기술조사보고서

트랜스포머 기반 최신 언어 모델 기법 분석

2021. 09

인공지능기술연구단

CONTENTS

목차

CONTENTS

그림목차

CONTENTS

표목차

제1장

서 론

제 1 절 언어 모델의 기술적 배경 및 연구 트렌드

1. 언어 모델의 기술적 배경

□ 기존 자연어 처리 기법

○ 확률론적 언어 모델(Probabilistic Language Model)

■ 언어 모델(Language Model; 줄여서 LM)은 단어의 나열(word sequence)로 표현되는 언어를 모델링하고자 단어의 나열 (또는 문장)에 확률을 할당하는 모델이다.

- 주어진 m 길이의 단어의 나열(word sequence)에 언어 모델은 확률 $P(w_1, w_2, \cdots, w_m)$을 할당

■ 단어의 나열에 확률을 할당하기 위해 사용되는 보편적인 방법은 언어 모델이 이전 단어들이 주어졌을 때 다음 단어를 예측하도록 하는 것이다.

- 즉, $P(w_i|w_1, w_2, \cdots, w_{i-1})$를 구하는 것이라 할 수 있다.

- 전체 단어의 나열 W에 대한 확률은 W를 구성하는 모든 단어에 대한 예측 확률이므로 다음과 같이 계산할 수 있다.

$$P(W) = P(w_1, w_2, \cdots, w_m) = \Pi_{i=1}^{m} P(w_i|w_1, w_2, \cdots, w_{i-1})$$

■ 이 확률을 구하기 위해 각 단어의 출현 횟수에 기반을 두어 확률을 계산함.

- 즉, $P(w_i|w_1, w_2, \cdots, w_{i-1})$를 계산하는데 있어 모든 말뭉치(corpus)에서 $w_1, w_2, \cdots, w_{i-1}$이 100번 출현하였는데, 그 다음 w_i는 50번 출현하였다면, 확률은 50%가 됨

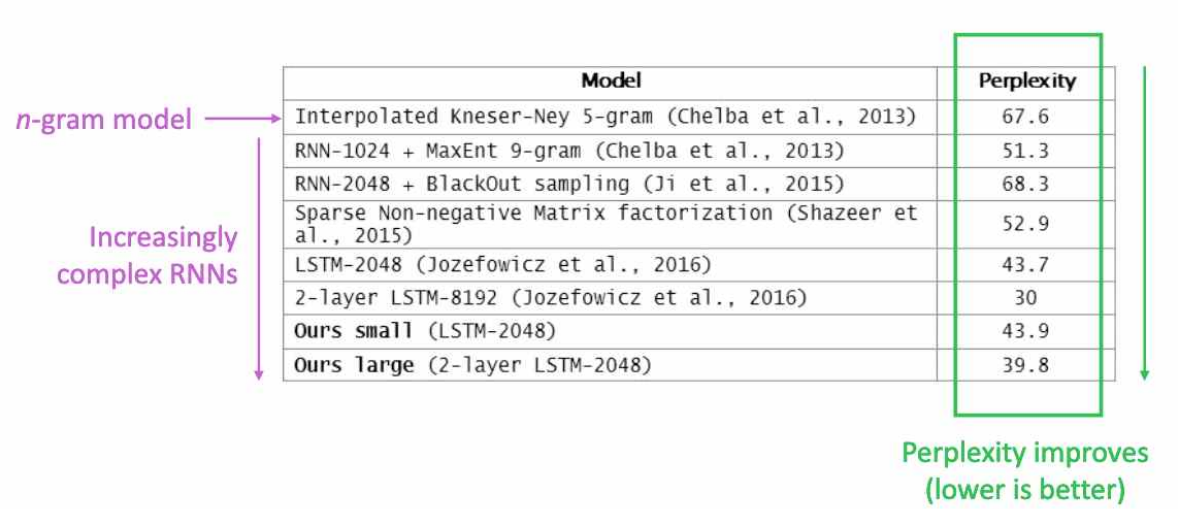
■ 그러나 현실에서의 확률 분포에서는 방대한 양의 말뭉치를 사용하고 있으므로, 해당 단어의 나열이 전체 말뭉치 크기에 비해 희소하므로 출현 횟수가 덜 빈번할 수 있으며 확률이 0과 견줄 정도로 미미한 값을 가지거나 계산할 수 없는 가능성이 커짐. 이러한 문제를 희소 문제(sparsity problem)이라 함

- 즉, $P(w_i|w_1,w_2,\cdots,w_{i-1})$를 계산하는 데 있어 단어의 나열이 길어짐에 따라 해당 코퍼스(corpus)에서 이 단어의 나열이 존재할 가능성은 갈수록 적어짐. 때문에 전체 단어의 나열에 대해 확률을 계산하지 않고, 이전 단어들 중 임의의 개수만 포함하여 출현횟수를 계산하는 방식으로 근사화하기 위한 n-gram 언어 모델이 도입됨
- n-gram 언어 모델은 n 개의 연속적인 단어의 나열을 하나의 토큰으로 간주함
  - 코퍼스의 각 문장들을 n 개의 단어 뭉치로 끊고, 이를 하나의 토큰으로 간주함
  - 가령 문장이 Bill Cosby released after assault conviction overturned이 주어진다고 할 때, n=1인 경우는 Bill, Cosby, released, after, assault, conviction, overturned로 끊어지며 이를 유니그램(unigram)이라고 함
  - n=2인 경우 바이그램(bigram)이라 하며, 이 경우, 위 예제 문장에 대한 bigram은 Bill Cosby, Cosby released, released, after, after assault, assault conviction, conviction overturned로 생성됨
  - n=3인 경우, 트라이그램(trigram)이라 하며, 위 예제 문장에 대한 trigram은 Bill Cosby released, Cosby released after, released after assault, after assault conviction, assault conviction overturned과 같이 생성되며, 그 이상의 n 값을 가지고 n-gram을 추출할 수 있음
- n-gram 언어 모델에서는 다음에 나올 단어의 예측은 오직 그 전의 gram에만 의존함. 예를 들어, Bill Cosby released after assault conviction _____ 과 같은 문장의 다음에 나올 단어 w_i를 예측하기 위해 트라이그램에서는 “after assault conviction” 3개의 단어만을 고려하여 처리함. 즉, $P(w_i|after\,assault\,conviction)$ 으로 확률을 계산 함
- n-gram 언어 모델은 희소 문제를 줄일 수는 있지만, 여전히 희소 문제의 발생 가능성이 존재할 수 있음. 더불어 적절한 n값을 선택하는 데 있어서의 trade-off가 발생함.
  - n 값을 작게 하면, 다음 단어 예측을 위해 봐야 할 단어의 개수가 줄어드

는 만큼 실제 문장 전체를 보는 것 대비 근사의 정확도는 멀어지게 됨. 반대로 n 값을 크게 하면, 실제 코퍼스에서 해당 n-gram이 출현할 확률은 적어지므로 희소 문제가 심각해질 수 있음.

- Perplexity(줄여서 PPL)는 두 개의 언어 모델 간의 성능을 비교하기 위해 사용되는 평가 척도임
  - 외부 평가(extrinsic evaluation)은 언어 모델들을 가지고 특정 태스크들을 수행한 결과를 바탕으로 평가하는 방식으로 SQuAD, GLUE 등과 같은 벤치마크가 이에 해당
  - 내부 평가(intrinsic evaluation)는 테스트 데이터에 대하여 빠르게 식으로 계산하는 평가 방식으로 Perplexity는 내부 평가 지표로 주로 활용하는 척도임
- Perplexity(PPL)은 단어의 수로 정규화(normalization)된 테스트 데이터에 대한 확률의 역수로 정의함
  - 문장 W의 길이가 m이라고 할 때 이 문장에 대한 PPL은 아래와 같이 정의할 수 있음. $PPL(W) = P(w_1, w_2, \cdots, w_m)^{-\frac{1}{m}} = \sqrt[m]{\frac{1}{P(w_1, w_2, \cdots, w_m)}}$
  - 바이그램 언어 모델의 경우 PPL은 다음과 같이 정의할 수 있음 $PPL(W) = \sqrt[m]{\frac{1}{\prod_{i=1}^{m} P(w_i|w_{i-1})}}$. 결국 PPL을 최소화한다는 것은 문장의 확률을 최대화하는 효과와 동일함
- 이 PPL을 이용하여 기존의 n-gram 언어 모델과 딥러닝 기반 언어 모델을 비교해보면 기존의 n-gram 언어 모델 대비 딥러닝 기반 언어 모델의 성능이 월등히 좋으며, 레이어가 깊어질수록 훨씬 나은 성능을 보이고 있음을 확인할 수 있음
- 이에 따라 최근의 연구들은 자연어 처리에서 보다 높은 성능을 보이는 딥러닝 기반 방식을 채택하고 있음
- 물론 전통적인 확률론적 언어 모델은 아직 단어 출현 횟수(term frequency)에 기반을 두는 정보 검색이나 기본적인 NLP 분야에서도 계속 사용되고 있으나, 딥러닝 모델을 활용한 방식이 이제는 전통적인 확률론적 언어 모델을 대체하여

자연어 처리 분야의 주류로 취급되고 있음



Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

[그림 1-1] n-gram 모델과 RNN 기반 모델의 비교
(출처: http://web.stanford.edu/class/cs224n/)

□ 딥러닝 기반 언어 모델

○ 개요

■ 딥러닝 기반 언어 모델은 크게 딥러닝 기반 **태스크 전용 모델**과 **범용의 언어 모델**로 구분될 수 있음

■ 태스크 전용 모델은 특정 자연어 처리 태스크(예, 품사 태깅, 관계 추출 등)를 수행할 수 있도록 고안된 태스크 전용의 딥러닝 모델을 이용하는 것임

- 태스크 전용 모델은 한 모델을 가지고 여러 개의 자연어 처리 태스크를 수행할 수 있도록 하는 다중 태스크 모델(Multi task model)과 하나의 자연어 처리 태스크만을 수행하도록 고안된 단일 태스크 모델(Single task model)로 분류할 수 있음

■ 범용의 언어 모델은 ELMo[Peters2018a], BERT[Devlin2018a] 등과 같이 대량의 코퍼스를 이용하여 사전학습(pre-training)을 수행한 언어 모델을 범용으로 이용함. 특정 태스크를 수행하기 위해서는 사전학습된 언어 모델을 가지고 그 위에 새로운 레이어를 올린 후, 특정 태스크에 적합한 훈련 데이터를 가지고 다시 학습하는 과정을 거침. 이러한 과정을 파인 튜닝(fine tuning)이

라 함

■ 즉 딥러닝 기반 범용 언어 모델은 대량의 코퍼스를 이용한 사전 학습과 이후 수행할 태스크에 따른 파인 튜닝을 거치는 방식으로 이용됨

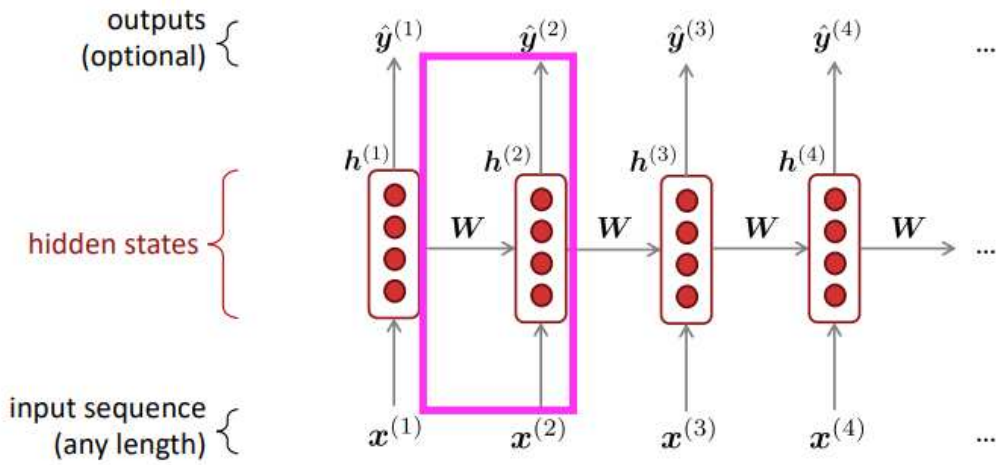
○ 딥러닝 기반 태스크 전용 모델

■ 태스크 전용 모델은 단어에 대한 벡터 표현을 먼저 단어 임베딩(word embedding) 기술을 이용하여 얻고 여기에 품사 정보나 위치 정보를 추가적으로 벡터화한 후 단어 임베딩 벡터와 결합하는 방식으로 입력 벡터를 구축함

- 단어 임베딩은 단어에 대한 주변 단어(context word)들과의 상관성을 학습하는 방법으로 단어들을 대표하는 벡터들을 생성해냄
- 대표적인 단어 임베딩으로는 word2vec [Mikolov2013a]이나 단어를 구성하는 부분어(subword) 정보까지 활용하는 FastText [Bojanowski2017a], gloVe [Pennington2014a] 등이 존재함
- 파서를 이용하여 얻은 각 단어에 대한 품사 정보(POS 태그)나 단어의 문장 내 출현 위치 정보 등을 원핫 인코딩(one-hot encoding) 기법 등으로 벡터화 하고 이를 해당 단어에 대한 임베딩 벡터와 결합하는 방식으로 모델에 대한 입력 벡터를 준비함

■ 태스크 전용 모델은 다양한 방식이 존재할 수 있으나, 대부분 LSTM(Long Short-Term Memory)[Sundermeyer2012a] 또는 GRU(Gated Recurrent Unit)[Cho2014a]를 유닛으로 활용하는 RNN[Mikolov2010a] 계열 모델이 주로 사용됨

- 자연어 문장의 경우 입력의 길이가 고정되지 않는 단어의 나열이나, CNN이나 완전연결 신경망 등은 고정 길이의 입력 벡터를 입력으로 함.
- 문장이 길어져도 모델 자체를 키울 필요가 없으며 가변 길이의 입력에 대해 RNN은 유연한 특징을 가짐
- 또한 RNN을 구성하는 데 있어 기울기 소실(vanishing gradients) 문제를 완화하기 위해 각 셀은 LSTM이나 GRU 등과 같은 유닛을 사용함



[그림 1-2] 자연어 처리를 위한 RNN 예시
(출처: http://web.stanford.edu/class/cs224n/)

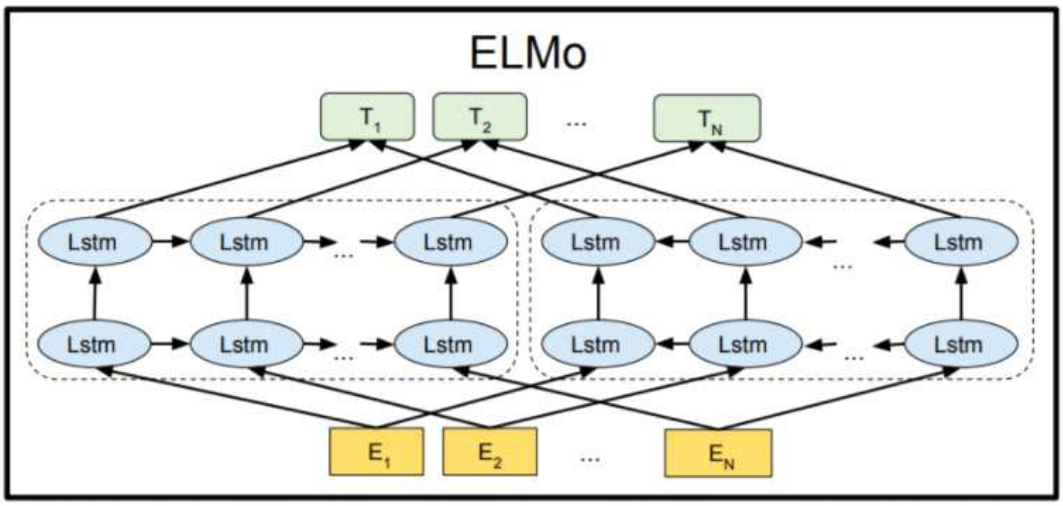
■ 태스크 전용 모델에서는 문장을 모델의 입력으로 하는 데 있어 문장을 단어들로 분리하고, 각 단어에 대응한 단어 임베딩 값을 입력 벡터로 함

- 이때 POS(Part-Of-Speech) 태그나 단어의 순서 정보 등을 원핫 인코딩(one-hot encoding) 등으로 벡터화한 후 이를 단어 임베딩과 결합하는 식으로 언어 처리용 자질(feature)을 추가하기도 함
- 태스크 전용 모델에서는 그동안 주로 LSTM을 셀로 하는 양방향(Bidirectional) RNN을 주된 모델로 하고, 단어 임베딩과 언어 처리용 자질을 결합한 벡터를 입력 벡터로 하여 특정 자연어 처리 태스크를 수행하도록 고안되어 왔음. 때문에 특정 태스크 별로 여러 모델들을 학습해야 할 필요가 존재하여 왔음

○ 딥러닝 기반의 범용 언어 모델

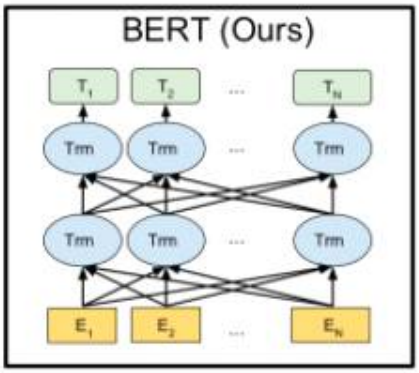
■ 딥러닝 기반의 범용 언어 모델은 대량의 코퍼스를 이용하여 언어 모델을 미리 학습해 두고, 이 미리 학습된 언어 모델(pre-trained language model)을 특정 자연어 처리 태스크를 위한 학습 데이터로 추가적으로 학습하는 방식(파인튜닝; fine tuning이라 함)으로 사용됨

- 범용 언어 모델의 사전 학습을 위해서는 대량의 코퍼스를 요구하며, 이를 기준으로 자가 학습(self-supervised learning)을 통해 언어 모델을 학습함
- 범용 언어 모델의 대표적인 모델로는 ELMo와 BERT, GPT, 그리고 이들의 다양한 개량형들이 존재함.
- ELMo의 경우, 2,048 개의 n-gram 컨볼루션 필터를 가지는 문자 수준 CNN을 먼저 이용하여 각 어휘에 대한 임베딩을 얻은 후, 이들 임베딩 벡터를 입력으로 하는 양방향 LSTM으로 모델을 구성함
- 양방향 LSTM은 언어 모델링에 있어 LSTM과 동일한 방식으로 학습함
  - 순방향 LSTM은 주어진 문장에서 시작부터 n 개의 단어를 보고 n+1 번째 단어를 예측하도록 학습
  - 역방향 LSTM은 반대로 주어진 문장의 끝부터 n개의 단어를 역순으로 보고, n-1 번째 단어를 예측하도록 학습함
  - 이때 순방향과 역방향 LSTM은 서로 파라미터를 공유하지 않고 완전히 분리된 상태로 학습이 진행되며, 학습 할 때는 log liklihood를 동시에 최대화하는 방식으로 학습을 수행함
  - Softmax 계층은 이 두 LSTM 위에 존재하고, 양방향의 LSTM이 생성한 은닉 상태(hidden state)를 결합한 벡터를 토대로 예측하고자 하는 단어를 맞추도록 학습을 수행하며, 이후 이 softmax를 제거하고, 특정 태스크를 수행하기 위한 계층을 붙이는 방식으로 사용
- BERT 언어 모델은 트랜스포머(transformer)라 명명된 멀티 헤드 어텐션 기법을 적용한 인코더-디코더 모델[Vaswani2017a] 중 인코더 모델을 이용하여 개발된 사전학습 기반 언어 모델임
  - 트랜스포머는 RNN을 사용하지 않고, 어텐션 기법만을 이용하여 기존의 인코더-디코더 또는 seq2seq 모델을 개선하기 위한 노력의 일환으로 고안됨
  - BERT는 일종의 마스킹된 언어 모델(Masked Language Model)의 일종으로 트랜스포머의 인코더 부분을 쌓아 올린 구조임. 기본 Base 모델은 총 12개, large 모델은 총 24개를 쌓아 올림



[그림 1-3] ELMo 언어 모델의 개략적인 구조 [Devlin2018a]

- 마스킹 된 언어 모델이란 입력 임베딩 중 일부를 마스킹(masking) 처리하고 출력 계층에서 해당 단어를 예측하도록 학습된 언어 모델을 의미함. 그림 1-4에서 보이는 바와 같이 BERT 모델 자체가 양방향성을 가지고 한 계층의 파라미터들이 다음 계층의 파라미터들의 학습에 영향을 주게 되어 있음. 이러한 양방향성 때문에 ELMo에서와 같이 순방향 또는 역방향으로 토큰들을 읽고 다음 토큰을 예측하는 방식이 아닌, 마스킹 된 토큰을 예측하는 방식으로 학습을 수행함
- 또한 BERT는 다음 문장 예측(NSP; Next Sentence Prediction)이라는, 두 문장 간의 선행 관계를 예측할 수 있도록 추가적인 학습을 통한 선행 학습을 수행함
- BERT의 경우 기존의 단어보다 더 작은 단위의 서브워드 토크나이저를 이용함. BERT의 경우 WordPiece 토크나이저를 이용하는데, 이는 글자로부터 서브워드들을 병합해 가는 방식으로 어휘 집합(vocabulary)를 만드는 방식임



[그림 1-4] BERT 모델 구조 개요
[Devlin2018a]

○ 이러한 범용의 언어 모델들은 모두 주어진 대량의 코퍼스를 가지고, 많은 학습을 수행하는 방식으로 대량 학습을 수행하고, 이후 주어진 태스크 별로 추가적인 학습 데이터를 가지고 파인 튜닝하는 방식으로 이용됨

○ 현재 자연어 처리 분야에서는 마스킹 된 언어 모델을 가져와 이를 파인 튜닝하는 방식이 기존 RNN 계열의 모델들을 밀어내고 거의 모든 분야에서 주요 모델들로 활용되고 있음.

2. 언어 모델의 연구 트렌드

□ 언어 모델의 지속적인 성장

○ 언어 모델의 발전은 자연어 처리 분야에서의 획기적인 성공을 가능하게 하였다. SQuAD 1.1 벤치마크[Rajpurkar2016a]의 경우 F1 스코어 91.2가 사람이 낼 수 있는 성능이라 여겨졌는데, 2019년 4월 처음 딥러닝 기반 모델들이 이 성능치를 넘은 바 있음. 또한 GLUE 벤치마크[Wang2018a]의 경우 87.1을 사람 성능이라 여겨졌는데 이 또한 2019년 7월부터 딥러닝 기반 모델들이 이를 상회한 바 있음

○ 이러한 언어 모델들 중 주류를 구성하는 것은 트랜스포머의 인코더 모델을 이용한 BERT 언어 모델 계열들임. 표 1-1은 이 보고서가 작성된 2021년 8월초경 GLUE 벤치마크 리더보드[1]를 보이고 있는데, 1~6위가 모두 BERT 모델의

1) http://gluebenchmark.com/leaderboard

개량형들이며, 이외에도 순위 상위권에 대부분 RoBERTa 등 BERT 개량형들이 포진하고 있음

[표 1-1] 2021년 8월 현재 GLUE 벤치마크 리더보드

Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI
1	AliceMind & DIRL	StructBERT + CLEVER		91.0	75.3	97.7	93.9/91.9	93.5/93.1	75.6/90.8	91.7	91.5	97.4
2	ERNIE Team - Baidu	ERNIE		90.9	74.4	97.8	93.9/91.8	93.0/92.6	75.2/90.9	91.9	91.4	97.3
3	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8	71.5	97.5	94.0/92.0	92.9/92.6	76.2/90.8	91.9	91.6	99.2
4	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3	91.1	97.8
5	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	91.3	97.5
6	liangzhu ge	Deberta + adv (ensemble)		90.4	72.7	97.3	92.7/90.3	93.2/92.9	75.6/90.8	91.7	91.5	96.4
7	T5 Team - Google	T5		90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	91.9	96.9
8	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART		89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	90.8	99.2
9	Huawei Noah's Ark Lab	NEZHA-Large		89.8	71.7	97.3	93.3/91.0	92.4/91.9	75.2/90.7	91.5	91.3	96.2
10	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)		89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	91.1	95.8
11	ELECTRA Team	ELECTRA-Large + Standard Tricks		89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	90.8	95.8
12	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)		88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	90.7	95.6
13	Junjie Yang	HIRE-RoBERTa		88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7	90.4	95.5
14	Facebook AI	RoBERTa		88.1	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8	90.2	95.4
15	Microsoft D365 AI & MSR AI	MT-DNN-ensemble		87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/89.9	87.9	87.4	96.0
16	GLUE Human Baselines	GLUE Human Baselines		87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0	92.8	91.2

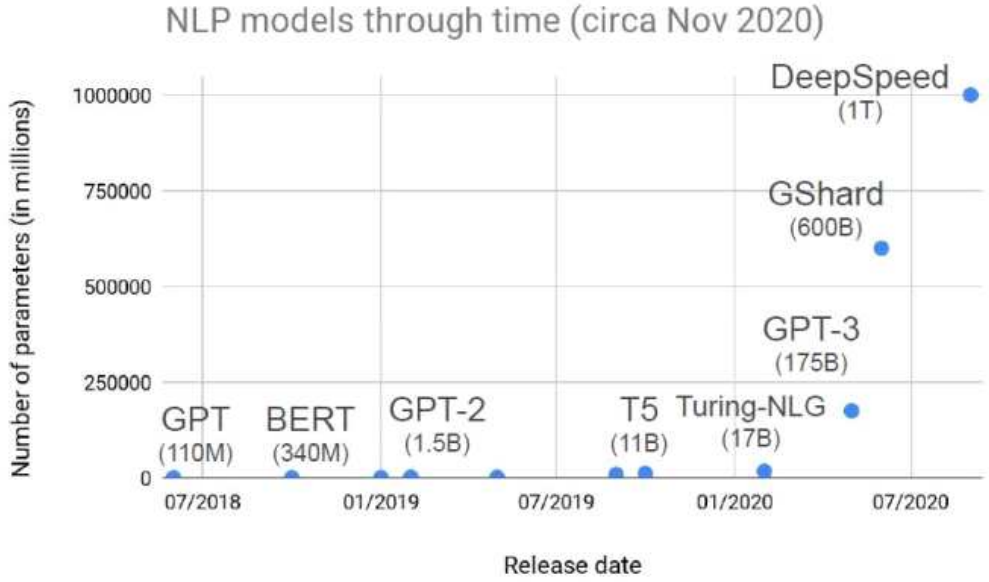
○ 이러한 언어 모델 발전의 이면에는 모델 크기의 급속한 증가라는 문제점 또한 포함하고 있는데, 그림 1-5는 최근 언어 모델의 크기를 비교하고 있음 [Ilharco2020a]

■ 그림에서 보이는 바와 같이 GPT 모델이 처음 공개되었을 때 약 110 백만 개의 파라미터로 구성되었던 반면에, T5는 약 110억 개 , DeepSpeed는 약 1조 개의 파라미터로 구성됨

■ 시간이 지남에 따라 보다 높은 성능을 위해 언어 모델의 크기는 지수적으로 급격하게 증가하는 경향을 보이고 있음

○ 이렇게 언어모델의 크기가 지수적(exponential)으로 크게 증가하는 이유는 좀 더 큰 모델이 보다 높은 성능을 보이고 있고, 이러한 성능 향상이 모델 크기 증가에 대해 아직까지 수렴하고 있지 않음에 기인함

■ Henighan 등의 연구[Henighan2020a]에 따르면, 이미지 처리 등에 사용되는



[그림 1-5] 언어 모델들의 크기 비교 [Illharco2020a]

모델들은 모델 크기가 크더라도 그 손실 값(loss)이 특정 부분에서 포화(saturate)되어 더 이상 개선이 되지 않는 반면에 언어 모델은 모델 크기가 커지는 경우 지속적으로 그 손실값이 줄어드는 경향을 보이고 있으며, 손실값이 줄어드는 기울기 또한 다른 분야의 모델 대비 완만한 특성을 보이고 있음 [그림 1-6] 참조

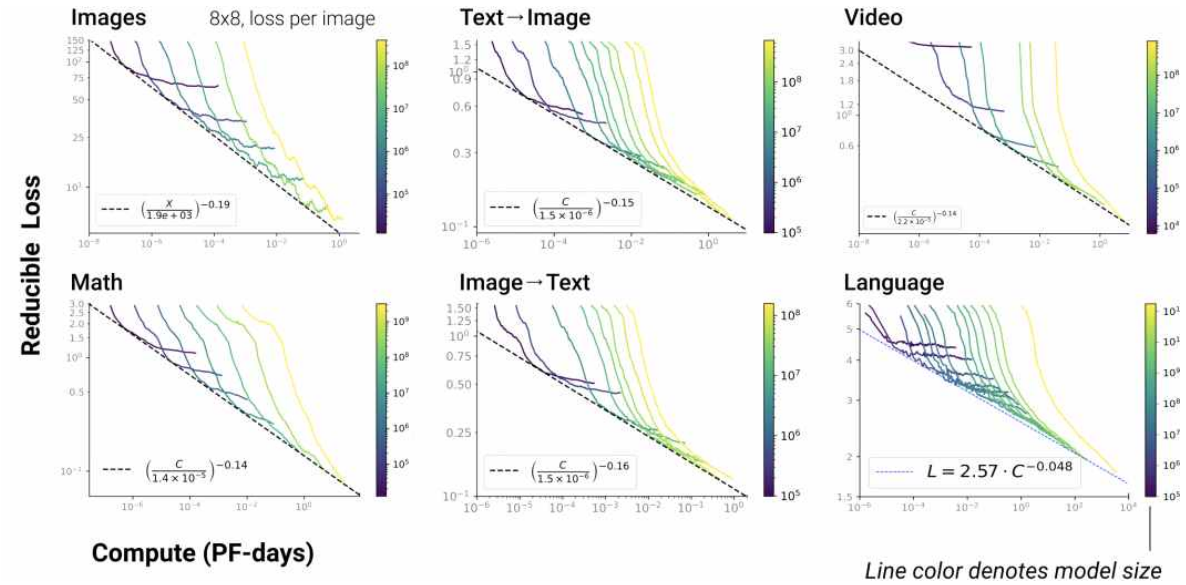
■ 또한 그림 1-8을 보면 파라미터 크기에 따른 언어 모델의 손실 값은 최근에야 손실 값이 떨어지는 기울기가 살짝 둔화되었을 뿐 포화되지는 않음을 보이고 있음.

■ 즉, 언어 모델링에 있어서 더 큰 모델의 크기가 자연어 처리 태스크의 정확도를 올리는데 긍정적인 영향을 미치고 있음을 확인할 수 있음

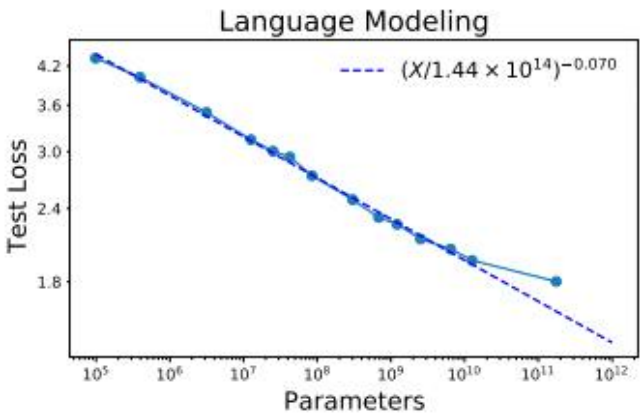
□ 언어 모델의 효과적인 학습 방법에 대한 연구

○ 현재의 언어 모델들은 대부분 트랜스포머[Vaswani2017a]의 인코더 또는 디코더를 이용하고, 대량의 코퍼스를 가지고 학습하는 방식으로 사전학습을 진행

■ 이러한 언어 모델들은 크게 트랜스포머의 디코더 방식을 이용하는 GPT 계



[그림 1-6] 분야별 모델 크기와 그 손실값 [Henighan2020a]



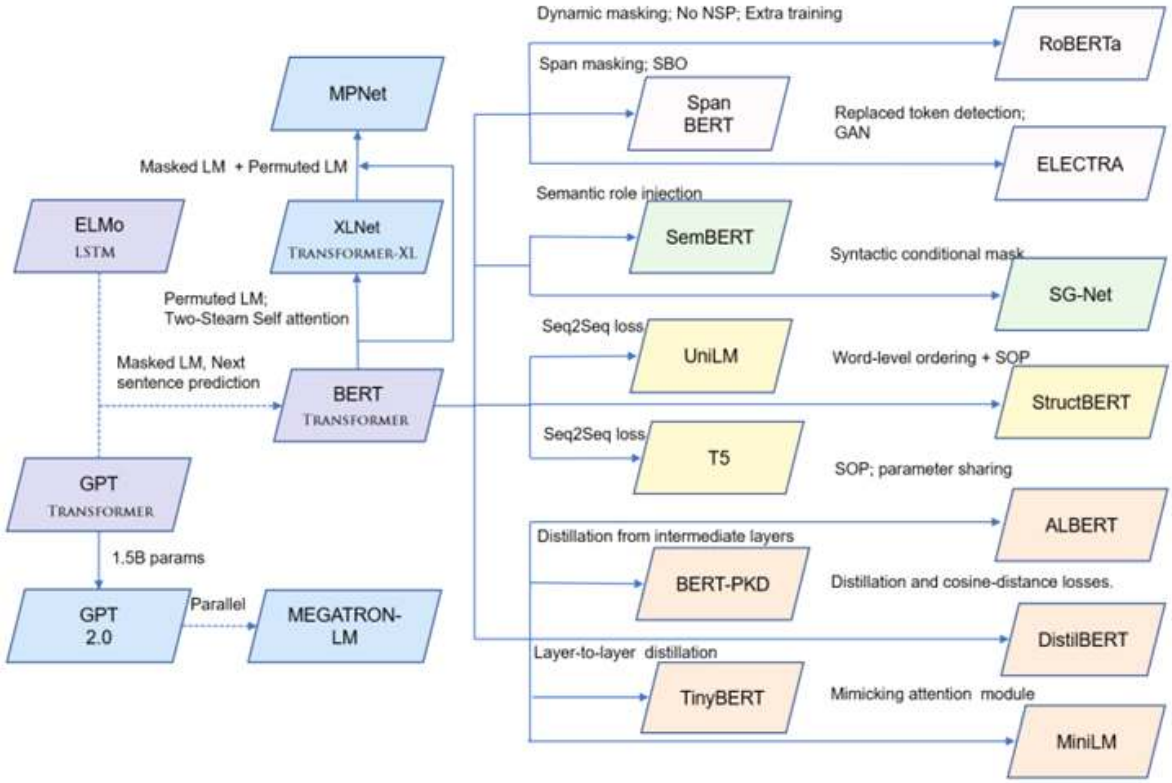
[그림 1-7] 언어 모델의 크기와 테스트 손실값[Henighan2020a]

열과 인코더 방식을 이용하는 BERT 계열로 구분할 수 있음

- 물론 T5[Raffel2020a]나 ELECTRA[Clark2020a]와 같이 인코더-디코더를 모두 이용하거나, 생성자(generator)-구별자(discriminator)를 두는 방식도 존재하나, 현재 트랜스포머 기반 언어 모델들은 이 두 방식 중 하나로 구분할 수 있음
- 일반적으로 BERT 계열과 같이 트랜스포머의 인코더 모델에 기반을 둔 언어 모델들은 언어 이해(Language Understanding)에, 디코더 모델에 기반을 둔

언어 모델들은 언어 생성(Language Generation)에 강점이 있는 것으로 알려짐

■ 여기에서는 BERT 모델을 기반으로 언어 모델의 효과적인 학습 기법에 대한 최근의 트렌드를 개략적으로 설명함



[그림 1-8] 언어 모델의 개괄적 분류 [Zhang2020a]

○ BERT 모델의 학습 기법에 대한 여러 개선책들이 제안되었으며, 주된 내용들은 NSP(Next Sentence Prediction) 태스크를 학습에 이용하는 것은 유의미한 개선을 보이지 못한다는 것임

■ 이에 따라 문장의 순서를 예측하거나, 문장의 선후 관계를 예측하도록 학습 태스크를 변경하는 방식이 제안되어 왔음(StructBERT, ALBERT 등)

○ 마스킹에 있어서도 고정된 방식으로 임의 코퍼스의 마스킹을 유지하는 것은 효과적이지 못하다는 사실이 알려짐

■ 때문에 마스킹하는 토큰의 수를 span 길이만큼 늘리거나, 또는 매 epoch마

다 동적으로 마스킹하는 토큰의 위치를 변경하는 방법들이 제안됨(SpanBERT, RoBERTa 등).

