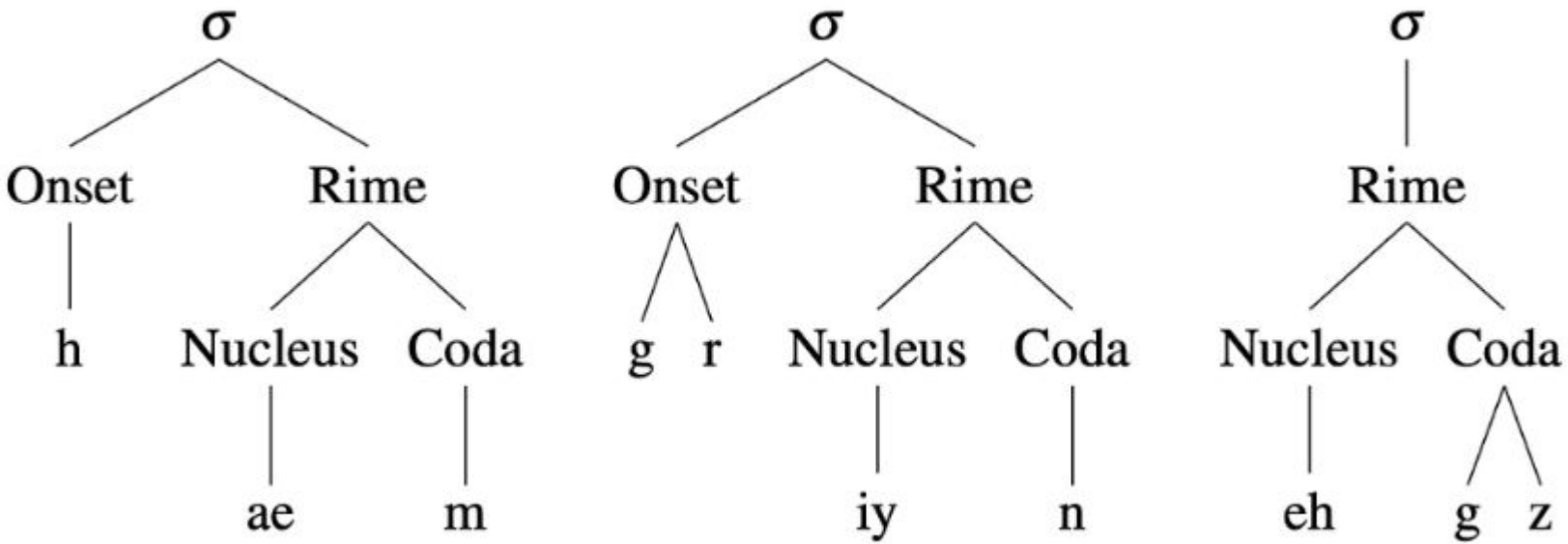
  피치 역시 높아진다고 느끼는 경향 있음

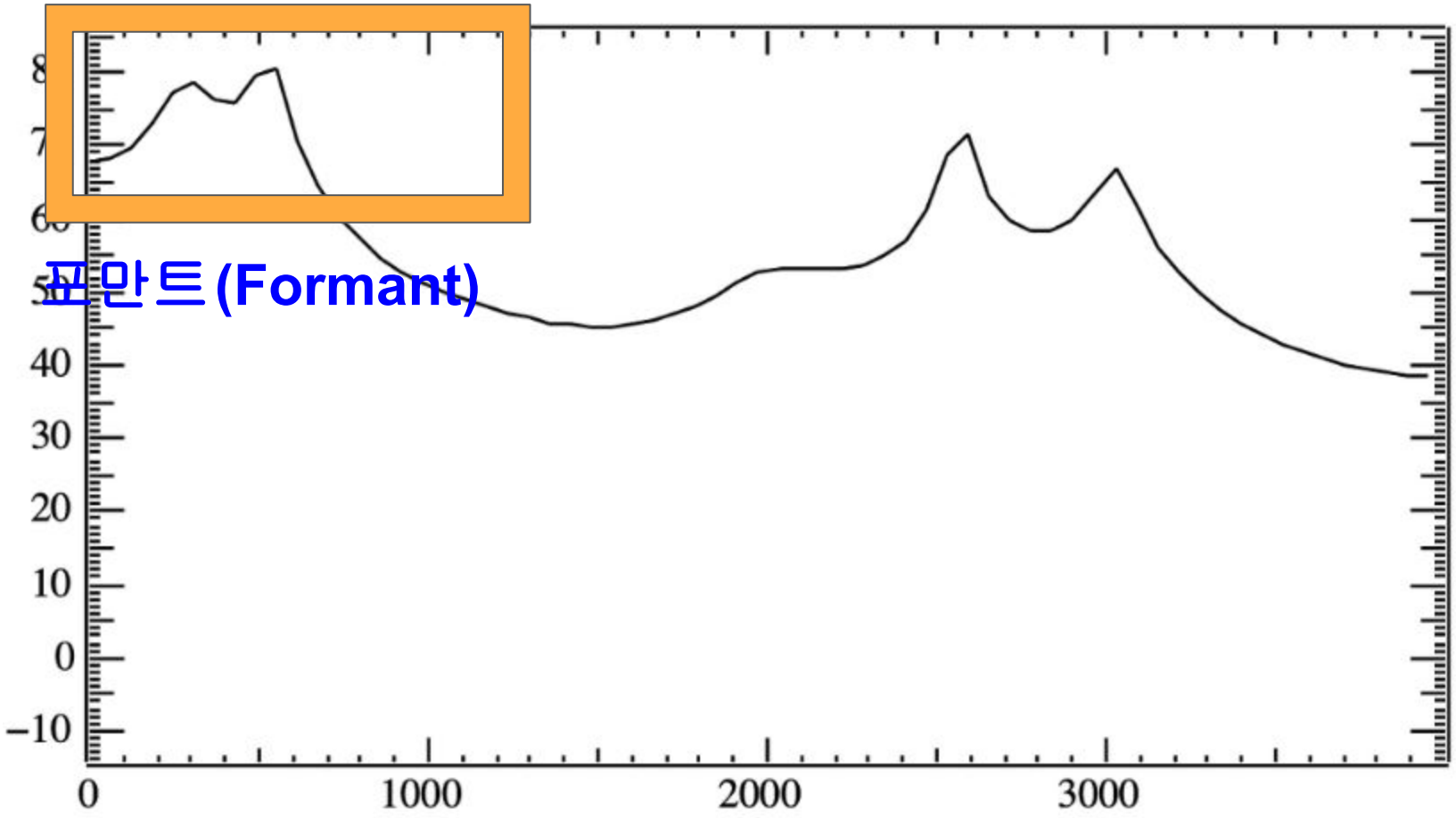
- 1000Hz 이상의 구간에서는 주파수와 피치 인식 사이의

  관계가 로그 형태를



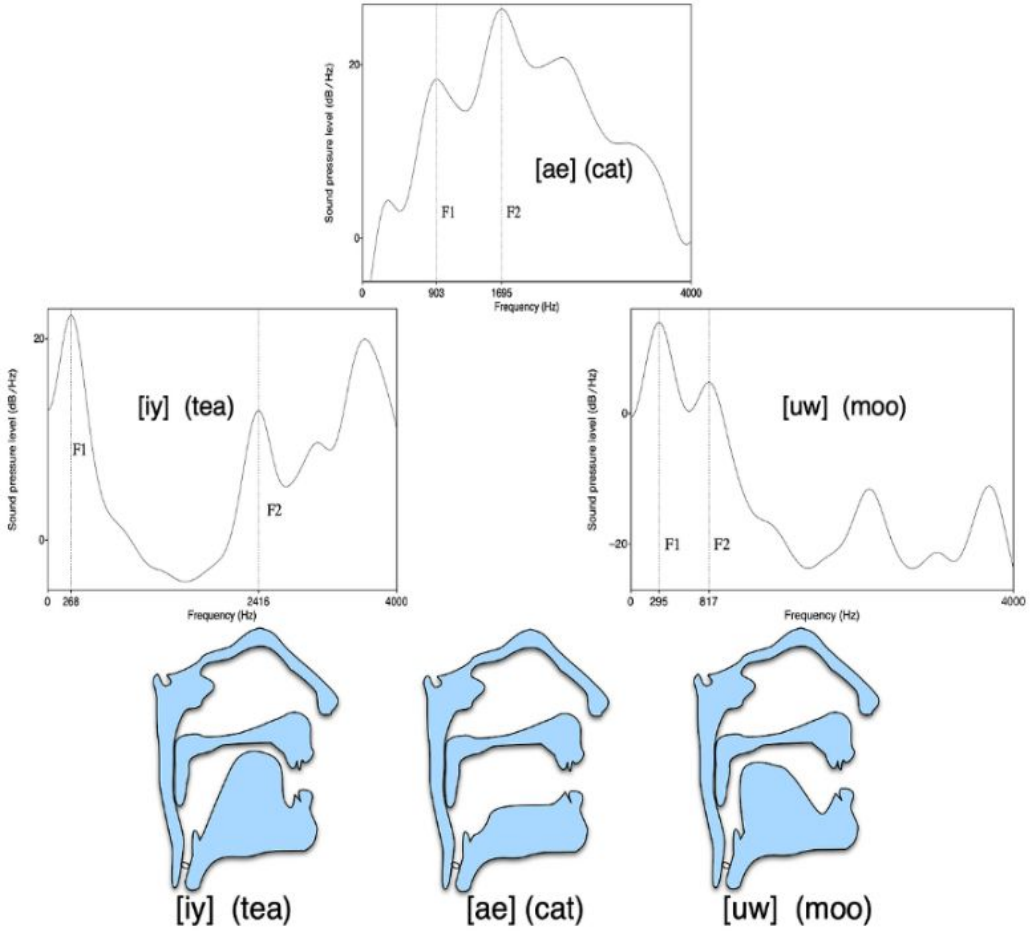
모음(Vowels)


σ
Onset
Rime
h
Nucleus
Coda
ae
m
σ
Onset
Rime
g
r
Nucleus
Coda
iy
n
σ
Rime
Nucleus
Coda
eh
g
z



x 축은 주파수, y 축은 진폭(amplitude)을 의미

The Source-Filter Model



인간의 음성 인식

lexical access : 음성을 단어 단위로 인식

- **frequency** : 사람은 빈도 높은 단어를 빠르게 인식
- **parallelism** : 여러 단어(예컨대 두 명 이상의 화자가 발화)를 한번에 알아들을 수 있음
- **cue-based processing** : 인간의 음성 인식은 다양한 단서(cue)에 기반함 (ex) 음성적 특징-포만트, 성대 진동개시시간 등/ 입모양 등)

