BrAIn Lab.
Brain and Artificial Intelligence Lab.

Wav2vec 2.0

Self-Supervised Learning of Speech Representations

컴퓨터공학과
23100493 김지윤

0. Author

BrAIn Lab.
Brain and Artificial Intelligence Lab.

Alexei Baevski

FOLLOW

Facebook AI Research
Verified email at fb.com

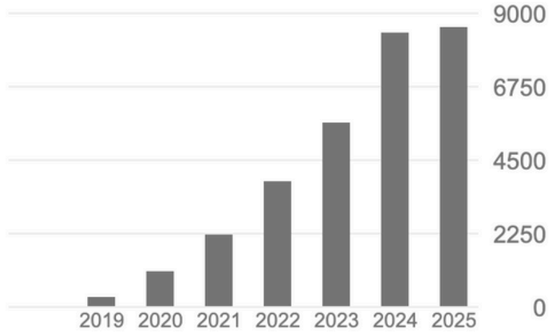
Machine Learning AI Natural Language Processing Automatic Speech Recogni…

GET MY OWN PROFILE

TITLE	CITED BY	YEAR
The llama 3 herd of models A Dubey, A Jauhri, A Pandey, A Kadian, A Al-Dahle, A Letman, A Mathur, ... arXiv e-prints, arXiv: 2407.21783	7629 *	2024
wav2vec 2.0: A framework for self-supervised learning of speech representations A Baevski, H Zhou, A Mohamed, M Auli arXiv preprint arXiv:2006.11477	7612	2020
fairseq: A fast, extensible toolkit for sequence modeling M Ott, S Edunov, A Baevski, A Fan, S Gross, N Ng, D Grangier, M Auli arXiv preprint arXiv:1904.01038	3495	2019
wav2vec: Unsupervised pre-training for speech recognition S Schneider, A Baevski, R Collobert, M Auli arXiv preprint arXiv:1904.05862	1939	2019
Data2vec: A general framework for self-supervised learning in speech, vision and language A Baevski, WN Hsu, Q Xu, A Babu, J Gu, M Auli International conference on machine learning, 1298-1312	1073	2022
Unsupervised cross-lingual representation learning for speech recognition	1021	2020

Cited by

	All	Since 2020
Citations	30162	29821
h-index	33	33
i10-index	40	40



Public access

VIEW ALL

0 articles	1 article
not available	available

Based on funding mandates

1. Background

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- Self-supervised Learning:

데이터 자체에서 정답(Label)을 만들어 모델을 스스로 학습시키는 방식으로,
사람이 직접 라벨을 달아주는 지도학습과 달리, 데이터의 일부를 문제로 만들고 나머지 부분을 정답으로 활용하여 데이터의 숨겨진 구조나 특징을 학습한다.
→ 라벨이 없는 방대한 양의 데이터를 활용하여 모델을 똑똑하게 만들 수 있다.

1. Background

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- **오디오 신호 복원(Reconstruction) 접근법**

입력된 오디오 신호의 일부를 변형하거나 손상시킨 뒤 모델이 이를 원래대로 복원하도록 학습하는 방법으로, Autoencoder 구조가 대표적이다.

- **다음 말을 예측하는 접근법**

오디오 신호의 과거 또는 현재 정보를 바탕으로 미래에 올 신호를 예측하도록 모델을 학습시키는 방법으로, Autoregressive Predictive Coding(APC) 등이 여기에 해당한다.

2. Transformer

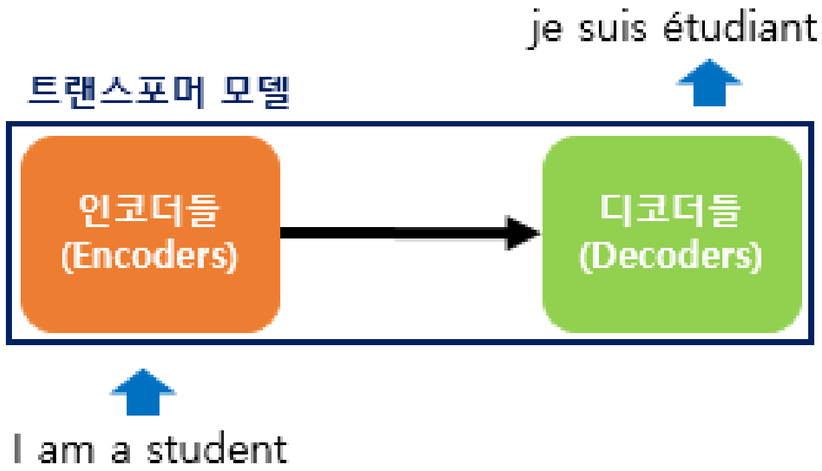
BrAIn Lab.
Brain and Artificial Intelligence Lab.

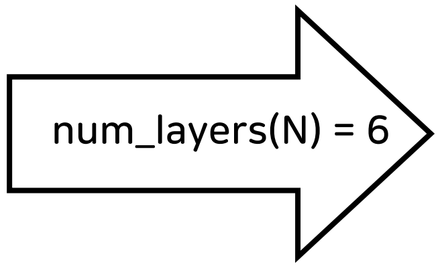
- Attention 의 기본 아이디어

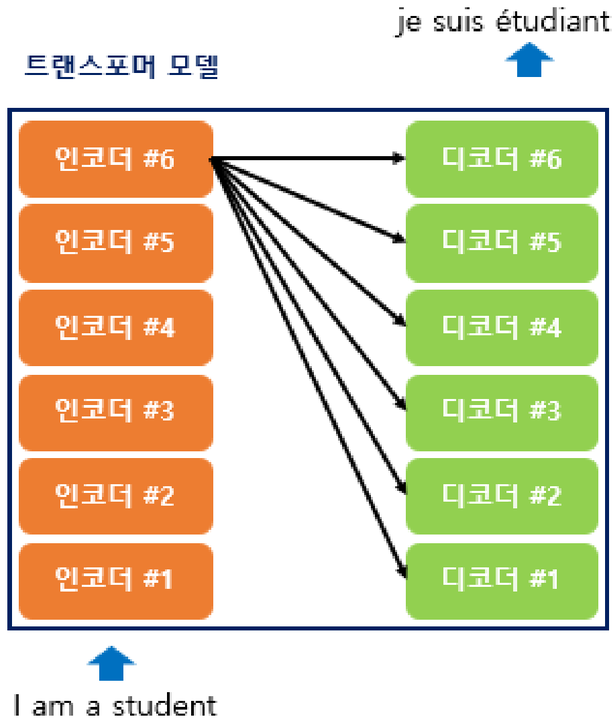
디코더에서 출력 단어를 예측하는 매 시점마다, 인코더에서의 전체 입력 문장을 다시 한 번 참고한다는 점이다.