- ■ 또한 마스킹된 토큰들의 순서를 랜덤하게 섞고, 마스킹된 토큰을 예측 시 이들 토큰 간의 옳은 출현 순서까지 예측하도록 학습 태스크를 기여하는 변경하는 방법들도 제안됨(StructBert 등)

○ 이러한 기법들의 특징은 BERT 계열 모델들의 구조적 차이는 크지 않으면서, 학습 시의 태스크나 입력을 조정함으로써 언어 모델이 개선될 수 있다는 점임

- ■ 보다 큰 코퍼스와 배치 크기는 모델의 성능에 긍정적인 영향을 미침
- ■ 학습 시 태스크를 변경함으로써 모델 성능을 향상 시킬 수 있음
- ■ 물론 학습 시 태스크가 복잡해지고 배치 길이 및 코퍼스가 커짐에 따라 언어 모델의 학습에 소요되는 시간과 비용은 상대적으로 증가할 수 밖에 없음
- ■ 때문에 언어 모델의 효율성 향상에 관한 연구도 최근에 많이 진행되고 있음

□ 언어 모델의 효율성 향상에 관한 연구

○ 언어 모델의 거대화에 따른 효율성 문제

- ■ BERT 등과 같은 트랜스포머 기반 언어 모델들은 그 크기, 즉 모델을 구성하는 파라미터 개수가 크게 증가하고 있음
  - 그림 1-6에서 보이는 바와 같이 BERT 자체도 약 3.3억 개의 파라미터로 구성되어 있으나, 이후 출현하는 모델들의 크기는 지수적으로 증가하는 경향을 보임
- ■ 모델 크기의 증가는 여러 제약 및 문제점을 야기할 수 있음
  - 우선 저성능의 전산 자원들을 가지고는 이러한 모델의 학습을 수행할 수 없음; 모델 학습을 적기에 완료하기 위해서는 많은 고성능의 전산자원이 요구됨
  - 또한 파라미터 개수가 많다는 것은 해당 모델의 구동 또한 모바일 기기 등 메모리나 처리 성능에 제약이 많은 플랫폼에서는 어려움을 의미함
  - 더불어 많은 양의 파라미터 개수에 따라 학습된 모델을 가지고 실제 응용

을 개발 시 입력 데이터에 대한 추론(inference)에 있어서도 많은 연산을 필요함에 따라 처리 속도가 늦어지고 응답이 지연될 수 있음; 이것은 실시간 대화형 시스템을 개발하는데 있어 현실적인 문제가 됨

■ 이러한 점을 극복하고자 언어 모델의 효율성 향상을 위한 연구들이 진행되어 왔음

○ 언어 모델의 효율성 향상 관련 연구 트렌드

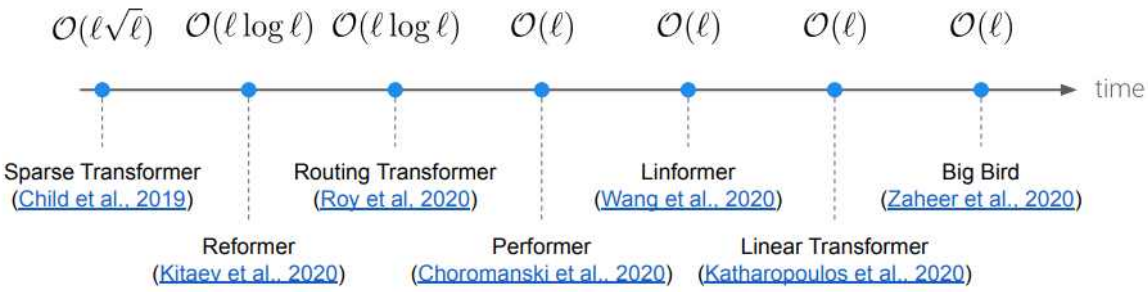
■ 언어 모델의 효율성 향상과 관련한 연구들로는 크게 1) 언어 모델의 기반이 되는 트랜스포머 구조의 효율성을 향상하기 위한 연구들과 2) 언어 모델 자체의 효율성 향상을 꾀하는 연구들로 구분할 수 있음

■ 언어 모델 자체의 효율성 향상을 위해서는 기존의 모델 경량화 기법들이 고려가 됨

- 모델 경량화 기술이란 모델의 정확도 손실(Accuracy loss)을 기존 모델 대비 최소화하면서 모델 크기와 연산량을 크게 줄임으로써 요구 메모리와 에너지, 요구 연산량 등 여러 면에서 학습, 추론 상의 효율성을 높이는 기술임 [Lee2020].
- 모델 경량화 기술들은 크게 1) 지식 증류(knowledge distillation), 2) 양자화(quantization), 3) 가지치기(pruning)로 분류 할 수 있음.
- 또한 경량 네트워크 설계 방식도 존재하나, 이 보고서에서는 언어 모델 분야에서는 이 방식은 별도로 트랜스포머 구조의 효율성 향상으로 따로 구분하겠음

■ 트랜스포머 구조의 효율성을 향상하기 위한 연구들은 멀티 헤드 어텐션 과정의 효율성을 향상시키기 위한 목적임

- 그림 1-9는 트랜스포머 모델의 어텐션 구조의 효율화가 어떻게 진행되어 왔는지를 간단히 도식으로 보이고 있음
- 입력 문장의 길이를 l이라 할 때 셀프 어텐션 기법이 적용됨에 따라 원래 트랜스포머의 어텐션의 복잡도는 $O(l^2)$으로 근사화할 수 있음
- 최근 연구들은 $O(l)$의 복잡도를 보일 수 있도록 개선됨



[그림 1-9] 트랜스포머의 어텐션 구조의 효율화 연구 진행 [Ilharco2020a]

[그림 1-10] 서로 다른 어텐션 구조 간의 성능 및 속도 비교 [Tay2021a]

- 그림 1-10은 각 트랜스포머의 효율성 개선을 위한 모델들과 트랜스포머모델을 처리 속도(X축)와 성능(Y축) 면에서 비교한 것임, 원의 크기는 각 모델의 메모리 상의 크기를 나타냄; LRA 점수는 Tay 등이 제안한 것으로 주로 긴 문장 입력에서의 컨텍스트나 의존성을 잡아내는 여러 태스크들을 종합하여 낸 점수임[Tay2021a]
- 그림 1-10을 보면 BigBird 모델이 가장 높은 LRA 점수를 기록하였으나,

Linformer나 Performer, Linear transformer 등 Kernel은 속도와 메모리 크기(memory footprint)에서 상대적으로 나은 트레이드오프를 보이는 것으로 나타남

○ 지식 증류 기법

■ 지식 증류 기법(knowledge distillation 또는 distillation이라 함)은 사전 학습된 여러 모델들을 가지고 모델 앙상블하여 이를 보다 작은 모델로 학습시키기 위해 고안된 기법임

- 사전 학습된 모델을 teacher 모델로 하고, 지식 증류 기법을 적용하여 얻은 작은 모델을 student 모델이라 명명함
- 때문에 teacher-student 기법이라고 불림
- 지식 증류 기법을 위해서는 사전 학습된 모델이 teacher 모델로서 준비가 되어 있어야 하며, teacher 모델은 BERT 등 언어 모델을 활용할 수 있음

■ 지식 증류 기법을 이용하여 사전 학습된 언어 모델을 가지고 이보다 작은 student 모델을 얻을 수 있음

- 대표적인 모델로는 BERT-PKD, DistllBERT, MobileBERT, TinyBERT 등이 존재함
- 이들 모델들은 BERT Large 모델을 teacher 모델로 하여 이보다 작은 student 모델을 학습하도록 고안됨

○ 양자화 기법

■ 양자화(Quantization) 기법은 FP32(32비트 부동소수점 타입)을 주로 사용하는 모델의 파라미터 값들을 이보다 낮은 비트너비(lower bit-width)를 갖는 1~K 비트 크기로 표현함으로써 모델의 메모리 크기(memory footprint)를 줄이는 방법임

- 원래 양자화란 기술 용어는 아날로그 신호를 디지털로 변환하는데 있어 아날로그 값을 비트열로 표현할 수 있는 디지털 값으로 변환하는 과정을 의미함
- 이 연구 분야에서의 양자화는 기존 큰 크기의 부동소수점 타입으로 표현

되던 모델의 파라미터 값을 이보다 적은 비트열로 표현하기 위해 변환하는 과정을 의미함

- 일반적으로 FP32 타입으로 표현된 파라미터 값을 그보다 작은 K 비트들로 표현하는 경우에는 32/K배만큼 모델의 메모리 크기를 줄일 수 있는 장점을 가짐
- 그러나 보다 적은 수의 비트로 표현할 수 있는 수의 범위는 크게 한정될 수밖에 없는데, 이에 따라 모델 성능이 저하되는 문제가 발생
- 때문에, 모델에 대한 양자화를 수행하는데 있어 목표는 보다 낮은 비트너비를 이용하여 모델의 메모리 크기를 줄이면서도, 기존 FP32 타입 대비 정확도 손실은 최소화하는데 있음
- 양자화 기법은 지식 증류나 가지치기 등 여러 경량화 기법과 함께 사용될 수 있는 장점을 가짐
- 양자화 과정에 있어서는 각 비트가 어떠한 값을 대표하는지를 결정하기 위해 코드북(codebook)을 이용하는데, 양자화 과정 전에 값이 이미 지정이 되어 있는 경우에는 고정 코드북(fixed codebook)이라 하며, 반대로 정확도 손실을 최소화하도록 양자화 과정 중에 양자화 값을 결정하는 코드북을 적응형 코드북(adaptive codebook)이라 함

■ BERT 언어 모델에 대한 대표적인 양자화 방식으로는 Q8BERT와 Q-BERT 등이 존재함

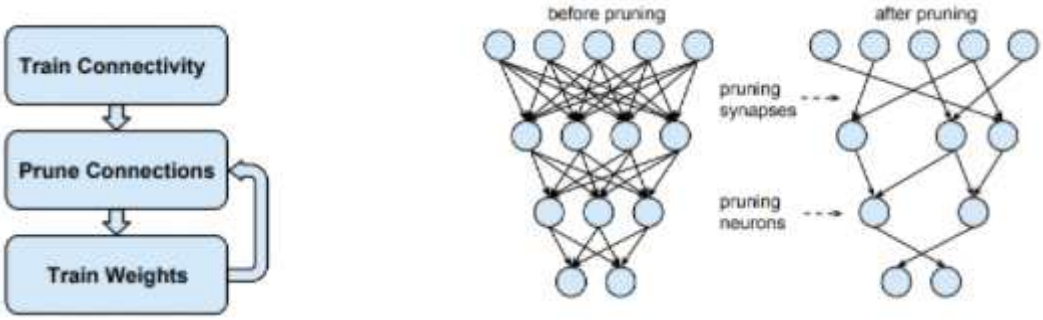
- Q8BERT의 경우 대칭적이며 선형적인 양자화를 수행하여 8 비트로 모델의 파라미터를 양자화 함
- Q-BERT의 경우 양자화 범위가 서로 동일하도록(uniform) 양자화를 수행함.
- 이외에 TernaryBERT와 같이 양자화와 지식 증류 기법을 같이 사용하는 방식 또한 존재함.

○ 가지치기 기법

■ 가지치기 기법은 덜 중요한 파라미터를 제거하여 모델의 크기를 줄이는 방

법임

- '모델의 모든 파라미터들이 추론에 동일한 영향을 미치는 것은 아니다'라는 관찰에서 출발하여, 먼저 모델을 학습한 후 추론에 있어 큰 영향을 미치지 못하는 파라미터들을 제거하고, 남아있는 파라미터들을 가지고 재학습하는 과정을 반복함으로써 덜 중요한 파라미터들이 상당 부분 제거된 모델을 얻는 방식임

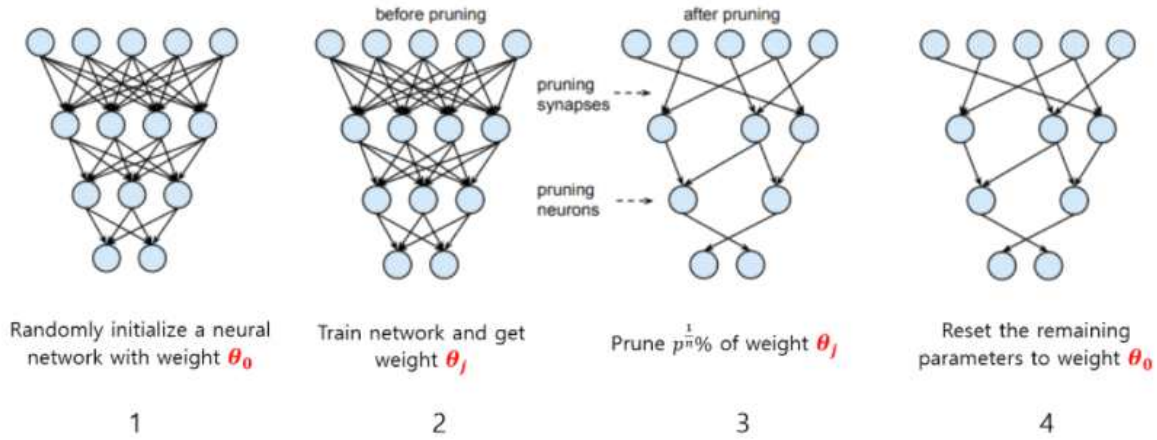


[그림 1-11] 가지치기 기법 개요 [Han2015]

- 하지만, 가지치기 뒤에 얻은 뉴럴 네트워크를 재학습 하는 과정에서 파라미터들을 랜덤 초기화시킨 후 처음부터 다시 학습하는 방식을 이용하면 가지치기 전과 비슷한 수준의 정확도를 얻기 어려운 문제가 발생
- 이러한 이유는 가지치기된 뉴럴 네트워크의 파라미터 개수가 적어지면서 그만큼 해당 네트워크의 허용량(capacity)가 작아지는 이유에 기인함

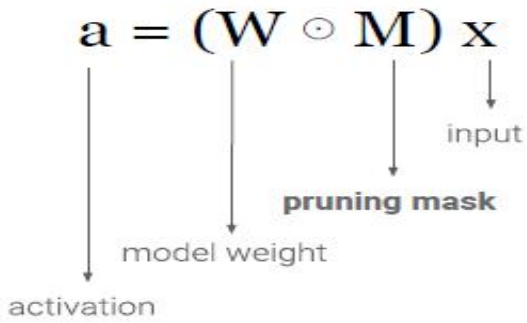
■ 복권 가설의 도입을 통한 가지치기 기법의 성능 개선

- 복권 가설(Lottery Ticket Hypothesis)[Frankle2019]은 우리가 복권에 당첨될 확률을 높이기 위해서는 복권을 더 많이 사야하는 것에 비유하여 가지치기 기법의 성능을 개선하는 시도임
- 하나의 큰 네트워크에 포함된 서브 네트워크들 중에는 당첨되지 않는 복권과 당첨되는 복권(Winning ticket)을 포함하는데 여기서 당첨되지 않는 복권을 선택하지 않아야 한다는 가설. 즉 불필요한 파라미터들을 제거함으로써 당첨이 될 복권인 서브 네트워크만으로 뉴럴 네트워크를 구성해야 한다는 제안임
- 그림 1-12는 복권 가설을 이용한 가지치기의 과정을 보임.



[그림 1-12] 복권 가설을 이용한 가지치기의 과정 [Frankle2019]

- 먼저 당첨될 복권을 찾는 과정을 진행함. 이 과정은 뉴럴 네트워크를 랜덤하게 초기화함(θ_0로 초기화됨)
- j번째 epoch까지 학습하게 되면, 같은 파라미터도 특정한 값(θ_j)에 도착할 것임. 이렇게 얻은 파라미터에 대해서 $p\%$만큼 가지치기를 수행하고 제거할 파라미터에 대한 마스크 M도 생성
- 가지치기를 하고 남은 파라미터에 대해서, 학습하기 전에 처음 랜덤하게 초기화 했던 것과 동일한 파라미터로 다시 리셋함. 생성한 마스크 M과 리셋하여 얻은 파라미터로 당첨되는 복권을 생성함.
- 파라미터를 리셋하는 이유는 다시 랜덤하게 서브네트워크를 초기화하면 더 이상 원래의 네트워크와 비슷한 성능을 낼 수 없기 때문임. 이 가설은 랜덤하게 초기화된 네트워크의 당첨되는 복권에 해당하는 서브 네트워크가 적어도 하나 이상은 존재할 것이라는 것으로 다시 초기화하면 이 가정을 쓸 수가 없음
- 요약하면, 중요한 뉴런이 무엇인지 저장된 마스크(mask)는 학습된 네트워크를 기준으로 가져오되, 가중치는 학습하기 전의 네트워크에서 가져오는 방식으로 파인튜닝을 수행



[그림 1-13] 가지치기를 위한 마스크 M의 도입[Frankle2019]

제2장

어텐션 기법과 트랜스포머 모델

제 1 절 어텐션 기법

1. Seq2Seq 모델

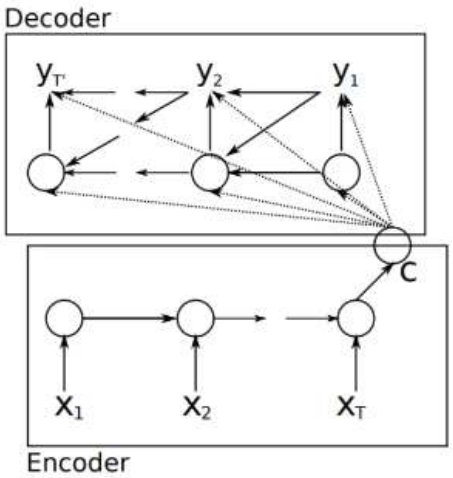
□ 개요

○ Seq2Seq는 입력된 시퀀스(sequence; 토큰의 순열 등)를 처리하여 다른 시퀀스로 출력하는 형태의 모델을 의미함

■ Q&A 등 대화형 시스템이나 기계 번역(Neural Machine Translation) 등이 대표적인 활용 분야임

- 예를 들어 Q&A 시스템의 경우 입력 시퀀스는 질의문이고, 출력 시퀀스는 질의문에 대한 대답이 해당이 됨

○ Seq2Seq는 그림 2-1에서 보이는 바와 같이 인코더와 디코더 두 개의 모듈로 구성이 됨



[그림 2-1] 인코더-디코더 모델 개요[Cho2014b]

■ 인코더는 입력 시퀀스를 순차적으로 입력 받은 후 이를 하나의 컨텍스트 벡

터(Context vector)로 만듦. 즉 입력 시퀀스의 모든 정보는 하나의 컨텍스트 벡터로 압축이 되며, 디코더는 이 컨텍스트 벡터를 받아 하나씩 순차적으로 처리 함.

■ 기존의 seq2seq 모델은 주로 LSTM이나 GRU 셀들로 구성되는 RNN 구조로 구성됨

- 인코더의 RNN 셀은 모든 단어를 입력 받은 뒤에 인코더 셀의 마지막 시점의 은닉 상태(hidden state)를 디코더의 RNN 셀로 넘겨주게 되며, 이것이 컨텍스트 벡터임.
- 컨텍스트 벡터는 디코더의 RNN 셀의 첫 번째 은닉 상태로 사용됨

□ 문제점

○ Seq2Seq 모델에서는 입력 시퀀스가 컨텍스트 벡터라는 하나의 고정된 크기의 벡터 표현으로 압축됨

○ 이때, 하나의 고정된 크기의 컨텍스트 벡터로 입력 시퀀스의 길이에 상관없이 모든 정보를 표현하고자 하니 정보 손실이 발생함

■ 특히 입력 시퀀스의 길이가 길어질수록 정보 손실 문제는 보다 심각해짐

○ 또한 입력 시퀀스가 길어지면, RNN 모델 자체에 기울기 소실 및 폭발 문제로 인해 학습이 곤란한 점이 존재함

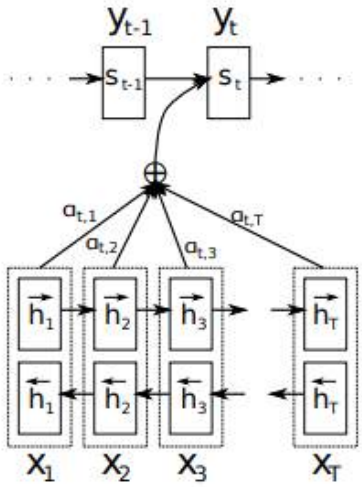
■ 입력 시퀀스가 길어지면, 그만큼 셀들 간에 전파되는 기울기가 소실되거나 폭발될 가능성이 높아짐에 따라 파라미터들이 특정 값으로 수렴(converge)되기 어려운 문제가 발생

○ 이러한 문제점을 해결하기 위해 어텐션(attention) 기법이 고안됨

2. Seq2Seq 모델에서 어텐션 기법의 도입

□ 어텐션 기법 개요

○ 어텐션은 디코더에서 매 시점(time step) 마다, 인코더의 여러 은닉 상태 정보



[그림 2-2] Bahdanau 등의 어텐션 구조[Bahdanau2015a]

들을 참조할 수 있게 함

■ 그림 2-2는 Bahadanau 등이 2015년에 제안한 어텐션 구조로 양방향 RNN으로 구성된 인코더의 은닉상태를 모두 가중치 합으로 더하고, 이를 디코더의 은닉 상태 계산에 참조하는 경우를 보임.

- 이때 각 입력 토큰 $X_1, X_2, \ldots, X_T$에 대해 출력 위치 I에서 참고하고자 하는 입력 토큰들의 은닉 상태들은 아래와 같이 가중치 합으로 계산되도록 함
- $c_i = \sum_{j=1}^{T} a_{ij} h_j$ 여기에서 a_{ij}는 j번째 토큰에 대한 은닉상태 h_j에 대한 출력 위치 i에 대한 가중치임

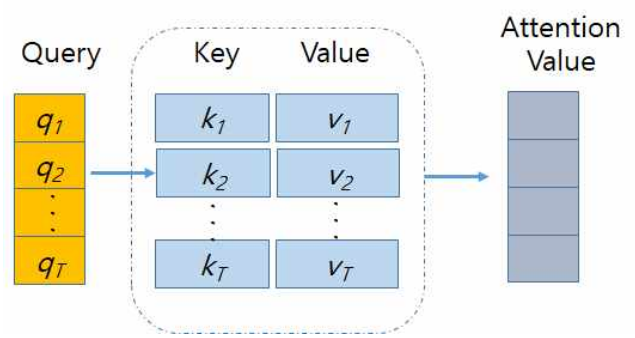
■ 이렇게 함으로써 디코더는 입력된 문장 중 유용한 부분에 집중할 수 있도록 함에 따라 긴 입력 시퀀스를 보다 효과적으로 처리할 수 있음

■ 이때, 인코더의 모든 은닉 상태를 사용하는 어텐션을 전역 어텐션(global attention)이라 하고, 일부 은닉 상태만을 사용하는 어텐션을 지역 어텐션이라 함

○ 어텐션은 기본적으로 키-값(key-value) 쌍으로 구성되는 자료형에 대한 검색과 유사한 개념임

■ 일반적으로 데이터베이스 등 자료 저장, 관리를 위해서는 키-값 쌍으로 데이터를 정렬하여 저장 관리하게 됨

- 예를 들어 학생 정보를 관리하는 시스템에서는 학번을 키로 하고, 이름이나 소속 학과 등을 값으로 하여 데이터를 저장하게 됨
- 이렇게 키-값 쌍으로 저장된 데이터에 대하여 우리는 질의어(Query)를 이용하여 검색을 수행하게 되는데, 이때 질의어가 원핫 인코딩(one-hot encoding)으로 코딩된 벡터라 하고, 이를 가지고 키 벡터와 닷 프로덕트(dot-product)를 수행한다면, 1로 설정된 벡터 아이템에 대응하는 키 값만이 선택되고, 해당하는 키 값에 대응하는 value 값을 찾아 출력할 수 있음 ([그림 2-3] 참조)



[그림 2-3] 어텐션의 질의, 키, 값

- 이런 방식으로 정확하게 데이터베이스 시스템에서의 일치 검색(exact match)의 결과와 동일한 결과를 얻을 수 있음
- 만약 질의어 벡터를 원핫 인코딩이 아닌 유사도 벡터라 한다면, 질의어 벡터와 키 벡터를 닷 프로덕트를 수행한 것은 일치 검색이 아닌, 질의와 키가 얼마나 유사한지에 대한 유사도 검색(similarity search)과 같음
- 왜냐하면 두 벡터 A, B 간 코사인 유사도(Cosine similarity)는 $\frac{A \cdot B}{\|A\| \times \|B\|}$ 인데, 이는 두 벡터를 닷-프로덕트(dot product)한 결과에 대해 두 벡터의 길이의 곱으로 scale을 조정한 것이기 때문임
- 따라서 그림 2-3에서 보이는 어텐션이라 하는 것은 결과적으로 질의 벡터와 키 벡터간 유사도에 따른 가중치에 기존 값(value)가 곱해지는 것과 같

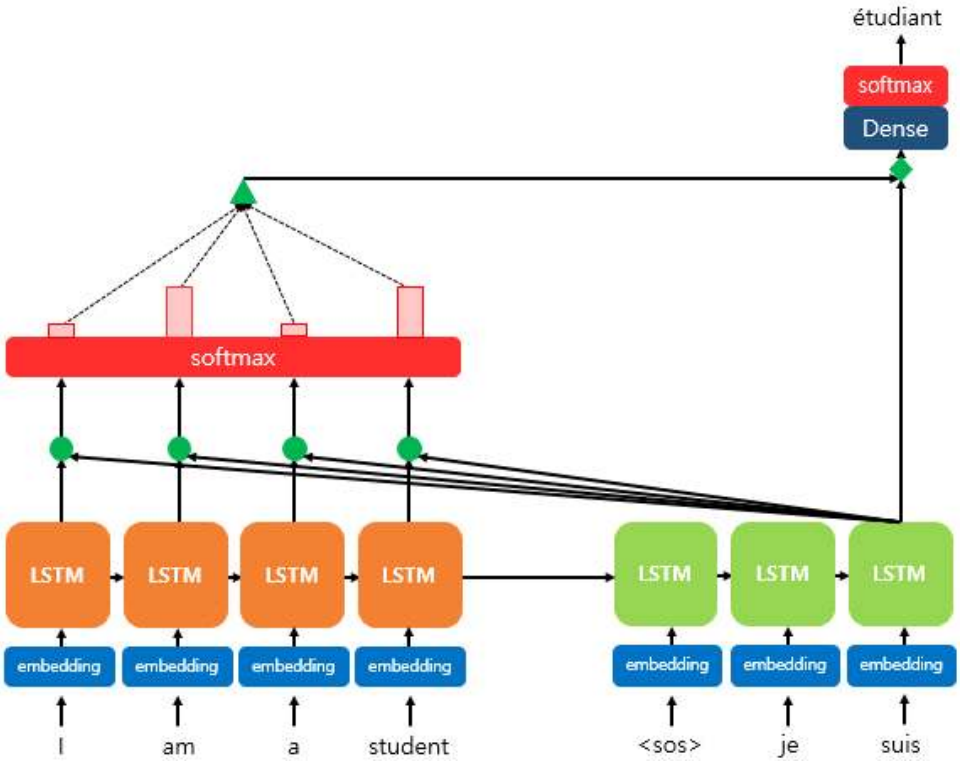
은 것이라 할 수 있임

- 여기에서는 닷 프로덕트를 기준으로 어텐션의 개념을 설명하였는데, 이외에 어텐션을 구하는 여러 방법들 또한 존재하며, 아래에서는 닷 프로덕트 어텐션이 Seq2Seq 모델에 어떻게 적용되는지를 보도록 함

□ 닷-프로덕트 어텐션

○ 어텐션 점수의 계산

■ 그림 2-4는 LSTM RNN으로 구성된 Seq2Seq 모델에서 닷 프로덕트 어텐션을 수행하는 예를 보이고 있음



[그림 2-4] Seq2Seq 모델에의 닷 프로덕트 어텐션 적용 예시 [유원준2021]

- 그림 2-4에서는 디코더의 세 번째 LSTM 셀에서 출력 단어를 예측할 때 어텐션 메커니즘을 보여줌.
- 인코더의 각 시점에서의 은닉 상태(hidden state)를 각 $h_1, h_2, \ldots h_N$라고 하고, 디코더의 현재 시점(time step) t에서의 디코더의 은닉 상태를 s_t 라

고 정의

- 시점 *t*에서 출력 단어를 예측하기 위해서 디코더의 셀은 이전 시점인 *t-1*의 은닉 상태와 이전 시점 *t-1*에 나온 출력 단어를 입력으로 함
- 어텐션 기법에서는 출력 단어 예측에 이 두 값 이외에 어텐션 값(attention value) a_t를 이용
- 닷-프로덕트 어텐션에서는 a_t을 구하기 위해 s_t를 전치(transpose)하고, 각 은닉상태 h_i와 닷 프로덕트를 수행하여 스칼라 값을 구함

■ softmax를 이용한 어텐션 분포 계산

- s_t와 인코더의 모든 은닉 상태의 어텐션 스코어의 모음값을 e^t라 하면, $e^t = [s_t^T h_1, \ldots, s_t^T h_N]$가 되고, e^t에 softmax 함수를 적용하여, 어텐션 가중치들의 모음인 어텐션 분포(attention distribution)를 구함; $\alpha^t = softmax(e^t)$.