MFCC

- 음성 인식과 관련해 불필요한 정보는 버리고 중요한

  특질만 남긴 것

- 뉴럴네트워크 기반 피처 추출 방식(딥러닝이 알아서 특징 추출) 과는 달리 음성 도메인의 지식과 공식에 기반한 추출 방법

Fourier Transform

- 시간에 따라 변하는

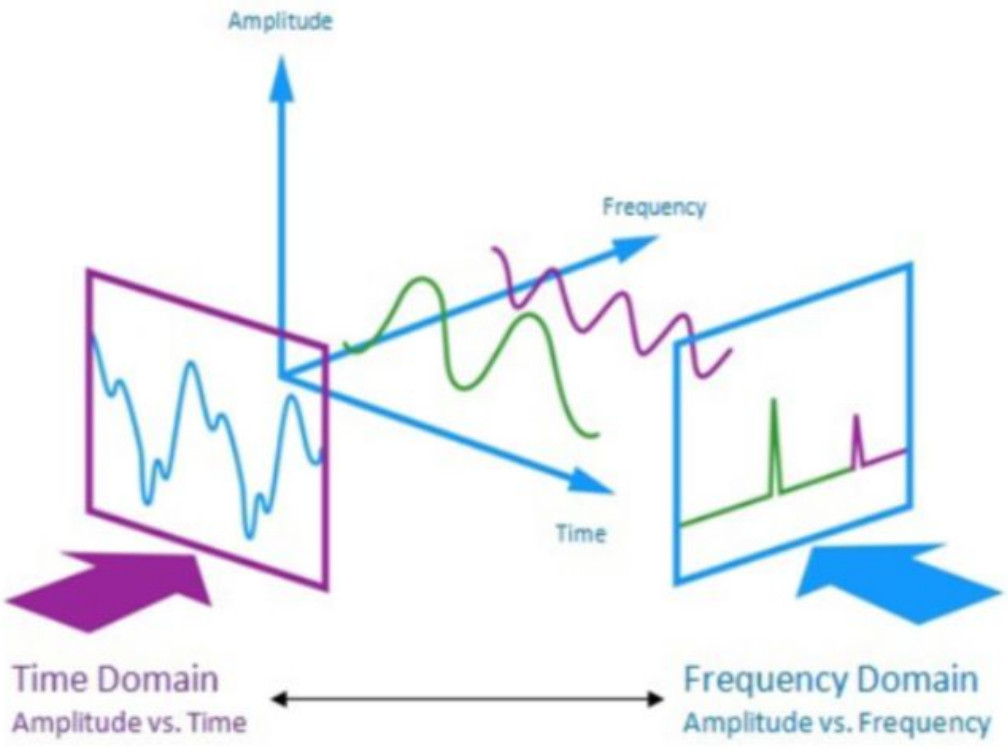
신호를 주파수 성분들의

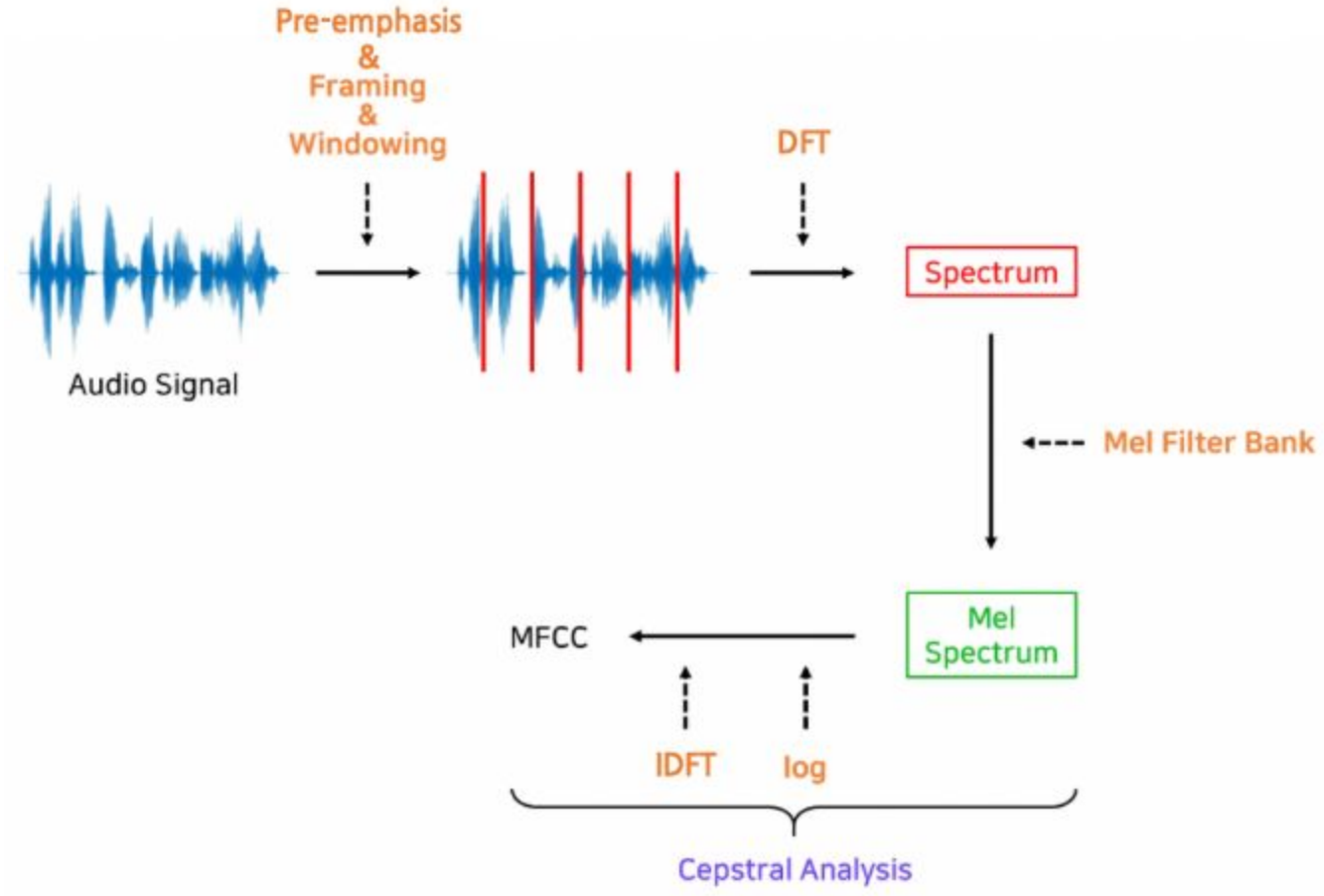
합으로 바꾸는 방법

- 입력 프레임 각각의

주파수 정보를 확인하는 과정

그림1 FOURIER TRANSFORM





Mel-Frequency Cepstral Coefficients(MFCC)를 만드는 전체 과정을 도식화한 그림

Step 1. Raw Wave Signal -> Preemphasis

Preemphasis

- 고주파 성분의 에너지를 올려주는 전처리 과정
- 사람 말소리를 스펙트럼(spectrum)으로 변환해서 관찰하면 일반적으로 저주파(low frequency) 성분의 에너지(energy)가 고주파(high frequency) 보다 많은 경향 (모음에서 이런 경향 많이 보임)
    - 고주파 성분의 에너지를 조금 올려주면 음성 인식 모델의 성능을 개선할 수 있음 $$\mathbf{y}_t = \mathbf{x}_t - \alpha \mathbf{x}_{t-1}$$