- Transformer

기존의 Seq2Seq의 구조인 인코더-디코더를 따르면서도 Attention만으로 구현한 모델

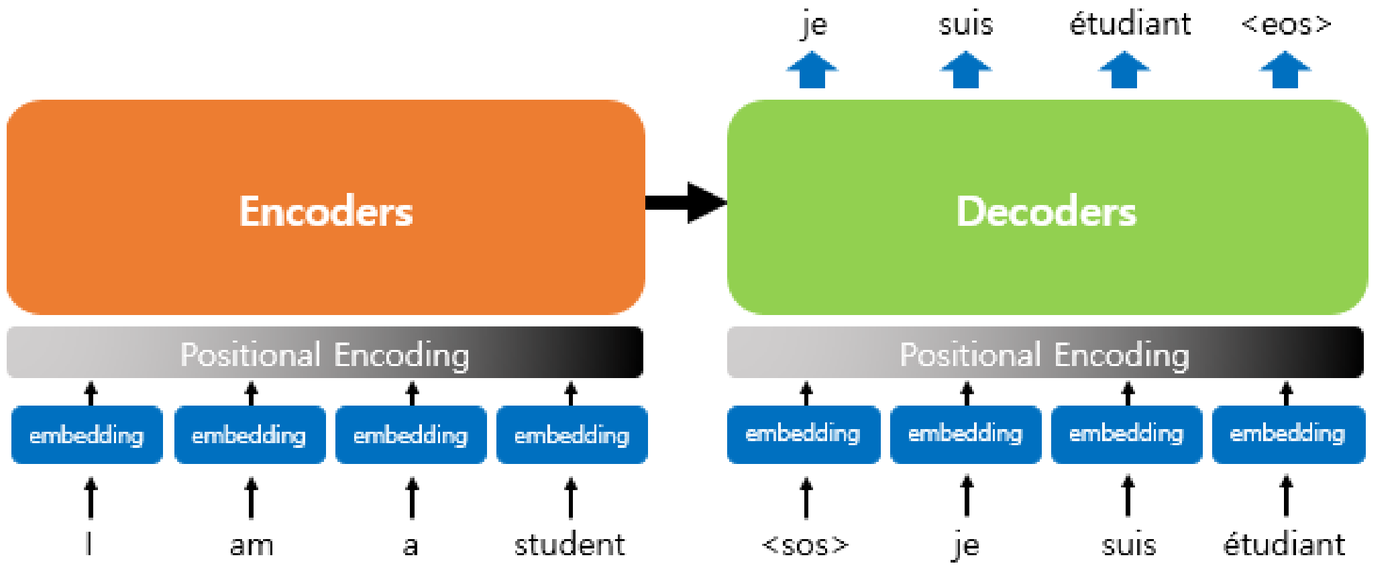




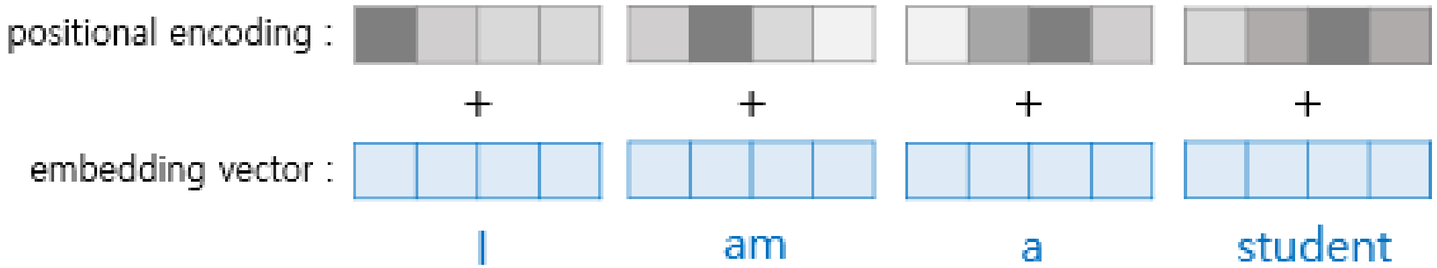


2. Transformer

디코더는 마치 기존의 Seq2Seq 구조처럼 시작 심볼 <sos>를 입력으로 받아 종료 심볼 <eos>가 나올 때까지 연산을 진행한다.



Transformer는 문장 전체를 한 번에 입력 받아 처리하기 때문에 단어 입력을 순차적으로 받는 방식이 아니므로, 단어의 위치 정보를 얻기 위해 각 단어의 임베딩 벡터에 위치 정보들을 더하여 모델의 입력으로 사용한다.



2. Transformer

BrAIn Lab.
Brain and Artificial Intelligence Lab.

해당 단어의 위치 정보를 담고 있는 positional encoding 벡터는 임베딩 벡터와 동일한 차원으로 만들어지며, 짝수 인덱스에는 사인 함수의 결과 값이, 홀수 인덱스에는 코사인 함수의 결과 값이 들어간다.

$$PE_{(pos,\,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,\,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

여기서 주파수는 차원 인덱스가 커질수록 점점 느리게 변하도록 설계되어 각 위치미다 고유한 패턴의 벡터가 생성된다.

문장에서 단어 위치(pos)	1번 값	2번 값	3번 값	4번 값
10	sin(1) = 0.84	cos(1) = 0.54	0.01	1.00
11	sin(1.1) = 0.89	cos(1.1) = 0.45	0.011	1.00
20	sin(2) = 0.91	cos(2) = -0.42	0.02	0.99

Positional Encoding의 핵심 원리는 한 위치에서 다른 위치로의 관계가 일관된 규칙으로 표현된다는 점이다. 모델은 이를 학습하여 "아, 이 단어는 내가 지금 보는 단어와 '세 칸 앞 변환' 관계에 있구나" 하고 상대적인 거리를 파악할 수 있게 된다.