■ 어텐션 값을 구한 뒤 적용

- 이후 각 인코더의 은닉 상태와 어텐션 가중치들을 가중치 합(weighted sum) $a^t = \sum_{i=1}^{N} \alpha_i^t h_i$을 수행하여 어텐션 값을 구함
- 이 어텐션 값과 디코더의 은닉상태 s_t를 결합(concatenate)하여 하나의 벡터를 만들고, 이를 Dense 계층의 입력으로 하여 새로운 은닉 상태 $\tilde{s_t} = \tanh(W_c[a_t; s_t] + b_c)$를 구함
- 그리고 $\tilde{s_t}$를 출력층의 입력으로 사용하여 출력 벡터 $\hat{y_t} = softmax(W_y \tilde{s_t} + b_y)$를 얻음

□ 다양한 종류의 어텐션

○ 여러 어텐션들의 차이는 어텐션 스코어 함수의 차이

■ 위의 어텐션이 닷-프로덕트 어텐션인 이유는 어텐션 스코어를 구하는 방법이 내적(dot product)이었기 때문

■ 어텐션 스코어를 구하는 방법은 여러 가지가 제시되어있으며, 현재 제시된 여러 종류의 어텐션 스코어 함수들은 아래와 같음

[표 2-1] 어텐션 스코어 함수의 종류

이름	스코어 함수	출처
dot-product	$score(s_t,\ h_i) = s_t^T h_i$	Luong2015
scaled dot-product	$score(s_t,\ h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	Vaswani2017
general	$score(s_t,\ h_i) = s_t^T W_a h_i$ // 단, W_a는 학습 가능한 가중치 행렬	Luong2015
concat	$score(s_t,\ h_i) = W_a^T\ tanh(W_b[s_t; h_i])score(s_t,\ h_i) = W_a^T\ tanh(W_b s_t + W_c h_i)$	Bahdanau2015
location-base	$\alpha_t = softmax(W_a s_t)$ // α_t 산출 시에 s_t만 사용하는 방법.	Luong2015

■ 닷 프로덕트 어텐션은 제안한 사람의 이름을 따서 루옹(Luong) 어텐션이라고도 함. concat이라는 이름의 어텐션은 바다나우(Bahdanau) 어텐션 이라고도 함

■ 스케일드 닷 프로덱트(scaled dot product) 어텐션은 트랜스포머에서 사용하는 어텐션 기법임

○ 초기에 어텐션은 seq2seq 모델 성능의 개선을 위해 고안되었으나, 이제는 어텐션 기법만으로 seq2seq 모델을 대체해 버림

■ 트랜스포머 모델이 어텐션 기법만으로 seq2seq 모델을 대체하는 기술이며, 현대의 언어 모델 또한 이 트랜스포머 모델을 기반으로 하여 고안됨

제 2 절 트랜스포머 모델

1. 모델 구조

□ 개요

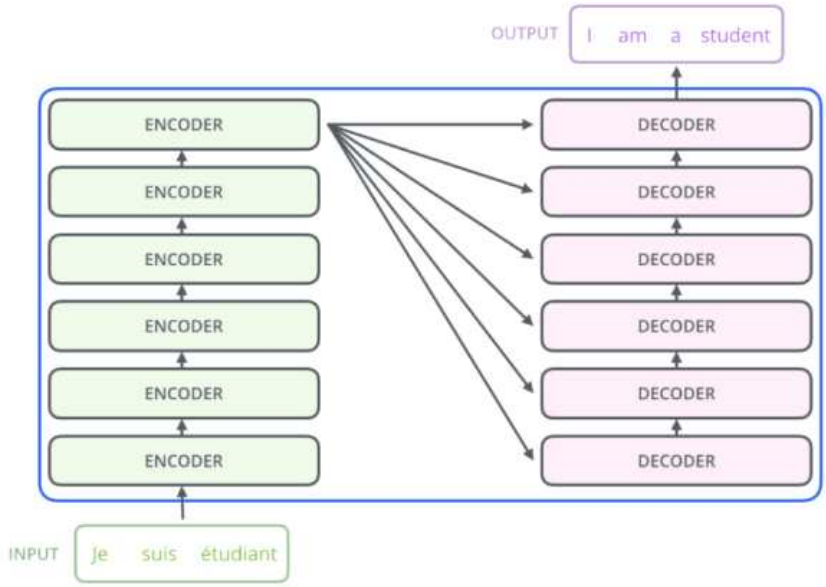
○ 어텐션 기법을 이용한 인코더-디코더 구조

■ 트랜스포머는 기존 Seq2Seq 모델처럼 인코더에서는 입력 시퀀스를 입력 받고, 디코더에서 출력 시퀀스를 출력하는 인코더-디코더 구조를 가짐

■ 트랜스포머에서는 인코더와 디코더가 각각 여러 개의 계층을 가질 수 있음

- 트랜스포머를 첫 제안한 논문에서는 인코더와 디코더의 수를 각각 6 개씩 사용

■ 기존 seq2seq 모델과의 차이점은 인코더의 경우 최상단 계층만 디코더에 은닉 상태를 전달하게 되는데, 이때 하나의 디코더 계층에 전달하는 것이 아니라 모든 계층의 디코더에 은닉 상태를 전달한다는 점이 다름



[그림 2-5] 트랜스포머에서의 인코더-디코더 구조 [Alammar2018a]

○ 위치정보 인코딩(Positional encoding)

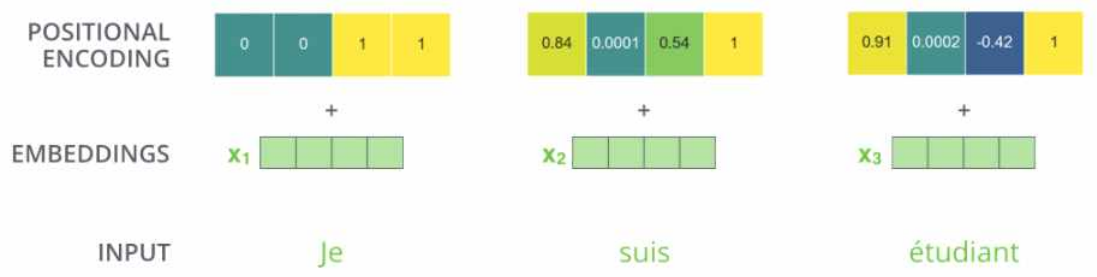
■ 트랜스포머는 입력 시퀀스를 한 번에 받기 때문에 입력 시퀀스의 토큰 또는 단어의 위치 정보를 추가로 입력해야 할 필요가 존재함

■ 입력 토큰의 임베딩 벡터에 위치 정보들을 더하여 인코더의 입력으로 사용

- 그림 2-6은 입력 임베딩 벡터에 포지셔널 인코딩 값이 더해지는 값을 시각화한 것임

■ Vaswani 등이 제안한 트랜스포머 모델[Vaswani2017a]의 경우 포지셔널 인코딩 값을 만들기 위에 아래의 두 수식을 사용함

- $PE_{(pos,\,2i)} = \sin(pos/10{,}000^{2i/d_{model}})$ (식 2-1)
  $PE_{(pos,\,2i+1)} = \cos(pos/10{,}000^{2i/d_{model}})$
- 식 2-1에서 pos는 입력 시퀀스에서의 해당 토큰의 위치를 나타내며, i는 해당 토큰에 대한 임베딩 벡터 내의 차원 인덱스를 의미함
- 또한 d_{model}은 트랜스포머의 모든 층의 출력 차원을 의미하는 초매개변수(hyperparameter)임. Vaswani의 트랜스포머 논문에서는 512로 설정됨
- 위와 같은 포지셔널 인코딩 방법을 사용하면 동일한 단어라 하더라도 입력 시퀀스 내 단어 위치에 따라 임베딩 벡터의 값이 달라지게 됨



[그림 2-6] 포지셔널 인코딩의 사용 예시 [Alammar2018a]

□ 트랜스포머 모델의 구조

○ 트랜스포머는 그림 2-7과 같은 구조를 가지고 있음

■ 먼저 트랜스포머는 인코더와 디코더로 구성되어 있으며, 인코더와 디코더는 각각 N 개만큼 중첩하여 여러 개를 가질 수 있음(그림에서는 Nx로 표기)

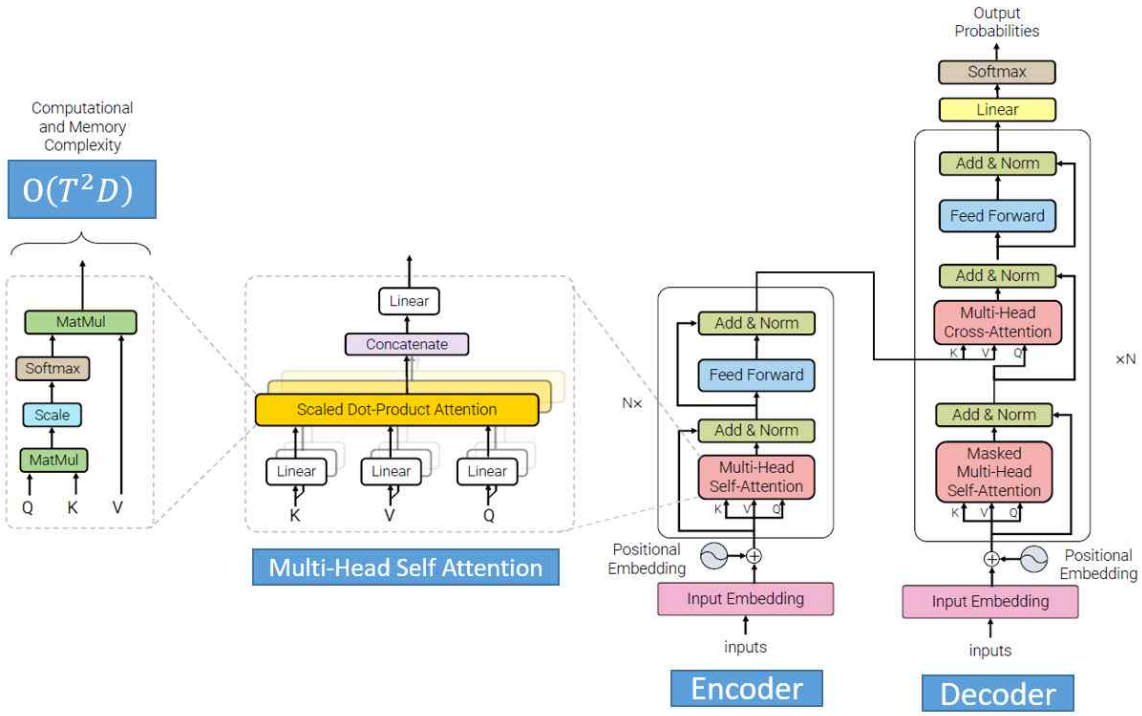
■ 인코더와 디코더에서 사용하는 어텐션 종류는 총 3 종류가 있음

- 이러한 어텐션의 종류는 질의(Q), 키(K), 값(V)가 무엇이냐에 따라 달라짐
- 트랜스포머의 인코더에서는 셀프 어텐션(self attention)을 수행함. 셀프 어텐션은 Q, K, V가 모두 같은 출처의 벡터를 활용한다는 특징이 있음
- 디코더의 경우 마스크된 셀프 어텐션(masked self attention)과 크로스 어텐션(cross attention)을 수행함.
- 크로스 어텐션에서는 질의 Q가 디코더의 벡터인 반면, 키 K와 값 V는 인코더의 벡터로 다름에 따라 크로스 어텐션이라 명명함

○ 트랜스포머의 인코더

■ 인코더에서의 셀프 어텐션

- 어텐션 함수는 주어진 질의 Q에 대해 모든 키 K와의 유사도를 구하고, 이 유사도를 가중치로 하여 키 K에 대응하는 V들을 가중합함
- 셀프 어텐션은 Q, K, V가 모두 입력 시퀀스의 모든 토큰들에 대한 임베딩 벡터들임

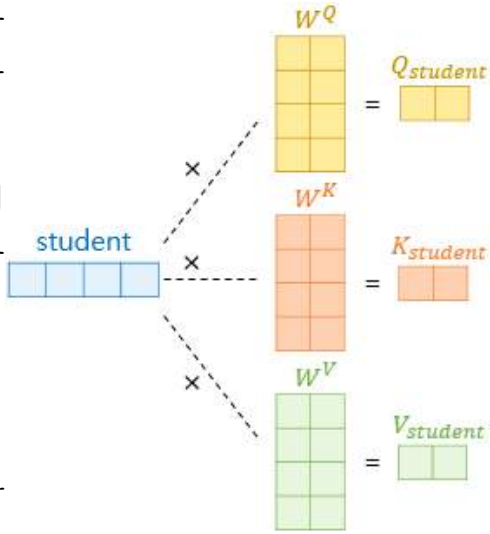


[그림 2-7] 트랜스포머 모델의 기본 구조 [Lin2021]

- 셀프 어텐션을 수행하기 위해서는 Q, K, V를 입력 시퀀스 내 토큰에 대응하는 임베딩 벡터로부터 얻어야 하는데, 트랜스포머의 출력 벡터 차원을 결정하는 d_{model} 을 멀티 헤드 어텐션에서의 헤드 개수를 지정하는 초매개변수 num_heads로 나눈 값으로 설정함. Vaswani의 논문에서는 num_heads를 8로 설정함에 따라 512/8 = 64 차원의 Q, K, V 벡터를 구함
- 64 차원의 Q, K, V 벡터는 기존 입력 임베딩 벡터에 대해 가중치 행렬을

곱함으로써 완성됨. 각 가중치 행렬은 $d_{model} \times (d_{model}/\text{num_heads})$의 크기를 가짐

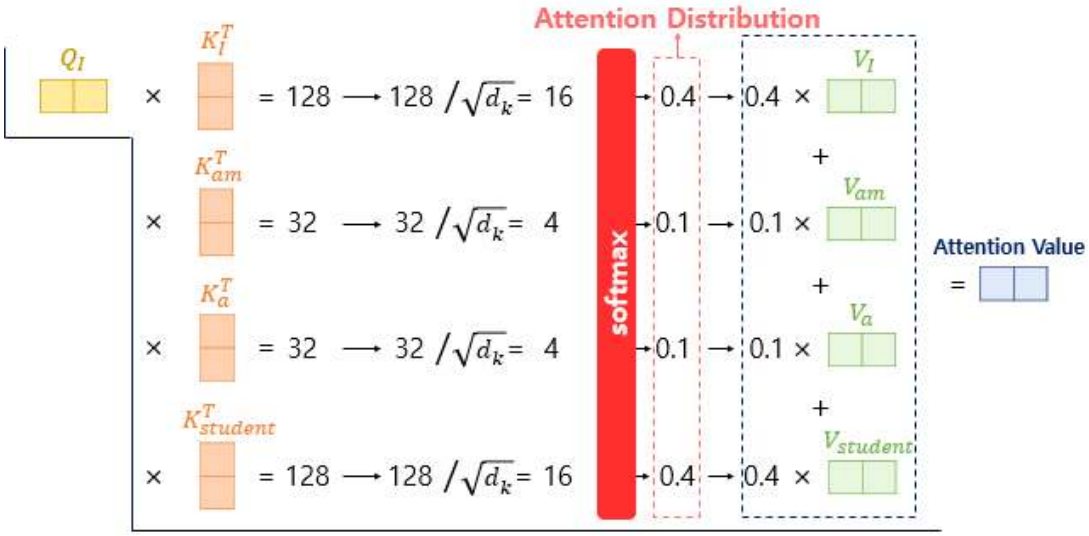
- 그림 2-8은 student 토큰에 대한 임베딩 벡터로부터 Q, K, V 벡터를 얻는 과정을 도식화 함



[그림 2-8] Q, K, V 벡터의 획득[유원준2021]

■ 스케일드 닷-프로덕트 어텐션

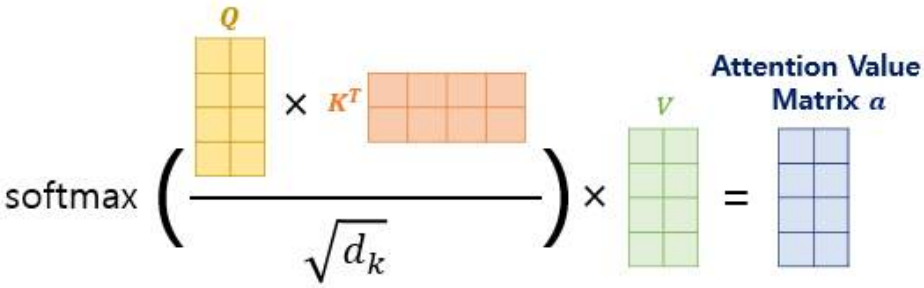
- 트랜스포머에서는 스케일드 닷-프로덕트 어텐션 기법을 이용함 (표 2-1 참조)
- 스케일드 닷-프로덕트는 기존 닷-프로덕트와 동일하나 어텐션 스코어를 $\sqrt{d_k}$으로 나누는 차이를 가짐. 여기에서 d_k는 K 벡터의 차원으로 현재 64이므로 $\sqrt{d_k}$는 8의 값을 가짐
- 이제 어텐션 스코어에 softmax를 취해 어텐션 분포를 구하고, 각 V 벡터와 가중합하여 어텐션 값을 구함. 이를 단어 I에 대한 어텐션 값이라 함.



[그림 2-9] 스케일드 닷-프로덕트 어텐션 예시

- 이 연산은 실제로는 벡터 연산이 아닌 아래와 같이 Q와 K, V 벡터 간의

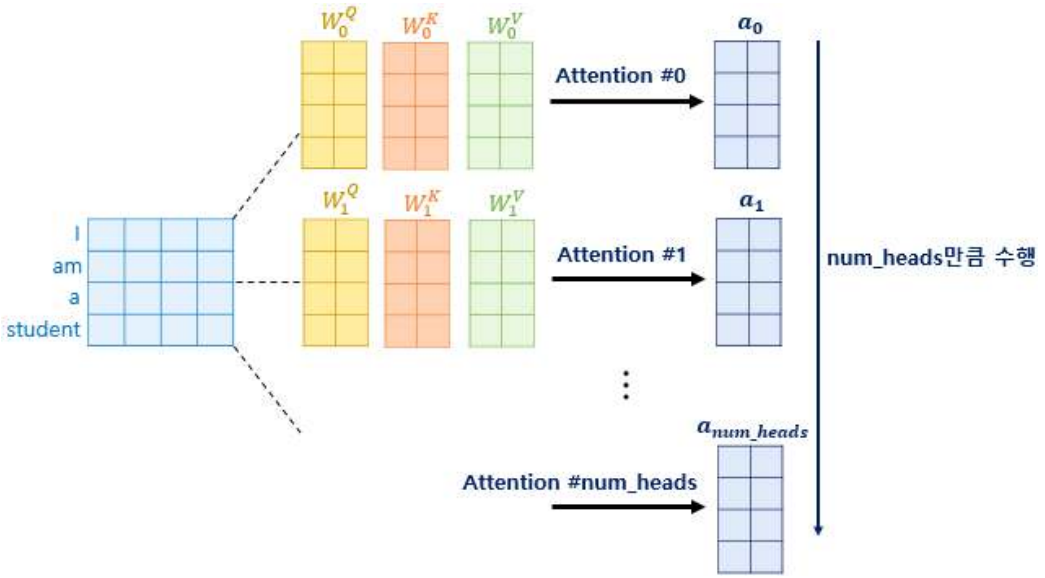
행렬 곱 연산을 통해 일괄 계산이 가능함



[그림 2-10] 행렬곱 연산을 통한 어텐션 계산

■ 멀티 헤드 어텐션

- 트랜스포머 모델은 한 번의 어텐션보다 여러 개의 어텐션을 수행하는 것이 더 효과적이라는 관찰에 따라 d_{model}/num_heads 차원을 가지는 Q, K, V 에 대해 num_heads 만큼의 어텐션을 수행함.
- Vaswani의 논문에서는 num_heads 값이 8로 설정됨에 따라, 총 8개의 어텐션 연산이 병렬로 수행되며, 이들의 결과를 합친 것은 64* 8 = 512개로 d_{model} 초매개변수 값과 같게 됨



[그림 2-11] 멀티헤드 어텐션 예시 [유원준2021]

- 그림 2-11은 4개의 토큰으로 구성된 입력 시퀀스에 대해 각각 어텐션을 수행하는 예시를 보임
- 어텐션을 수행한 결과들은 모두 결합(concatenate)시켜 d_{model} 차원을 갖는 어텐션 행렬을 만듦. 어텐션 행렬의 크기는 입력 시퀀스 길이 seq_len * d_{model} 이 됨
- 멀티 헤드 어텐션에서의 어텐션 함수는 선택적으로 패딩 마스크를 입력으로 가질 수 있는데, 이는 입력 시퀀스에 <PAD>라는 특수 토큰이 있을 경우 어텐션 연산에서 이를 제외하기 위해 이용함

■ 잔차 연결과 계층 정규화

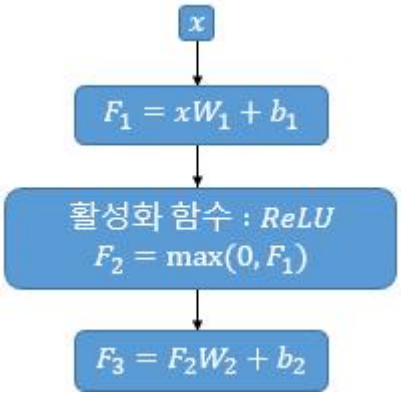
- 그림 2-7의 구조에서 Add & Norm이라 명명된 블록은 잔차 연결(residual connection)과 계층 정규화(layer normalization)을 수행함을 의미함
- 잔차 연결은 ResNet 등에서 제안된 방법[He2016a]으로 현 계층의 입력을 현 계층의 출력에 더하는 방식으로 깊은 계층의 뉴럴 네트워크에서 학습 효과를 증가시키기 위한 방법임.
- 잔차 연결을 거치고 나면 계층 정규화를 수행하는데, 계층 정규화는 d_model 차원의 텐서에 대하여 각 자질 차원에서 평균과 분산을 구한 뒤, 이를 가지고 정규화를 수행함을 의미함.
- 벡터 x_i의 각 차원을 k라고 할 때 $x_{i,k}$, 즉 x_i의 각 k 차원의 값은 x_i의 평균 u_i와 분산 σ_i^2을 가지고 $\frac{x_{i,k}-u_i}{\sqrt{\sigma_i^2+\epsilon}}$으로 정규화를 수행함

■ 포지션-와이즈 FFNN

- 포지션-와이즈 FFNN(Position-wise Feed Forward Neural Netwrok)는 트랜스포머의 인코더와 디코더 양쪽에 공통적으로 존재하는 계층으로 완전 연결 FFNN 으로 그림 2-12와 같은 연산을 수행함
- 그림에서 x는 멀티헤드 어텐션 결과로 나온(seq_len, d_{model})의 크기를 가지는 행렬이고, 가중치 행렬 W_1은 (d_{model}, d_{ff})의 크기를, W_2는 (d_{ff}, d_{model})의 크기를 가짐. d_{ff}는 FFNN의 은닉층의 크기로 논문에서는 2,048의 크기

로 정의됨

- 이들 가중치 행렬들은 하나의 인코더 층 내에서는 입력에 상관없이 공통적으로 사용되나, 다른 인코더 계층들 간에는 공유되지 않음. 단, 최근의 트랜스포머 개선 연구들에서는 다른 계층 간에도 공유하는 시도들이 있기는 함



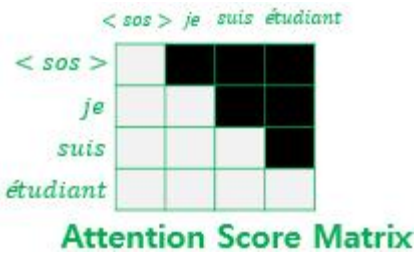
[그림 2-12] 포지션-와이즈 FFNN [유원준2021]

○ 트랜스포머의 디코더

■ 디코더도 인코더와 동일하게 입력 임베딩과 포지셔널 인코딩 벡터를 더하여 시퀀스를 생성함

■ 룩-어헤드 마스크

- RNN으로 작성된 디코더에서는 입력 단어를 매 time step마다 순차적으로 받으므로 다음 단어 예측에 현재 시점 이전에 입력된 단어들만 참고할 수 있음
- 반면에 트랜스포머 모델은 입력 시퀀스를 한 번에 입력받는 형태이므로 입력 시퀀스 내에 예측할 단어가 있다면, 이를 참고할 수 있게 되는 일종의 컨닝이 가능함
- 이러한 문제를 해결하기 위해 트렌스포머의 디코더에서는 특정 시점의 예측에서 그 시점보다 미래에 있는 단어들을 참고하지 못하도록 룩-어헤드 마스크(look-ahead mask)를 이용할 수 있음
- 룩-어헤드 마스크는 디코더의 첫 번째 어텐션 층에서 이뤄지는데, 기존 멀티 헤드 어텐션 층과 동일하지만 어텐션 스코어 행렬에서 각 단어의 다음 단어들에 대해 보지 못하도록 마스킹을 수행한다는 점이 다름. 이렇게 함으로써 어텐션 스코어 행렬에서는 자기 자신과 그 이전 단어들만을 참고할 수 있도록 함



[그림 2-13 룩 어헤드 마스킹의 예시 [유원준2021]

■ 크로스 어텐션

- 크로스 어텐션 디코더의 두 번째 어텐션 모듈로 질의 Q가 디코더의 벡터인 반면, 키 K와 값 V는 인코더의 벡터로 다름
- 그림 2-7에서 보이는 바와 같이 키 K와 값 V는 인코더의 마지막 계층으로부터 받으며 Q는 디코더의 하위 계층으로부터 받게 됨.
- 즉, 인코더의 어텐션 적용한 K, V를 입력으로 받아 디코더의 Q를 가지고 어텐션을 수행하도록 고안됨

2. 트랜스포머 모델의 계산량

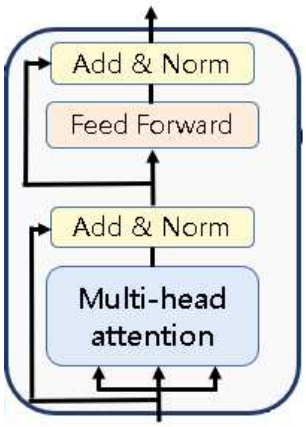
□ 개 요

○ 트랜스포머의 구조

■ 앞서 살펴본 바와 같이 트랜스포머 구조의 핵심 구성요소는 (1)멀티 헤드 어텐션 구조, (2) 포지션-와이즈 FFNN, (3) 계층 정규화 그리고 (4) 잔차 연결 모듈로 구성되어 있다고 요약할 수 있음

■ 따라서 이들 각각의 계산량을 먼저 파악해 둔다면, 트랜스포머 모델을 기반으로 하는 언어 모델의 계산량 및 효율성을 고려하는데 도움을 줄 수 있음

■ 여기에서는 그림 2-14에서 보이는 트랜스포머 모델의 핵심 구성 요소에 대하여 계산량을 점검해 보도록 함



[그림 2-14] 트랜스포머 모델의 핵심 구성요소

□ 어텐션 모듈의 연산

■ 트랜스포머의 멀티 헤드 어텐션에서 각 헤드에 대한 연산은 질의 Q, 키 K, 값 V에 대해 스케일드 닷 프로덕트 연산으로 식 2-3와 같이 계산이 됨

■ 트랜스포머 모델에서는 어텐션 연산을 닷 프로덕트 연산 결과에 $\sqrt{d_k}$로 나누어 스케일을 조정함

■ 질의와 키를 $(\mu,\sigma)=(0,1)$인 랜덤 변수로 가정하는 닷프로덕트로 연산된

$\sum_{i=1}^{d_k} q_i k_i^T$ 값이 $(\mu,\sigma)=(0,d_k)$이므로 $\frac{1}{\sqrt{d_k}}$으로 스케일을 조정함. 이때 d_k는 키 벡터의 차원임

■ i 번째 헤드에서 j번째 토큰으로의 어텐션은 아래와 같이 계산이 됨

$$head_{ij} = Attention(Q_i, K_j, V_j) = Attention(XW_i^Q, XW_j^K, XW_j^V) \quad \text{(식 2-2)}$$

$$head_{ij} = Attention(Q_i, K_j, V_j) = softmax(\frac{Q_i K_j^T}{\sqrt{d_k}}) V_j \quad \text{(식 2-3)}$$

$W_i^Q, W_j^K \in R^{d_m \times d_k}$, $W_j^V \in R^{d_m \times d_v}$

■ 멀티헤드 어텐션에 대한 연산은 아래와 같음

$$MheadAtt(Q,K,V) = Concat(head_1, head_2, ..., head_n) \times W^O \quad \text{(식 2-4)}$$

출력 projection W^O matrix는 $W^O \in R^{hd_v \times d_m}$에 속함.

■ $head_i$의 연산량은 병렬 연산으로 $O(B \times N \times N_h \times \frac{d}{N_h})$만큼의 계산량이 요구됨 (B : batch크기). 길이 N인 입력 시퀀스에 대해 QK^T 의 연산은 N^2 연산량의 메모리와 시간을 필요로 하는 제약점(quadratic 연산량)이 따르게 됨

□ FFN 연산과 Connected Residual 연산

■ 트랜스포머 블록은 수식 2-4의 멀티 헤드 어텐션 이후에 다음과 같이 두 계층 FFNN으로 계층 정규화와 함께 연산하여 position-wise 연산을 수행함.

$$\hat{h_l} = LN(h_{l-1} + MheadAtt(h_{l-1})) \quad \text{(식 2-5)}$$

$$h_l = LN(\hat{h_l} + FFN(\hat{h_l})) \quad \text{(식 2-6)}$$

- MheadAtt : 멀티헤드 어텐션 계층으로 심층 양방향 및 컨텍스트 임베딩을 얻기 위함
- $FFN(\hat{h_l})$: position-wise feed forward neural network, 포지션 임베딩을 이용하는 어텐션에서 놓칠 수 있는 포지션-와이즈 학습을 진행.

- $LN(x+f(x))$: 잔차 연결과 계층 정규화; 모델이 깊어지면서 발생하는 기울기 소실(Vanishing Gradient) 또는 폭발(Exploding Gradient) 문제 완화를 위한 방법으로 Kaming He의 ResNet 모델[He2016a]을 모방하여 적용

□ 트랜스포머 계층의 구성

○ 모델의 크기

■ 트랜스포머 기본 구조를 이용한 BERT 기본 크기 언어 모델은 트랜스포머의 인코더 모듈 12개로 구성되고, BERT Large 모델은 24개의 계층으로 구성됨

■ 트랜스포머의 디코더 구조를 이용하는 GPT 버전 2 small 언어 모델은 12개의 디코더 모듈로 구성되며, GPT 버전 2 Extra large 모델은 48 개의 디코더 모듈로 구성됨.

■ 가장 최근의 GPT 버전 3(GPT-3)은 트랜스포머 디코더 모듈 96 개로 구성됨

■ [표 2-2]에 GPT-3모델 유형들이 나열되어 있는데 모두 3000 억개의 토큰들을 어휘(vocabulary)로 하여 학습.

[표 2-2] GPT-3 모델들의 크기, architecture, learning hyper-parameter들.

Model	$n_{parameters}$	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

□ 트랜스포머 모듈(셀프 어텐션, FFNN)의 연산량

○ 모델 크기 [표 2-2]의 d_model에 해당하는 은닉 차원의 크기를 D 그리고 입력

시퀀스 길이를 n라고 할 때, 보통 표준 트랜스포머의 FFFN의 중간 계층의 차원 크기는 은닉 차원 크기의 4배인 $4D$이다(Vaswani et al., 2017).

○ 이에 셀프 어텐션 모듈과 포지션 와이즈 FFNN 모듈의 연산 복잡도와 파라미터 개수는 [표 2-3]을 통해 확인할 수 있듯이, 특히 연산의 병목이 되는 부분은 포지션 와이즈 FFFN 모듈임을 확인할 수 있음

[표 2-3] 셀프 어텐션과 포지션 와이즈 FFNN 모듈의 연산복잡도와 파라미터 갯수

모듈명	연산복잡도	n_parameters
셀프 어텐션	$O(n^2 \cdot D)$	$4D^2$
포지션 와이즈 FFNN	$O(n \cdot D^2)$	$8D^2$

○ 이에 따라, 셀프 어텐션과 포지션-와이즈 FFNN의 연산 복잡도를 중심으로 트랜스포머 모델 성능을 개선하고자 하는 시도들이 다수 출현하게 됨

제3장

언어 모델의 확장

제 1 절 BERT 기본 모델

1. 개요

□ 트랜스포머 기반의 사전 학습 모델

○ BERT(Bidirectional Encoder Representation from Transformer)는 트랜스포머 모델 구조 중 인코더 모듈만으로 구현된 사전 학습된 마스킹 기반의 언어 모델임 [Devlin2018a]

■ GPT[Radford2018] 계열의 언어 모델의 경우 트랜스포머 모델 구조 중 디코더 모듈로 구현됨

■ T5[Raffel2020a]와 같은 언어 모델처럼 트랜스포머의 인코더-디코더 모듈을 모두 사용하는 경우도 있음

■ ELECTRA[Clark2020a]의 경우 트랜스포머의 인코더를 이용하여 generator와 discriminator 모듈 두 개를 생성, 이용하는 경우도 존재함

○ BERT는 트랜스포머 인코더를 이용한 언어 모델

■ BERT의 기본 구조는 트랜스포머의 인코더를 쌓아 올린 구조임

- 기존 트랜스포머 모델이 켜켜이 쌓인 6개의 인코더 모듈을 포함한 인코더 디코더 구조라면, BERT Base 모델은 총 12개의 인코더로, Large 모델은 총 24개의 인코더로 구성됨.