Step 2. Framing

- 음성 신호가 stationary하다고 가정해도 좋을 만큼 원시 음성 신호를 아주 짧은 시간 단위(대개 25ms)로 잘게 쪼개기
- 적용 이유 : 원시 음성 신호는 아주 빠르게 변화함 (non-stationary)

Step 3. Windowing

- 각각의 프레임에 특정 함수를 적용해 경계를 스무딩하는 기법
- 해밍 윈도우(Hamming Window) 쓰는 이유:
  - Rectangular Window : 원시 음성 신호를 짧은 구간 단위로 잘게 쪼개는 framing을 수행
  - Rectangular Window로 자른 프레임의 양끝에서는 신호가 살아 있다가 갑자기 죽는(=0) 상황이 발생
  - 이같은 프레임에 이후 푸리에 변환을 실시하게 되면 불필요한 고주파(high frequency) 성분이 살아남게 됨
- 해밍 윈도우를 적용하게 되면 양끝 부분이 스무딩되어 부작용 완화

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$$

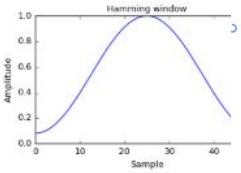
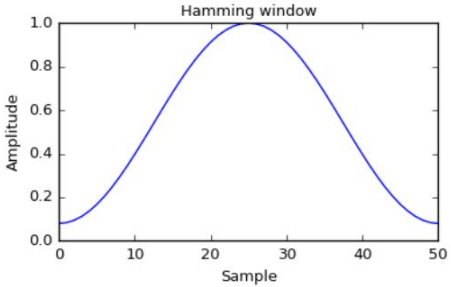
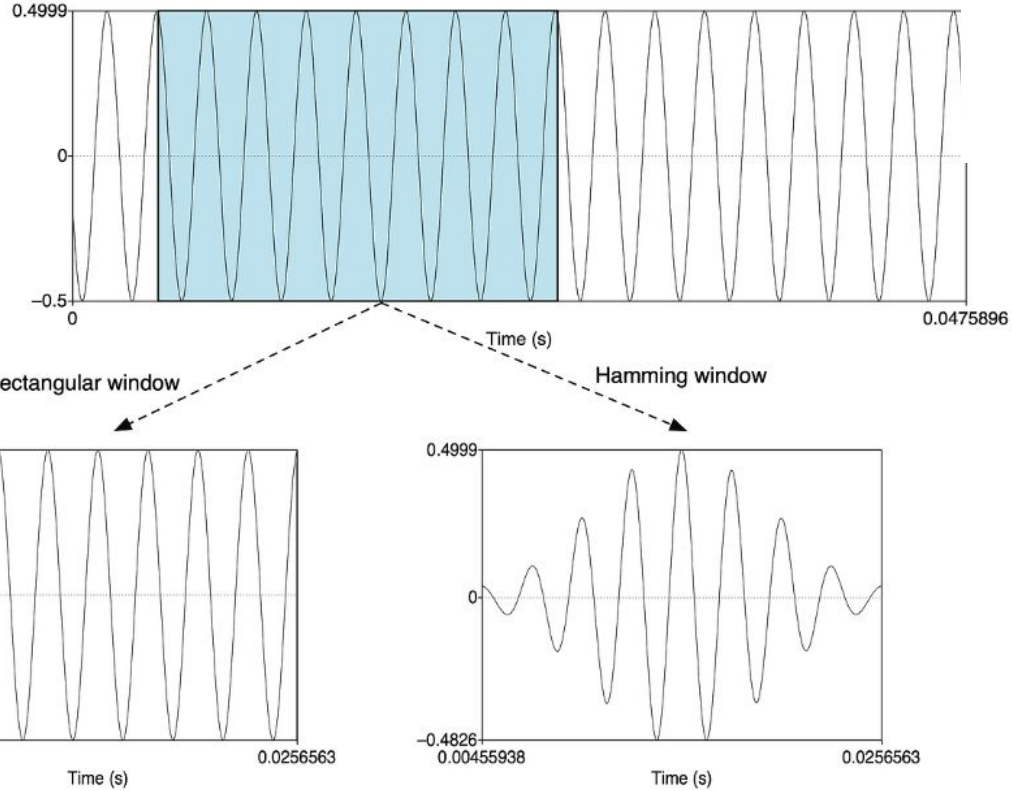