2. Transformer

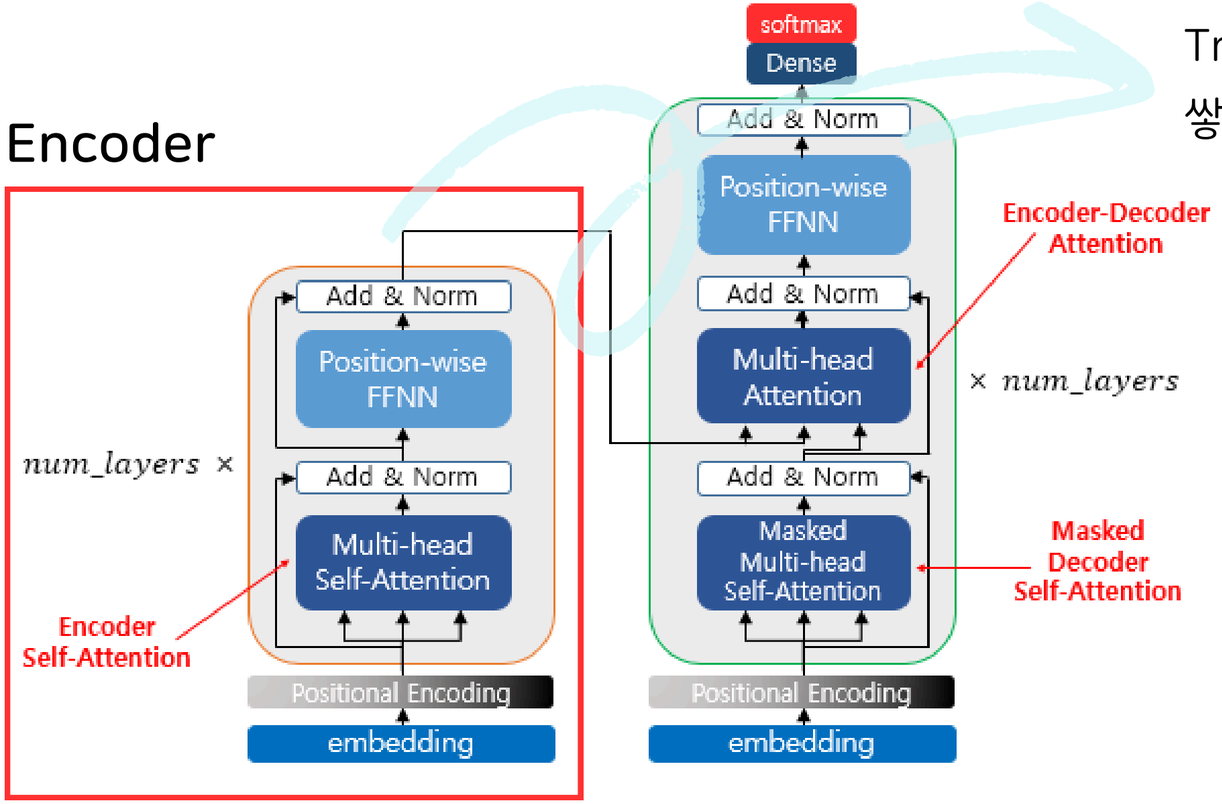
BrAIn Lab.
Brain and Artificial Intelligence Lab.

이제 Transformer의 입력은 순서 정보가 고려된 임베딩 벡터가 된다.

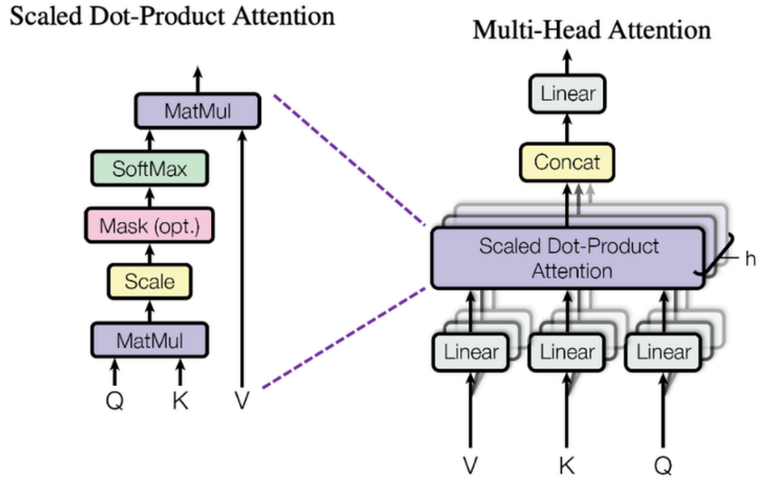
→ 같은 단어라 하더라도 문장 내의 위치에 따라 Transformer의 입력으로 들어가는 임베딩 벡터 값이 달라진다.

- **Transformer에서 사용되는 세 가지의 Attention**

Encoder



Transformer는 하이퍼파라미터인 “num_layers” 개수 만큼 인코더 층을 쌓게 되는데, 하나의 인코더 층은 크게 총 2개의 sublayer로 나뉜다.

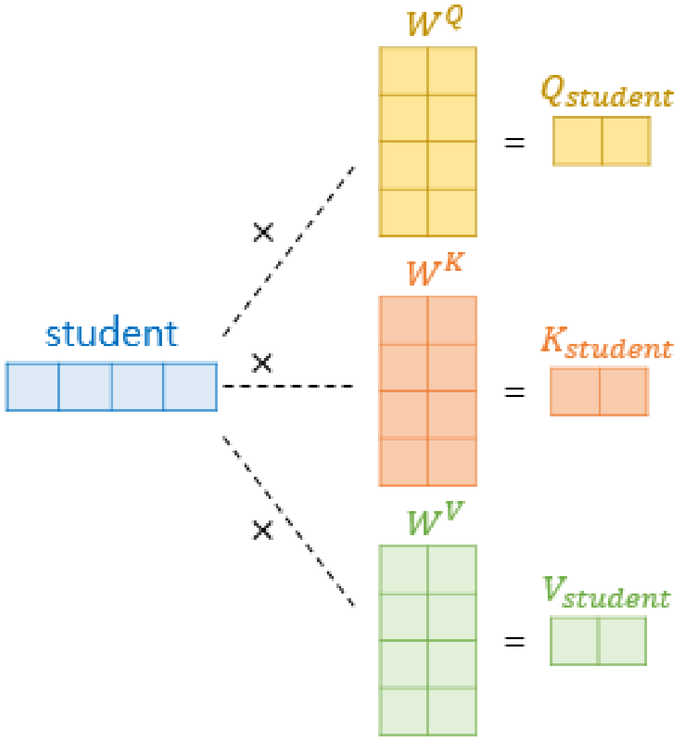


2. Transformer

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- Encoder의 Self-Attention

인코더의 초기 입력인 단어 벡터들을 사용하여 Self-Attention을 수행하는 것이 아니라, 각 단어 벡터들로부터 Q, K, V 벡터를 얻는 작업을 거친다.

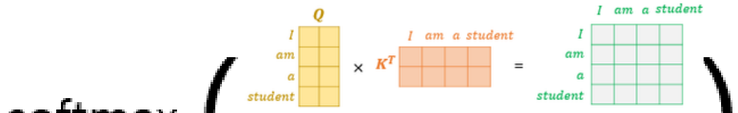


인코더의 입력인 각 단어 벡터: $1 \times d_{model}$ → 각 가중치 행렬: $d_{model} \times \frac{d_{model}}{num\_heads}$ → Q, K, V 벡터들: $\frac{d_{model}}{num\_heads}$

각 단어의 Q 벡터는 모든 단어의 K 벡터에 대해서 Attention Score를 구하고 이를 사용하여 모든 단어의 V 벡터를 가중합하여 Attention Value 또는 Context Value를 구하게 된다.