- BERT Base 모델은 또한 출력 벡터의 차원이 768이고, 어텐션 헤드의 개수는 12개로 1.1억 개의 파라미터를 가지고 있으며, 이는 GPT 버전 1과의 비교를 위하여 GPT 버전 1과 동일한 파라미터를 가지도록 구성되었음

- BERT Large 모델은 출력 벡터 크기가 1024, 어텐션 헤드는 16개를 가지도록 구성되어 약 3.4억 개의 파라미터를 가지도록 구성됨

○ 문맥 임베딩(Contextual embedding)

■ BERT는 ELMo나 GPT 버전 1과 마찬가지로 문맥 임베딩(contextual embedding) 기법을 사용하고 있음

- BERT의 입력은 768 차원의 임베딩 벡터가 지정된 시퀀스 길이 512 개가 되고, CLS 같은 특수 토큰들을 포함하여 각 단어에 대해서 동일하게 되어 있음.
- 또한 출력에 있어서도 768 차원의 벡터를 출력하도록 되어 있음. 즉 은닉 차원의 크기는 입력과 출력에서 변함이 없음
- BERT 내 어텐션 모듈들의 연산을 거친 후 출력 임베딩은 입력 시퀀스 즉 문장의 문맥을 모두 참고한 문맥이 반영된 임베딩이 됨

○ 서브워드 토크나이저(subword tokenizer)

■ BERT는 입력 시퀀스를 구성하는 토큰들을 구성하는데 있어 단어들을 단어보다 더 작은 구성단위로 분할하는 서브워드 토크나이저(subword tokenizer)를 이용함

- BERT가 사용하는 토크나이저는 WordPiece 토크나이저로 Byte Pair Encoding(BPE)와 유사한 형태의 알고리즘임
- BERT에서 토큰화는 먼저 입력 토큰이 어휘 집합(Vocabulary)에 존재하는지를 검사하고, 어휘 집합에 이미 있는 단어라면 토큰을 분리 하지 않음
- 반면 어휘집합에 존재하지 않는 경우에는 해당 토큰을 서브워드(subword)로 분리하고, 첫 번째 서브워드를 제외한 나머지 분할된 서브워드들은 앞에 “##”라는 특수 심볼을 붙여서 토큰들로 함
- 이러한 방식은 어휘집합(vocabulary)에 토큰이 없는 경우 발생하는 OOV(Out-Of-Vocabulary)를 해결할 수 있게 함

■ BPE(Byte Pair Encoding) 알고리즘

- BPE 알고리즘은 원래 데이터 압축 알고리즘으로 고안되었으나, 자연어 처리 분야에서 단어를 서브워드로 분리하기 위한 알고리즘으로 응용이 됨
- BPE는 기본적으로 연속적으로 가장 많이 등장한 글자의 쌍(pair)을 찾아서

하나의 글자로 병합하는 방식을 수행함. 예를 들어서 아래와 같은 문자열이 주어졌을 때,

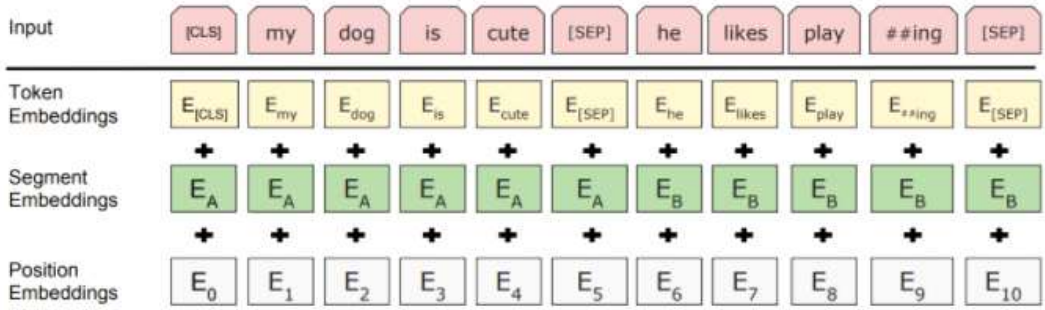
- aababababcca와 같은 문자가 주어진다면, 해당 시퀀스에서 가장 많이 등장한 글자의 쌍을 찾아 하나의 문자로 치환함.
- 위 예에서는 aa, ab, ba, bc, cc, ca 쌍이 존재하며 각 쌍들의 출현 횟수는 aa=1, ab=3, ba=2, bc=1, cc =1, ca=1이며, 가장 많은 출현횟수를 가진 ab를 Z로 치환하면, aZZZcca가 됨. 이제 다시 이 aZZZcca를 가지고 가장 많은 출현횟수를 가지는 쌍을 구하면 aZ =1, ZZ=2, Zc=1, cc= 1, ca =1회이므로 ZZ를 X로 치환하면 aXZcca로 줄일 수 있음. 이렇게 각 쌍들을 더 이상 병합할 쌍이 없을 때까지 반복하여 압축하는 것이 BPE 알고리즘임
- 자연어 처리에서 BPE는 서브워드 분할을 위해 사용되는데, 글자 단위에서 점차적으로 단어 집합을 만들어 내도록 하는데 사용됨
- 이를 위해 모든 단어들을 글자(character) 단위로 분리함. 즉 어휘 사전(dictionary)의 초기 구성은 글자 단위로 분리된 상태임
- 이후 BPE를 이용하여 가장 빈도수가 높은 글자의 쌍을 하나의 단어로 통합하는 과정을 반복함. 이 반복횟수는 사용자가 지정할 수 있음
- 주어진 반복횟수만큼 반복하게 되면, 빈도수에 따라 단어가 토큰 단위로 분할되는 특성을 가짐
- 이러한 방식은 어휘 사전에 토큰이 없는 경우 발생하게 되는 OOV 문제를 글자단위까지 분할된 토큰들이 어휘에 있게 되면서 없어지는 효과와 함께 빈도가 높은 단어 또는 서브워드들이 어휘 사전에 구성되는 장점을 가짐
- BERT에서는 BPE를 코퍼스의 likelihood를 가장 높이는 쌍으로 병합도록 변경된 Wordpeice 모델[Wu2016a]을 이용

○ 입력 임베딩

■ BERT 모델의 입력은 입력 토큰 단위의 임베딩 벡터들의 시퀀스임

- 한 토큰의 임베딩은 토큰 임베딩(token embedding), 세그먼트 임베딩(segment embedding), 포지션 임베딩이라는 세 개의 임베딩의 합으로 구

성이 됨 (그림 3-1 참조)



[그림 3-1] BERT 입력의 구성 [Devlin2018a]

- 토큰 임베딩은 앞에서 설명한 WordPeice 모델에 따라 구성된 어휘 사전의 토큰에 대한 임베딩 값임.

■ 포지션 임베딩은 각 토큰의 위치 정보를 추가하기 위해 이용

- 한 시퀀스의 길이가 512개이기 때문에 최대 512개의 포지션 임베딩 벡터를 학습
- 트랜스포머에서 사용한 sin, cos 함수와 같은 주기 함수를 이용한 임베딩이 아님

■ 세그먼트 임베딩

- 입력 세그먼트를 구분하기 위한 임베딩으로 임베딩 벡터의 종류는 문장의 최대 개수인 2개임; 이는 Q&A 시스템 등으로의 활용을 위해 질문과 응답을 구성하는 두 개의 세그먼트를 구분하기 위해 사용
- BERT에서는 한 세그먼트의 길이를 512 토큰으로 제한함.
- 한 세그먼트에 두 문장 이상을 넣기 위해서는 [SEP] 이라는 특수 토큰을 이용해 여러 문장을 한 세그먼트에 기입할 수 있음

□ 사전학습과 파인 튜닝

○ BERT는 대량의 코퍼스를 이용한 모델의 사전 학습과 파인 튜닝을 수행

■ LSTM RNN과 달리 BERT는 양방향성을 갖기 때문에 이전 토큰이나 이후 토큰을 예측하는 방법이 아닌 마스킹된 토큰을 예측하는 태스크로 사전학습

을 진행

- 랜덤으로 15%의 토큰들을 선택
- 이중 80%의 단어들을 [MASK]로 변경
- 이중 10%의 단어들은 랜덤으로 토큰을 다른 토큰으로 변경
- 이중 10%의 단어들은 동일하게 그대로 둠

○ BERT는 사전학습 태스크로 마스크된 단어를 예측하도록 함

■ 이렇게 하는 이유는 선택된 모든 토큰들을 [MASK]로 모두 마스킹 할 경우 [MASK] 토큰이 파인 튜닝 단계에서는 나타나지 않으므로 사전 학습 단계와 파인 튜닝 단계에서의 불일치가 발생

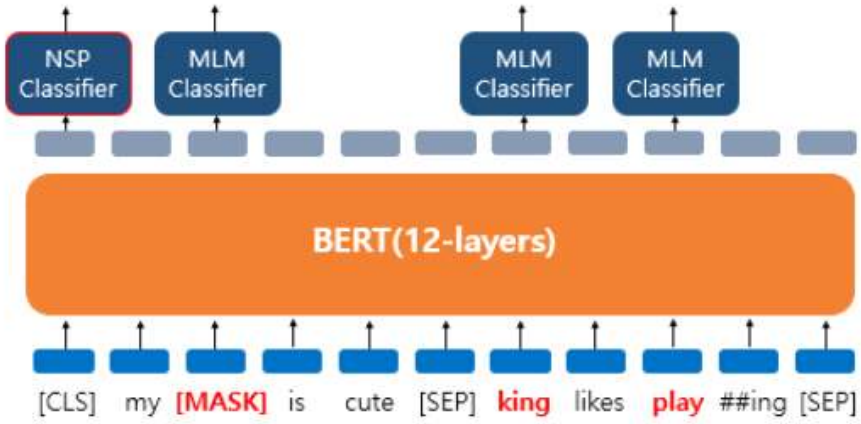
- 이 문제를 완화하기 위해 랜덤으로 선택된 15%의 단어들의 모든 토큰을 [MASK]로 마스킹 하지 않음
- 즉, 마스킹된 단어 뿐만 아니라 랜덤으로 변경된 단어와 그대로 유지된 단어에 대해서도 출력 층에서 소프트맥스로 원래 단어가 무엇인지 예측할 수 있게 함으로써 보다 많은 토큰들에 대해 단순히 마스킹된 단어 뿐만 아니라 다양한 토큰들을 예측할 수 있게 함
- 그림 3-2는 이렇게 마스킹된 토큰과, 변경된 단어, 변경되지 않은 단어 모두에 대해 토큰 예측을 수행하는 예시를 보임

○ BERT는 사전학습 태스크로 다음 문장 예측을 수행함

■ BERT는 두 문장을 주고 이 문장이 서로 이어지는 문장인지를 맞추는 방식으로도 훈련함

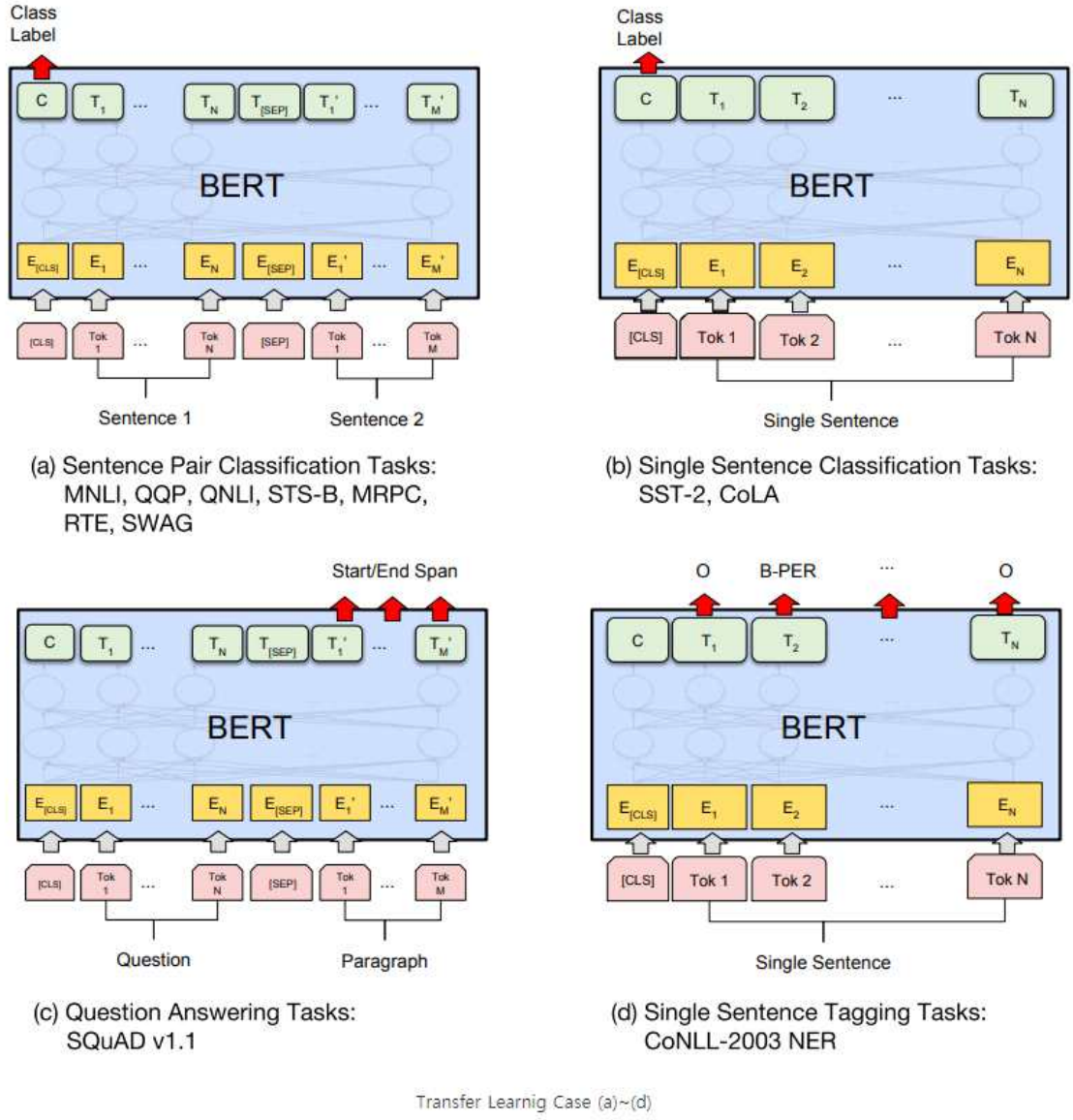
- 이를 위해 50:50의 비율로 실제 이어지는 두 개의 문장과 랜덤으로 붙인 두 개의 문장을 주고 훈련시킴
- 두 문장이 서로 연결되는지, 아닌지를 이진 분류를 하도록 학습을 진행 함
- 이러한 태스크를 추가하는 것은 Q&A(Question & Answering)이나 NLI(Natural Language Inference)와 같이 두 문장 관계를 이해하는 것이 중요한 태스크들이 존재하기 때문임

○ 파인 튜닝을 통한 실제 NLP 태스크의 수행



[그림 3-2] 마스킹 된 토큰의 예측 및 다음 문장 예측 [유원준2021]

- 사전 학습된 BERT 모델을 가지고 풀고자 하는 태스크의 데이터를 추가로 학습시키는 과정을 파인 튜닝이라 하며 이 과정을 통해 BERT 언어 모델을 여러 NLP 태스크에 적용시킬 수 있음.
- 텍스트 쌍에 대한 분류 문제
  - 자연어 추론(NLI) 등의 문제에 해당하는 것으로 [SEP]로 구분되는 두 텍스트가 어떠한 관계에 있는지를 분류하는 유형임
  - 그림 3-3의 (a)에 해당하는 태스크로 소제목의 약어들은 이 NLP 태스크에 대응하는 벤치마크 데이터셋을 의미함
- 한 문서에 대한 텍스트 분류
  - 이 유형은 한 문서에 대한 분류 유형으로, 뉴스나 리뷰 등의 텍스트를 분류하는 유형으로 문서의 시작에 [CLS]라는 토큰을 입력
  - 분류 문제에서는 이 [CLS] 토큰 위치의 출력층에 Dense 또는 완전 연결 계층을 추가하여 분류에 대한 예측을 수행함. 그림 3-3의 (b)에 해당
- 질의 응답 문제
  - 그림 3-3(a)와 같이 텍스트 쌍을 입력받는 또 다른 태스크로 질의문과 본문이라는 두 개의 텍스트를 쌍으로 입력 받는데, 첫 텍스트인 질의문에 대해 두 번째 텍스트인 본문의 일부분을 추출하여 질의문에 대답하는 유형임(그림 3-3 (c) 참조)



[그림 3-3] 파인 튜닝을 통한 여러 NLP 태스크의 수행 [Devlin2018a]

■ 각 토큰에 대한 태깅 작업

- 이 유형은 입력 시퀀스의 각 토큰에 대하여 태깅을 수행하는 것으로 품사 태깅이나 개체명 인식 등이 이에 해당; 그림 3-3의 (d)에 해당
- 토큰 단위로 각 토큰에 대해 밀집 층을 이용한 분류 예측을 수행함

제 2 절 언어 모델의 학습 기법 개선 기법

1. 개 요

□ 다양한 마스크된 언어 모델을 위한 학습 기법이 제안됨

○ BERT는 대량의 코퍼스를 입력으로 사전학습 태스크로 다음 문장 예측과 마스킹된 토큰 예측을 수행함

○ 이 사전 학습 태스크를 개선함으로써 좀 더 사전학습의 성능을 올리고자 하는 다양한 시도들이 존재함

■ RoBERTa, ALBERT, SpanBERT, StructBERT, DeBERTa 등이 이렇게 학습 태스크의 유형을 변경함으로써 성능향상을 꾀하는 방식임

○ 또한 BERT의 어텐션 메커니즘이나 모델을 변경함으로써 성능 향상을 꾀하고자 하는 시도들이 존재

■ 4장에서 소개할 어텐션 메커니즘의 효율성 향상을 위한 시도들 이외에도 DeBERTa와 같이 어텐션 가중치 표현을 변경한다거나 또는 T5, Electra처럼 트랜스포머의 인코더만 가지고 언어 모델을 구성하는 것이 아닌 인코더-디코더 모델, 생성자-구별자 모델과 같은 다양한 언어 모델도 제안됨

○ 이외에 사전학습을 위해 코퍼스뿐만 아니라 기존에 잘 작성되어 구축되어 있는 DBPedia, Yago, Freebase 등과 같은 지식그래프(Knowledge Graph; KG)의 정제된 정보를 코퍼스와 함께 학습시킴으로써 모델 성능 향상을 꾀하고자 하는 시도도 존재함

■ 대표적인 연구로는 K-BERT, KnowBert, KEPLER, ERNIE 등이 존재함. 이에 대해서는 3절에서 설명함

2. 사전 학습 기법 변경을 통한 언어 모델 개선

□ RoBERTa [Liu2019a]

○ RoBERTa는 기본적으로 BERT와 동일한 모델 구조를 이용함

■ FACEBOOK AI 팀은 처음 발표된 BERT[Devlin2018a] 모델이 학습이 제대로

안되어 초매개변수(hyperparameter)들이 최적으로 설정되지 않았음을 지적함

■ 이에 따라 사전학습(pre-training) 과정에서 BERT 모델을 더 견고하게 최적화하기 위해 초매개변수의 설정과 관련한 여러 학습 방법들을 포함함

○ 개선 방법

■ 동적 마스킹(dynamic masking)

- 기존 BERT 모델이 사전학습에서 사용하는 마스킹 기법은 무작위로 입력 토큰에 마스크를 씌우고 그것을 예측하는데 있어, 학습 시작 전에 마스킹을 하게 됨에 따라 같은 마스크를 처리하게 됨(static masking)
- 이를 보완하고자 BERT는 동일한 문장을 여러 개 복사하여 복사된 문장들을 무작위로 마스킹 하는 방법을 이용하나, 이럴 경우 학습 과정 중에 메모리를 더 사용하는 문제가 발생

■ 배치 및 학습 데이터 크기의 증가

- RoBERTa는 배치 크기의 학습 성능의 영향을 점검하기 위한 실험을 진행하여, 같은 step 수에도 배치 크기가 클수록 성능이 좋아지는 경향을 실험을 통해 확인함
- 또한, 학습 데이터 크기가 모델 성능에 영향을 줌에 따라 RoBERTa는 BERT에 비해 더 많은 데이터(160GB, BERT의 약 10배), 더 큰 배치 사이즈(8K, BERT의 약 32배로)로 더 오래 학습을 진행하였음

■ 입력 형식 변경과 다음 문장 예측(NSP) 태스크의 제거

- 기존 BERT 모델에서는 두 개의 문장을 이어 붙여 입력 시퀀스를 생성하였고, 두 문장이 문맥상 연결된 문장인지를 판단하는 다음 문장 예측(NSP; Next Sentence Prediction) 태스크를 사전 학습 단계에서 마스크된 단어 예측 태스크와 같이 진행하였음
- RoBERTa에서는 이 NSP 태스크가 모델 학습에 그리 큰 기여를 하지 못한다는 실험 결과를 얻어 마스크된 단어 예측 태스크만으로 사전 학습을 수행함
- 이에 따라 굳이 512 크기의 입력 시퀀스를 생성하는데 있어 꼭 두 문장만으로 구성할 이유가 없어짐에 따라 문장 길이를 BERT에서의 입력 최대

길이인 512를 넘어가지 않는 선에서 문장을 최대한 이어 붙여 full-sentence 형식으로 문장을 구성함

□ ALBERT[Lan2020]

○ BERT 모델의 경량화 된 버전

■ BERT 모델의 성능을 향상시키기 위해서는 큰 크기의 모델로 학습해야 하나 이럴 경우 파라미터 수 증가에 따른 학습 시간 및 메모리 요구량 증가

■ 지식 증류 기법을 이용하여 학습된 BERT 모델로부터 경량화 된 모델을 구할 수 있으나, 추가적인 개발 비용 소요

■ 아예 **처음부터 파라미터 수를 감소시킨 BERT 모델을 가지고 학습을 수행하는 방법**을 제안하여, 파라미터 수는 BERT보다 18배 적으면서 약 1.7배 빠르게 학습될 수 있게 함

○ 개선 방법

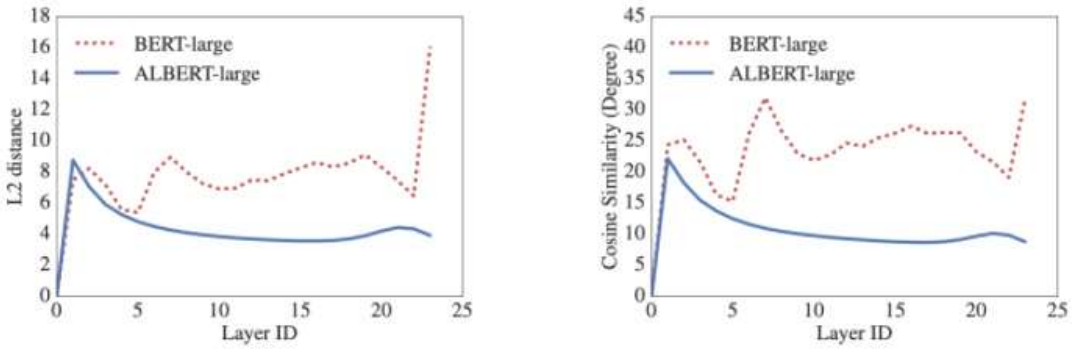
■ Factorized embedding parameterization

- 기존 BERT 모델에서는 WordPiece 임베딩 크기 E와 은닉 계층 크기 H를 동일하게 설정함(E = H). 어휘사전 크기가 V라 할 때 어휘사전 전체에 대한 임베딩 크기는 V x E로 무척 커지게 됨.
- 이를 줄이기 위해 은닉 공간으로 사상하는 계층 (E x H)를 하나 더 만들어 줌으로써 파라미터 숫자를 O(V x H)에서 O(V x E+ E x H), H>>E로 줄임으로서 모델 크기를 줄임

■ 레이어 간 파라미터 공유

- 기존 BERT 모델에서는 계층 간에 파라미터들이 서로 공유되지 않음.
- ALBERT에서는 모든 계층에서 파라미터 값들의 공유를 수행함(cross-layer parameter sharing). 즉 BERT 모델이 12개의 트랜스포머 블록에서 블록별로 각 파라미터들을 학습해왔다면 ALBERT에서는 1개의 트랜스포머 블록을 12번 돌면서 학습함
- 파라미터 공유를 통해 파라미터 개수도 획기적으로 줄일 수 있을 뿐(1/계층 수)만 아니라, 그림 3-4에서 볼 수 있듯이 계층이 깊어짐에도 안정적으

로 학습이 됨



[그림 3-4] ALBERT의 계층별 L2 거리와 코사인 유사도 [Lan2020]

■ 문장 순서 예측 태스크의 도입

- RoBERTa에서 지적하였듯이 기존 BERT의 다음 문장 예측(NSP) 태스크는 실제 성능 향상에 큰 도움을 주지 못함을 지적함
- BERT의 NSP 태스크에서 두 번째 문장은 실제 문장 또는 임의로 뽑은 문장인데, 임의로 뽑은 문장은 첫 문장과 완전히 다른 주제의 문장일 확률이 높으므로 문장 간 연관 관계를 학습하기보다 두 문장이 같은 주제에 대해 말하는지를 판단하는 주제 예측(topic prediction) 문제에 가까움
- 이를 계선하기 위해 ALBERT에서는 문장 순서 예측(SOP; Sentence Order Prediction) 태스크를 사전 학습에 이용
- SOP 태스크에서 학습 데이터는 실제 연속인 두 문장(positive example)과 두 문장의 순서를 앞뒤로 바꾼 문장(negative example)로 구성되고 문장의 순서가 옳은지 여부를 예측하는 방식으로 학습을 수행.
- 표 3-1에서 보이는 바와 같이 SOP로 학습해도 NSP 태스크에 대한 예측이 어느 정도 가능함에 반해, NSP 태스크로 학습 시 SOP 태스크에 대한 정확도는 많이 떨어지며, 다운스트림 태스크에 대해서도 SOP가 NSP 대비 나은 성능을 보여주고 있음

[표 3-1] NSP와 SOP 태스크의 비교 [Lan2020]

	Intrinsic Tasks			Downstream Tasks					
SP tasks	MLM	NSP	SOP	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
None	54.9	52.4	53.3	88.6/81.5	78.1/75.3	81.5	89.9	61.7	79.0
NSP	54.5	90.5	52.0	88.4/81.5	77.2/74.6	81.6	**91.1**	62.3	79.2
SOP	54.0	78.9	86.5	**89.3/82.3**	**80.0/77.1**	**82.0**	90.3	**64.0**	**80.1**

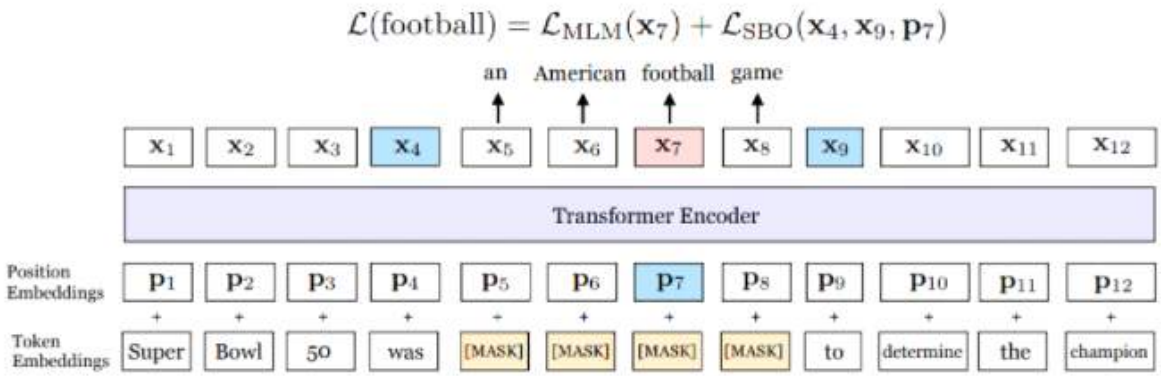
□ SpanBERT [Joshi2020]

○ 질의응답과 동일지시어 해소에 효과적인 BERT 모델 학습 방법

■ 질의응답(Question and Answering)과 동일지시어 해소(Coreference Resolution)에 있어서는 한 토큰이 아니라 여러 개의 토큰들(token span)이 대상이 되는 경우가 많음

■ 자연어처리 태스크가 이렇게 두 개 이상의 토큰 스팬 사이의 관계에 대한 추론을 포함하는 경우를 위해 SpanBERT는 BERT에 임의 연속 스팬 마스킹과 마스킹된 스팬 전체를 그 경계의 토큰으로부터 유추하는 SBO(Span Boundary Objective)를 도입

■ 기존 BERT 모델 구조는 그대로 이용함



[그림 3-5] SpanBERT에서의 사전학습 방법 예시 [Joshi2020]

○ 개선 방법

■ 스팬 마스킹(Span Masking)

- 입력 시퀀스의 15%가 마스킹 되도록 하는데 토큰 스팬의 길이는 기하분포($(l \sim \geq o(p))$)에 의해 선택하고, 균등 분포에 따라 임의로 토큰 스팬의

시작 위치를 선택. 논문에서는 $p = 0.2, l_{\max} = 10$로 설정.

- 또한 스팬의 길이는 서브워드 토큰이 아닌 완전 단어 단위로 설정함. BERT에서 10%를 임의로 토큰 교체하는 것을 SpanBERT에서는 토큰 단위가 아닌 스팬 단위로 대체하도록 함

■ SBO 도입 및 NSP 제거

- SpanBERT에서는 스팬의 끝에 최대한 많은 스팬 내부 컨텐츠를 압축하기 위해 경계의 토큰을 사용하여 마스킹된 스팬을 예측하는 SBO(Span Boundary Objective) 태스크를 도입
- BERT의 경우 순차적으로 오는 두 개의 문장을 하나의 입력 시퀀스로 하여 NSP 태스크를 수행하는데 반해 SpanBERT는 하나의 긴 문장을 처리하는 방식으로 사전학습 수행

□ StructBERT [Wang2020c]

○ 언어 구조 정보를 활용한 BERT 모델의 성능 향상

■ BERT 모델 구조를 그대로 이용하면서도 언어 구조 정보를 추가로 학습시키게 함으로써 BERT 모델 성능을 향상시키는 것을 목표로 함

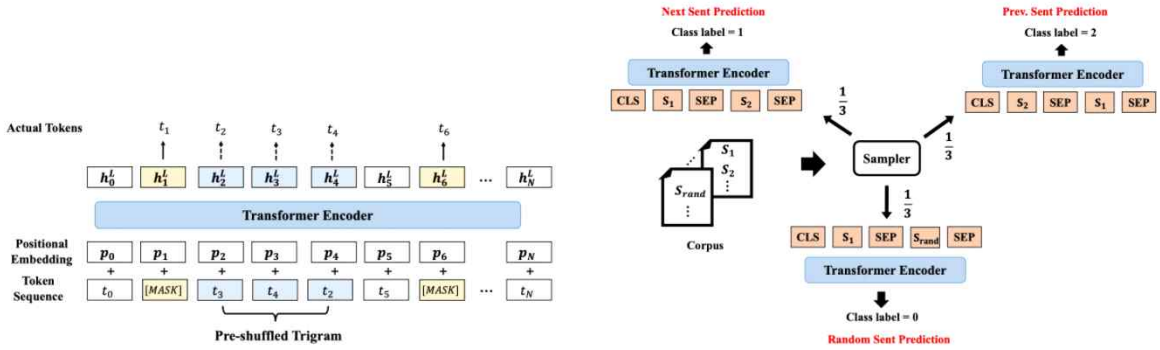
■ BERT의 사전학습 태스크는 언어 구조상의 고차원적인 의존성을 정확히 모델링하지 못한다는 가정에 출발하여 BERT 언어 모델이 단어뿐만 아니라 문장 간 구조도 잘 학습될 수 있도록 2개의 추가적인 태스크를 부여함.

■ 기본적으로 BERT 모델 구조를 그대로 사용하며, 입력 시퀀스 형태 또한 동일함

○ 개선 방법

■ 단어 구조의 학습(WOO; Word Ordering Objective)

- StructBERT에서는 마스킹된 토큰의 예측 분류 이외에 임의로 순서가 뒤바뀐 토큰들을 주고, 해당 위치에서 올바른 토큰의 우도가 최대가 되도록 학습함(그림 3-6(a)), 즉 이 문제에 대한 목표 함수(objective function)는 $\arg max_\theta = \sum \log P(pos_1 = t_1, pos_2 = t_2, \ldots pos_K = t_K | t_1, t_2, \ldots, t_K, \theta)$로 정의 가



[그림 3-6] StructBERT의 2개의 추가 학습 태스크 [Wang2020c]

능

- StructBERT 논문에서 순서정보를 변경하는 토큰의 개수 K는 3으로 설정하였으며, 마스킹된 토큰 예측 태스크 이외에 마스킹이 되지 않은 트라이그램(trigram, K=3) 중 일부를 임의로 선택하여 해당 토큰들의 순서를 변경. 그리고 이를 인코더로 인코딩 후 softmax 계층에서 원래 토큰 순서를 예측하게 함. 그림 3-6에서는 2-4번째 토큰이 순서가 변경되어 입력으로 들어가고 이 토큰들의 순서를 예측하게 함
- 이 WOO는 동일한 모델로 마스킹된 토큰 예측(MLM)과 같이 한 번에 학습되며, MLM과 같은 가중치로 손실 값이 합산됨

■ 문장 구조의 학습(Sentence Structural Objective)

- StructBERT에서는 다음 문장 예측(NSP) 태스크에 추가하여 이전 문장 예측을 같이 수행하도록 함
- 즉 한 입력 시퀀스를 구성하는 두 개의 세그먼트 A, B가 있다고 할 때, NSP 태스크가 B가 A의 다음 문장인지 아닌지를 판별한다면, StructBERT에서는 B가 A의 이전 문장으로 오는 것이 적절한지도 예측함. 즉 기존 NSP가 이진 분류(binary classification)이라면, StructBERT는 3개 클래스로의 분류를 수행함
- 이를 위해 입력 시퀀스를 세그먼트 A 뒤에 세그먼트 B가 문맥에 맞는 구성(positive sample), 세그먼트 A 뒤에 임의로 추출한 문장을 B로 구성하는 경우(negative sample), 그리고 세그먼트 A 뒤에 A 이전의 문장으로

세그먼트 B를 구성하는 경우(이전 문장 예측)로 구분하며 이 3가지 경우를 동일한 확률 1/3으로 샘플링함

[표 3-2] GLUE 벤치마크 결과 [Wang2020c]

System	CoLA 8.5k	SST-2 67k	MRPC 3.5k	STS-B 5.7k	QQP 363k	MNLI 392k	QNLI 108k	RTE 2.5k	WNLI 634	AX	Average
Human Baseline	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0/92.8	91.2	93.6	95.9	-	
BERTLarge [6]	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7/85.9	92.7	70.1	65.1	39.6	80.5
BERT on STILTs [19]	62.1	94.3	90.2/86.6	88.7/88.3	71.9/89.4	86.4/85.6	92.7	80.1	65.1	28.3	82.0
SpanBERT [12]	64.3	94.8	90.9/87.9	89.9/89.1	71.9/89.5	88.1/87.7	94.3	79.0	65.1	45.1	82.8
Snorkel MeTaL [22]	63.8	96.2	91.5/88.5	90.1/89.7	73.1/89.9	87.6/87.2	93.9	80.9	65.1	39.9	83.2
MT-DNN++ [14]	65.4	95.6	91.1/88.2	89.6/89.0	72.7/89.6	87.9/87.4	95.8	85.1	65.1	41.9	83.8
MT-DNN ensemble [14]	65.4	96.5	92.2/89.5	89.6/89.0	73.7/89.9	87.9/87.4	96.0	85.7	65.1	42.8	84.2
StructBERTBase	57.2	94.7	89.9/86.1	88.5/87.6	72.0/89.6	85.5/84.6	92.6	76.9	65.1	39.0	80.9
StructBERTLarge	65.3	95.2	92.0/89.3	90.3/89.4	74.1/90.5	88.0/87.7	95.7	83.1	65.1	43.6	83.9
StructBERTLarge ensemble	68.6	95.2	92.5/90.1	91.1/90.6	74.4/90.7	88.2/87.9	95.7	83.1	65.1	43.9	84.5
XLNet ensemble [32]	67.8	96.8	93.0/90.7	91.6/91.1	74.2/90.3	90.2/89.8	98.6	86.3	90.4	47.5	88.4
RoBERTa ensemble [15]	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8/90.2	98.9	88.2	89.0	48.7	88.5
Adv-RoBERTa ensemble	68.0	96.8	93.1/90.8	92.4/92.2	**74.8/90.3**	**91.1/90.7**	98.8	**88.7**	89.0	**50.1**	88.8
StructBERTRoBERTa ensemble	**69.2**	**97.1**	**93.6/91.5**	**92.8/92.4**	**74.4/90.7**	90.7/90.3	**99.2**	87.3	**89.7**	47.8	**89.0**

- 추가적인 2종류의 사전 학습 태스크를 수행한 결과는 표 3-2와 같아 GLUE 벤치마크의 9종류 태스크 중 2종류를 제외하고 기존 BERT 모델 학습 기법들 보다 나은 성능을 보임

□ DeBERTa [He2020]

○ RoBERTa를 기반으로 하여 추가적인 성능 개선

■ DeBERTa는 RoBERTa를 기반으로 다음의 두 기법을 적용하여 성능을 개선

- Disentangled 어텐션 기법 : 각 단어의 어텐션 가중치를 서로 독립인 두 벡터 content 와 position 벡터의 조합으로 표현
- 또한 위치 값 임베딩을 상대적인 임베딩 값으로 함. 특정위치의 단어를 기준으로 일정 길이(윈도우) 이내에 위치한 단어와의 상대적 거리 관계 정보를 반영.