그림2 HAMMING WINDOW





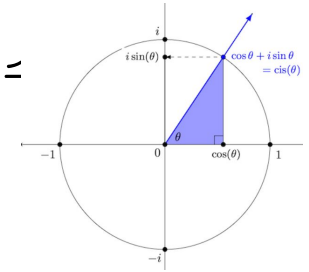
RECTANGULAR WINDOW VS HAMMING WINDOW

Step 4. Fourier Transform

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn} \Longleftrightarrow X[k] = \sum_{n=0}^{N-1} x[n]\left(\cos\frac{2\pi kn}{N} - j\sin\frac{2\pi kn}{N}\right)$$

- 고속 푸리에 변환(Fast Fourier Transform) 적용 : 시간 도메인의 음성 신호를 주파수(frequency) 도메인으로 바꾸는 과정
- 진폭은 이 주파수 성분의 크기를, 위상은 해당 주파수

  (복소평면상 단위원상)위치를 나타냄

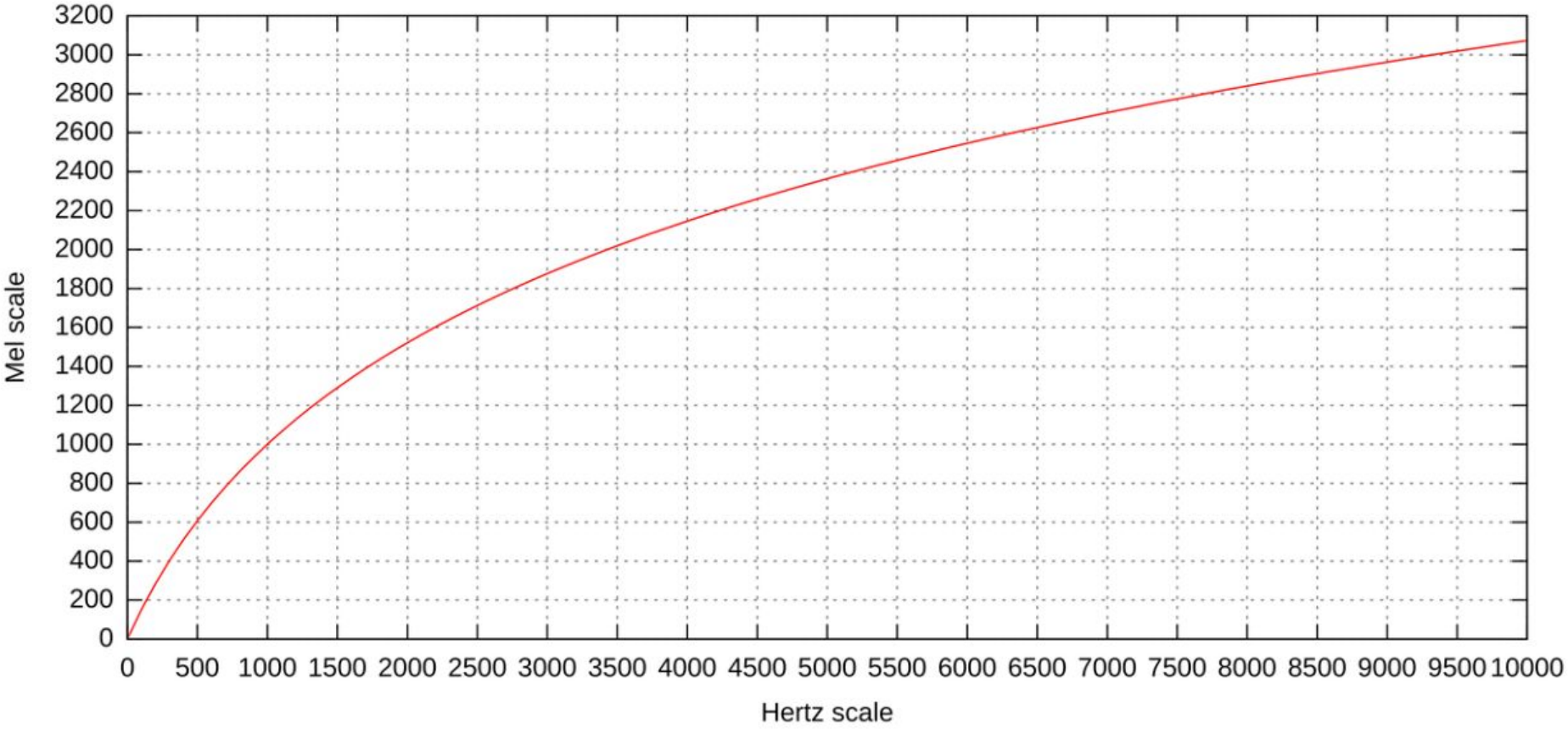


- MFCC를 구할 때는 음성 인식에 불필요한 위상 정보는

  없애고 진폭 정보만을 남김

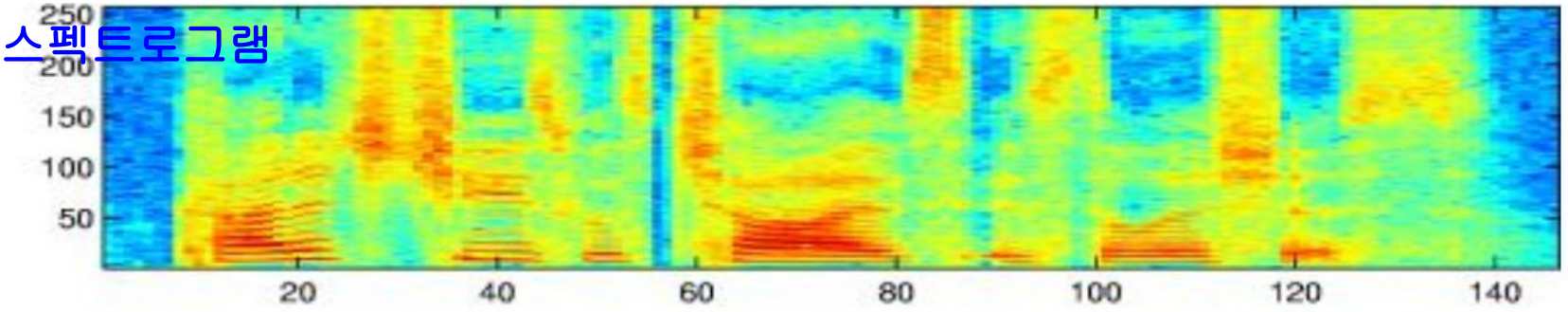
Step 5. Log-Mel Spectrum

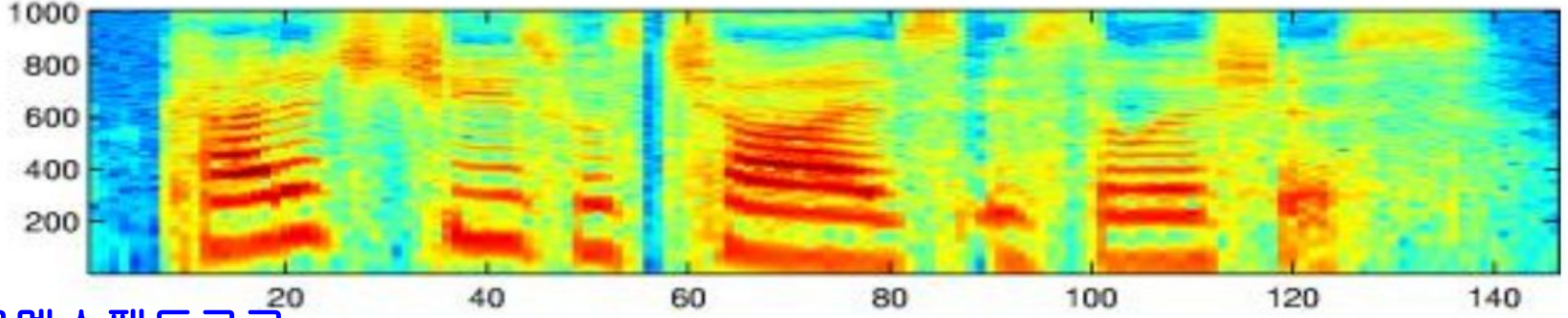
- 사람의 소리 인식은 로그(log) 스케일에 가깝다
  - 사람이 두 배 큰 소리라고 인식하려면 실제로는 에너지가 10배 큰 소리여야 한다
- 멜 스펙트럼 혹은 로그 멜 스펙트럼은 태생적으로 피처(feature) 내 변수 간 상관관계(correlation)가 존재 → DCT적용(독립적으로 변환)


Mel scale
3200
3000
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0 500 1000 1500 2000 2500 3000 3500 4000 4500 5000 5500 6000 6500 7000 7500 8000 8500 9000 9500 10000
Hertz scale