$$score(q,k) = \frac{q \cdot k}{\sqrt{n}} (= \frac{d_{model}}{num\_heads}).$$

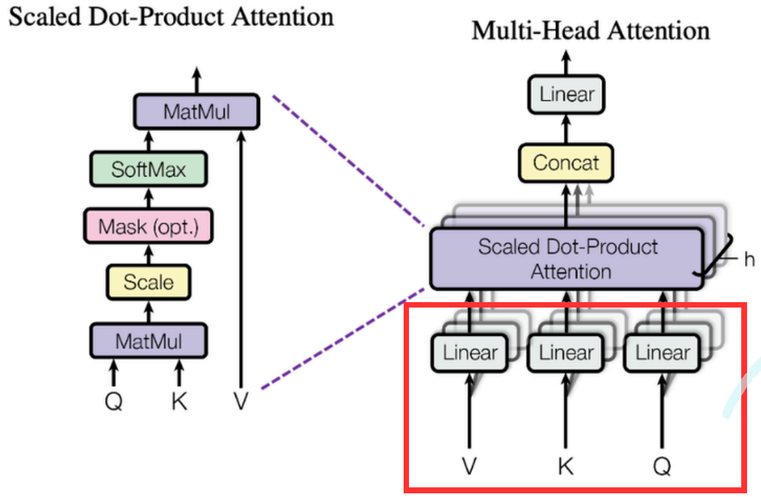
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) \times V = \text{Attention Value Matrix } a$$



2. Transformer

- “Multi-Head” Self-Attention

각 헤드가 단어 하나씩을 맡아서 처리하는 것이 아니라, 입력된 문장 전체(모든 단어 벡터)가 모든 헤드에 각각 들어가지만 각 헤드는 서로 다른 관점에서 문장 내부의 관계를 파악하여 각자의 결과물을 내놓는다.



Head 1

Head 2

. . .

입력: “I am a student”

각 단어에 대해 “Head 1(의미적 관점) 30%, Head 2(문법적 관점) 50% …” 반영하여 최종 결과를 내놓는다.

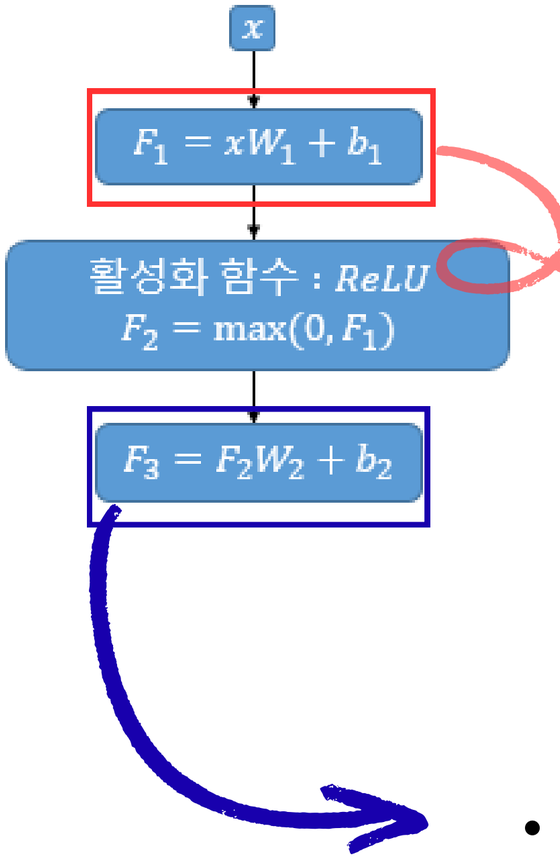
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(Z_1, Z_2, \ldots, Z_h) \cdot W_o$$

→ 여러 헤드의 다양한 관점이 종합된 최종적인 하나의 결과 행렬 Z가 생성되며, 이는 다시 원래의 입력과 동일한 차원을 갖게 된다.

2. Transformer

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- Position-wise FFNN(Fully-connected Neural Network)



Multi-Head Self-Attention을 거쳐 나온 결과 행렬 $(\text{단어 개수} \times d_{model})$ 이 있을때, PFFN은 이 행렬 전체를 한 번에 입력받는 것이 아닌 행렬의 각 행을 하나씩 독립적으로 통과시키게 되며, 동일한 가중치를 공유하게 된다.