■ 어텐션을 구하는 행렬 연산 과정에서 이를 활용. 각 단어 간의 상대적인 위치 정보(pair-wise positional relationship)를 파악할 수 있음

■ Enhanced mask decoder: 마스킹된 토큰을 예측 시 해당 예측을 수행하는 계층에서 절대 위치 정보를 추가하여 확장된 마스크 디코더를 이용

■ 현재 바이두 그룹의 ERNIE에 이어 GLUE 벤치마크에서 SOTA 2위를 기록

3. 파인 튜닝에서의 개선 사항

□ 개요

○ 사전 학습 언어 모델의 파인 튜닝 학습에 있어 학습의 불안정성을 보이는 조건으로 거대 학습 데이터를 거대 모델로 학습한 것(bert-large, roberta-large 등 멀티 헤드 어텐션 모듈 24계층 이상의 모델)에 비해, 파인 튜닝에 사용되는 학습데이터의 크기가 작은 경우 (<10K(1만 개 샘플) 미만) 그 파인 튜닝 과정에서 학습의 불안전성을 보이는 경우가 자주 보고되어 왔음[Zhang2021a, Mosbach 2021a].

○ 사전학습 모델을 기반으로 하는 전이 학습(transfer learning)의 한 종류로 볼 수 있는 파인 튜닝 학습에 있어 그 학습의 불안전성의 문제를 해결하는 방안은 크게 학습 방법에 대한 개선과 기존 사전학습 모델 데이터로의 과적합(over-fitting)된 파라미터를 재초기화 하는 두 가지 방식을 설명할 수 있다.

□ 최적화기(Optimizer)의 개선

○ Bert ADAM 기법

■ Devlin 등은 사전 학습모델인 BERT 언어 모델 학습에 있어 ADAM 알고리즘(Kingma & Ba, 2014)의 bias 값을 업데이트 하지 않는 BERT ADAM 알고리즘을 사용함 [Devilin2018a]

■ BERT Adam 알고리즘에서 Adam알고리즘을 변형하여 생략한 9번째와 10번째 라인의 역할은 bias 수정을 위해 epoch별로 learning rate을 $\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$ 비율로 rescaling하는 효과를 위함임

■ Zhang 등의 연구 결과에 따르면, bias를 수정하지 않은 버전의 BERT Adam 알고리즘을 이용하게 될 경우, 파인 튜닝 학습 시간을 단축하면서 안정된 결과를 얻을 수 있다고 함[Zhang2021]

■ Mosbach 등은 bias를 수정한 Adam 알고리즘[Kingma&BA2014]을 이용하게 될 경우, 학습 초기 학습 비율(learning rate) 크기를 늘려서 탐색 범위를 넓히는 역할을 하게 되어 일종의 웜업(warm-up) 구간을 두는 것과 같은 역할

을 하게 된다고 봄

- 이를 통해 파인 튜닝 학습을 오랜 시간 안정된 학습을 하면서, 단기간 학습한 것에 비해 더 향상된 실험 결과를 얻을 수 있다고 주장함 [Mosbach2021]

[표 3-3] BERT ADAM 알고리즘

Adam 알고리즘에 대해 Bert Algorithm에서 bias를 업데이트 하지 않는 Bert Adam 알고리즘을 사용함(아래 코드의 9~10라인을 제외시킴)[Devilin2018a]	
α: learning rate, $\beta_1, \beta_2 \in [0,1)$: moment 추정을 위한 exponential decay rate, $f(\theta)$: 학습 parameter θ의 통계적 목적함수 θ_0 : 초기 parameter 벡터, λ : decoupled weight decay	
1	$m_0 \leftarrow 0$ (첫 번째 moment vector 초기화)
2	$v_0 \leftarrow 0$ (두 번째 moment vector 초기화)
3	$t \leftarrow 0$ (time stamp 초기화)
4	**while** θ_t not converged **do**
5	$t \leftarrow t+1$
6	$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (시점 t에서 목적함수에 따라 gradients값을 취함)
7	$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (첫 번째 biased moment vector 업데이트)
8	$v_t \leftarrow \beta_2 \cdot m_{t-1} + (1-\beta_2) \cdot g_t^2$ (두 번째 biased moment vector 업데이트)
9	~~$\hat{m}_t \leftarrow m_t / (1-\beta_1^t)$ (bias 수정된 moment vector 추정치 계산)~~
10	~~$\hat{v}_t \leftarrow v_t / (1-\beta_2^t)$ (bias 수정된 두 번째 moment vector 추정치 계산)~~
11	
12	$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (parameter 업데이트)
13	**end while**
	return θ_t

□ 재초기화(Re-initialization)

○ Radford 등은 GPT-2 모델 제안 시 GPT-1의 성능을 향상시키기 위해 모델 구조의 변화와 함께 재초기화를 적용함[Radford2019]

■ Pre-activation 구조로 각 블록 앞에 계층 정규화를 적용하고 마지막 셀프 어텐션 이후에 추가로 계층 정규화를 적용하였는데, 이는 ResNet[He2016]과 유사한 구조를 적용한 것임.

■ 잔차 연결이 누적되는 깊이에 재초기화를 적용함

○ Zhang 등 또한 파인 튜닝 시 깊은 계층들을 재초기화시킴으로써 BERT의 파인 튜닝 성능을 향상시킴 [Zhang2021].

제 3 절 지식 주입 기법

1. 지식 주입 기법 개요

□ 언어 모델에 코퍼스 이외에 지식 정보를 추가하기 위한 기법임

○ 추가 학습 정보로 기존에 잘 조직화되어 있는 지식 정보를 이용하여 언어 모델의 성능 향상을 목표로 함

■ 언어 모델을 학습하기 위해 모아놓는 코퍼스 이외에 추가로 이미 사람이 가지고 있는 지식 정보를 활용할 수 없을까라는 시도로 최근에 시도되는 연구임

■ Wikipedia와 같은 사전 정보들은 텍스트로 구성되어 있고, 이미 이러한 텍스트들은 BERT와 같은 언어 모델 학습에 있어 기본적인 코퍼스로서 들어가 있으므로, 실제 아무런 지식 정보 없이 언어 모델을 학습하는 것은 아님

■ 또한 텍스트들이 기본적으로 지식을 내포하고 있으므로, 지식 정보를 추가로 활용한다는 것에는 어폐가 있을 수 있음

■ 오히려 여기에서는 텍스트 이외의 형식으로 구성되어 있는 지식 정보를 코퍼스 이외에 같이 활용하는 방식으로 이해해야 할 필요가 있음

□ 주로 지식 그래프의 활용에 관한 연구가 많음

○ 지식 그래프는 기본적으로 텍스트로 구성되어 있던 정보들 중 개체(entity)들을 식별하고 이들 개체들 간의 관계성을 조직화 한 것

■ 이들 지식 그래프는 주로 RDF(Resource Description Framework)이라는 W3C로 구성됨. 물론 RDF만이 지식 그래프를 구성하기 위한 파일 형식은 아님

■ RDF는 지식을 주어(Subject), 술어(Predicate), 목적어(Object)라고 불리는 3개의 조합(triple)으로 정보를 구성

■ 텍스트에서 추출한 각 개체들은 RDF의 주어 또는 목적어로 조직화되고 이들 간의 관계는 술어로 구성됨

- 예를 들어 <“Steve Jobs”, “CEO”, “Apple Inc.”>라는 트리플은 “Steve Jobs는 Apple Inc.의 CEO이다”라는 문장에서 두 개체 Steve Jobs와 Apple Inc.를 식별하고, 이들 간의 관계 CEO를 추출하여 구성된 것이라 할 수 있음

■ RDF 트리플들은 모여져서 지식 그래프(KG; Knowledge Graph)를 구성

- RDF 트리플은 지식 그래프에서 2개의 노드와 이 두 노드를 연결하는 간선을 의미함
- 그림 3-7에서 사각형들은 주어 또는 목적어를 의미하며, 방향성 있는 화살표는 술어를 의미함.
- 가령, 그림 3-7에서 <author2, memberOf, “KISTI”>라는 트리플은 두 노드를 연결하는 하나의 간선으로 표현되며, 다시 <author2, name, “K.H. Lee”>는 author2 노드를 기준으로 뻗어나가는 또 다른 간선으로 표현됨

○ 대표적인 지식 그래프로는 DBpedia, Freebase, YAGO, OpenCyc, WikiData, Google Knowledge Graph 등이 존재 [Färber2018]

■ 그러나 Wikipedia를 언어 모델의 코퍼스로 활용함에 따라 Wikipedia를 대상으로 하여 생성한 DBpedia나 WikiData보다는 다른 지식 그래프의 효용성이 더 높을 것임

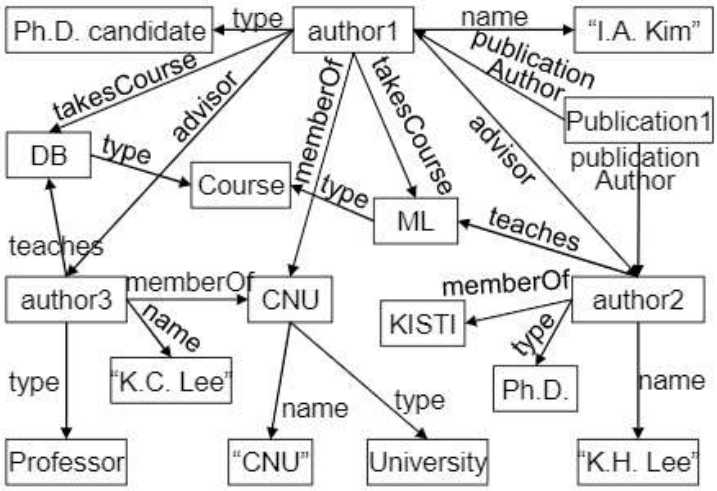
■ 또한, 특정 분야의 텍스트 처리에 효과적인 언어 모델을 개발하기 위해서는 그 분야의 지식 그래프를 활용하는 것이 효과적일 것임

□ 그래프 데이터 구조에 대한 처리가 필요

○ 지식 그래프는 기본적으로 그래프 데이터 모델로 토큰들의 시퀀스를 입력으로 하는 언어 모델과 데이터 형식부터 차이가 있음

■ 이에 따라 언어 모델 학습에 있어서 그래프 데이터 모델로 표현된 지식 그래프를 어떤 방법으로 활용할 것인지에 대한 논의가 필요함

■ 현재는 그래프 데이터 모델을 별도의 시퀀스로 만드는 방법, 별도의 임베딩



[그림 3-7] 지식 그래프 예시

벡터로 만들어 토큰 단위 임베딩과 합하는 방법, 또는 모델을 확장하여 그래프 임베딩을 추가하는 방법 등이 제시되어 있음

○ 입력 시퀀스에 대응하는 지식 그래프의 검색 필요

■ 입력 시퀀스에 대응하는 추가적인 지식을 지식 그래프를 이용하여 제공하기 위해서는 정확하게 해당 입력 시퀀스에 대응하는 지식 정보를 제공할 필요가 존재함

- 예를 들어 “Steve Jobs was the CEO of Apple Inc.”와 같은 입력 시퀀스를 입력받았을 때, Steve Jobs에 대한 추가적인 정보를 제공하기 위해서는 이 시퀀스가 인물에 관한 것으로, 동명의 책이나, 영화, 또는 의류 업체를 의미하지는 않음을 판별하고, Apple Inc.의 전 CEO였던 Steve Jobs와 관련된 정보만을 제공해야 함을 의미한다.
- 특히 동음이의어의 처리가 중요한데, 영단어에서 ‘bank’와 같은 경우 문맥에 따라 둑이나 제방 또는 은행으로 해석됨에 따라 문맥에 따른 처리가 요구된다.

■ 지식 그래프 자체를 그대로 시퀀스화 하는 것은 곤란하며, 결국 부분탐색 질의가 지원이 되어야 함

- 그래프의 경우 그래프의 평균 차수(한 노드에서 다른 노드로의 간선 수)만

큼 복잡도가 늘어남.

- 그래프 자체의 노드들을 서로 연결하여 h 길이의 1차원의 시퀀스화 한다면, 결국엔 평균 차수 N 개인 그래프로부터 생성할 수 있는 시퀀스의 개수는 $O(h^N)$가 됨에 따라 코퍼스의 크기가 급격히 증가하게 됨
- 때문에 주어진 입력 시퀀스에 대하여 지식 그래프에서 정확하게 관련된 사실만을 제공해야 하는 것이 중요한데 지식 그래프 탐색 쪽에서 이는 부분 그래프 또는 유사 부분 그래프 탐색 문제에 해결되며 이를 지원하는 지식 그래프 DB 등의 기술이 같이 요구됨

2. 지식 주입 관련 제안 기술들

□ K-BERT [Liu2020c]

○ 특정 분야의 전문 지식의 부족을 해결하기 위한 언어 모델의 확장 제안

■ BERT 언어 모델은 대량의 코퍼스(large-scale corpora)를 이용하여 일반적인 언어 표현을 잘 학습하여 왔으나, 특정 분야의 전문 지식(domain-specific knowledge)는 활용하지 못함

■ 반면, 지식 그래프는 특정 분야의 전문 지식을 정제된 형태로 표현함에 따라 이를 활용하고자 하는 시도임

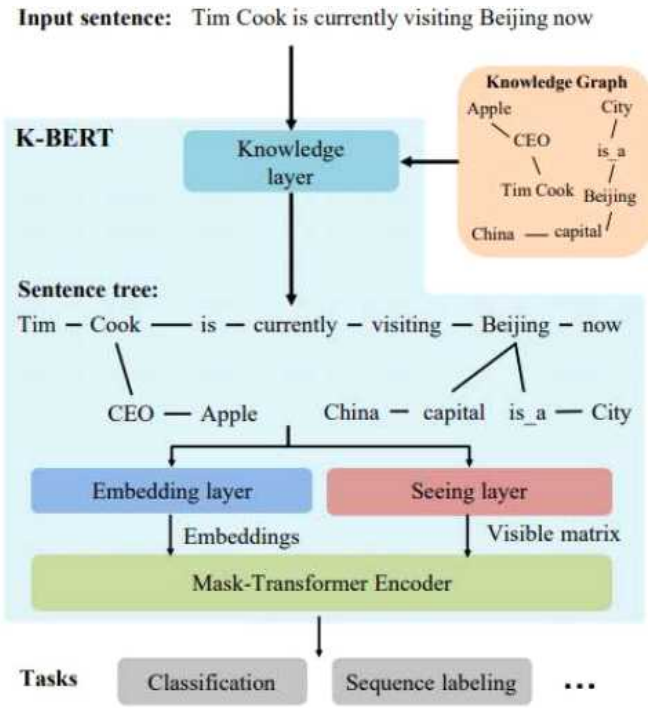
○ 지식 그래프의 시퀀스화

■ 그래프로 표현된 특정 분야의 전문 지식을 학습시키도록 하기 위해 K-BERT는 지식 그래프의 시퀀스화를 수행함

- 그림 3-8은 K-BERT의 모델 구조를 보이는 것으로, 지식 그래프로부터 입력 문장에 대한 적절한 지식 그래프의 일부를 가져와 이를 추가적인 입력 시퀀스로 생성하여 학습하는 과정을 보임

■ 단일 입력 시퀀스로의 처리를 위한 눈에 보이는 행렬

- 여기에서 문제점은 BERT의 학습 과정에서 입력 시퀀스는 토큰들의 나열인데 추가적인 시퀀스를 제공하게 됨에 따라 이것이 그림에서 보이는 바와 같은 문장 트리(sentence tree)를 구성하게 됨

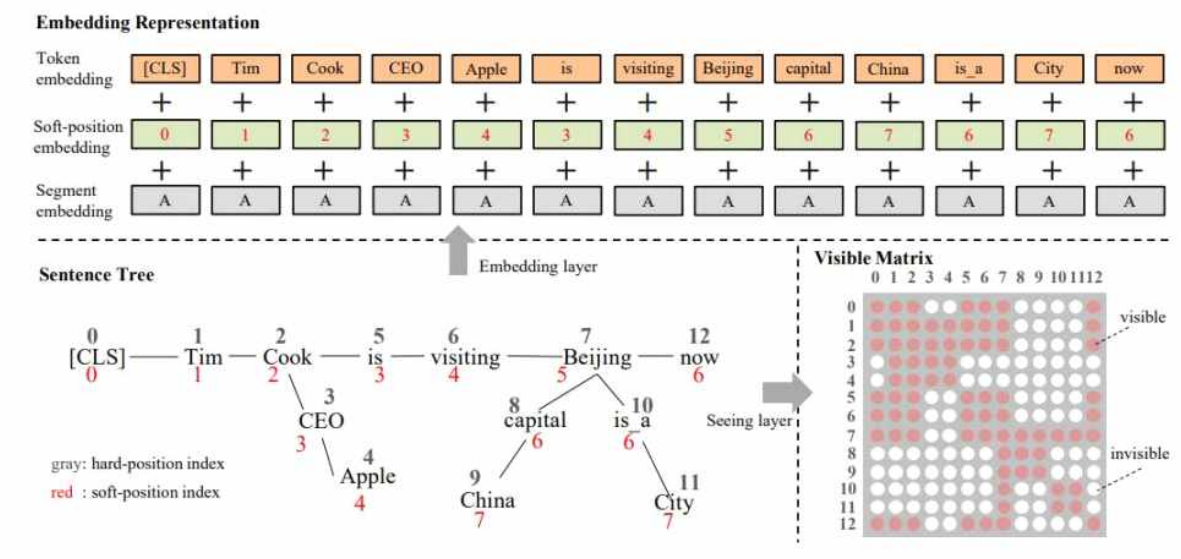


[그림 3-8] K-BERT의 모델 구조 [Liu2020c]

- 이에 따라 K-BERT에서는 그림 3-9와 같이 1D 형태로 시퀀스 화를 수행하는데, 기본 원리는 지식 그래프에서 제공된 새로운 시퀀스들의 토큰들을 기존 토큰 시퀀스 군데군데 삽입하는 것임. 그림 3-9에서 보면 Cook - CEO- Apple이라는 지식 정보가 원래의 입력 시퀀스에 삽입된 것을 확인할 수 있음
- 이때 문제점은 새로 입력된, 지식 그래프로부터 제공된 시퀀스 토큰들에 대해서도 동일하게 어텐션 점수를 계산하게 된다는 점임. 이를 해결하기 위해 K-BERT에서는 Visible 행렬이라는 것을 두어 이렇게 새로 삽입된 토큰에 대해서는 어텐션 점수를 구하지 않도록, 즉 보이지 않도록 함
- 이를 통해 모델의 변경을 최소화할 수 있는 장점을 가짐

■ 제한된 분야에서의 활용 가능성을 보임

- 표 3-4에서 보이듯 K-BERT는 특정 분야에 대해서 성능 측정을 주로 하였으며, 아주 큰 성능 향상을 보이지는 못하였음
- 지식 그래프를 활용하여 유의미한 성능 차이를 보일 수 있는 일부 분야에



[그림 3-9] K-BERT에서의 입력 시퀀스 처리 [Liu2020c]

대해서만 일단 효과적인 판별이 될 것임

- 주어진 입력 시퀀스에 대해 지식 그래프에서 적절한 지식을 추출하는 것에 대한 논의가 이뤄지지 않았으며, 이에 대한 고려가 필요함

[표 3-4] K-BERT의 성능 비교 [Liu2020c]

Table 1: Results of various models on sentence classification tasks on open-domain tasks (*Acc. %*)

Models\Datasets	**Book_review**		**Chnsenticorp**		**Shopping**		**Weibo**		**XNLI**		**LCQMC**	
	Dev	*Test*	*Dev*	*Test*	*Dev*	*Test*	*Dev*	*Test*	*Dev*	*Test*	*Dev*	*Test*
Pre-trainied on WikiZh by Google.												
Google BERT	88.3	**87.5**	93.3	94.3	96.7	96.3	98.2	98.3	76.0	75.4	88.4	86.2
K-BERT (HowNet)	88.6	87.2	**94.6**	**95.6**	**97.1**	**97.0**	98.3	98.3	**76.8**	**76.1**	**88.9**	86.9
K-BERT (CN-DBpedia)	**88.6**	87.3	93.9	95.3	96.6	96.5	98.3	98.3	76.5	76.0	88.6	**87.0**
Pre-trained on WikiZh and WebtextZh by us.												
Our BERT	**88.6**	87.9	94.8	95.7	96.9	**97.1**	98.2	98.2	77.0	76.3	89.0	86.7
K-BERT (HowNet)	88.5	87.4	**95.4**	95.6	96.9	96.9	98.3	**98.4**	**77.2**	**77.0**	**89.2**	**87.1**
K-BERT (CN-DBpedia)	88.8	87.9	95.0	**95.8**	**97.1**	97.0	98.3	98.3	76.2	75.9	89.0	86.9

□ KnowBERT [Peters2019a]

○ 기존 BERT에의 외부 지식 결합 방법 제안

■ BERT와 같은 언어 모델은 실세계 개체에 대한 사실 정보를 확인하기 어려움

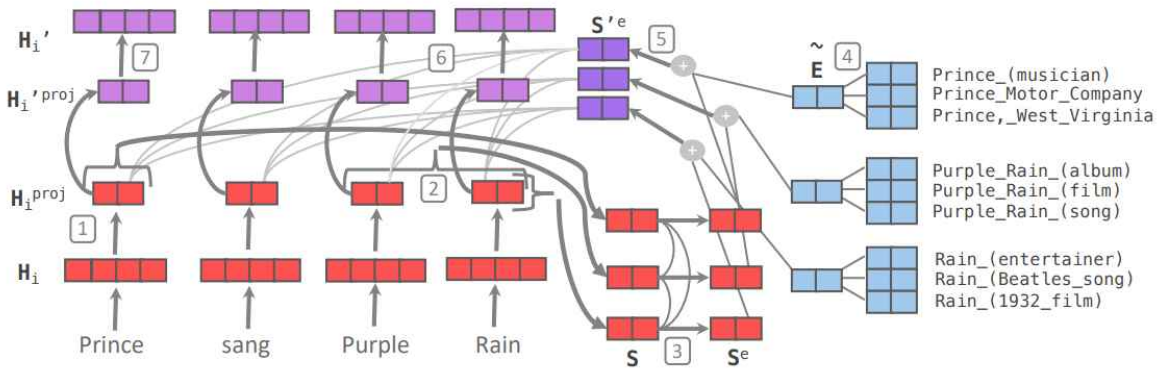
■ 지식 베이스(knowledge base)를 활용하면 원문에서 적은 빈도로 등장하거나 긴 범위 의존성(long range dependency) 문제로 인해 학습이 곤란한 정보까지 같이 학습 할 수 있을 것임

■ 이를 위해 KAR(knowledge attention & recontextualization)라는 기법을 제안하였으며, BERT-Base 모델에 Wikipedia와 WordNet 정보를 결합하였을 때 Bert-Large 모델보다 더 나은 성능을 확인함

○ 지식 베이스의 활용

■ 지식 베이스에 있는 K개의 개체(entity)들은 E 차원의 임베딩 벡터로 표현되어 있음

- 지식 베이스는 주어진 문장에서 C 개의 개체 후보와 그들의 시작, 끝 위치를 찾아주는 개체 후보 선택기(Entity Candidate Selector) 모듈을 가지고 있고, 또한 각 개체가 어떤 타입 인지에 대한 사전 확률(prior)도 가지고 있다고 가정함. 개체 후보 선택기가 한 입력 시퀀스에 대해 제공하는 개체 후보는 30 개로 제한함
- BERT는 트랜스포머의 인코더 모듈이 여러 계층을 두고 쌓아두는 구조인데, KnowBERT는 이 인코더 모듈 사이에 지식 정보 활용을 위한 추가적인 계층을 둠 (그림 3-10 참조)
- 그림 3-10에서 빨간색으로 표기된 입력 토큰들이 오면, 개체 후보 선택기가 제공하는 개체 목록과 각 개체들의 span 정보를 가지고 이 토큰들에 대한 은닉 벡터들을 생성 (그림의 S에 해당)
- 각 개체의 span 안에 있는 단어들에 대한 은닉 벡터들을 셀프 어텐션 스팬 풀링을 이용하여 각 개체 당 하나의 은닉 벡터를 선택
- 여기에 지식 베이스의 개체 임베딩을 반영한 새로운 개체-스팬 표현을 만들어내고, 이들과 기존 은닉 벡터들을 대상으로 어텐션을 수행함
- 지식 베이스에서 제공하는 개체 임베딩은 초기에 한번 학습된 후 고정되는 값임, 언어 모델에서의 MLM 손실 값과 개체 연결기(entity linker)의 손실 값을 더하여 학습을 수행함. 개체 연결기는 지식 베이스에서 어떤 개체 후보를 이용할지 구분하는 역할을 함



[그림 3-10] KnowBERT의 KAR 구조 [Peters2019a]

□ ERNIE [Zhang2019a]

○ 지식 정보를 개체 임베딩으로 입력하여 처리

■ ERNIE 또한 KnowBERT와 유사하게 개체 임베딩을 언어 모델 학습에 활용하기 위한 목적을 고안됨

■ 개체 임베딩을 처리하기 위한 BERT 모델의 확장

- 그림 3-11에서 보이는 바와 같이 BERT 모델을 확장하여 T-Encoder 모듈과 K-Encoder 계층이 교차하면서 쌓이는 구조임

- T-Encoder는 BERT 모델에서의 기본적인 트랜스포머 구조와 완전히 동일하며, K-Encoder는 개체 임베딩을 처리하도록 새로 고안된 모듈임. 즉 기존 BERT 모델의 각 인코더 계층 사이에 K-Encoder를 껴 넣은 구조임

- K-Encoder는 T-Encoder에서 처리한 각 토큰에 대한 은닉 벡터를 그대로 받고, 여기에 개체 임베딩에 대한 멀티 헤드 어텐션을 수행하여 이를 토큰의 은닉 벡터와 합치는 작업을 수행함

○ 사전 학습 태스크의 변경

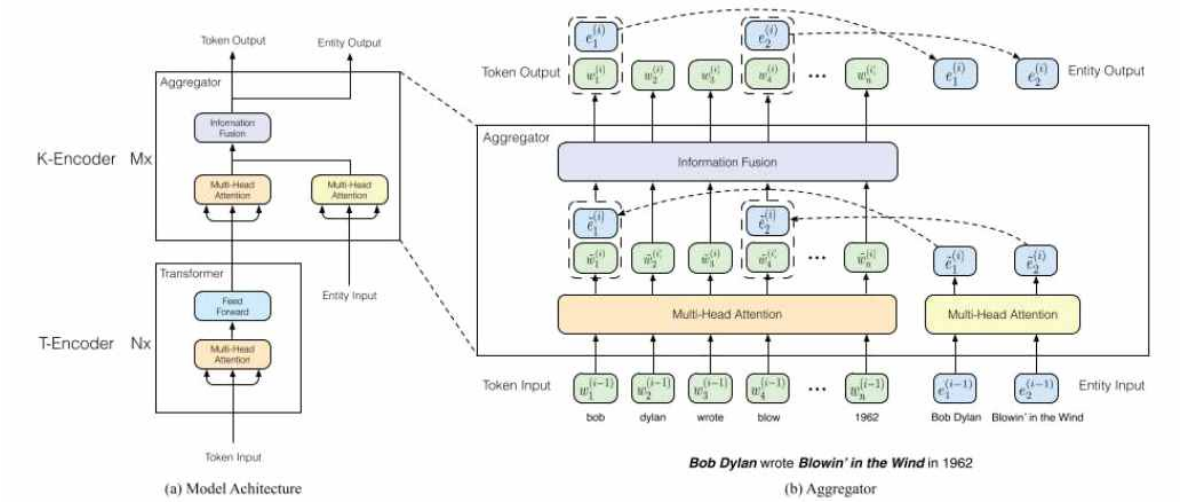
■ ERNIE는 BERT의 사전 학습 태스크인 마스크된 토큰 예측과 다음 문장 예측 이외에 디노이징 개체 오토 인코더(dEA; Denising Entity AUto-Encoder) 태스크를 추가하여 활용

- 마스크된 토큰 예측과 다음 문장 예측 태스크는 BERT의 경우와 완전히

동일한 태스크임

■ 디노이징 개체 오토 인코더 태스크는 개체 임베딩을 입력으로 하는 학습 방법으로 학습 데이터에 노이즈를 추가함으로써 모델을 좀 더 견고하게 만들기 위한 방법으로 아래와 같은 방식을 취함

- 5%의 개체를 임의로 다른 개체로 변경
- 15%의 토큰에 대해 해당하는 개체가 있어서 없애버림
- 80% 토큰에 대해서는 해당하는 개체를 같이 학습하게 함



[그림 3-11] ERNIE 모델 구조 [Zhang2019a]

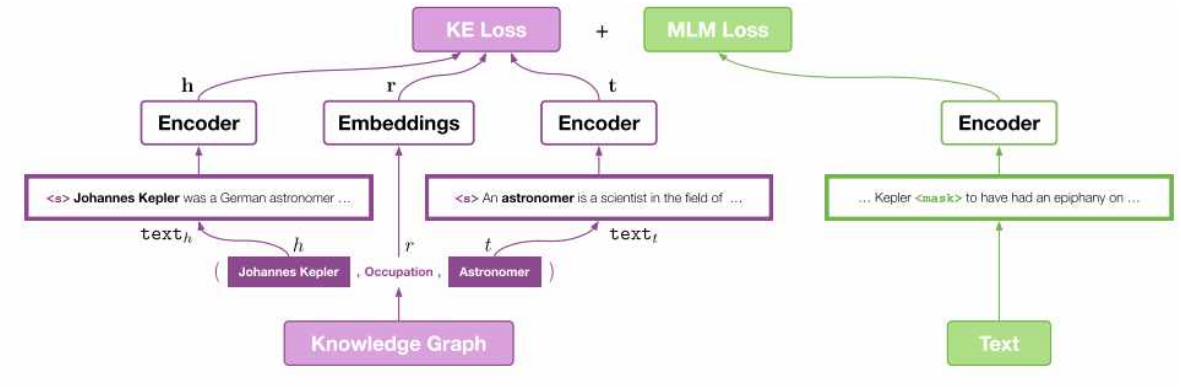
□ KEPLER [Wang2021b]

○ 사전 학습 언어 모델과 지식 임베딩의 결합

■ 사전 학습 언어 모델은 사실 지식(factual knowledge)을 텍스트로부터 잘 얻지 못하는 반면, 지식 임베딩(KE; Knowledge Embedding) 기법들은 지식 그래프로부터의 사실 지식을 잘 내포하고 있음. 그러나 지식 임베딩은 텍스트에 내포하는 지식을 확보하지는 못함

■ KEPLER는 지식 임베딩과 언어 모델을 결합하는 방법을 통해 이를 해결하고자 함

○ 지식 임베딩과 언어 모델 학습의 손실 함수를 결합하여 학습



[그림 3-12] KEPLER 모델 구조 [Wang2021b]

- 그림 3-12에서 보이는 바와 같이 KEPLER는 묵시적으로 사실 지식을 언어 표현에 포함시키기 위해 두 개의 손실 함수를 결합하여 활용 함
- 텍스트 인코더(그림 3-12의 오른쪽)는 기존의 트랜스포머 모델을 그대로 이용하며, 바이트쌍 인코딩(BPE)을 토크나이저로 하는 RoBERTa 학습 방법으로 사전 학습을 수행하며, 이 트랜스포머 구조에 아무런 변경을 가하지 않음
- 지식 그래프에 존재하는 개체들과 이들 간의 관계를 벡터화하기 위해 지식 임베딩 기술들을 도입함. 지식 임베딩 기술들은 다양하게 존재하는데, 이 지식 임베딩 벡터를 구하기 위한 손실함수와 마스크된 언어 모델의 손실 함수를 결합하게 같이 학습시키는 것이 KEPLER의 주된 아이디어임
- 즉, KEPLER에서는 ERNIE와 다르게 지식 임베딩이 미리 계산된 후 이를 저장하고 쓰는 것이 아니라, 사전 학습 과정 중에 같이 학습이 되는 점이 다름

[표 3-5] TACRED 벤치마크를 이용한 KEPLER와 다른 모델과의 성능 비교 [Wang2021b]

Model	P	R	F-1
BERT	67.2	64.8	66.0
$\text{BERT}_{\text{LARGE}}$	–	–	70.1
MTB	69.7	67.9	68.8
MTB ($\text{BERT}_{\text{LARGE}}$)	–	–	71.5
$\text{ERNIE}_{\text{BERT}}$	70.0	66.1	68.0
$\text{KnowBert}_{\text{BERT}}$	**73.5**	64.1	68.5
RoBERTa	70.4	71.1	70.7
$\text{ERNIE}_{\text{RoBERTa}}$	**73.5**	68.0	70.7
$\text{KnowBert}_{\text{RoBERTa}}$	71.9	69.9	70.9
Our RoBERTa	70.8	69.6	70.2
KEPLER-Wiki	71.5	**72.5**	**72.0**
KEPLER-WordNet	71.4	71.3	71.3
KEPLER-W+W	71.1	72.0	71.5
KEPLER-Rel	71.3	70.9	71.1
KEPLER-Cond	72.1	70.7	71.4
KEPLER-OnlyDesc	72.3	69.1	70.7
KEPLER-KE	63.5	60.5	62.0

■ 표 3-5는 TACRED 벤치마크를 이용하여 KEPLER와 다른 언어 모델을 비교한 것으로 Wikipedia를 지식그래프로 사용한 KEPLER 모델이 F-1 척도 기준으로 타 모델 대비 나음을 보임

□ REALM [Guu2020]

○ 잠재 지식 검색(latent knowledge retriever)을 이용한 언어 모델의 사전 학습 강화

■ BERT와 같은 마스킹 기반 언어 모델(MLM; Masked Language Model)은 코퍼스로부터 묵시적으로 학습된 지식을 가짐. 1 장에서 보인 바와 같이 언어 모델의 크기를 키울수록 더 많은 묵시적인 지식(implicit knowledge)을 학습할 가능성은 커짐