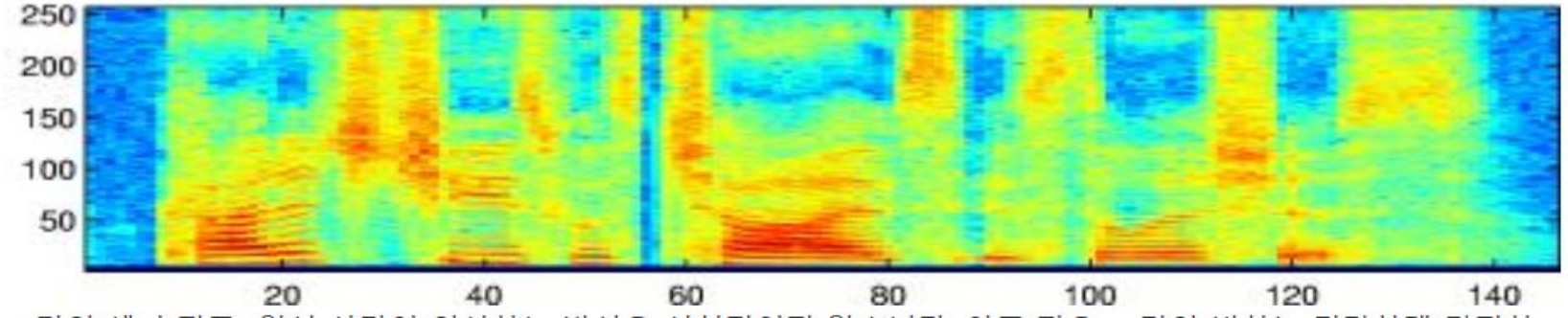
Mel Scale

멜스펙트로그램

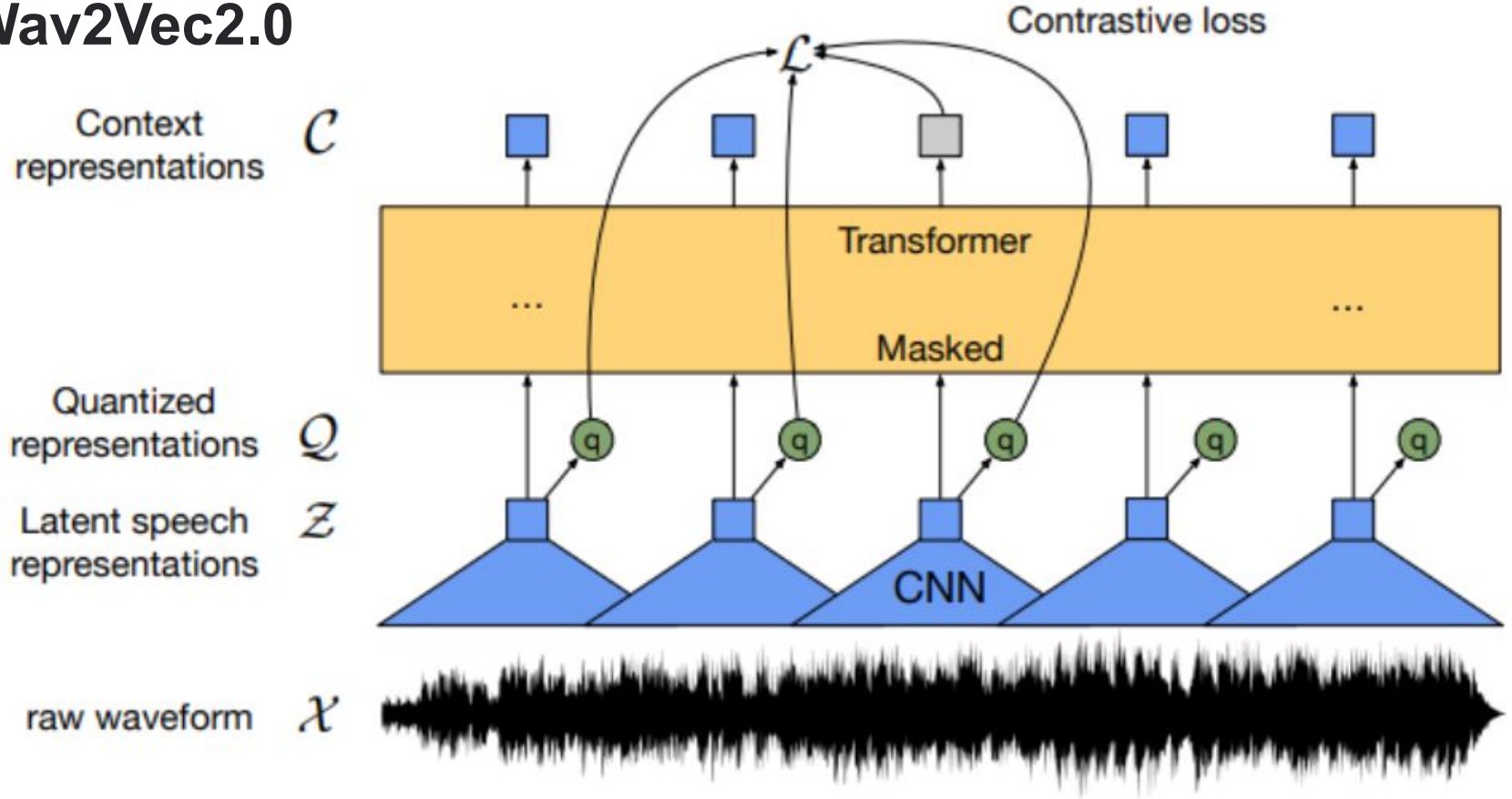




로그멜스펙트로그램

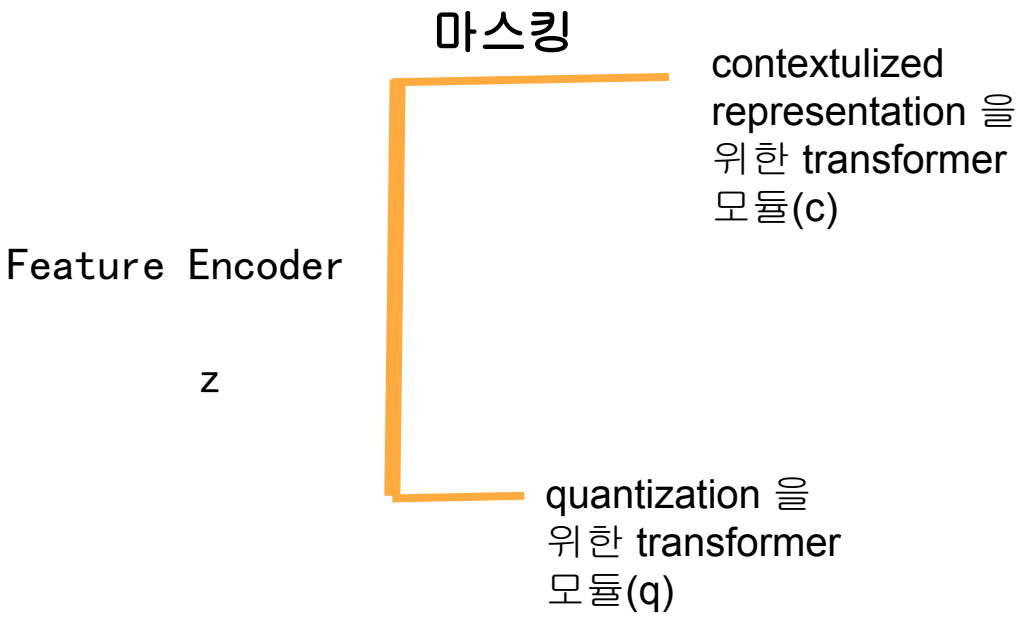


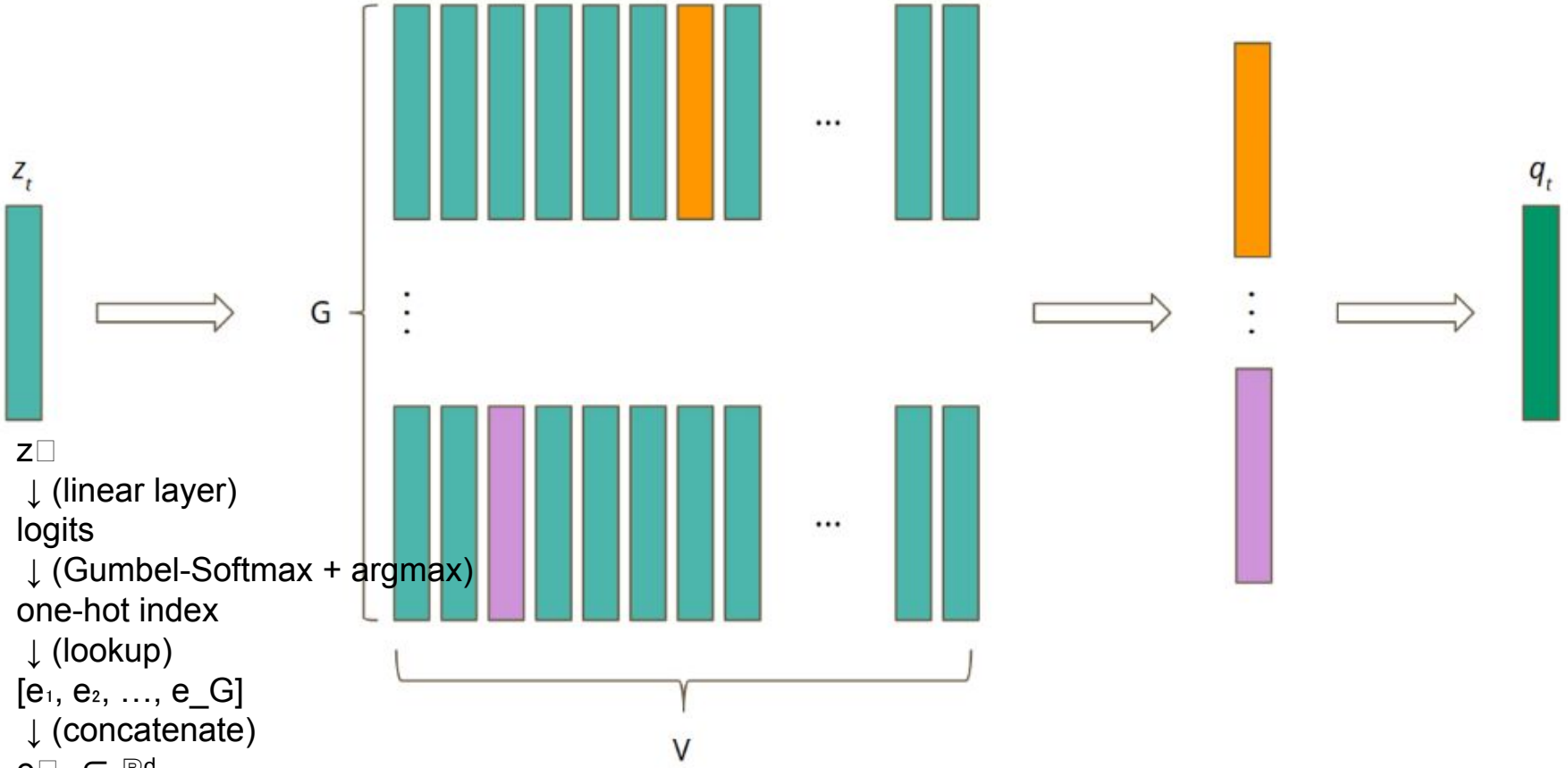
Wav2Vec2.0



[그림02] pre-training 과정에서의 Wav2Vec 2.0 모델 아키텍처

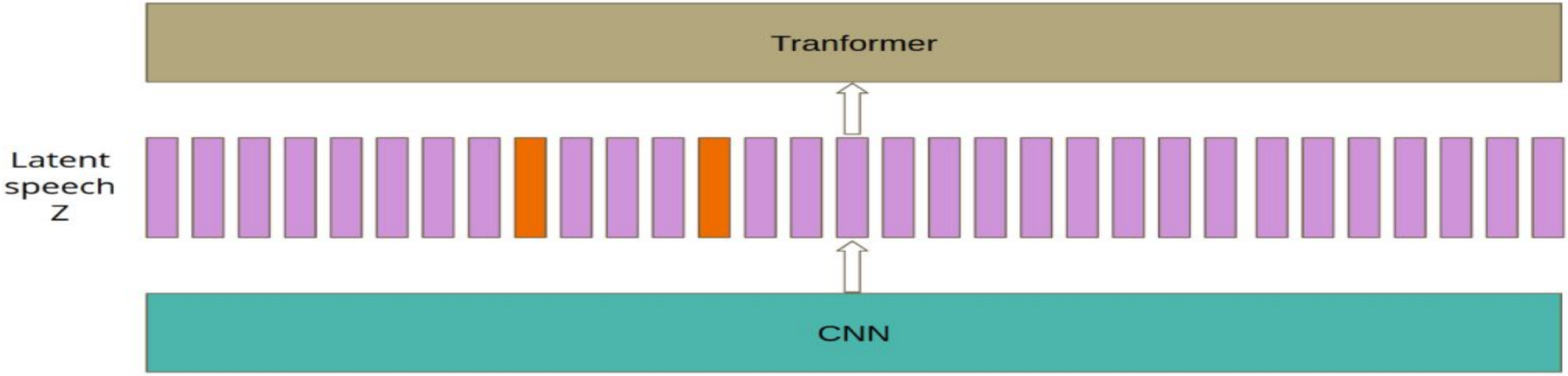
Pre-training



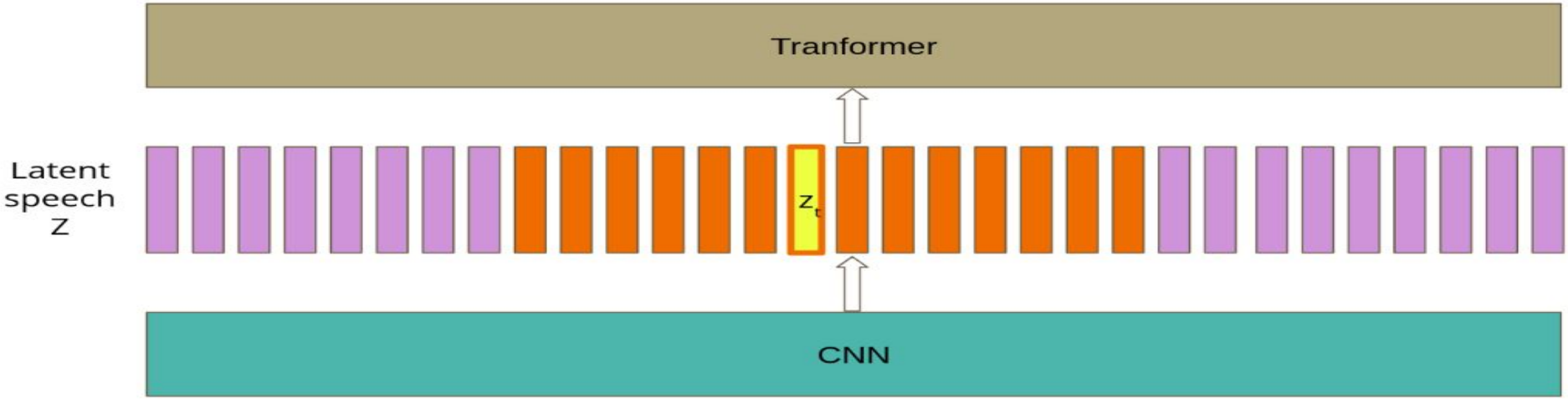


z_t
↓ (linear layer)
logits
↓ (Gumbel-Softmax + argmax)