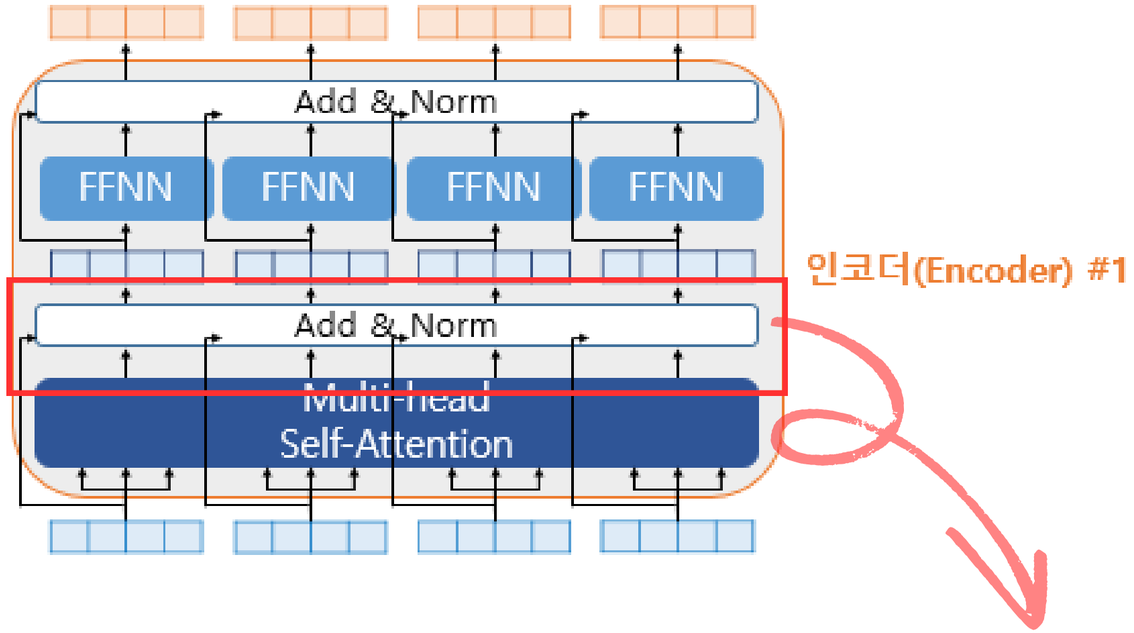
- **차원 확장**

"사과"라는 단어가 "음식" 관점 vs "IT 기기" 관점 모두 희미 → 더 뚜렷한 방향으로

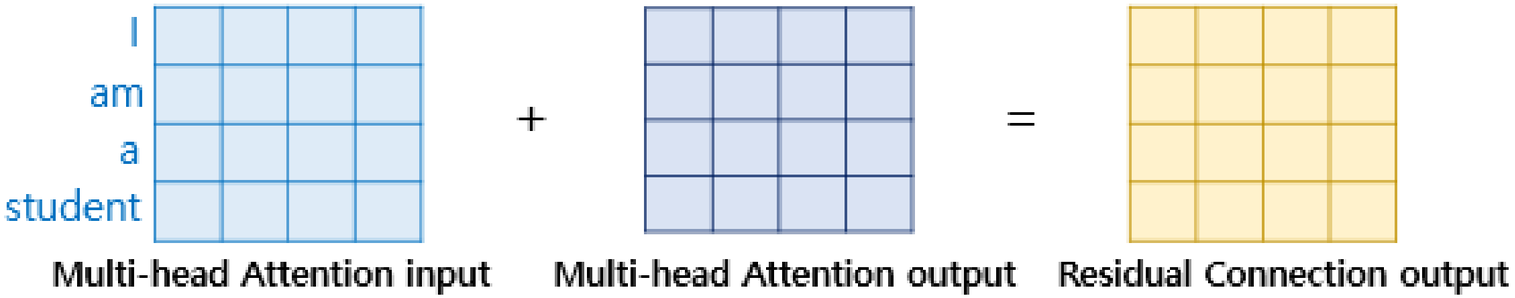
"먹다" + "사과" → "과일을 섭취하는 행위"

- **다시 원래 차원으로**

2. Transformer



- 잔차 연결(Add) & 층 정규화: Layer Norm(x + Multi-Head Attention(x))



3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- CNN Encoder: 1D CNN(Temporal CNN) → Layer Normalization → GELU 활성화 함수

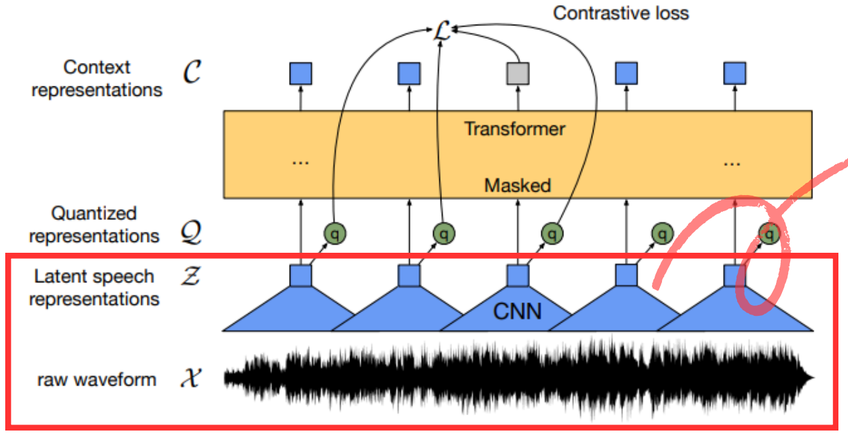
Layer 1: Stride=5, Filter Sizes=10, Channel=1 → 512

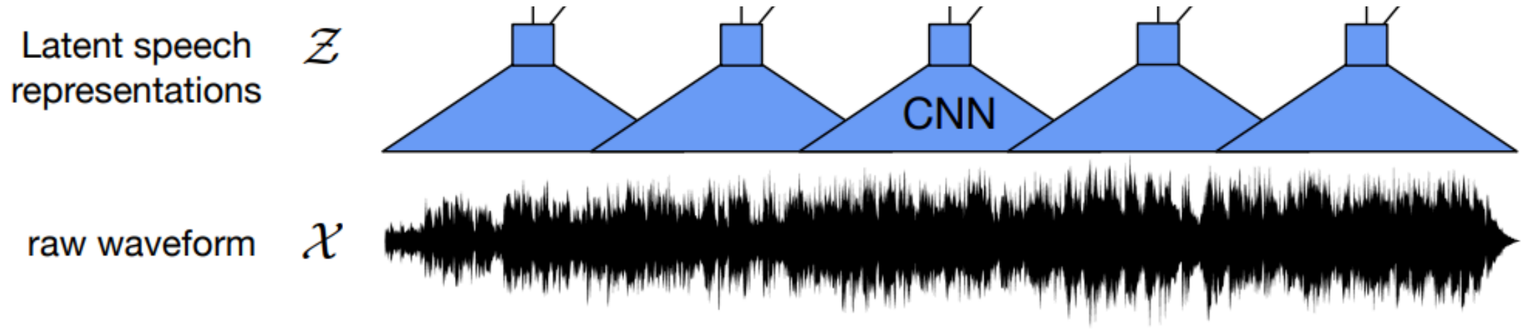
Layer 2: Stride=2, Filter Sizes=3, Channel=512 → 512

...

Layer 6: Stride=2, Filter Sizes=2, Channel=512 → 512

Layer 7: Stride=2, Filter Sizes=2, Channel=512 → 512





3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- CNN Encoder: 1D CNN(Temporal CNN) → Layer Normalization → GELU 활성화 함수

입력: [0.1, 0.3, 0.8, 0.5, -0.2, ...] → 샘플링된 원본 음성(Raw Audio)

1D 필터: [+1, +1, -1]과 같은 1D CNN 필터가 적용된다. → “값이 잠시 올라갔다가 뚝 떨어지는” 패턴을 감지

→ CNN Encoder는 하나의 긴 음성 파형을 약 20ms 간격으로 (줄어든 시간 길이, 512)인 $[z_1, z_2, \ldots, z_T]$ 을 내놓으며, 이때 특정 시점 t의 정보를 512개의 다른 종류의 음향 특징 값으로 담고 있다.

- F1 : 날카로운 어택 강도 ('ㅌ' 같은 소리)
- F2 : 모음 지속성 ('ㅏ' 같은 소리)
- F3 : 고주파 잡음
- F4 : 저주파 웅웅거림

```
      F1    F2    F3    F4
t=1 [ 0.1,  0.2,  0.8,  0.1 ]  (잡음이 많은 구간)
t=2 [ 1.8,  0.1,  0.2,  0.0 ]  ('ㅌ' 소리가 터지는 구간)
t=3 [ 0.2,  1.5,  0.1,  0.1 ]  ('ㅏ' 소리가 시작되는 구간)
t=4 [ 0.1,  1.3,  0.0,  0.2 ]  ('ㅏ' 소리가 지속되는 구간)
t=5 [ 0.3,  0.2,  0.1,  0.9 ]  (다른 소리가 시작되는 구간)
```

Layer 1 → Layer 2

목표: Layer 2의 한 필터가 “타” 음절 패턴을 찾는다.

이 필터의 목표는 **'날카로운 어택(F1)이 먼저 나온 뒤, 모음 지속성(F2)이 뒤따라 나오는' 3개 시간 길이의 패턴을** 찾는 것이다.