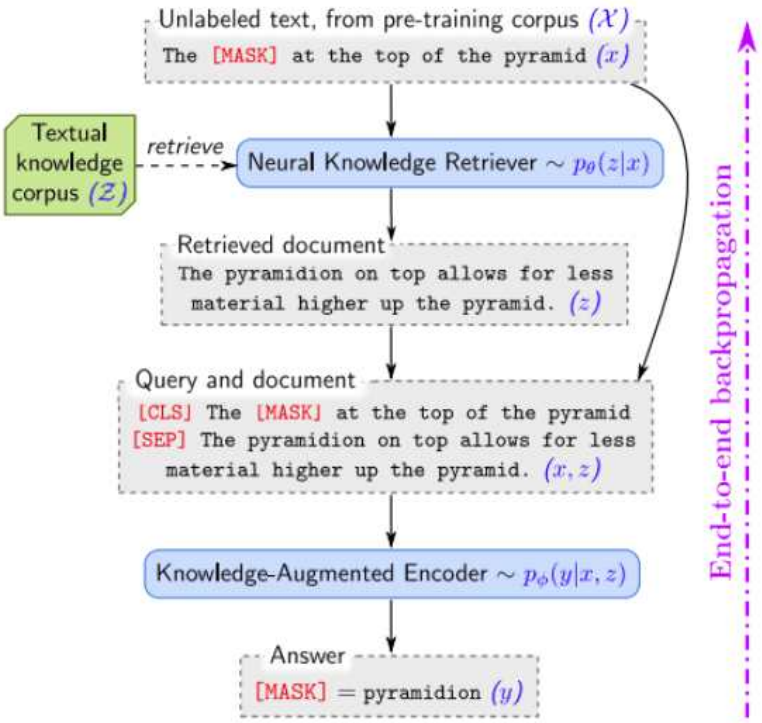
■ 그러나 이렇게 언어 모델의 크기를 키우는 것은 메모리나 연산량 면에서 보면 매우 비효율적임

■ 제안하는 방법은 언어 모델이 추론 중에 어떤 지식을 검색하고 사용할지 결정하도록 언어 모델에 요청하여 실세계 지식의 역할을 명시적으로 노출함

- 마스킹 기반 언어 모델을 사용하고 수백만 개의 문서로부터 검색 단계를 통해 잠재 지식 검색을 비지도 학습 방식으로 학습되게 함
- 예측 전에 언어 모델은 잠재 지식 검색을 통해 Wikipedia 같은 대량의 코퍼스에서 문서를 검색하고, 해당 문서를 탐색하여 예측을 수행함 (그림 3-13 참조)
- 검색을 통해 얻어진 문서가 언어 모델의 perplexity를 개선하는 경우 보상을 받으며, 그렇지 않을 경우 페널티를 받게 됨.
- 그림 3-13에서 “The ___ at the top of the pyraid”의 빈칸을 채워야 하는 경우 잠재 지식 검색이 “ The Pyramidion on top allows for less material higher up the pyramid”라는 문서를 선택하면 보상을 받게 됨

■ 비동기적인 업데이트

- 사전 학습 동안에 대규모의 뉴럴 지식 검색 모듈을 학습하기 위해서는 수백만 개의 문서를 고려해야하고, 이 결정에 대해 역방향전파를 수행해야



[그림 3-13] REALM에서 언어 모델 사전 학습을 뉴럴 지식 검색으로 보강하는 방법 예시 [Guu2020]

하므로 계산상 제약이 됨

- 이를 감소시키기 위해 각 문서에 대해 수행된 계산을 캐싱하고 비동기적으로 갱신할 수 있도록 뉴럴 지식 검색기를 구현함

■ Retrieve-then-Predict 전략

- 제안하는 방법의 구현을 위해 REALM은 크게 뉴럴 지식 탐색기(neural knowledge retriever)와 지식 보강 인코더(Knowledge Augmented Encoder)를 가지며, 이를 하나의 수식으로 표현하면 아래와 같음

$$p(y|x) = \sum_{z \in Z} p(y|z,x)p(z|x) \qquad \text{(식 3-1)}$$

- 그림 3-13에서 Z는 검색될 수 있는 후보 문서 그룹이고, x와 y는 각각 마스킹된 입력과 마스킹된 토큰의 원래 값임. 즉 주어진 입력 x 에 대해 관련된 문서를 찾고($p(z|x)$), 검색된 문서를 고려해서 마스킹된 토큰을 예측($p(y|z,x)$)을 하는 목적 함수를 가짐

- 뉴럴 지식 검색은 문서와 질의를 BERT 모델의 입력으로 하고, 이 둘 간의 유사도는 둘의 dot-product로 계산함. 전체 검색 후보 중에서 실제 관련이 있는 문서가 가장 높은 점수를 갖도록 크로스 엔트로피를 함수로 학습
- 사전 학습 시 모든 문서들에 대해 점수를 계산하는 것은 연산량 면에서 사실상 불가능하므로, top-k 개의 후보군에 대해서만 점수 계산을 수행
- 마스킹된 입력 x와 관련이 있는 문서를 가지고 마스킹된 토큰을 예측 함. 이를 위해 단순히 [SEP]토큰으로 마스킹 된 입력과 관련 문서의 텍스트를 연결하여 시퀀스를 구성하고, 기존 BERT와 같은 방식으로 학습 진행

■ T5 대비 상대적으로 적은 크기로 좋은 성능을 보임

- 표 3-6은 Open-Q&A 데이터 셋에 대해서 REALM과 BERT, T5 및 기존 2단계 기반 베이스라인 모델들과 성능 비교를 수행한 결과임
- 기존 기법 대비 큰 성능 향상을 보여 주었으며 동일한 Dense Retrieval + Transformer 접근을 사용한 ORQA보다도 약 7% 높은 성능 향상을 보임
- 또한 30배 이상 큰 T5모델과 비교해도 Q&A 태스크에서 6% 높은 성능을 보임

[표 3-6] REALM의 성능 비교(NQ: NaturalQuestions-Open, WQ: WebQuestions, CT: CuratedTrec)

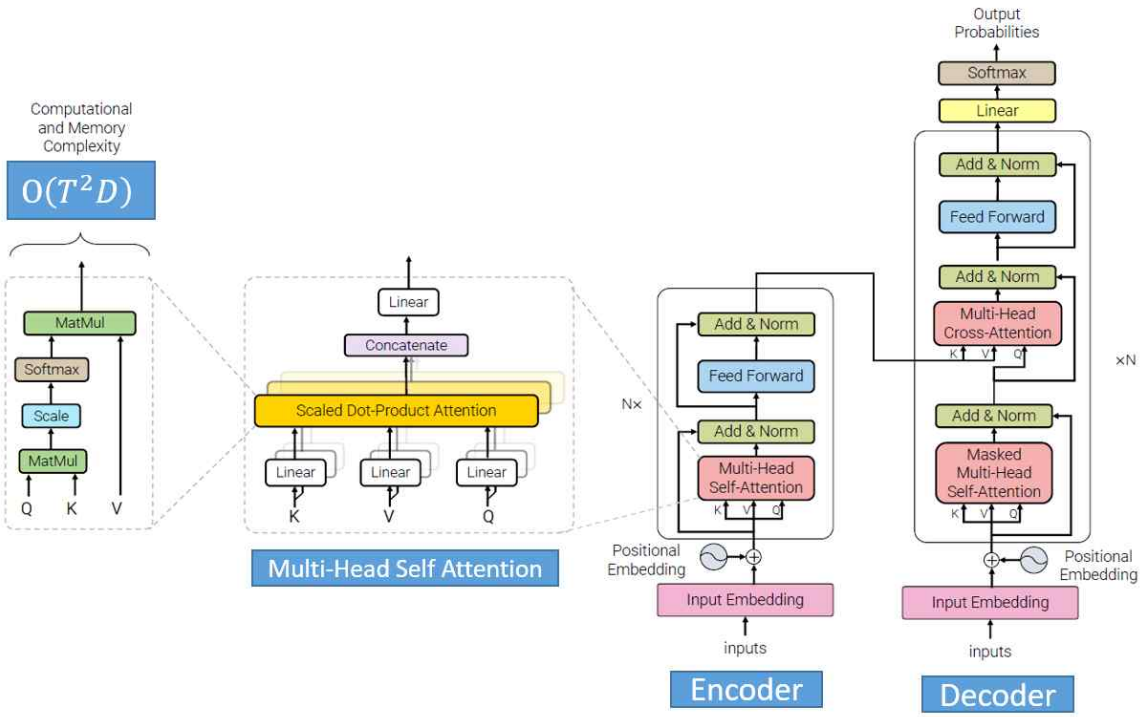
Name	Architectures	Pre-training	NQ (79k/4k)	WQ (3k/2k)	CT (1k /1k)	# params
BERT-Baseline (Lee et al., 2019)	Sparse Retr.+Transformer	BERT	26.5	17.7	21.3	110m
T5 (base) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	27.0	29.1	-	223m
T5 (large) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	29.8	32.2	-	738m
T5 (11b) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	34.5	37.4	-	11318m
DrQA (Chen et al., 2017)	Sparse Retr.+DocReader	N/A	-	20.7	25.7	34m
HardEM (Min et al., 2019a)	Sparse Retr.+Transformer	BERT	28.1	-	-	110m
GraphRetriever (Min et al., 2019b)	GraphRetriever+Transformer	BERT	31.8	31.6	-	110m
PathRetriever (Asai et al., 2019)	PathRetriever+Transformer	MLM	32.6	-	-	110m
ORQA (Lee et al., 2019)	Dense Retr.+Transformer	ICT+BERT	33.3	36.4	30.1	330m
Ours ($\mathcal{X}$ = Wikipedia, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	39.2	40.2	**46.8**	330m
Ours ($\mathcal{X}$ = CC-News, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	**40.4**	**40.7**	42.9	330m

제4장

언어 모델의 효율성 개선

제 1 절 개요

1. 트랜스포머 모델 구조개요



[그림 4-1] 트랜스포머의 기본 구조 [Lin2021]

□ 트랜스포머 기반 언어 모델

○ 최근 언어모델은 그림 4-1에서 보이는 바와 같이 트랜스포머 모델 구조를 기반으로 인코더 또는 디코더 모듈을 여러 계층으로 쌓아올려 최적의 초매개변수로 구성한 구조를 지님. 즉, 그림 4-1에서의 어텐션 계층이 반복되는 구조를 가짐

○ 이는 [표 2-2]과 같이 GPT-3의 다양한 버전의 예시를 통해서도 확인할 수 있음

■ 이에 효율적인 연산을 위해 트랜스포머 구조 내에 반복적으로 존재하는 모듈의 각 구성 요소들에 대한 여러 변형들이 제시되어 왔음.

■ 이들 중 특히, 어텐션 모듈에 대한 변형 제안이 다수를 차지함

2. 트랜스포머 모델 변형의 분류

□ [그림 4-2]는 트랜스포머 구조를 구성하는 각 모듈에 대한 변형 방법들을 나열

○ 위치별 FFN(Position-wise) FFN 모듈, 계층 정규화(Layer Normalization) 모듈, 어텐션 모듈 및 위치 인코딩(positional encoding) 모듈들에 대한 변형 방법들이 제시되어 왔음

○ 이 장에서는 계층 정규화 모듈과 위치 인코딩의 변형에 대해서는 2 절에서 기술하며, 특히 앞서 연산에 있어 병목이 되는 FFN 모듈과 어텐션 모듈에 대해서는 3절과 4절에서 주목하여 기술함

제 2 절 계층 정규화와 위치 인코딩의 개선

1. 계층 정규화의 변형

□ 계층 정규화 사용의 이유

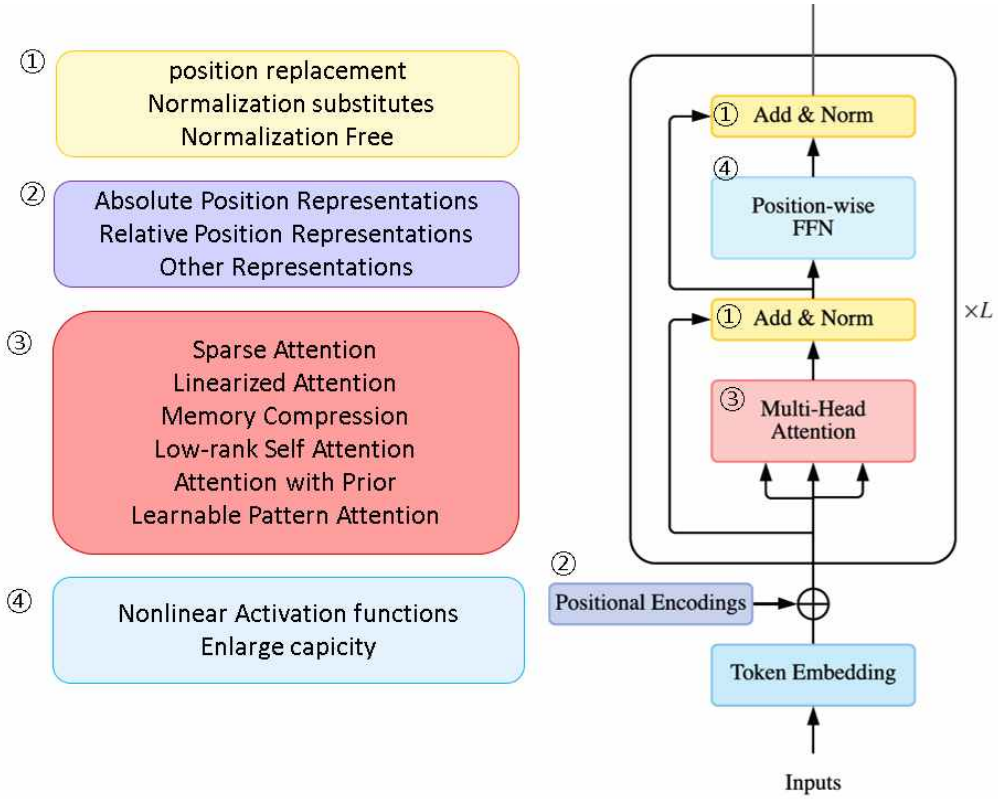
○ 트랜스포머(Transformer)에서는 layer normalization은 학습 도중에 발생한 불량 설정된 기울기 값(ill-posed gradients)들을 수정하는 역할을 함

■ 안정된 학습결과를 도출하기 위하여 필요한 계층이다.

■ 컴퓨터 비전 영역에서 주로 사용하는 CNN(Convolutional Neural Network_에서는 배치 정규화(batch normalization)를 주로 사용하는데, 이는 학습에 사용하는 이미지 데이터들은 동일한 크기들을 가지고 있어, 배치 크기가 모두 일정하게 계산됨에 따라 배치 정규화를 통해 학습 불안전성을 해소할 수 있기 때문임

■ 그러나 문장 시퀀스의 길이가 일정하지 않은 텍스트 처리를 위해 사용하는

트랜스포머 모델에서는 배치 정규화를 수행하는 경우 성능 저하가 있다는 보고가 있음. 때문에, 트랜스포머 기반 언어 처리 모델에서는 배치 정규화 대신 계층 정규화를 주로 사용함



[그림 4-2] 트랜스포머 모델 변형의 분류

□ 사전과 사후 계층 정규화

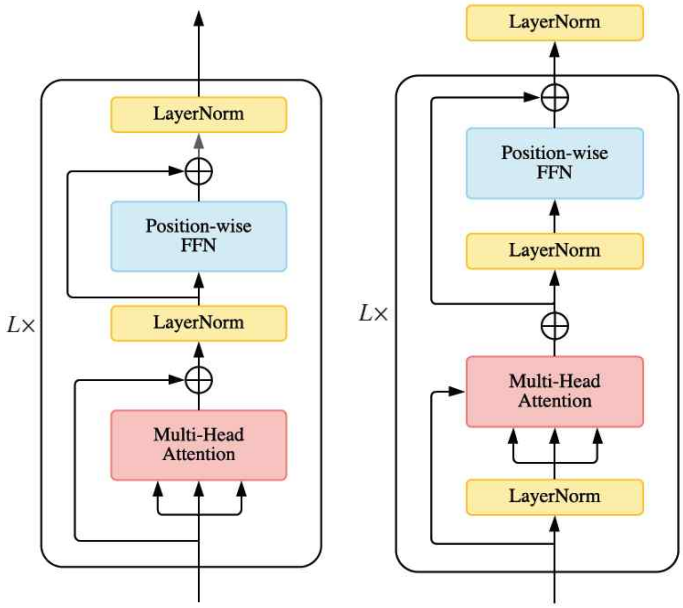
○ 계층 정규화는 현재까지 가장 널리 이용되는 정규화 기법 중 하나로 크게, 사후 계층정규화(Post-LN; Post Layer Normalization)와 사전 계층 정규화(Pre-LN; Pre Layer Normalization) 두 종류로 구분할 수 있음.

○ 사후 계층 정규화는 트랜스포머의 초기 모델에 제시된 것으로 멀티 헤드 어텐션(multi-head attention) 이후 계층 정규화를 수행함. 반면, ResNet의 사전 활성화(pre-activation) 기법과 유사하게 사전 계층 정규화는 멀티 헤드 어텐션 이전에 계층 정규화를 이후 FFN(Feed Forward Network) 이전에 계층 정규화 수행 후 잔차 연결을 합하는 연산 수행을 반복하고, 소프트맥스 연산 이전에 계층 정규화를 수행한다.

■ [그림 4-3]의 (a) post-LN, (b) pre-LN 참조

○ Xiong 등의 연구에 따르면, pre-LN에 비해 post-LN의 경우 학습 초기 출력 기울기 값이 크게 나오게 되어 학습의 불안정성을 초래하므로 학습율(learning rate)의 웜업(warm-up) 구간을 설정하는 것이 학습 시 안전성을 가져옴.

○ 이에 비해 웜업 구간 없이 pre-LN을 수행해도 안정적인 학습을 드러냄을 실험을 통해 입증함 [Xiong2020]



(a) post-LN (b) pre-LN

[그림 4-3] 계층 정규화의 두 대표 모델 [Lin2021]

○ Liu 등의 연구에 따르면, post-LN의 초기 학습 불안정성의 원인은 초기에 잔차 연결에 대한 의존성이 커져 증폭효과가 있음으로 확인됨

○ 이러한 분석을 바탕으로, 학습안정화를 위해 post-LN의 residual branch에 대한 의존도를 낮추고자 parameter들을 추가함. 실험을 통해 추가 parameter와 함께 안정적인 학습과 pre-LN에 비해 미세하게 더 좋은 성능을 보임을 입증함 [Liu2020a]

[표 4-1] 언어 번역에서 Post-LN과 Pre-LN의 계층 크기별 BLUE 스코어

Dataset	IWSLT'14 De-EN	WMT' 14 En-Fr		WMT' 14 En-De		
#Enc-#Dec	6L-6L (small)	6L-6L	6L-12L	6L-6L	12L-12L	18L-18L
Post-LN	35.64±0.23	41.29	failed	27.80	failed	failed
Pre-LN	35.50±0.04	40.74	43.10	27.27	28.26	28.38

○ Bachilechner 등의 ReZero 연구에서는 identity 연산을 위해 각 계층을 초기화

■ 각 계층에서 입력 신호 x에 대해 잔차 연결을 설정하는데, 이 때 유일한 학습 파라미터 α 는 계층 $F(X)$의 변형을 조절하는데 이용

■ 학습이 시작될 때 $\alpha = 0$으로 설정. 초기에 F를 정의하는 모든 파라미터들은 사라지지만, 초기 학습 단계에서 적절한 값으로 동적으로 변형 [Bachilechner2021]

$$x_{i+1} = x_i + \alpha_i F(x_i) \quad \text{(식 4-1)}$$

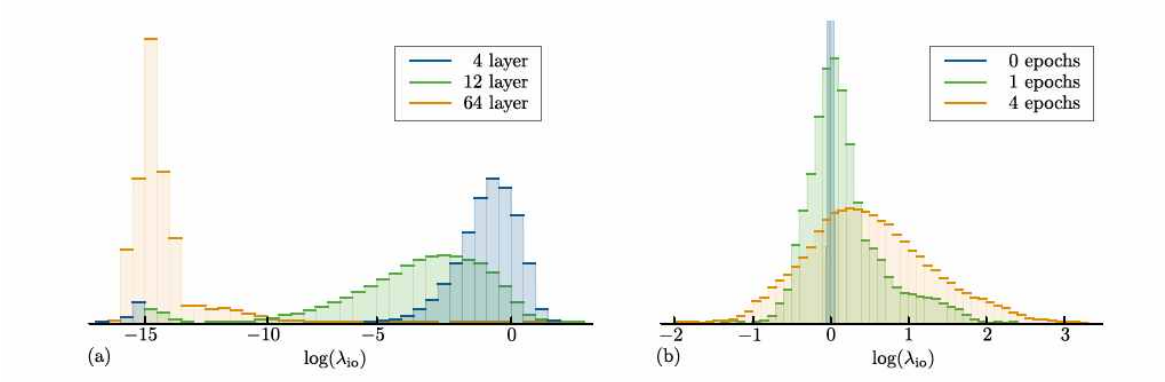
[표 4-2] 딥 뉴럴 네트워크에서 널리 이용되는 구조

	모델	수식
1	Deep Network	$x_{i+1} = F(x_i)$
2	Residual Network	$x_{i+1} = x_i + F(x_i)$
3	Deep Network + Norm	$x_{i+1} = Norm(F(x_i))$
4	Residual Network+ Pre-Norm	$x_{i+1} = x_i + F(Norm(x_i))$
5	Residual Network+ Post-Norm	$x_{i+1} = Norm(x_i + F(x_i))$
6	**ReZero**	$x_{i+1} = x_i + \alpha_i F(x_i)$

■ 기존의 딥 뉴럴 네트워크로 구성된 트랜스포머 인코더의 경우 깊은 계층(64계층)으로 구성하고 학습 시, 학습 초기와 학습 도중의 입출력 야코비안 행렬(Jacobian matrix)의 singular value의 log값($\log(\lambda_{io})$)에 대한 히스토그램을 확인하면 그 분산(variance)이 매우 크고 평균이 일정하지 않아 정규화 과정이 필요함을 확인 가능

■ ReZero를 통해 학습한 모델의 경우 ($\log(\lambda_{io})$)값의 평균 singular value=1이어서 정규화 과정이 불필요함을 확인할 수 있음. [그림 4-4] 참조.

■ [그림 4-4]에서 (a) Transformer encoder network으로 그 깊이가 4, 12, 64계층의 초기 값. (b) ReZero Transformer Encoder network의 64계층의 학습



[그림 4-4] 학습 초기와 학습 도중의 입출력 야코비안 행렬의 singular value의 log값($\log(\lambda_{io})$) [Bachilechner2021]

초기와 학습 과정중의 $\log(\lambda_{io})$. Rezero는 평균 singular value $\lambda_{io} \approx 1$ 로 dynamical isometry에 가까움.

- 이러한 근거로 ReZero를 활용해 빠른 수렴(converge)을 통해 연산 시간을 단축할 수 있고, 깊은 계층(64 계층 이상)으로 갈 때 잘 수렴한다는 장점을 지닌 뉴럴 네트워크 구조[Bachilechner2021]
- Liu의 연구에서는 [표 4-3]와 같이 6 계층을 가지고 pre-LN 및 post-LN으로 정규화 수행 시, ReZero에 비해 더 나은 성능을 보임을 확인. 가장 널리 활용되는 12계층~24계층에서 두 구조에 대한 장기간 학습 시 성능 비교가 아직 공개되지 않음[Liu2020a, Bachilechner2021]

[표 4-3] ReZero 성능(6 계층의 인코더와 디코더로 구성)

Models	Admin	Post-LN	Pre-LN	ReZero	ReZero+Post-LN
BLEU	35.67±0.15	35.64±0.23	35.50±0.04	33.67±0.14	34.67±0.08

2. 위치 인코딩의 변형

□ 계층 정규트랜스포머 모델은 일반적인 컨볼루션 네트워크나 순환 네트워크(recurrent network)와는 달리 입력 시퀀스에서 위치 정보를 같이 활용하는 모델

○ 이에 따라 단어들의 순서가 중요함. 입력 시퀀스를 구성하여 트랜스포머 모델의 입력으로 하기 위해 위치 정보를 벡터 값으로 표현하고, 이를 토큰 임베딩과 합하는 형태로 구성함.

○ 이렇게 위치 정보 값을 벡터화 하는 과정을 위치 인코딩(positional encoding)이라 함

○ 위치 인코딩(Positional Encoding)의 방법으로는 절대 위치 정보(absolute position), 상대적 위치 정보(relative position) 표현 방법이 있음

□ 절대 위치 정보 표현

○ 처음 제안된 트랜스포머 모델[Vaswani2017a]에서는 단어(token)의 문장 내 위치 정보를 주기 함수(sinusoidal)를 이용하여 인코딩하였음.

■ 각 위치 인덱스 t에 대해, 입출력 토큰을 d_{model}차원의 인코딩 벡터 $p_t = PE(t) \in R^{d_{model}}$로 표현. 이 벡터는 미리 정의된 주파수와 인덱스로 표현되는 sinusoidal(sin/cos) 함수를 이용하여 생성

$$PE(t)_i = \begin{cases} \sin(w_i t) \text{ if } i \in even, \\ \cos(w_i t) \text{ if } i \in odd, \end{cases} \quad \text{(식 4-2)}$$

여기에서 w_i 는 차원 크기에 따라 수동 정의된 주파수임.

초기 트랜스포머 모델[Vaswani2017a]의 경우 $w_i = 1/10000^{\frac{i}{d_{model}}}$ 로 설정함

○ Wang 등은 식 4-2와 같은 절대 위치 정보 표현 방식을 이용하되, w_i 값의 설정을 입력 데이터를 기준으로 인코딩하는 방식을 제안함 [Wang2021]

○ Floatter 등은 연속 동적 시스템으로 위치 정보를 표현하고 Neural ODE 모델(ResNet의 연속 시퀀스 버전)을 차용하여 역전파(back propagation)를 이용한 end-to-end 학습 방식을 사용함. 이를 통해 입력 시퀀스 길이의 제한을 두는 위치 임베딩 방식의 제약성도 넘어서고, NMT, NLU 등의 태스크에서 비교 시스템들보다 나은 성능을 입증함 [Liu2020b].

○ AL-Rfoud 등은 깊은 계층(64계층 이상)의 문자 단위의 언어모델 학습 성능 향

상을 위해 중간 시퀀스 위치를 중간 계층에 추가하여 손실 값을 계산하는 형태를 취함[AL-Rfoud2019]

○ Dehghani도 작은 데이터 집합에 대해 유연하게 동작하는 언어모델을 제안하기 위하여 같은 방법으로 중간 시퀀스 위치를 중간계층에 추가하는 형태를 취함[Dehghani2019]

□ 상대적 위치 정보 표현

○ 개별 토큰들의 절대적 위치 값을 사용하는 대신, 토큰 간의 상대적 위치를 표현하는 방법

■ 기본 아이디어는 각 토큰의 절대 거리 계산보다는 입력되는 토큰 쌍 간의 상대적인 방향과 거리 계산이 셀프 어텐션 학습에 있어 더 중요하다는 관점

■ Shaw 등은 어텐션 기법에서 키값 학습 가능한 상대적 위치 임베딩 개념을 추가함. 식 4-3의 어텐션 연산에 입력 요소 간 간선 a_{ij}^{K}에 대한 개념을 추가하여 식 4-4와 같이 변경함 [Shaw2018]

$$Attention(Q_i, K_j, V_j) = softmax(\frac{Q_i K_j^T}{\sqrt{d_k}}) V_j \quad \text{(식 4-3)}$$

$$Attention(Q_i, K_j, V_j) = softmax(\frac{Q_i K_j^T + a_{ij}^K}{\sqrt{d_k}})(V_j + a_{ij}^V) \quad \text{(식 4-4)}$$

■ 이때 상대적 위치 정보의 최댓값인 k는 가장 긴 입력 시퀀스의 길이로 설정. 입력 요소에 대해 $2k+1$의 유일한 간선 레이블들만을 고려함. 최대 거리를 clipping함으로써 학습 기간 동안 연산되지 않은 시퀀스 길이까지 모델을 일반화함

$$a_{ij}^K = w_{clip(j-i,k)}^K,\ a_{ij}^V = w_{clip(j-i,k)}^V \quad \text{(식 4-5)}$$

$$clip(x,k) = \max(-k, \min(x,k)), \quad \text{(식 4-6)}$$

- 이를 통해 어텐션 키와 값에 대한 가중치에 대해 $2K+1$의 상대적 위치(Relative position)가 학습됨. $w^K = \{w_{-k}^K, ..., w_k^K\}$ $w^V = \{w_{-k}^V, ..., w_k^V\}$

■ InDIGO는 문장 생성에 있어 그 순서를 왼쪽에서 오른쪽으로 생성하던 모델

들과 달리 문장 입력에 따라 생성 순서에 자율을 둔 non-auto regressive 생성 모델임. InDIGO에서는 K값을 3으로 둠 [Gu2019]

- Music Transformer는 상대적 위치 정보를 이용하여 중간 메모리 요구량을 줄이는 방법을 제안 [Huang2019]
- Raffel 등은 단순버전의 상대적 위치 정보 임베딩을 어텐션 가중치 계산을 위해 이용되는 대응 점수에 추가된 학습 가능한 스칼라 값으로 간주 [Raffel2020]
- Transformer-XL에서는 어텐션 점수 계산을 아래와 같이 재설계함. 기존 트랜스포머 모델은 주기함수를 이용한 위치 인코딩을 이용하였으나, contents 정보와 position 정보를 융합하여 어텐션 점수 계산을 함 [Dai2019]

$$A_{ij} = q_i k_j^T + q_i (Re_{i-j} W^{K,R})^T + u^1 k_j^T + u^2 (Re_{i-j} W^{K,R})^T, \qquad \text{(식 4-7)}$$

  - 식 4-7에서 행렬 $W^{K,R} \in R^{d_m \times d_k}$, 벡터 $u^1, u^2 \in R^{d_k}$은 학습 가능한 파라미터들이며, 일반적인 어텐션 계산과 달리 다음 변수들이 대체됨.
  - Re은 기존의 절대 위치 인코딩 행렬을 absolute position encoding matrix를 대신하는 상대적 위치 인코딩 행렬로 주기함수로 인코딩된 값들을 가짐
  - k_j^T의 가중치로 content 정보 행렬을 이용하고, $W^{K,R}$은 위치 정보를 나타내는 가중치.
  - u^1, u^2는 모든 위치에서 같은 값으로 대체되는 질의 벡터임
- DeBERTa[He2020]는 상대적 위치 정보 임베딩[Shaw2018]이나 Transformer-XL[Dai2019]과 유사한 disentangled 형식의 임베딩을 모델에 적용

$$A_{ij} = q_i k_j^T + q_i (\delta_{ij} W^{K,R})^T + k_j (\delta_{ij} W^{Q,R})^T, \qquad \text{(식 4-8)}$$

  - $W^{K,R}, W^{Q,R} \in R^{d_m \times d_k}$은 학습 가능한 파라미터이며, δ_{ij} 는 [Shaw2018]에서와 유사하게 토큰 i의 토큰 j에 대한 상대적 위치 정보 임베딩인데, 두 토큰 사이의 최대 거리 k에 대해 $\delta_{ij} \in [0, 2k]$이며, 이 값은 $\delta_{ij} = \begin{cases} 0 & , \text{if } i-j \le -k \\ 2k-1 & , \text{if } i-j \ge k \\ i-j+k & , others \end{cases}$ 에 따라 계산됨. $q_i k_j^T$은 콘텐트 간 어텐션

(content-to-content attention)이며, $q_i(r_{ij}W^{K,R})^T$은 콘텐트-위치정보 간 어텐션(content-to-position attention), $k_j(r_{ij}W^{Q,R})^T$은 위치정보-콘텐트 간 어텐션(position-to-content attention)임

제 3 절 어텐션 모듈의 효율성 개선

1. 어텐션 연산의 제약 및 개선

□ 일반적으로 Attention 모듈이 갖는 두 가지 제약점으로 (1) 연산의 복잡도와 (2) 적은 양의 데이터로 학습할 경우 모델이 과적합(over-fitting)기 쉬운 점이 있음

○ 셀프 어텐션은 구조적인 사전 정보(prior knowledge)의 부재로, 입력에 대한 어떤 bias도 가정하지 않고, 입력 데이터의 순서도 입력 데이터로부터 학습됨에 기인함

□ 이러한 문제를 해결하고자 [그림 4-2]에서 소개된 것과 같이 기존 멀티 헤드 어텐션 모듈에 대한 여러 개선책이 제안됨

○ 제안된 개선책들은 (1) 희소 어텐션(sparse attention), (2) 선형화된 어텐션(linearized attention), (3) 질의 프로토타이핑(query prototyping), (4) 메모리 압축(memory compression), (5) 저순위 셀프 어텐션(low rank self-attention), (6) 사전 분포를 이용한 어텐션(attention with prior), 그리고 (7) 학습 가능한 패턴 어텐션(learnable pattern attention) 방법으로 분류할 수 있다.

2. 어텐션 모듈의 개선 연구들

□ 어텐션 변형 모듈의 분류

○ 희소 어텐션(Sparse attention)

■ 계산 복잡도를 줄이기 위해 어텐션 계산에 sparsity bias를 적용

○ 선형화된 어텐션(linearized attention)

■ 계산 복잡도를 선형(linear)화하는 것으로 목표로 어텐션 연산을 커널 맵을

이용하여 재해석 후 역순으로 연산 풀이함

○ 질의 프로토타이핑(query prototyping)

■ 어텐션 행렬의 크기를 줄이기 위해 어텐션 질의의 수를 줄이는 방식

○ 메모리 압축(memory compression)

■ 여러 토큰들을 한 번에 접근하도록 추가적인 메모리 모듈을 활용하는 방법으로, 글로벌 메모리 개념과 풀링(pooling) 접근 방법이 있음

- 글로벌 메모리 접근 방법은 모든 입력 시퀀스에서 접근 가능한 방법이고, 풀링 접근 방법은 전체 시퀀스를 대표하는 토큰들을 컨텍스트로 활용하여 메모리에 두고 접근하는 방법
- ETC[Ainslie2020a]와 Longformer [Beltagy2020]가 메모리 압축 방식을 이용하는 방식임.