one-hot index
↓ (lookup)
$[e_1, e_2, \ldots, e_G]$
↓ (concatenate)
$e_t \in \mathbb{R}^d$
↓ (linear projection)
$q_t \in \mathbb{R}^f$

[그림03] G개의 codebook, V개의 code words를 활용한 Wav2Vec 2.0의 quantization 모듈



[그림04] masking 인덱스의 시작점을 선택함



[그림05] masking을 진행함

Loss function

$$\mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d$$

Contrastive Loss

Q: 후보자(t+1) = q(정답) + distractor(t)

$$\mathcal{L}_m = -\log \frac{\exp\left(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa\right)}{\sum_{\tilde{\mathbf{q}} \sim \mathbf{Q}_t} \exp\left(sim(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa\right)}$$

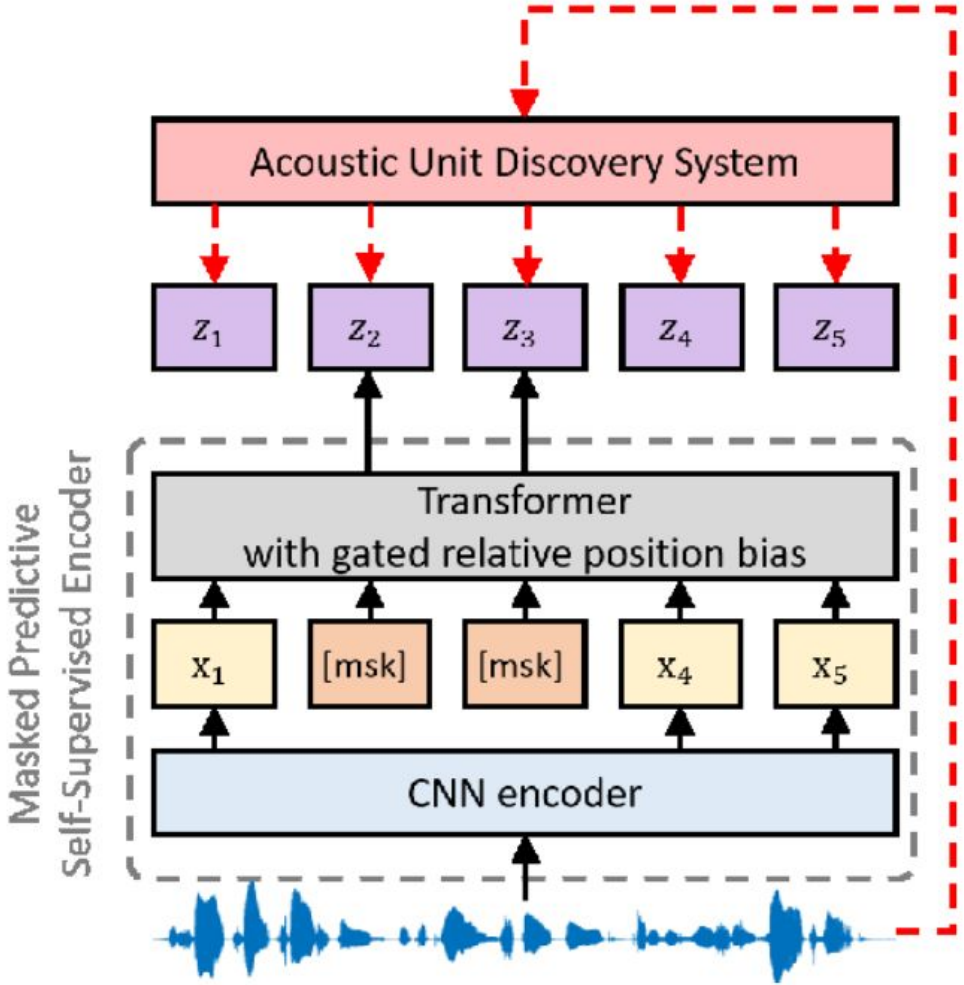
Diversity Loss

$$\mathcal{L}_d = \frac{1}{GV} * (-H(\bar{p}_g)) = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log(\bar{p}_{g,v})$$