- 필터의 모양: 가중치 행렬 (3 x 4 크기) = “타” 음절의 이상적인 패턴 템플릿

```
       F1    F2    F3    F4
t-1 [+1.5, -0.5, -0.2, -0.1 ]  <- 't-1' 시점에서는 F1(어택)이 중요!
t   [-0.8, +1.2, -0.1, -0.2 ]  <- 't' 시점에서는 F2(모음)가 중요!
t+1 [-0.3, +1.0,  0.0, -0.1 ]  <- 't+1' 시점에서도 F2(모음)가 중요!
```

3. Wav2vec 2.0

- **Transformer의 Encoder에 들어가기 직전 Masking**

CNN 인코더 출력인 전체 시간 스텝 중 무작위 6.5%를 마스킹의 시작 위치로 샘플링하며, 그 지점부터 연속된 10개의 시간 스텝인 (10, 512)을 하나의 덩어리로 묶어 Masking 한다.

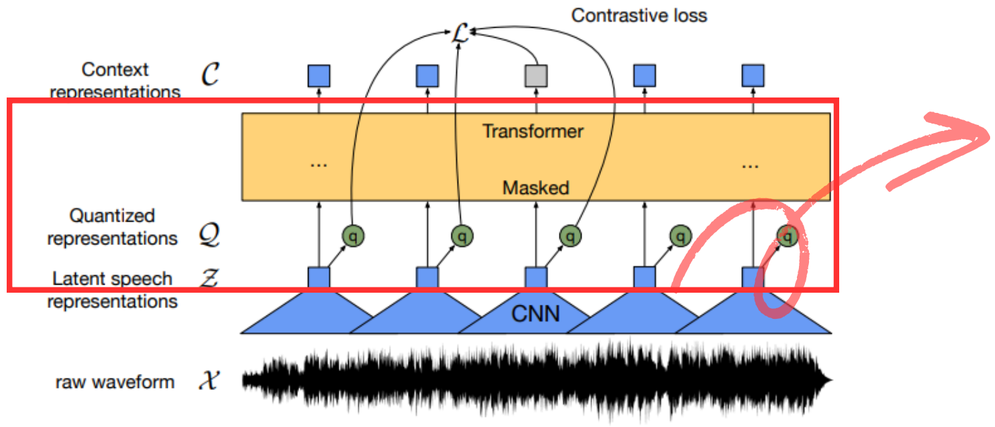
→ Masking 덩어리들은 서로 중첩될 수 있기 때문에 최종적으로 마스킹되는 시간 스텝은 전체의 약 49%

- **Transformer 구조의 Encoder만 사용**

원본 음성이 CNN Encoder를 통과해 나온 음성 시퀀스 Z를 Masking 하여 입력으로 넣어 문맥 정보가 풍부하게 담긴 또 다른 표현 시뭔스를 만드는 것이다.

→ CNN Encoder를 통과한 후 여전히 언어, 화자, 주변 소음 등 정보가 섞여 있는 “날 것”의 상태이기에 Transformer는 마스킹된 연속적인 입력을 보고 이산적인 정답이 무엇일지 예측하도록 학습된다.

3. Wav2vec 2.0



- **Quantization module: 어떠한 소리의 대표 심볼(정답지)**

CNN Encoder의 출력이자 마스킹하기 전 Z는 Codebook에 있는 모든 Codevector들 각각과의 유사도를 계산한다.

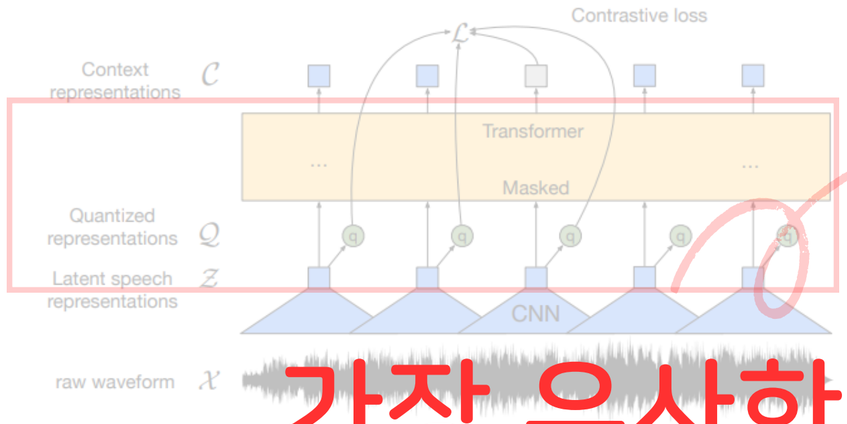
→ 유사도 점수를 통해 가장 유사한 Codevector 하나를 "선택"

- 전체 손실 함수: $L = L_m + \alpha L_d$

$$L_m + \alpha L_d = -\sum_{t \in M} log \frac{exp(sim(c_t, q_t)/k)}{\sum_{\tilde{q} \in Q_t} exp(sim(c_t, \tilde{q})/k)} + \alpha(-\frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} p_{g,v} log\ p_{g,v})$$

"마스킹 된 부분의 최종 문맥 벡터 C"를 Quantization module을 통해 나온 정답지와 유사해지도록, 다른 오답과는 멀어지도록 "대조 손실"을 바탕으로 학습한다.

3. Wav2vec 2.0



- Quantization module: 어떠한 소리의 대표 심볼(정답지)

CNN Encoder의 출력이자 마스킹하기 전 Z는 Codebook에 있는 모든 Codevector들 각각과의 유사도를 계산한다.

→ 유사도 점수를 통해 가장 유사한 Codevector 하나를 "선택"

가장 유사한 Codevector 하나를 "선택"

→ 미분 불가능!!

- 전체 손실 함수: $L = L_m + \alpha L_d$

$$L_m + \alpha L_d = -\sum_{t \in M} log \frac{exp(sim(c_t, q_t)/k)}{\sum_{\tilde{q} \in Q_t} exp(sim(c_t, \tilde{q})/k)} + \alpha \left(\frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} p_{g,v} log\, p_{g,v} \right)$$

"마스킹 된 부분의 최종 문맥 벡터 C"를 Quantization module을 통해 나온 정답지와 유사해지도록, 다른 오답과는 멀어지도록 "대조 손실"을 바탕으로 학습한다.

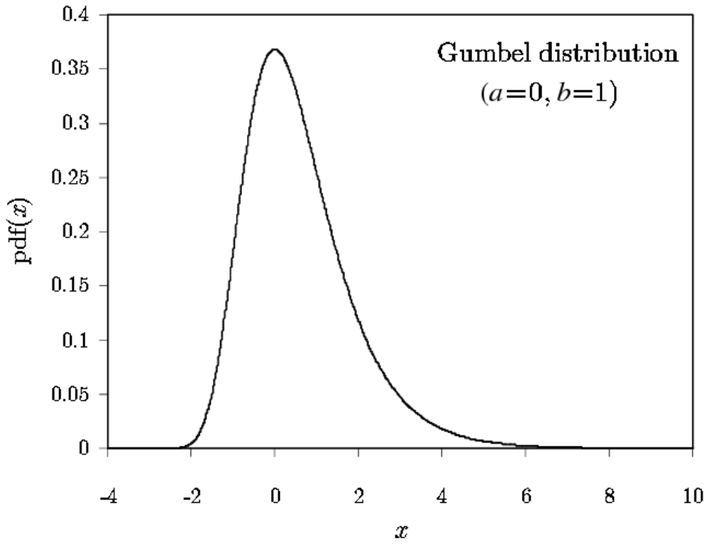
3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- **Gumbel-Softmax**

순전파 시에는 가장 확률 높은 하나의 Codevector를 선택했지만, 역전파 시에는 Gumbel-Softmax가 계산한 "soft한" 확률값들을 활용한다.