○ 저순위 셀프 어텐션(low-rank self-attention)

■ 셀프 어텐션 행렬에 저 순위 근사화 기법(low-rank approximation)을 적용하여 행렬 연산량을 줄임

- Linformer[Wang2020b]가 이에 해당

■ 셀프 어텐션 행렬을 커널화 하여 접근

- performer[Choromanski2020], Linear transformer[Katharopoulus2020]가 이에 해당

■ 사전 분포를 이용한 어텐션(attention with prior) : 입력 데이터의 사전 분포(prior distribution) 특징을 이용하여 어텐션 행렬의 보조 수단으로 이용하거나 이를 대체 연산

■ 학습 가능한 패턴 어텐션(learnable pattern attention)

- 입력 데이터에 따라 학습 가능한 패턴을 찾아가는 방식으로 토큰들의 상관관계를 보고 이를 클러스터 및 버켓에 할당하여 연산
- Reformer[Kitaev2020]는 해시 함수 기반의 유사도 측정으로 토큰들을 클러스터링하여 청크(chunk) 단위로 연산함.
- Sinkhorn Sorting Network[Tay2020a]은 입력 시퀀스 블록들을 정렬하여

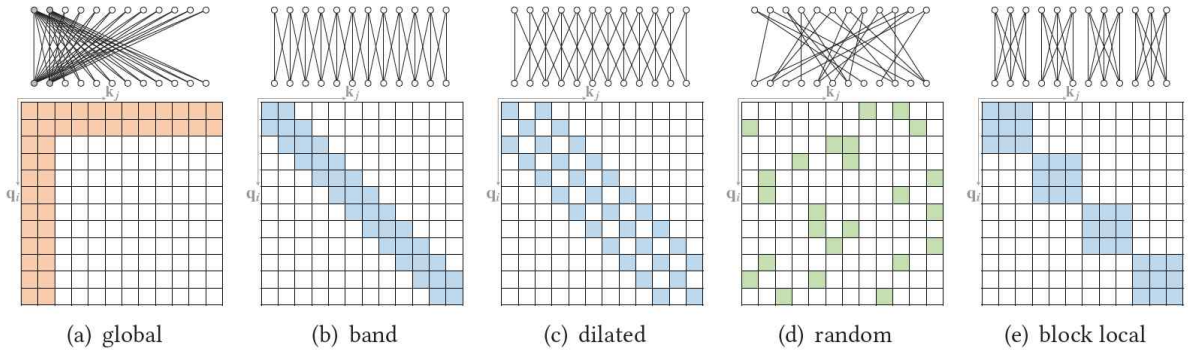
어텐션 가중치들의 희소성(sparsity)를 드러냄

□ 어텐션 모듈에 대한 분석

○ 희소 어텐션(Sparse Attention)에서는 미리 정의된 패턴의 질의, 키 쌍들에 대해서만 어텐션 연산을 수행함으로써 연산량을 줄임

■ 미리 정의된 패턴에 속하지 않은 질의 i, 키 j쌍의 어텐션은 $A_{ij} = -\infty$로 정의하고 메모리에 저장 안함.

○ 희소 어텐션 구현 모듈의 원소 패턴들로는 (a) 글로벌, (b) 밴드, (c) 확장, (d) 랜덤, (e) 블록 로컬 어텐션이 존재함(아래 [그림 4-5] 참조).



[그림 4-5] 희소 어텐션을 구성하는 대표적인 원소 패턴들의 행렬 [Lin2021]

■ 글로벌 어텐션(global attention): 희소 어텐션에서 긴 길이의 의존관계 표현의 부족을 보완하고자 노드들 사이의 정보 전달 허브 역할을 하는 글로벌 노드를 추가함. 글로벌 노드는 입력 시퀀스 내의 모든 노드들에 대해 어텐션 연산을 수행하고, 모든 시퀀스는 글로벌 노드에 대해 어텐션 수행. [그림 4-5]의 (a)참조

■ 밴드 어텐션(band attention): 밴드 어텐션은 슬라이딩 윈도우 어텐션 또는 로컬 어텐션으로 불림. 대부분의 언어 시퀀스 내의 노드들이 전후 노드에 큰 영향을 받는 성질을 기반으로 어텐션 연산을 수행하는 노드를 주변 노드들로 제한을 두는 방법임. [그림 4-5]의 (b)참조

■ 확장 어텐션(dilated attention): dilated CNN(Convolutional Neural

Network)에서의 방법과 유사하게 밴드 어텐션에 확장 차(dilation gap)을 지닌 확장 윈도우(dilated window) 혹은 스트라이드 된 윈도우(strided window)를 통해 연산 복잡도를 줄이면서 수용 영역(receptive field)을 확대함. [그림 4-5]의 (c) 참조

- 랜덤 어텐션(random attention): 비 지역 상호작용(non-local interaction)을 높일 수 있도록 각 질의에 대한 어텐션 간선을 임의로 선택하도록 하는 방법으로 [그림 4-5]의 (d)참조. 이는 랜덤 그래프가 그래프 내의 랜덤 워크로 빠른 처리 시간과 함께 완전 그래프와 유사한 특징을 보임을 기인함(Erdos-Renyi 랜덤 그래프)
- 블록 로컬 어텐션(block local attention): 이는 입력 시퀀스를 몇몇 비겹침 질의 블록(non-overlapping query block)들로 분리 후, 각각 대응되는 메모리 블록 키에 대해서만 어텐션 연산을 수행하는 방법. 이 때 각 질의 블록은 로컬 메모리 블록(local memory block)에 해당됨. [그림 4-5]의 (e)참조.

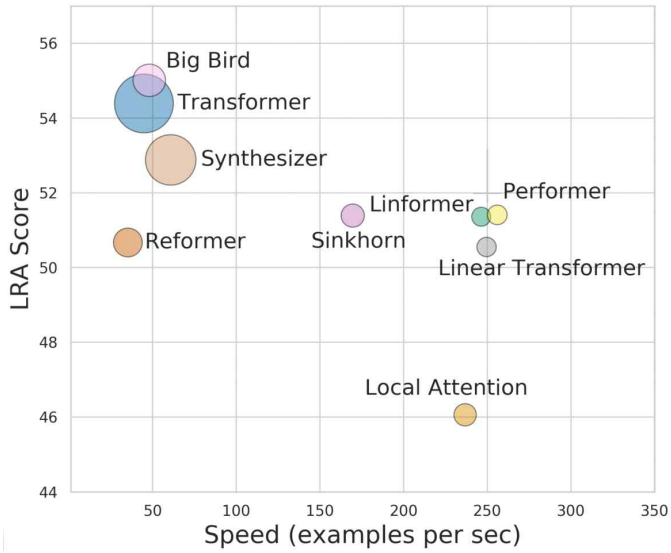
3. 어텐션 모듈 개선과 관련한 대표 논문들

□ 어텐션 모듈 개선 기법들에 대한 성능 비교

○ Tay 등[Tay2021a]은 효율적인 트랜스포머 모델에 대한 벤치마크를 수행하였는데, Long Range Arena라 명명한 평가 지표를 통해 Performer [Choromanski2021]를 비롯한 최근 어텐션 연산의 개선 모델들에 대한 성능을 그 효과성(LRA score)과 효율성(examples per sec) 측면에서 비교하였음

- 그림 4-6에서 X축은 단위 시간(초)당 처리하는 데이터 개수이고, Y축은 해당 논문에서 정의한 LRA 점수 임. 그림에서 오른쪽 위에 배치될수록 효과성과 효율성 모두가 좋은 모델임. 원의 크기는 모델의 크기를 상대적으로 표현하는데, 가령 초기 트랜스포머 모델과 Big Bird 모델을 비교하면 Big Bird 모델이 트랜스포머 모델보다 크기가 작음을 확인할 수 있음
- 그림 4-6을 통해 확인할 수 있듯이, BigBird[Zaheer2020]는 LRA 점수가 트랜스포머 모델보다 향상된 성능을 보이며, Performer 및 Linformer, Linear Transformer 등이 상대적으로 작은 모델 크기로 빠른 처리가 가능함을 보임

■ [표 4-4]에서 보이듯이 문서 압축 비교 검증 및 영상 분류에 있어 Sparse Transformer[Child2019]가 우수한 성능을 보임. 텍스트처리 및 Pathfinder와 같은 추론 문제에 있어서 저 순위 근사화 기술을 적용한 Performer 및 Linear Transformer의 성능이 우수함을 표를 통해 확인할 수 있음.



[그림 4-6] 어텐션 모듈을 개선한 트랜스포머 모델들의 성능 비교 [Tay2021a]

[표 4-4] Long Range Arena 벤치마크 기준 트랜스포머 논문들의 성능 비교

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	FAIL	54.39
Local Attention	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Sparse Trans.	17.07	63.58	**59.59**	**44.24**	71.71	FAIL	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
Linformer	35.70	53.94	52.27	38.56	76.34	FAIL	51.36
Reformer	**37.27**	56.10	53.40	38.07	68.50	FAIL	50.67
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	FAIL	51.39
Synthesizer	36.99	61.68	54.67	41.61	69.45	FAIL	52.88
BigBird	36.05	64.02	59.29	40.83	74.87	FAIL	**55.01**
Linear Trans.	16.13	**65.90**	53.09	42.34	75.30	FAIL	50.55
Performer	18.01	65.40	53.82	42.77	**77.05**	FAIL	51.41
Task Avg (Std)	29 (9.7)	61 (4.6)	55 (2.6)	41 (1.8)	72 (3.7)	FAIL	52 (2.4)

■ Tay 등이 제안한 Long Range Arena(LRA) 점수[Tay2021]는 표 4-4에서 보이

는 바와 같이 아래의 태스크들 각각에 대해 성능을 측정하고 이들의 평균 값으로 산정함

- ListOps 태스크는 긴 시퀀스($\leq$2K)에서 문맥을 추론하는 능력을 확인함
- Text 태스크는 바이트 레벨 텍스트를 분류하는 능력을 검증함. 예, 스팸, 사기, 봇 감지 및 상업광고 문서 분류 등
- Retrieval 태스크는 바이트 레벨로 문서를 이진 분류(binary classification)함. 이를 위해 문서를 인코딩하고 압축 저장 후, 압축된 표현들의 유사도를 비교하여 문서 검색 성능을 비교함
- Image 분류 태스크는 CIFAR10 데이터 집합을 이용하여 $N \times N$ 크기의 이미지를 일련의 시퀀스로 하여 N^2 픽셀들을 보고 이에 대한 분류를 수행하여 성능을 비교
- Path finder 챌린지는 이미지 내의 연결된 경로를 찾는 문제로 검은 바탕에 점으로 표현된 하얀색 경로들이 선택된 두 점에 대해 연결되었는지의 여부를 이진 분류하는 문제. 이미지 크기는 32×32으로, 이를 펼쳐 만드는 시퀀스 길이는 $32^2 = 1{,}024$ 로 설정함
- Path finder-X 챌린지는 path finder challenge와 동일한 태스크이나 이미지 크기를 128×128로 늘리고, 이에 따른 시퀀스 길이는 $128^2 = 16K$로 Path finder 챌린지 대비 시퀀스 길이가 16배임

□ 어텐션 모듈의 개선과 관련된 대표 논문들

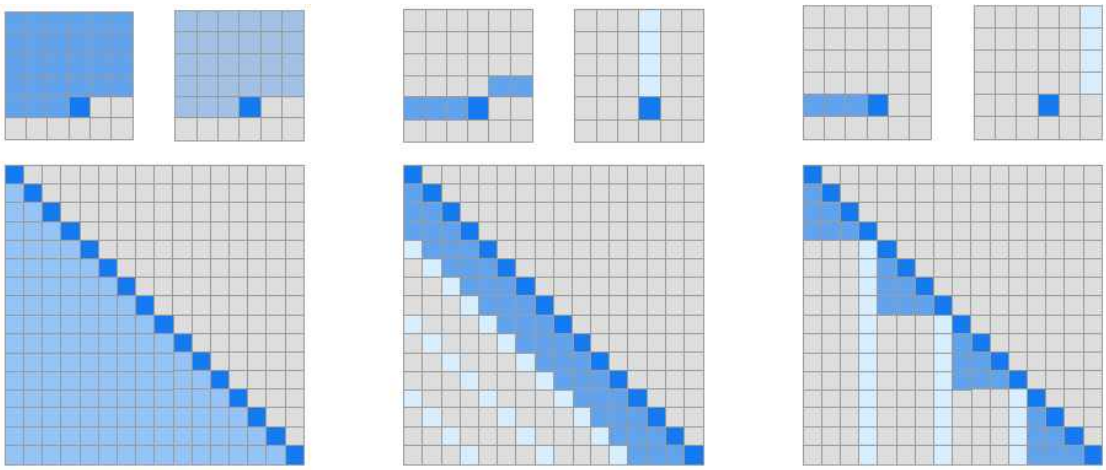
○ Sparse Transformer [Child2019]

■ 이 기법은 멀티 헤드 어텐션 연산 시(이미지 분류에 있어 Correlation 연산 시), 어텐션 행렬의 로컬 어텐션과 strided 어텐션을 결합하여 연산량을 줄임

■ 메모리와 파라미터 요구량 : 질의 Q, 키 K, 값 V가 기존 트랜스포머와 동일하므로 파라미터 갯수는 변화 없으나, 어텐션 계층의 메모리 복잡도는 $O(nlogn)$임. 여기에서 n은 입력 시퀀스 길이

■ 행렬 곱셈 연산의 특정 블록 희소 연산 구현을 지원하는 GPU에서는 이용가능하나, TPU에서는 이용불가하다는 제약이 따름. 커스텀 화된 CUDA 커

널과 TVM 프로그래밍을 필요로 함



(a) 트랜스포머 (b) 희소 트랜스포머(strided) (c)희소 트랜스포머(fixed)

[그림 4-7] 완전 트랜스포머와 인수 분해된 어텐션 기반의 희소 트랜스포머 연산 비교 [Child2019]

- [그림 4-7]에서 아래 그림은 출력 connectivity 어텐션 행렬로 상위 그림의 6x6이미지의 두 어텐션 헤드의 연산 대상이 되는 입력 픽셀의 위치를 표시함. 희소 트랜스포머는 연산 파라미터 갯수를 줄여 향상된 연산속도를 확보함.

○ ETC [Ainslie2020]

■ 이 기법은 희소 어텐션 방식의 개선 연구로, 새로운 글로벌-로컬 어텐션 기법(global-local attention mechanism)을 제시함

■ 제시된 글로벌-로컬 어텐션 기법은 4종류로 나뉘는데, (1) global-to-global (g2g), (2) global-to-local(g2l), (3) local-to-global(l2g), 그리고 local-to-local(l2l)로 구분됨

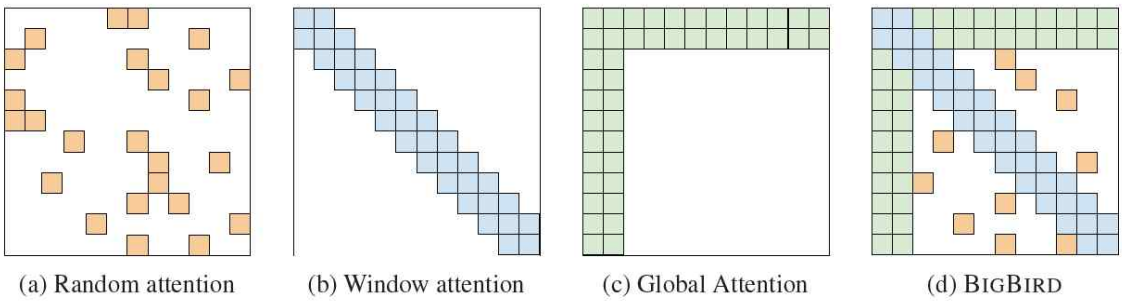
■ ETC는 원래의 입력 외에 접두어(prefix)로 n_g개의 보조 글로벌 토큰을 가짐. 이는 글로벌 시퀀스 정보를 수집하는 메모리 형태의 풀링 연산이며, 메모리 요구량은 $O(n^{2_g}+n_g n)$임

■ 글로벌 어텐션 연산으로는 causal mask 연산이 불가하여 autoregressive decoding에 사용될 수 없다는 제약이 따름

○ Big Bird [Zaheer2020]

■ Big Bird는 어텐션 연산을 위해 그림 4-5에서 보이는 바와 같이 3개의 원자 패턴들(글로벌 어텐션, 밴드 어텐션, 랜덤 어텐션)을 결합하여 아래 그림 4-8과 같이 희소한 어텐션 모델을 구성함

■ Big Bird 모델은 그림 4-8에서 보이는 바와 같이 (a) r=2로 랜덤 어텐션 (b) w=3으로 밴드 어텐션(또는 윈도우 어텐션), (c) g=2로 글로벌 어텐션을 결합하여 어텐션 연산을 구현



[그림 4-8] Big Bird모델의 어텐션 결합 [Zaheer2020]

■ 어텐션 연산에 희소성을 높임으로써 연산 가능 시퀀스의 길이를 늘여 긴 문서에 대한 컨텍스트 파악(ListOps), 텍스트, 검색 등의 태스크에서 장점을 드러냄 (표 4-4 참조)

■ 글로벌 어텐션에 있어 Big Bird가 기존의 희소 트랜스포머 계열과 다른 점은 글로벌 어텐션을 시퀀스에 존재하는 토큰들에서 선별하여 그 인덱스를 토큰으로 사용함으로써 전체 시퀀스에 대해 어텐션 연산의 대상으로 함. 이를 Bird-ITC(Internal Transformer Construction)라 칭함.

■ CLS와 같은 전체 컨텍스트에 영향을 줄 수 있는 특별 토큰의 처리도 고려하였는데, 이에 대해서도 앞서 설명된 인덱스를 통해 글로벌 어텐션을 수행하는 형태이고, 이를 Bird-ETC(Extended Transformer Construction)이라 칭함. ITC는 그 개수가 적어 글로벌 어텐션에 유용하고, ETC는 태스크 별 특정 글로벌 컨텍스트 유지에 유리함.

■ ETC와 마찬가지로, 글로벌 어텐션 연산으로 causal mask 연산이 불가하여

autoregressive decoding에 사용될 수 없는 인코더 전용 모델이라는 제약이 따름. 모델의 메모리 요구량과 파라미터 복잡도는 선형적인 장점이 있음. 즉 $O(n)$임.

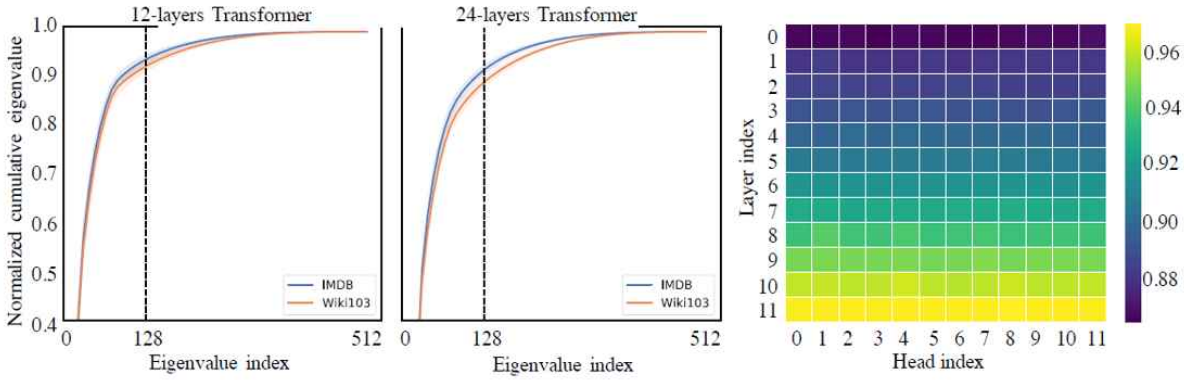
- Big Bird에서의 희소 어텐션은 완전 어텐션(full attention)에 비해 효율성은 향상시켰으나, 정확도는 그에 미치지 못함. 더불어 1 개의 조밀한 어텐션 계층(Dense Attention layer)으로 계산이 가능한 간단한 문제 해결을 위해서도 n-개의 희소 어텐션 계층(Sparse Attention layer)을 필요로 한다는 지적도 있음

○ Linformer [Wang2020b]

- Linformer는 셀프 어텐션 행렬에 대해 저 순위 근사화(low-rank approximation)를 적용하여 행렬 연산량을 줄임.
- 사전 학습된 트랜스포머의 셀프 어텐션 행렬 분석을 통해 singular value 값이 계층이 깊을수록, 그리고 rank가 작을수록 대부분의 어텐션 헤드 값을 좌우함을 그림 4-9에서 보이는 바와 같이 확인함.
  - 그림 4-9에서 왼쪽 그림은 입력 시퀀스 길이 n=512인 사전 학습된 트랜스포머의 셀프 어텐션 행렬의 스펙트럼 분석으로 Y축은 정규화 된 누적 컨텍스트 매핑 행렬 P의 singular value이고 X축은 largest Eigen value의 index를 표현함. 오른쪽 그림은 서로 다른 계층과 어텐션 헤드들에 대해 128번째 가장 큰 Eigen value에서의 normalized 누적 eigenvalue를 히트맵으로 그린 것임.
- Linformer에서는 head-wise, layer-wise 공유 E,F $(E=F)$ 행렬을 둠. 모든 계층에 대해 단일 프로젝션을 수행함. 이에 따라 식 4-3을 변형하여 식 4-9, 4-10과 같이 변경함.

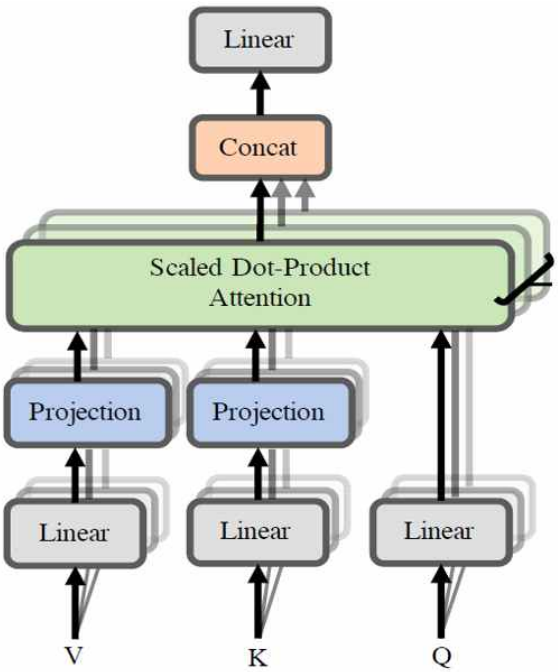
$$head_{ij} = Attention(Q, K, V) = Attention(XW_i^Q, EXW_j^K, FXW_j^V) \quad \text{(식 4-9)}$$

$$Attention(Q, K, V) = \underbrace{Softmax(\frac{XW_i^Q(EXW_j^K)^T}{\sqrt{d_k}})}_{P:\, n\times k} \cdot \underbrace{FXW_j^V}_{k\times d} \quad \text{(식 4-10)}$$



[그림 4-9] Linformer에서 저순위 근사화를 적용한 근거가 된 셀프 어텐션 행렬 분석 [Wang2020b]

■ Linformer의 셀프 어텐션의 연산 구조를 도식화하면 아래 그림 4-10과 같음.



[그림 4-10] Linformer의 셀프 어텐션 연산 구조 [Wang2020b]

■ 시퀀스 길이의 projection은 차원 방향으로 여러 시퀀스 정보를 혼합하게 됨. 즉, 어텐션 스코어 계산 시, 과거와 미래 정보를 혼합하게 됨. 이는 causal masking을 유지하는 것을 불가능하게 하여 autoregressive decoding에 사용될 수 없다는 제약이 따름.

■ Linformer의 메모리와 파라미터 복잡도는 셀프 어텐션에 대한 메모리 복잡도에 선형적임. 즉 $O(n)$임.

○ Linear Transformer [Katharopoulos2020]

■ Linear Transformer는 셀프 어텐션에서의 행렬 곱셈 연산을 커널 기반 연산으로 대체하여 연산 속도를 선형적이 되도록 향상시킴.

■ Auto-regressive decoding에서 이용되는 causal masking의 어텐션도 RNN으로 줄여 연산속도를 상수화 함.

■ Linear Transformer에서 위치 i에 대한 질의 Q_i에 대하여 어텐션 연산 $A_i(x)$은 아래의 식으로 정의함.

$$A_i(x) = \frac{\sum_{j=1}^{p} sim(Q_i, K_j) V_j}{\sum_{j=1}^{p} sim(Q_i, K_j)} \quad \text{(식 4-11)}$$

- 이 수식에서 $p = n$인 경우, full unmasked 어텐션(fully visible mask)이고, $p = i$인 경우 causal masking임.

- 일반적인 softmax 어텐션은 $sim(q,k) = \exp(\frac{q^T k}{\sqrt{d_k}})$ 형태이나, Linear Transformer는 이를 커널 형태 $sim(q,k) = \Phi(q)^T \Phi(k)$로 대체함.

- 여기서 Φ는 고차원 피처 맵으로,

$$A'_i(x) = \frac{\Phi(Q_i)^T S_p}{\Phi(Q_i)^T Z_p} \quad \text{(식 4-12)}$$

$$A'_i(x) = \frac{\Phi(Q_i)^T \sum_{j=1}^{p} \Phi(K_j) V^{T_j}}{\Phi(Q_i)^T \sum_{j=1}^{p} \Phi(K_j)} \quad \text{(식 4-13)}$$

- $p = n$인 fully unmaksed attention에 대해 S_n, Z_n에 대해 한 번 계산 후 $0 \le i \le N$ 범위의 모든 position에 대해 그 값을 재사용함.

- causal masking 어텐션에서는 식 4-12의 S_i, Z_i는 RNN의 상태로 간주할 수 있고, 초기 조건 $S_0 = 0, Z_0 = 0$와 함께 순환 연계(recurrence relation)로 업데이트함.

$$S_i = S_{i-1} + \Phi(K_i)\, V^{T_i} \quad \text{(식 4-14)}$$

$$Z_i = Z_{i-1} + \Phi(K_i) \quad \text{(식 4-15)}$$

- 메모리와 파라미터 복잡도 : 질의, 키, 값의 차원이 d 이고 커널 연산 비용이 $O(c)$일 때 실행 연산 복잡도는 $O(Ncd)$임. Φ 피처 맵의 선택으로 모델의 연산 복잡도는 $O(Nx^2)$임. 순방향, 역방향 연산은 적재합을 이용해서 선형의 시간이 걸리며 메모리 요구량이 일정함.

○ Performer [Choromanski2021]

■ Performer는 랜덤 커널을 이용한 어텐션 기법을 제안함

■ 일반적인 bi-directional attention은 다음과 같음.

- $Q, K, V \in R^{n \times d}$ 에 대해 (n은 입력 시퀀스 길이, d는 은닉 차원)

$$Att = D^{-1}AV,\ A = \exp(QK^T/\sqrt{d}),\ D = diag(A1_L) \quad \text{(식 4-16)}$$

- 여기에서 1_L은 길이 L의 모든 값이 1인 벡터, exp(.) 연산은 element-wise 연산, diag(.)는 입력 벡터를 가지고 직교 행렬(diagonal matrix)를 만드는 함수

- 식 4-16에서 표현한 어텐션의 복잡도는 시간 복잡도의 경우 ~ $O(n^2 d)$, 메모리 복잡도의 경우 ~ $O(n^2 + nd)$

■ 이를 커널 버전으로 해석한 어텐션 정의는 다음과 같음

$$Att = [g(Q_i^T)K(Q_i^T K_j^T), h(K_j^T)] \quad \text{(식 4-17)}$$

- K는 $d \times d$를 스칼라 R에 매핑 하는 커널 함수, g,h는 d를 스칼라 R에 매핑 하는 함수

■ performer는 FAVOR(Fast Attention via Orthogonal Random Feature)중 ORF(Orthogonal Random Feature)를 기반으로 $n \times n$ 어텐션 행렬의 이차

방정식(quadratic) 크기의 연산량을 줄이고자 함.

- Performer의 어텐션 출력 Y 연산은 ORF(Orthogonal Random Feature)를 기반으로 식 4-18과 같음. 식 4-19~4-21은 식 4-18 해석에 필요한 보조 수식임
- Performer는 Linear Transformer와 같은 형태의 커널 기반의 함수인데, performer의 kernel은 식 4-21과 같음.

$$Y = \hat{D}^{-1}(Q'((K')^T V)) \quad \text{(식 4-18)}$$

$$, \ \hat{D} = diag(Q'((K')^T 1_N)), \ Q' = D_Q \Phi(Q^T)^T, \ K' = D_K \Phi(K^T)^T \quad \text{(식 4-19)}$$

$$, \ D_Q = g(Q_i^T), \ D_K = h(K_i^T) \quad \text{(식 4-20)}$$

$$, \ \Phi(X) = \frac{c}{\sqrt{M}} f(Wx + b)^T \quad \text{(식 4-21)}$$

- c는 상수, $W \in R^{M \times d}$는 랜덤 피쳐 행렬
- M은 랜덤 피쳐 갯수를 조절하는 행렬의 차원을 나타냄

- 메모리와 파라미터 복잡도: 양방향 FAVOR의 연산 복잡도는 $O(Md + Nd + MN)$임, M은 랜덤 피쳐 차원

제 4 절 트랜스포머의 FFN 모듈의 효율성 개선

□ FFN 모듈의 개선

○ Dong 등의 분석에 따르면, FFN 계층 없이 셀프 어텐션 계층만 쌓게 될 경우 rank collapse 문제를 일으키게 되어 토큰이 동일하게 분포하는 inductive bias를 야기 시킨다. 이러한 문제를 완화하기 위해 FFN 계층이 필요하고, 다음과 같은 변형 모듈이 존재한다[Dong2021]

□ FFN 모듈의 활성 함수 개선

○ Vanilla Transformer는 두 계층의 FFN의 non-linearity를 위해 ReLU를 사용

○ Ramanchandran 등은 ReLU를 Swish 함수 $f(x) = x sigmoid(\beta x)$로 대체하였으며, WMT2014 데이터 집합에서 영어를 독일어로 번역하는데 있어 향상된 성

능을 보임 [Ramanchandran2017]

○ GPT 모델[Radford2018]에서는 사전학습 모델 학습에 Gaussian Error Linear UnitGELU)[Hendrycks2016]를 이용하였으며 사전학습 모델에서 거의 표준처럼 이용하고 있음

○ Shazeer등은 Gated Linear Units(GLU)[Dauphin2017] 및 이에 대한 변형 활성함수를 아래 수식(4.32)~(4.36)와 같이 FFN 계층에 적용함[Shazeer2020].

■ 추가 파라미터들이 있는 관계로 베이스라인 모델과 모델 크기를 맞추기 위해 FFN의 중간 계층의 파라미터 수를 줄임. 학습기간 동안 향상된 성능을 표 4-5와 같이 확인함. 사전 학습 후 파인 튜닝 태스크에서도 표 4-6과 같이 향상된 성능을 확인함.

$$FFN_{GLU}(x, W, V, W_2) = (\sigma(x\,W) \otimes x\,V)\,W_2 \quad \text{(식 4-22)}$$

$$FFN_{BiLinear}(x, W, V, W_2) = (x\,W \otimes x\,V)\,W_2 \quad \text{(식 4-23)}$$

$$FFN_{ReGLU}(x, W, V, W_2) = (\max(0, x\,W) \otimes x\,V)\,W_2 \quad \text{(식 4-24)}$$

$$FFN_{GEGLU}(x, W, V, W_2) = (GELU(x\,W) \otimes x\,V)\,W_2 \quad \text{(식 4-25)}$$

$$FFN_{SwiGLU}(x, W, V, W_2) = (Swish(x\,W) \otimes x\,V)\,W_2 \quad \text{(식 4-26)}$$

[표 4-5] GLU 활성 함수 및 이를 변형한 활성함수 모델들의 GLUE 벤치마크 성능 비교

	Score Average	CoLA MCC	SST-2 Acc	MRPC F1	MRPC Acc	STSB PCC	STSB SCC	QQP F1	QQP Acc	MNLIm Acc	MNLImm Acc	QNLI Acc	RTE Acc
FFN_{ReLU}	83.80	51.32	94.04	**93.08**	**90.20**	89.64	89.42	89.01	91.75	85.83	86.42	92.81	80.14
FFN_{GELU}	83.86	53.48	94.04	92.81	**90.20**	89.69	89.49	88.63	91.62	85.89	86.13	92.39	80.51
FFN_{Swish}	83.60	49.79	93.69	92.31	89.46	89.20	88.98	88.84	91.67	85.22	85.02	92.33	81.23
FFN_{GLU}	84.20	49.16	94.27	92.39	89.46	89.46	89.35	88.79	91.62	86.36	86.18	92.92	**84.12**
FFN_{GEGLU}	84.12	53.65	93.92	92.68	89.71	90.26	90.13	89.11	91.85	86.15	86.17	92.81	79.42
$FFN_{Bilinear}$	83.79	51.02	**94.38**	92.28	89.46	90.06	89.84	88.95	91.69	**86.90**	**87.08**	92.92	81.95
FFN_{SwiGLU}	84.36	51.59	93.92	92.23	88.97	**90.32**	**90.13**	**89.14**	**91.87**	86.45	86.47	**92.93**	83.39
FFN_{ReGLU}	**84.67**	**56.16**	**94.38**	92.06	89.22	89.97	89.85	88.86	91.72	86.20	86.40	92.68	81.59
[Raffel et al., 2019]	83.28	53.84	92.68	92.07	88.92	88.02	87.94	88.67	91.56	84.24	84.57	90.48	76.28
ibid. stddev.	0.235	1.111	0.569	0.729	1.019	0.374	0.418	0.108	0.070	0.291	0.231	0.361	1.393

[표 4-6] 트랜스포머 모델의 held-out log-perplexity

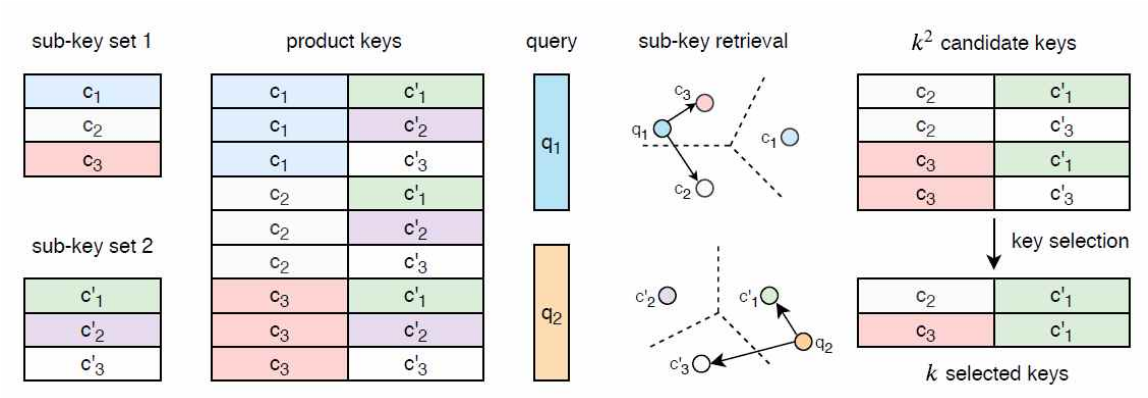
Training Steps	65,536	524,288
FFN_{RELU}(baseline)	1.997(0.005)	1.677
FFN_{GELU}	1.983(0.005)	1.679
FFN_{Swish}	1.994(0.003)	1.683
FFN_{GLU}	1.982(0.006)	1.663
$FFN_{Bilinear}$	1.960(0.005)	1.648
FFN_{GEGLU}	1.942(0.004)	1.633
FFN_{SwiGLU}	1.944(0.010)	1.636
FFN_{REGLU}	1.953(0.003)	1.645

□ 대용량 연산을 위한 FFN 모듈의 변형

○ Lample 등은 FFN을 product-key 메모리 계층으로 대체함 [Lample2019a]

■ product-key 메모리 계층은 질의 네트워크, 키 선택 모듈(두 종류의 sub-key들을 포함), 값 룩업 테이블 구성

■ (1) 입력 데이터를 질의 네트워크를 이용하여 은닉 공간에 project함, (2) 생성된 질의를 가지고 k-최인접 키를 찾기 위해 키 값과 비교함. 이 때 키 값은 키 선택 모듈을 통해 구해지며, 두 종류 sub-key들의 카티시안 곱임. (3) k 최인접 키를 이용하여 값 룩업 테이블에서 대응되는 값들을 찾고 출력 값을 생성함. 이 과정은 생성 질의를 큰 규모의 글로벌 키-값 쌍으로 어텐션을 구한다는 점 기존 어텐션 기법과 유사함(그림 4-11 참조). 이 과정을 통해 추가 연산량을 무시할 정도로 작게 함으로써 현격히 향상된 성능을 얻음.