Fine tuning

- 사전학습된 CNN과 Transformer를 사용 → context representation(C)을 추출
- linear projection layer와 CTC Loss사용

WavLm


Acoustic Unit Discovery System
z_1
z_2
z_3
z_4
z_5
Masked Predictive
Self-Supervised Encoder
Transformer
with gated relative position bias
x_1
[msk]
[msk]
x_4
x_5
CNN encoder

(a)

Gated Relative Position Bias

- 입력된 음성 Feature의 순서와 위치 정보를 더 잘 반영

→ 긴발화, 잡음환경, 화자 변화에 더 강함

$$g_i^{(\text{update})}, g_i^{(\text{reset})} = \sigma(\mathbf{q}_i \cdot \mathbf{u}), \sigma(\mathbf{q}_i \cdot \mathbf{w})$$

$$\tilde{r}_{i-j} = w g_i^{(\text{reset})} d_{i-j}$$

$$r_{i-j} = d_{i-j} + g_i^{(\text{update})} d_{i-j} + (1 - g_i^{(\text{update})}) \tilde{r}_{i-j}$$

전체 파이프라인

input_values (raw waveform)

→ feature_extractor (CNN 7층, 시간 다운샘플)

→ feature_projection (512→768 차원 맞춤)

→ encoder (pos conv embed + Transformer 12층)

→ output hidden states (범용 음성 표현)

Feature Extractor: CNN 7층 (raw → latent frames)

(feature_extractor): WavLMFeatureEncoder(

(conv_layers): ModuleList(

(0): WavLMGroupNormConvLayer(Conv1d(1→512, k=10, s=5) + GN + GELU)

(1-4): 4 x Conv1d(512→512, k=3, s=2) + GELU

(5-6): 2 x Conv1d(512→512, k=2, s=2) + GELU

))

Feature Projection: (512 → 768) 차원 정렬

(feature_projection): WavLMFeatureProjection(

  LayerNorm(512)

  Linear(512→768)

  Dropout(0.1)

)

Encoder 앞단: Positional Conv Embedding (위치 정보 주입)

(pos_conv_embed): WavLMPositionalConvEmbedding(

  Conv1d(768→768, k=128, groups=16, padding=64) + GELU

)

Transformer Encoder: 12층 (문맥 학습)

(layers): ModuleList(

  (0): WavLMEncoderLayer(...)

  (1-11): 11 x WavLMEncoderLayer(...)

)

WavLM Transformer Encoder Layer (전체 뼈대)

```
class WavLMEncoderLayer(nn.Module):
    def __init__(self, hidden_size=768, num_heads=12, ff_size=3072):
        super().__init__()

        # --- Self-Attention ---
        self.attention = WavLMAttention(hidden_size, num_heads)
        self.dropout = nn.Dropout(0.1)
        self.layer_norm = nn.LayerNorm(hidden_size)

        # --- Feed Forward ---
        self.feed_forward = WavLMFeedForward(hidden_size, ff_size)
        self.final_layer_norm = nn.LayerNorm(hidden_size)

    def forward(self, x, attention_mask=None, relative_position_bias=None):
        # x: (B, T, 768)

        # 1) Self-Attention + Residual
        attn_output = self.attention(
            x,
            attention_mask=attention_mask,
            relative_position_bias=relative_position_bias
        )
        x = x + self.dropout(attn_output)
        x = self.layer_norm(x)

        # 2) Feed Forward + Residual
        ff_output = self.feed_forward(x)
        x = x + self.dropout(ff_output)
        x = self.final_layer_norm(x)
```

WavLMAttention (핵심 차별점 포함)

```
class WavLMAttention(nn.Module):
    def __init__(self, hidden_size, num_heads):
        super().__init__()
        self.num_heads = num_heads
        self.head_dim = hidden_size // num_heads

        # Q K V projection
        self.q_proj = nn.Linear(hidden_size, hidden_size)
        self.k_proj = nn.Linear(hidden_size, hidden_size)
        self.v_proj = nn.Linear(hidden_size, hidden_size)
        self.out_proj = nn.Linear(hidden_size,
hidden_size)

        # --- WavLM 핵심 ---
        # Query 기반 gate 생성 (GRU-style)
        self.gru_rel_pos_linear = nn.Linear(64,
num_heads)

        # 상대 위치 bias embedding
        self.rel_attn_embed = nn.Embedding(320,
num_heads)
```

Attention forward (gated relative position bias 포함)

```
def forward(self, x, attention_mask=None,
relative_position_bias=None):
    B, T, C = x.shape

    # 1) Q, K, V 생성
    Q = self.q_proj(x)  # (B, T, C)
    K = self.k_proj(x)
    V = self.v_proj(x)

    # (B, num_heads, T, head_dim)
    Q = Q.view(B, T, self.num_heads, self.head_dim).transpose(1, 2)
    K = K.view(B, T, self.num_heads, self.head_dim).transpose(1, 2)
    V = V.view(B, T, self.num_heads, self.head_dim).transpose(1, 2)

    # 2) 기본 attention score
    attn_scores = torch.matmul(Q, K.transpose(-2, -1)) /
math.sqrt(self.head_dim)
```

WavLM 핵심: gated relative position bias

```
gate = torch.sigmoid(
        self.gru_rel_pos_linear(Q.mean(dim=-1))  # (B, num_heads, T)
    )

    # gate를 relative position bias에 적용
    gated_rpb = gate.unsqueeze(-1) * relative_position_bias
    attn_scores = attn_scores + gated_rpb
```

Attention 결과 계산

```
if attention_mask is not None:
        attn_scores = attn_scores + attention_mask

    attn_probs = torch.softmax(attn_scores, dim=-1)
    context = torch.matmul(attn_probs, V)

    # (B, T, C)
    context = context.transpose(1, 2).contiguous().view(B, T, C)
    return self.out_proj(context)
```

Feed Forward Network (표준 Transformer)

```
class WavLMFeedForward(nn.Module):
    def __init__(self, hidden_size, ff_size):
        super().__init__()
        self.intermediate_dense = nn.Linear(hidden_size, ff_size)
        self.activation = nn.GELU()
        self.output_dense = nn.Linear(ff_size, hidden_size)
        self.dropout = nn.Dropout(0.1)

    def forward(self, x):
        x = self.intermediate_dense(x)
        x = self.activation(x)
        x = self.output_dense(x)
        return self.dropout(x)
```