어떤 음성 특징 Z에 대해 각 Codevector가 선택될 확률: 유사도 점수 + Gumbel-noise → Softmax
역전파 시 이 확률 값들이 일정의 "가중치" 역할을 하여, Codevector e1 (70%), e2(25%), 나머지 ei(5%)
→ Codevector e1은 입력 Z의 방향으로 가장 많이 업데이트, e2는 약간 업데이트 …



lity density function of the standard Gumbel distribution.

3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- Gumbel-Softmax

순전파 시에는 가장 확률 높은 하나의 Codevector를 선택했지만, 역전파 시에는 Gumbel-Softmax가 계산한 "soft한" 확률값들을 활용한다.

어떤 음성 특징 Z에 대해 각 Codevector가 선택될 확률: 유사도 점수 + Gumbel-noise → Softmax

역전파 시에 확률 값들이 일정한 "가중치" 역할을 하여, Codevector e1(70%), e2(25%), 나머지 ei(5%)

→ Codevector e1은 입력 Z의 방향으로 가장 많이 업데이트, e2는 약간 업데이트 ...

Gumbel-Softmax를 활용하여 미분 가능!!

→ Codevector 업데이트 가능!!

3. VQ (Vector Quantization)

BrAIn Lab.
Brain and Artificial Intelligence Lab.

VQ는 연속적인 신호나 데이터를 미리 정해둔 한정된 개수의 대표 벡터(심볼, Codevector) 중 하나로 바꿔 표현하는 기술이다.

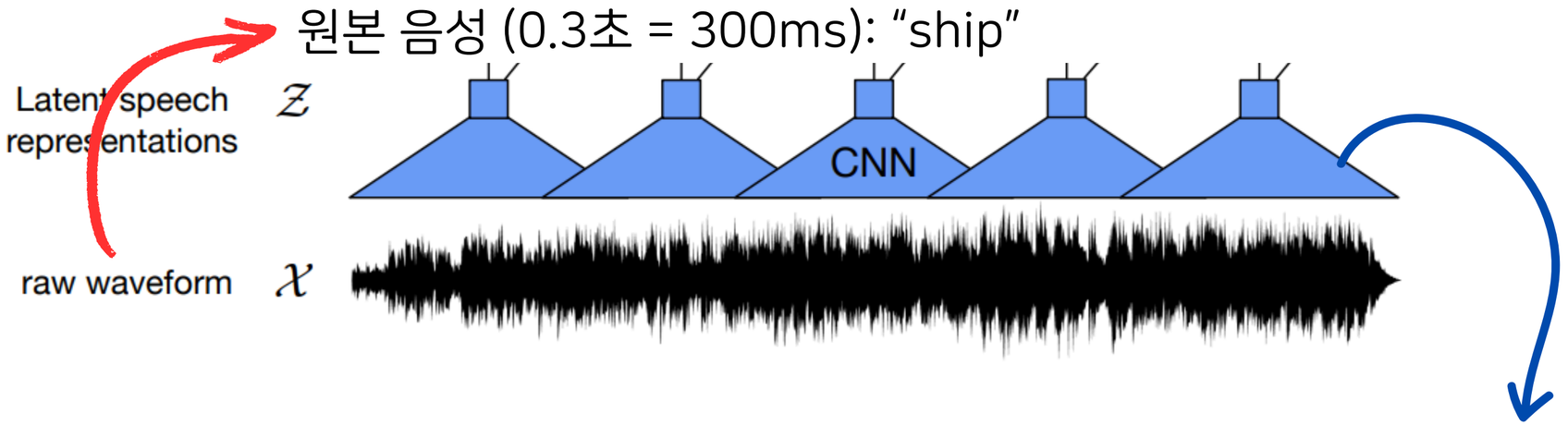
→ 정보의 손실을 최소화하면서 데이터를 압축하고 이산화(discrete)하는 기술

$$L_{vq} = ||sg(z_e(x)) - e_k||_2^2$$

$$e_k = \frac{1}{N_k} \sum_{z \in S_k} z$$

손실 함수를 최소화하는 것은 특정 Codevector가 자신을 선택한 모든 CNN Encoder의 중심으로, 클러스터의 중심을 만드는 것과 같다.

3. Wav2vec 2.0



20ms 간격으로 특징 벡터를 출력하므로, 0.3초의 음성은 약 15개의 연속적인 특징 벡터 시퀀스로 변환된다.

$z_1 \sim z_6$: ‘/ʃ/’(sh) 소리의 특징을 담고 있다.
$z_7 \sim z_{12}$: ‘/ɪ/’(i) 소리의 특징을 담고 있다.
$z_{13} \sim z_{15}$: ‘/p/’(p) 소리의 특징을 담고 있다.

3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

Step 0. 초기 상태

음성 '/ʃ/'의 특징 벡터 z1: [0.8, 0.1, 0.9]

랜덤 초기화된 Codebook E: $e_1 = [0.1, 0.9, -0.2]$

$e_2 = [0.7, -0.6, 0.3]$

$e_3 = [-0.5, 0.2, 0.4]$

$e_4 = [-0.8, -0.1, 0.5]$

코사인 유사도가 우연히 가장 높다!!

→ e2가 가장 큰 폭으로 업데이트되어 z1에 가장 많이 가까워진다.

Step 50,000. 학습 중기

수많은 '/ʃ/' 소리 벡터 z들이 e4를 직간접적으로 업데이트하여, e4가 클러스터의 중심으로 이동!!