[그림 4-11] product key 메모리 계층의 연산 과정 [Lample2019a]

○ Lample 등은 이 기법을 적용한 XLM 모델[Lample2019a]과 100개 이상의 다

국어를 함께 학습 시킨 XLM-R 모델[Lample2019b]과 함께 공개함[1)]

제 5 절 언어 모델의 효율성 개선

□ 개요

○ 언어 모델의 효율성을 향상시키는 연구는 그 목적을 크게 (1) 거대 데이터 대형 모델(Pre-trained model)을 학습시키는 자원과 시간을 절약하기 위함, (2) 기 학습된 모델의 크기를 줄여 저장 용량과 태스크 별 파인 튜닝 시 실행 시간을 단축시키기 위해 작은 모델을 만들되, 기 학습된 모델에 대응하는 성능을 보일 수 있게 함, (3) 기 학습된 모델에서 추론 속도를 줄이는 효율성 개선을 목적으로 함

○ 그간 이와 관련하여 제시된 연구들은 크게 (1) 가중치 공유, (2) 양자화 또는 정밀도 조정, (3) 지식 증류, (4) 신경망 구조 탐색, (5) 태스크 어댑터 방식 등 있음

□ 가중치 공유(Weight Sharing) 기법

○ Universal Transformer 는 각 계층에 걸쳐 어텐션과 트랜지션 가중치를 공유 [Dehghani2019]

○ Quaternion Transformer[Tay2019]는 Linear transformation 계층에서 각 요소들을 지엽적으로 공유하는 Hamilton 프로덕트를 기반으로 가중치 공유.

○ ALBERT는 각 계층에 걸쳐 어텐션과 트랜지션 가중치를 공유하여 사용

■ 표 4-7에서 보이듯 가중치 공유를 통해 학습 파라미터의 개수를 기존 트랜스포머 대비 약 10배 정도 줄일 수 있는 효과를 가짐

■ 표 4-8를 통해 Bert와 Albert를 비교해 볼 때, 계층을 보다 깊이 쌓은 ALBERT xlarge 모델이 BERT large 모델에 비해 학습 파라미터 개수는 적으나 정확도 측면에서 더 나은 성능을 보임.

■ 반면, ALBERT xlarge 모델은 BERT base모델에 비해 학습 속도 측면에서는

1) https://github.com/facebookresearch/XLM#v-product-key-memory-layers-pkm

8배정도 오히려 느린 것을 확인할 수 있음. 한편 이 표들을 종합해 볼 때, ALBERT를 통해 학습 시간이 절약되고 파라미터 개수를 줄이는 효과를 볼 수는 있으나 parameter 개수를 줄이는 효과를 볼 수 있으나 가중치 공유 전략의 성능의 한계가 있음을 확인할 수 있음[Lan2020]

[표 4-7] 가중치 공유 전략에 따른 효과

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base E=768	all-shared	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8
	shared-attention	83M	89.9/82.7	80.0/77.2	84.0	91.4	67.7	81.6
	shared-FFN	57M	89.2/82.1	78.2/75.4	81.5	90.8	62.6	79.5
	not-shared	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base E=128	all-shared	12M	89.3/82.3	80.0/77.1	82.0	90.3	64.0	80.1
	shared-attention	64M	89.9/82.8	80.7/77.9	83.4	91.9	67.6	81.7
	shared-FFN	38M	88.9/81.6	78.6/75.6	82.3	91.7	64.4	80.2
	not-shared	89M	89.9/82.8	80.3/77.3	83.2	91.5	67.9	81.6

[표 4-8] BookCorpus와 Wikipedia에 대해 사전학습 모델의 성능 비교

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

□ 양자화 또는 정밀도 조정(Quantization/Mixed Precision) 기법

○ FairSeq 프레임워크는 복수개의 정밀도를 가진 부동소수점 타입(mixed precision) 알고리즘으로 sequential encoder-decoder구조를 기반으로 확장 가능한 sequential modeling지원. loss가 없는 Mixed Precision을 위해 Forward 연산 시 FP16으로, Backward 연산 시 FP32를 채택함[Ott2019]

○ Q-bert 모델은 이차 헤시안(Hessian) 정보를 활용한 양자화 기술로 BERT 모델의 파라미터 표현에 2, 4, 8 비트 정밀도를 적용하면서도 정확도 손실을 최대 2.3% 정도로 낮춤

■ [표 4-11](b)에서와 같이 32 비트로 파라미터가 표현된 모델에 비해 가중치 양자화 비트를 2, 3비트로 멀티 헤드 어텐션 계층에 적용하더라도 큰 loss 손실이 없음. CNN(Convolutional Neural Network) 모델에서 컨볼루션 계층의 양자화에 비해 완전 연결 계층(fully connected layer)의 양자화 손실이

큰 것과 마찬가지로, BERT에서도 멀티 헤드 어텐션 계층에서의 양자화보다 완전 연결 계층을 양자화 할 시에 손실이 더 큼 [Shen2020]

[표 4-9] Q-Bert의 Quantization 실험 결과

(a) quantization effect on embedding

Method	ew-bits	ep-bits	SST-2	MNLI-m	CoNLL-03	SQuAD
Baseline	32	32	93.00	84.00	95.00	88.69
Q-BERT	8	8	92.88	83.83	94.79	88.47
Q-BERT	4	8	91.74	82.91	94.44	87.55
Q-BERT	8	4	89.11	82.84	93.86	72.38
Q-BERT	4	4	85.55	78.08	84.32	61.70

(b) quantization of multi-head attention versus fully-connected layer

Method	s-bits	f-bits	SST-2	MNLI-m	CoNLL-03	SQuAD
Baseline	32	32	93.00	84.00	95.00	88.69
Q-BERTMP	1/2MP	2/3MP	89.56	73.66	91.74	75.81
Q-BERTMP	2/3MP	1/2MP	85.89	70.89	87.55	68.71
Q-BERTMP	2/3MP	2/3MP	92.08	81.75	93.91	86.95

- 표 4-9에서 ew-bits, ep-bits, s-bits, f-bits의 약어는 각각 다음을 나타냄. ew-bits: word embedding bits, ep-bits: position embedding bits, s-bits: multi-head attention layer, f-bits: fully-connected layer
- (a)에서는 가중치와 활성화 함수를 8bit으로 precision,
- (b)에서는 embedding과 활성화 함수를 8bit으로 precision하여 실험 진행.

○ Fan 등은 사전 학습된 CNN 모델 및 트랜스포머 모델 파라미터들을 기준으로 프로덕트 양자화(Product Quantization) 기술을 적용하여 RoBERTa Base 모델 대비 파라미터 숫자를 13배 이상 줄이면서도(480MB→38MB), MNLI 태스크에 대해서 정확도 손실이 1.2% 정도로 크게 줄임. 해당 기술로 Efficient Net7 대비 파라미터 크기를 80배 이상 줄이면서(260MB→3.3MB), ImageNet 데이터 분류 태스크에 대해서는 정확 손실이 4.4%정도 발생함 [Fan2021]. 아래 표 4-10,

4-11 참조.

[표 4-10] Roberta-Base모델 압축 성능비교, MNLI task

Model	MB	MNLI
RoBERTa Base + LD (Fan et al., 2019)	480	84.8
BERT Base (Devlin et al., 2018)	420	84.4
PreTrained Distil (Turc et al., 2019)	257	82.5
DistilBERT (Sanh et al., 2019b)	250	81.8
MobileBERT* (Sun et al.)	96	84.4
TinyBERT† (Jiao et al., 2019)	55	82.8
ALBERT Base (Lan et al., 2019)	45	81.6
AdaBERT† (Chen et al., 2020)	36	81.6
Quant-Noise	38	83.6
Quant-Noise + Share + Prune	14	82.5

[표 4-11] EfficientNet-B7모델 압축 성능비교, ImageNet

Model	MB	Acc.
EfficientNet-B7 (Tan & Le, 2019)	260	84.4
ResNet-50 (He et al., 2015)	97.5	76.1
DenseNet-169 (Huang et al., 2018)	53.4	76.2
EfficientNet-B0 (Tan & Le, 2019)	20.2	77.3
MobileNet-v2 (Sandler et al., 2018)	13.4	71.9
Shufflenet-v2 ×1 (Ma et al., 2018)	8.7	69.4
HAQ 4 bits (Wang et al., 2018)	12.4	76.2
iPQ ResNet-50 (Stock et al., 2019)	5.09	76.1
Quant-Noise	3.3	80.0
Quant-Noise + Share + Prune	2.3	77.8

□ 지식 증류(Knowledge Distillation) 기법

○ DistilBert는 사전 학습된 모델에서 지식 증류를 통해 BERT 모델을 약 40% 크기로 줄이면서도 그 정확도 성능은 기존 대비 97% 수준을 유지하고 처리 속도는 약 60%향상시킴.

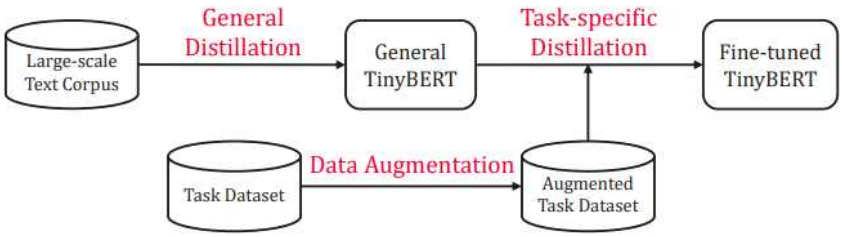
- DistilBert의 Student 모델은 teacher 모델인 BERT와 동일하나 토큰 타입 임베딩과 풀링 계층이 없으며, 트랜스포머 인코딩 블록을 두 배로 줄임. Student 모델 초기화는 teacher 모델의 계층 두 개당 하나 씩을 취하는 형태를 사용함[Sanh2021]

○ Tang 등[Tang2019]은 BERT를 Bi-LSTM 계층을 가지는 모델로 지식 증류를 수행함. 각 태스크별 성능 비교는 표 4-12 참조

[표 4-12] SST-2, QQP, MLNI-m과 MNLI-mm task에서의 Distilled BiLSTM 성능 비교

#	Model	SST-2	QQP	MNLI-m	MNLI-mm
		Acc	F_1/Acc	Acc	Acc
1	$BERT_{LARGE}$ (Devlin et al., 2018)	94.9	72.1/89.3	86.7	85.9
2	$BERT_{BASE}$ (Devlin et al., 2018)	93.5	71.2/89.2	84.6	83.4
3	OpenAI GPT (Radford et al., 2018)	91.3	70.3/88.5	82.1	81.4
4	BERT ELMo baseline (Devlin et al., 2018)	90.4	64.8/84.7	76.4	76.1
5	GLUE ELMo baseline (Wang et al., 2018)	90.4	63.1/84.3	74.1	74.5
6	Distilled $BiLSTM_{SOFT}$	**90.7**	**68.2/88.1**	**73.0**	**72.6**
7	BiLSTM (our implementation)	86.7	63.7/86.2	68.7	68.3
8	BiLSTM (reported by GLUE)	85.9	61.4/81.7	70.3	70.8
9	BiLSTM (reported by other papers)	$87.6^{\dagger}$	– /$82.6^{\ddagger}$	66.9^{*}	66.9^{*}

○ TinyBERT 모델은 지식 증류 기술을 이용하여 Student 모델을 학습하되, 아래 그림 4-12와 같이 이 단계 학습 방법을 활용하여 태스크 별로 향상된 성능을 얻는 지식 증류 기반 학습 기법에 대해 소개함 [Jiao2020]. 이를 통해 DistilBert와 같은 학습 파라미터들에 대해 태스크 별로 다소 향상된 성능을 표 4-15와 같이 확보함



[그림 4-12] TinyBERT의 학습 방법 [Jiao2020]

□ 신경망 구조 탐색(Neural Architecture Search) 기법

○ Wang 등은 신경망 구조 탐색 기술을 통해 HAT에서 서로 다른 하드웨어(CPU, GPU, IoT 기기)에 따른 효율적인 트랜스포머 모델을 탐색함. 특히 WMT14 영어-독일어, 영어-프랑스어 번역에 있어 라즈베리 Pi-4에서 기존 트랜스포머 모델 대비 3.7배 더 작은 크기 모델로 3배 빠른 추론 시간을 지니지만 BLEU 점

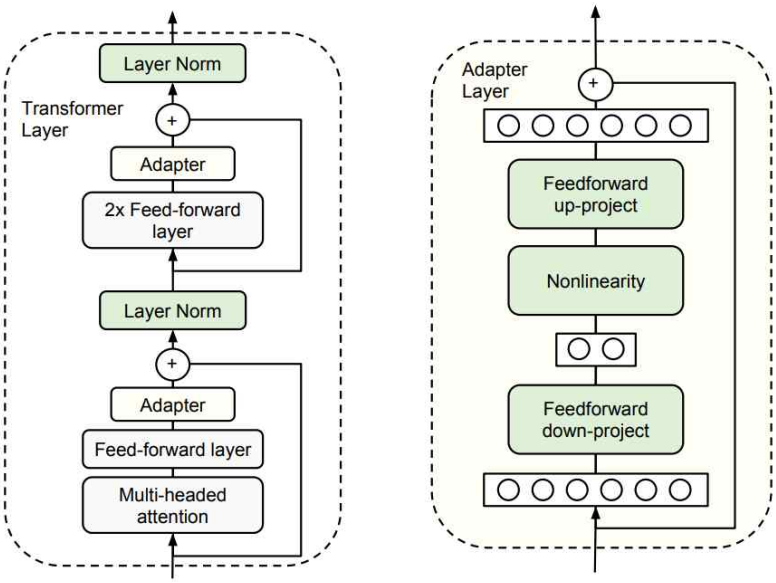
[표 4-13] TinyBERT와 기존 모델간의 압축 성능 및 태스크 정확도 성능 비교

System	#Params	#FLOPs	Speedup	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg
$BERT_{BASE}$ (Teacher)	109M	22.5B	1.0x	83.9/83.4	71.1	90.9	93.4	52.8	85.2	87.5	67.0	79.5
$BERT_{TINY}$	14.5M	1.2B	9.4x	75.4/74.9	66.5	84.8	87.6	19.5	77.1	83.2	62.6	70.2
$BERT_{SMALL}$	29.2M	3.4B	5.7x	77.6/77.0	68.1	86.4	89.7	27.8	77.0	83.4	61.8	72.1
$BERT_4$-PKD	52.2M	7.6B	3.0x	79.9/79.3	70.2	85.1	89.4	24.8	79.8	82.6	62.3	72.6
$DistilBERT_4$	52.2M	7.6B	3.0x	78.9/78.0	68.5	85.2	91.4	32.8	76.1	82.4	54.1	71.9
$MobileBERT_{TINY}$†	15.1M	3.1B	-	81.5/81.6	68.9	**89.5**	91.7	**46.7**	80.1	**87.9**	65.1	**77.0**
$TinyBERT_4$ (ours)	14.5M	1.2B	9.4x	**82.5/81.8**	**71.3**	87.7	**92.6**	44.1	**80.4**	86.4	**66.6**	**77.0**
$BERT_6$-PKD	67.0M	11.3B	2.0x	81.5/81.0	70.7	89.0	92.0	-	-	85.0	65.5	-
PD	67.0M	11.3B	2.0x	82.8/82.2	70.4	88.9	91.8	-	-	86.8	65.3	-
$DistilBERT_6$	67.0M	11.3B	2.0x	82.6/81.3	70.1	88.9	92.5	49.0	81.3	86.9	58.4	76.8
$TinyBERT_6$ (ours)	67.0M	11.3B	2.0x	**84.6/83.2**	**71.6**	**90.4**	**93.1**	**51.1**	**83.7**	**87.3**	**70.0**	**79.4**

수가 비슷한 성능을 확인함. Nvidia GPU에서도 기존 트랜스포머 모델 대비 2.7배 빠른 속도로 비슷한 BLEU 점수를 확인함 [Wang2020]

□ 태스크 어댑터(Task Adapter) 기법

○ Houlsby 등은 언어 모델을 파인 튜닝 시 다운스트림 태스크 별로 재사용성을 높일 수 있도록 태스크 어댑터 개념을 처음 제안함. 파인 튜닝 시 학습 파라미터를 여러 태스크에서 같이 재사용하는 태스크 어댑터를 통해 전체적으로 파라미터 개수를 줄임. 어댑터 계층을 추가한 트랜스포머의 구성은 아래 그림 4-13와 같음. [Houlsby2019]



[그림 4-13] 트랜스포머 모델에 추가된 어댑터 구조 [Houlsby2019]

○ Pfeiffer 등은 MAD-X 논문을 통해 BERT 모델과 XLM-R 모델에 대해 cross-lingual 환경에서 서로 다른 태스크에서 활용 가능한 어댑터를 구현하여 성능을 비교 평가함. MAD-X는 개체명 식별(NER), 상식 추론(common sense reasoning), 질의응답 태스크에서 경쟁적인 성능을 확보함 [Pfeiffer2020]

제5장

결 론

BERT로 대표되는 뉴럴 네트워크 기반의 언어 모델은 자연어 처리에 있어 가장 중요한 딥러닝 모델이며, 모델 크기가 시간이 지남에 따라 기하급수적으로 커지고 있다. 이러한 모델 크기는 모델의 성능을 올리는데 도움을 주나, 반대로 많은 메모리와 연산량을 요구함에 따라 중소기업이나 대학에서 이를 개발하기에는 곤란한 상황이 도래하고 있다. 본 보고서에서는 뉴럴 네트워크 기반의 언어 모델을 구성하는 기본 개념인 어텐션과 트랜스포머 구조를 설명하고, 현 언어 모델의 상황을 짚어보았다. 또한, 현재 언어 모델의 한계를 개선하기 위한 여러 방법들을 조사하고 이를 기록하였다. 특히 특정 분야의 전문 지식을 다루기 위한 언어 모델을 개발하는데 있어 그 성능 뿐만 아니라 효율성을 고려한 개발 방법 위주로 살펴보았다. 향후 뉴럴 네트워크 기반의 언어 모델의 개발과 성능 향상에 있어 이 보고서의 내용이 작지만 바른 방향을 가리키는 이정표가 되기를 바래본다.

Acknowledgement

○ 이 보고서는 한국과학기술정보연구원의 주요사업 과학기술 텍스트 분석을 위한 지능 기술 연구(K-21-L04-C01)의 결과물로 작성되었음.

○ 이 보고서의 2장 중 트랜스포머와 BERT 모델에 대한 설명의 경우 유원준 외 1인의 딥러닝을 이용한 자연어 처리 입문, https://wikidocs.net/book/2155의 글과 그림을 일부 인용하고 있음을 밝힘

참고 문헌

[Al-Rfou2019a] AL-Rfou Rami, et al. Character-level language modeling with deeper self-attention. In Proceedings of the AAAI, 2019. p. 3159-3166.

[Abadi2016a] Abadi, Martin et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016

[Alammar2018a] Alammar, Jay, The Illustrated Transformer, https://jalammar.github.io/illustrated-transformer/, 2018

[Aggarwal2018a] Aggarwal, Charu C., Neural Networks and Deep Learning, ISBN 978-3-319-94463-0, Springer, 2018

[Ainslie2020a] Ainslie, Joshua, et al. ETC: Encoding long and structured inputs in transformers. In Proceedings of the 2020 EMNLP Conference, pages 268-284, 2020

[Bahdanau2015a] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." In Proceedings of ICLR, 2015

[Bachilechner2021] BACHLECHNER, Thomas, et al. Rezero is all you need: Fast convergence at large depth. arXiv preprint arXiv:2003.04887, 2020.

[Beltagy2020] Beltagy, Iz; PETERS, Matthew E.; COHAN, Arman. Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150, 2020.

[Bojanowski2017a] Bojanowski, Piotr, et al. "Enriching word vectors with subword information." Transactions of the Association for Computational Linguistics 5 (2017): 135-146.

[Child2019] Child, Rewon, et al. Generating long sequences with Sparse Transformers. arXiv preprint arXiv:1904.10509, 2019.

[Cho2014a] Cho, Kyunghyun, et al. "On the properties of neural machine translation: Encoder-decoder approaches." arXiv preprint arXiv:1409.1259 (2014).

[Cho2014b] Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).

[Choromanski2020] Choromanski, Krzysztof, et al. Masked language modeling for proteins via linearly scalable long-context transformers. arXiv preprint arXiv:2006.03555, 2020.

[Choromanski2021] Choromanski, Krzysztof, et al. Rethinking attention with performers. ICLR, 2021.

[Clark2020a] Clark, Keven et al., “ELECTRA: Pre-training text encoders as discriminators rather than generators”, In Proceedings of the ICLR Conference, 2020.

[Dai2019] Dai, Zihang, et al. Transformer-XL: Attentive language models beyond a fixed-length context. Proc of the 57th the Association for Computational Linguistics, pp. 2978?2988, 2019.

[Dauphin2017] Dauphin, Yann N., et al. Language modeling with gated convolutional networks. In: International conference on machine learning. PMLR, p. 933-941, 2017.

[Dehghani2019] Dehghani, Mostafa, et al. Universal transformers. In Proceedings of ICLR, 2019.

[Devlin2018a] Devlin, Jacob, et al. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[Dong2021] DONG, Yihe; CORDONNIER, Jean-Baptiste; LOUKAS, Andreas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. arXiv preprint arXiv:2103.03404, 2021.

[Fan2021] Fan, Angela, et al. Training with quantization noise for extreme model compression. In Proceedings of ICLR, 2021.

[Färber2018] Färber, Michael, and Achim Rettinger. "Which Knowledge Graph Is Best for Me?." arXiv preprint arXiv:1809.11099, 2018

[Frankle2019] Frankle, Jonathan, and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks." International Conference on Learning Representations. 2019.

[Gu2019] Gu, Jiatao; LIU, Qi; CHO, Kyunghyun. Insertion-based decoding with automatically inferred generation order. Transactions of the Association for Computational Linguistics, V. 7, pp.661-676, 2019.

[Guo2019] Gui, Qipeng, et al. Star-transformer. In Proceedings of NAACL, 2019.

[Guu2020] Guu, Kelvin, et al. Realm: Retrieval-augmented language model pre-training. arXiv preprint arXiv:2002.08909, 2020

[Han2015] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).

[He2016a] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[He2020] He, Pengcheng, et al. DeBERTa: Decoding-enhanced BERT with disentangled attention. In Proceedings of ICLR, 2021.

[Henighan2020a] Henighan, Tom, et al. "Scaling laws for autoregressive generative modeling."

arXiv preprint arXiv:2010.14701 (2020).

[Hendrycks2016] HENDRYCKS, Dan; GIMPEL, Kevin. Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415, 2016.

[Ho2019] HO, Jonathan, et al. Axial attention in multidimensional transformers. arXiv preprint arXiv:1912.12180, 2019.

[Houlsby2019] Houlsby, Neil, et al. Parameter-efficient transfer learning for NLP. In: International Conference on Machine Learning. PMLR, pp. 2790-2799, 2019.

[Huang2019] Huang, Cheng-Zhi Anna, et al. Music transformer. In proceedings of the ICLR conference, 2019.

[Ilharco2020a] Ilharco, Gabriel, et al. "High Performance Natural Language Processing." Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts. 2020.

[Jiao2020] Jiao, Xiaoqi, et al. Tinybert: Distilling bert for natural language understanding. In Proceedings of EMNLP, 2020.

[Joshi2020] Joshi, Mandar, et al. "SpanBERT: Improving pre-training by representing and predicting spans." Transactions of the Association for Computational Linguistics 8 (2020): 64-77.

[KamingHe2016]K. He, X. Zhang, S. Ren, and J. Sun, ``Identity mappings in deep residual networks,'' in Proc. Eur. Conf. Comput. Vis. Cham, Switzerland: Springer, pp. 630645, 2016.

[Katharopoulos2020] Katharopoulos, Angelos, et al. Transformers are rnns: Fast autoregressive transformers with linear attention. In: International Conference on Machine Learning. PMLR p. 5156-5165, 2020.

[KingMa&Ba2014] KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[Kitaev2020] Kitaev, Nikita; KAISER, Łukasz; LEVSKAYA, Anselm. Reformer: The efficient transformer. In Proceedings of ICLR, 2020.

[Kovaleva2019] KOVALEVA, Olga, et al. Revealing the dark secrets of BERT. arXiv preprint arXiv:1908.08593, 2019.

[Lample2019a] LAMPLE, Guillaume, et al. Large memory layers with product keys. In Proceedings of NeurIPS, 2019.

[Lample2019b] Lample, Guillaume; CONNEAU, Alexis. Cross-lingual language model pretraining. arXiv preprint arXiv:1901.07291, 2019.

[Lan2020] Lan, Zhenzhong, et al. Albert: A lite bert for self-supervised learning of language

representations. In Proceedings of ICLR Conference, 2020.

[Lee2020] Lee, Kyong-Ha, Kim, Eunhui, Analysis of lightweighting techniques for Deep Neural Network, ISBN 976-89-294-1152-7, 2020

[Li2020a] LI, Zhuohan, et al. Train large, then compress: Rethinking model size for efficient training and inference of transformers. PMLR, 2020.

[Li2020b] Li, Xiaoya, et al. Sac: Accelerating and structuring self-attention via sparse adaptive connection. In Proceedings of NeurIPS, 2020.

[Lin2021] Lin, Tianyang, et al. A Survey of Transformers. arXiv preprint arXiv:2106.04554, 2021.

[Liu2019a] Liu, Yinhan, et al. "Roberta: A robustly optimized bert pretraining approach." arXiv preprint arXiv:1907.11692 (2019)

[Liu2020a] LIU, Liyuan, et al. Understanding the difficulty of training transformers. In Proceeding of EMNLP, 2020.

[Liu2020b] Liu, Xuanqing, et al. Learning to encode position for transformer with continuous dynamical model. In: PMLR, pp. 6327-6335, 2020.

[Liu2020c] Liu, Weijie, et al. "K-bert: Enabling language representation with knowledge graph." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. No. 03. 2020.

[Mikolov2010a] Mikolov, Tomáš, et al. "Recurrent neural network based language model." Eleventh annual conference of the international speech communication association. 2010.

[Mikolov2013a] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

[Mosbach2021] MOSBACH, Marius; ANDRIUSHCHENKO, Maksym; KLAKOW, Dietrich. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. ICLR, 2021.

[Oord2016] Oord, Aaron van den, et al. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499, 2016.

[Ott2019] Ott, Myle, et al. fairseq: A fast, extensible toolkit for sequence modeling. Proceedings of NAACL-HLT 2019: Demonstrations, pp. 48-53, 2019.

[Parmar2018] Parmar, Niki, et al. Image transformer. In: International Conference on Machine Learning. PMLR, pp. 4055-4064, 2018.

[Pennington2014a] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "GloVe: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[Peters2018a] Peters, Matthew E., et al. "Deep contextualized word representations." arXiv preprint arXiv:1802.05365, 2018

[Peters2019a] Peters, Matthew E., et al. "Knowledge enhanced contextual word representations." arXiv preprint arXiv:1909.04164, 2019

[Pfeiffer2020] Pfeiffer, Jonas, et al. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. Proceedings of the 2020 EMNLP, pp. 7654-7673, 2020.

[Radford2018] A. Radford;, K. Narasimhan; T. Salimans; I. Sutskever, Improving language understanding with unsupervised learning. Technical Report, OpenAI, 2018.

[Radford2019] RADFORD, Alec, et al. Language models are unsupervised multitask learners. OpenAI blog, 1.8: 9, 2019.

[Raffel2020a] Raffel, Colin, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research 21,pp. 1-67, 2020.

[Rajpurkar2016] Rajpurkar, Pranav, et al. "Squad: 100,000+ questions for machine comprehension of text." arXiv preprint arXiv:1606.05250 (2016).

[Ramanchandran2017]RAMACHANDRAN, Prajit; ZOPH, Barret; LE, Quoc V. Searching for activation functions. arXiv preprint arXiv:1710.05941, 2017.

[Roy2021] Roy, Aurko, et al. Efficient content-based sparse attention with routing transformers. Transactions of the Association for Computational Linguistics, V. 9, pp.53-68, 2021.

[Sanh2019] Sanh, Victor, et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. EMC^2: 5th Edition Co-located with NeurIPS'19, 2019.

[Shaw2018] Shaw, Peter; USZKOREIT, Jakob; VASWANI, Ashish. Self-attention with relative position representations. Proceedings of NAACL-HLT, pp. 464-468, 2018.

[Shazeer2020] SHAZEER, Noam. Glu variants improve transformer. arXiv preprint arXiv:2002.05202, 2020.

[Shen2020] Shen, Sheng, et al. Q-bert: Hessian based ultra low precision quantization of bert. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 8815-8821, 2020.

[Sundermeyer2012a] Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling." Thirteenth annual conference of the international speech communication association. 2012.

[Tang2019] Tang, Raphael, et al. Distilling task-specific knowledge from bert into simple neural networks. arXiv preprint arXiv:1903.12136, 2019.

[Tay2019] Tay, Yi, et al. Lightweight and efficient neural natural language processing with

quaternion networks. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 1494-1503, 2019.

[Tay2020a] Tay, Yi, et al. Sparse sinkhorn attention. In: International Conference on Machine Learning. PMLR, pp. 9438-9447, 2020.

[Tay2020b] TAY, Yi, et al. Efficient transformers: A survey. arXiv preprint arXiv:2009.06732, 2020.

[Tay2021a] TAY, Yi, et al. Long range arena: A benchmark for efficient transformers. In Proceedings of ICLR, 2021.

[Vaswani2017a] Vaswani, Ashish, et al. Attention is all you need. In Proceedings of the 31st Int'l Conf. on NeurIPS, 2017.

[Wang2018a]Wang, Alex, et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding." arXiv preprint arXiv:1804.07461 (2018).

[Wang2019a] Wang, Alex, et al. "Superglue: A stickier benchmark for general-purpose language understanding systems." arXiv preprint arXiv:1905.00537 (2019).

[Wang2020a] Wang, Hanrui, et al. HAT: Hardware-aware transformers for efficient natural language processing. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 7675-7688, 2020.

[Wang2020b] Wang, Sinong, et al. Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768, 2020.

[Wang2020c] Wang, Wei, et al. "StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding." International Conference on Learning Representations. 2019.

[Wang2021] Wang, Benyou, et al. On position embeddings in BERT, In Proceedings of ICLR Conference, 2021

[Wang2021b] Wang, Xiaozhi, et al. "KEPLER: A unified model for knowledge embedding and pre-trained language representation." Transactions of the Association for Computational Linguistics 9 (2021): 176-194.

[Wu2016a] Wu, Yonghui, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).

[Xiong2020] XIONG, Ruibin, et al. On layer normalization in the transformer architecture. In Proceedings of ICLR. PMLR, pp. 10524-10533, 2020.

[Zaheer2020] Zaheer, Manzil, et al. Big Bird: Transformers for Longer Sequences. In Proceedings of NeurIPS, 2020.

[Zhang2019a] Zhang, Zhengyan, et al. "ERNIE: Enhanced Language Representation with Informative Entities." Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019.

[Zhang2020a] Zhang, Zhuosheng, Hai Zhao, and Rui Wang. "Machine reading comprehension: The role of contextualized language models and beyond." arXiv preprint arXiv:2005.06249 (2020).

[Zhang2021] ZHANG, Tianyi, et al. Revisiting few-sample BERT fine-tuning, In Proceedings of ICLR, 2021.

[유원준2021], 유원준, 안상준, 딥 러닝을 이용한 자연어 처리 입문, https://wikidocs.net/book/2155

트랜스포머 기반 최신 언어 모델 기법 분석

저자
이경하, 김은희

2021년 10월 05일 인쇄
2021년 10월 05일 발행

발 행 처
KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information
www.kisti.re.kr
대전광역시 유성구 대학로 245
㉾34141
전화 : 042-869-1004
등록 : 1991년 2월 12일 제 5-259호

발행인
김 재 수

인쇄처
엠에스기획

비매품/무료
93560
9 788929 411879
ISBN 978-89-294-1187-9