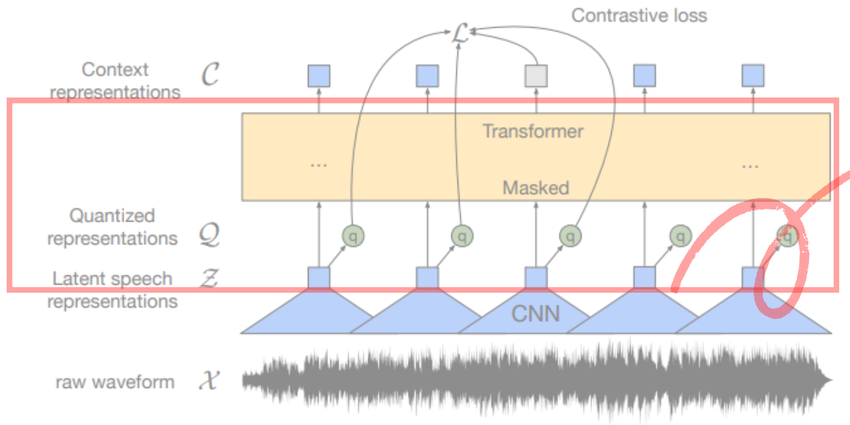
업데이트된 Codebook E:

- e_1 → 'a' 모음 전문가: `[ 0.2, 0.9, 0.1]`
- e_2 → 't' 파열음 전문가: `[ 0.8, -0.7, -0.1]`
- e_3 → 'k' 파열음 전문가: `[-0.9, 0.1, -0.3]`
- **e_4 → '/ʃ/' 소리 전문가: `[ 0.75, 0.05, 0.85]`**

압도적으로 높다!!

새로운 '/ʃ/' 특징 벡터 z_new 입력:[0.9, -0.05, 0.85]

3. Wav2vec 2.0



- Quantization module: 어떠한 소리의 대표 심볼(정답지)

CNN Encoder의 출력이자 마스킹하기 전 Z는 Codebook에 있는 모든 Codevector들 각각과의 유사도를 계산한다.

→ 유사도 점수를 통해 가장 유사한 Codevector 하나를 "선택"

- 전체 손실 함수: $L = L_m + \alpha L_d$

$$L_m + \alpha L_d = -\sum_{t \in M} log \frac{exp(sim(c_t, q_t)/k)}{\sum_{\tilde{q} \in Q_t} exp(sim(c_t, \tilde{q})/k)} + \alpha(-\frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} p_{g,v} log\ p_{g,v})$$

학습이 한 쪽으로 치우치지 않도록 강제하는 제약의 역할로,
Codebook의 모든 Codevector를 가능한 한 골고루 사용하도록 장려하여
Codebook 전체가 풍부한 표현력을 갖도록 돕는다.

3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

라벨링된 학습 데이터가 매우 적은 “저자원” 환경에서 성능 비교표

음성 인식 모델의 성능을 나타내는 단어 오류율(WER)을 사용
→ 수치가 낮을수록 좋은 모델

사전학습에 사용된 라벨 없는 데이터 Librispeech(LS-960)일 때와 훨씬 더 큰 LibriVox(LV-60k)일 때의 성능 차이

Model	Unlabeled data	LM	dev		test	
			clean	other	clean	other
10 min labeled						
Discrete BERT [4]	LS-960	4-gram	15.7	24.1	16.3	25.2
BASE	LS-960	4-gram	8.9	15.7	9.1	15.6
		Transf.	6.6	13.2	6.9	12.9
LARGE	LS-960	Transf.	6.6	10.6	6.8	10.8
	LV-60k	Transf.	4.6	7.9	4.8	8.2
1h labeled						
Discrete BERT [4]	LS-960	4-gram	8.5	16.4	9.0	17.6
BASE	LS-960	4-gram	5.0	10.8	5.5	11.3
		Transf.	3.8	9.0	4.0	9.3
LARGE	LS-960	Transf.	3.8	7.1	3.9	7.6
	LV-60k	Transf.	2.9	5.4	2.9	5.8
10h labeled						
Discrete BERT [4]	LS-960	4-gram	5.3	13.2	5.9	14.1
Iter. pseudo-labeling [58]	LS-960	4-gram+Transf.	23.51	25.48	24.37	26.02
	LV-60k	4-gram+Transf.	17.00	19.34	18.03	19.92
BASE	LS-960	4-gram	3.8	9.1	4.3	9.5
		Transf.	2.9	7.4	3.2	7.8
LARGE	LS-960	Transf.	2.9	5.7	3.2	6.1
	LV-60k	Transf.	2.4	4.8	2.6	4.9
100h labeled						
Hybrid DNN/HMM [34]	-	4-gram	5.0	19.5	5.8	18.6
TTS data augm. [30]	-	LSTM			4.3	13.5
Discrete BERT [4]	LS-960	4-gram	4.0	10.9	4.5	12.1
Iter. pseudo-labeling [58]	LS-860	4-gram+Transf.	4.98	7.97	5.59	8.95
	LV-60k	4-gram+Transf.	3.19	6.14	3.72	7.11
Noisy student [42]	LS-860	LSTM	3.9	8.8	4.2	8.6
BASE	LS-960	4-gram	2.7	7.9	3.4	8.0
		Transf.	2.2	6.3	2.6	6.3
LARGE	LS-960	Transf.	2.1	4.8	2.3	5.0
	LV-60k	Transf.	1.9	4.0	2.0	4.0

3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- 단일 모델 아키텍처(End-to-End) 구조
- 대규모 비지도 데이터 활용:

라벨링 되지 않은 53,000 시간 이상의 방대한 데이터로 사전학습 수행

- 파인 튜닝의 효율성:

사전 학습된 모델을 가져와 “아주 적은 양의 라벨링된 데이터 (단 10분)”만으로도 특정 도메인에 대한 음성 인식 모델을 만들 수 있으며 높은 성능을 보인다.

→ 음성 분야에서도 NLP의 BERT 처럼 “사전 학습 → 파인 튜닝” 패러다임이 성공적으로 정착!!
→ 라벨링 데이터 부족 문제를 해결하여, 한국어와 같이 데이터가 상대적으로 부족한 언어의 음성 인식 기술 발전에도 크게 기여할 수 있는 가능성

3. Wav2vec 2.0

BrAIn Lab.
Brain and Artificial Intelligence Lab.

- 막대한 사전 학습 비용

소량의 데이터로 “미세 조정”이 가능한 것은 사실이지만 그 전제 조건인 “사전 학습”에는 어마어마한 컴퓨팅 자원이 필요하다.

→ LARGE 모델의 경우 5만 3천 시간의 음성 데이터를 학습하기 위해 128개의 GPU로 5.2일 소요

- 실시간 스트리밍 처리의 어려움

Wav2vec 2.0의 Transformer 구조는 문장이나 발화 전체의 문맥을 파악해야 최상의 성능을 내기 때문에 전체 음성이 끝날 때까지 기다려야 하므로, 실시간으로 들어오는 음성은 지연 시간 문제가 발생한다.

- 문맥 처리의 한계 및 복잡한 음향 환경 대응의 한계

사전학습 데이터에 등장하지 않은 단어는 음소 기반으로 추측할 수밖에 없어 오류율이 높아지며,
여러 명의 화자가 동시에 말하는 "칵테일 파티 문제"나 매우 심한 소음 및 반향이 있는 환경에서는 여전히 성능이 저하될 수 있다.

→ 모델이 언어적 정보와 비언어적 노이즈를 분리하는 데는 한계가 있기 때문!!