시간효율적 Bert 튜닝을 위한 Casual Grid Search 기법

박준하, 장성섭, 권형우, 이성준

고려대 디지털융합금융학과 2기
head4ths@gmail.com

Casual Grid Search Method for Time-efficient Bert Tuning

Junha Park, Sungseop Chang, Hyungwoo Kwon, Sungjun Lee
Korea University, Digital Finance Convergence Engineering

요 약

BERT 의 성능 튜닝을 위한 주요 파라미터로는 배치사이즈, 러닝레이트, 에포크 수가 있고 이중 배치 사이즈는 실험환경에서 GPU 의 메모리가 허용하는 최대크기의 60~80% 정도 크기로 고정하고 러닝레이트와 에포크를 중점적으로 Grid Search 하며 최적의 조합을 찾는 방법이 시간 효율적이다. 그 외 패딩과 어텐션 마스크를 사용하여 학습시간을 줄일 수 있으며 최대 토큰 길이 등도 학습시간에 많은 영향을 주므로 적절히 설정해야 한다. 파라미터간 독립성을 파악하여 특정 파라미터를 튜닝할 때 독립적인 파라미터 들은 최대한 빠른 수행이 가능한 조합으로 설정해 놓는 방식의 테스트용-최종제출용 파라미터 이중화를 적극 활용해야 한다.

주제어: Bert, 튜닝, Grid Search,

1. 서론

본 과제는 크게 두가지로 진행되었다. 첫째는 Naver sentiment movie corpus 국문 DataSet 에 대하여 Binary 감정분석을 진행하는 것이고 둘째는 Friends 대화(영문) Data Set에 대하여 Multi Class 감정분석을 진행하는 과제였다.

기술의 발전과 선행연구자 특히 구글에 감사하게도 앙상블 기법처럼 이전의 복잡하고 컴퓨팅 파워를 많이 필요로 하는 모델을 사용하지 않고도 Pretrained BERT 모델을 활용하여 전이학습을 수행 하는 것 만으로도 준수한 성능을 낼 수 있는 시대가 되었다.

본 과제를 수행함에 있어서 직장과 병행하는 코스의 특성과 Kaggle 리더보드 경쟁을 위해 최적의 성능을 내는 것에 주안점을 두어 활용과 성능에 초점을 맞추어 진행하였으며 그에 따라 다양한 실험적인 시도는 해보지 못한 점은 다소 아쉬운 점이다.

하지만 개인적으로는 BERT 라는 최신 기술을 실제 과제에 접목하여 전처리를 수행하고 성능을 향상시키기 위하여 다양한 시도를 하면서 향 후 회사 업무 및 대학원 졸업과제에 대한 다양한 아이디어 및 활용 가능성을 얻었으며 많은 의미가 있었다. 또한 회사에서 본 과정을 지원을 하는 목적도 학술적 성과보다는 최신 기술의 습득과 그를 업무에 접목함에 있다고 생각한다.

따라서 본 소논문에서는 BERT 의 일반적인 동작원리와 전이학습의 일반적인 프로세스에 대하여는 기술하지 않고 Friends 데이터셋에 대하여 전처리를 수행한 방식과 BERT 튜닝시 어떤 하이퍼 파라미터 들이 있고 어떤 방식으로 시간효율적인 BERT 튜닝을 하였는지 위주로 진행하고자 한다.

2. 데이터 특성

먼저 국문 Naver sentiment movie corpus 데이터 셋은 각각의 독립된 문장에 대하여 긍정/부정으로 라벨링이 되어 있으며 binary 라서 class imbalance 문제가 크지 않다.

두번째 Friends 드라마 대화 데이터셋은 5~10 개의 문장들이 전후관계를 가지고 대화 형태로 연결된 컨텍스트를 가지고 있으며 하나의 Sentence는 감정을 분석하기에 부족한 수준으로 짧다. 따라서 문맥을 활용하기 위한 전처리가 필요하다.또한 화자에 따라 감정이 바뀌는 부분도 있으므로 전처리 시 반영이 필요하다. 또한 8개의 감정이 라벨링 된 multi class 이며 class imbalance 문제를 가지고 있다.

3. 전처리

국문 Naver sentiment movie corpus 은 BERT 의 SentencePiece Tokenizer를 그대로 활용했으며 그 외의 과제만을 위한 전처리를 별도로 하지 않아도 되었다. BERT 처리를 위한 특수 토큰인 [CLS] [SEP] 만 표시해주는 정도만으로 아래처럼 SentencePiece Tokenizer 처리가 되었다.

```
원문 : [CLS] 정말 많이 울었던 영화입니다. [SEP]
토큰화 : ['[CLS]', '정', '##말', '많이', 'ㅇ', '##ㅜㄹ', '##었던', '
영화', '##입', '##니다', '.', '[SEP]']
```

물론 SKT KoBERT, ETRI Korbert 등 한글 전용으로 학

습된 BERT 모델을 활용한다면 형태소로 전처리 하는 것도 도움이 되겠으나 본 과제에서는 bert-base-multilingual-uncased 모델을 사용하여서 일반적인 SentencePiece 토크나이징을 사용하였다.

4. 전처리 - 담화자 처리

Friends 대화 Data Set 에는 담화자가 있다. 담화자에 따라서 화를 잘 내는 사람, 기분이 좋은 편인 사람 등등 감정분류에 영향을 줄 수 있는 요소가 있다. 이런 정보를 활용하기 위해 다음과 같이 여러가지 경우를 테스트하였다.

```
담화자 : 문장
담화자 문장
[담화자] 문장
[담화자] [Says] 문장
```

이중 가장 좋은 성능을 보인 "담화자 : 문장" 의 형태를 채택하여 후속 테스트를 진행하였다.

담화자 출연 빈도를 분석하여 일정 비율 이하의 엑스트라 출연자는 생략하면 더 성능을 개선할 수도 있을 것이다.

5. 전처리 - Dialog Context 처리

휴먼의 감정인식을 살펴보면 대화의 감정분석은 상식적으로 전후 맥락을 살펴서 이루어지게 된다. 특히 특정 문장의 감정은 직전 문장의 영향을 크게 받는다. 다음과 같은 사례를 생각해보자

```
A : 너 화났니? (중립)
B : 응 매우 (화났음)
```

만약 직전 문장을 고려하지 않고 "응 매우" 만 학습을 시킨다면 아무리 뛰어난 모델도 해당 문장을 화난 것으로 판단하지 못할 것이다.

그런데 직후 문장도 고려를 해야할까? 아래의 예를 보면 직후 문장의 경우 영향도가 매우 떨어진다.

```
B : 응 매우 (화났음)
A : 미안해 … (슬픔)
```

위와 같은 직관을 가지고 여러가지 경우를 테스트해보았다. 마찬가지로 BERT 모델에서 활용하기 위하여 [CLS] 로 시작하는 토큰을 붙이고 문장마다 [SEP] 로 나누어 준다. 담화자도 각각의 문장에 포함시켰다.

```
[CLS] 본문장 [SEP] 직전문장 [SEP]
[CLS] 직전문장 [SEP] 본문장 [SEP]
[CLS] 본문장 [SEP] 직후문장 [SEP]
[CLS] 직전문장 [SEP] 본문장 [SEP] 직후문장 [SEP]
[CLS] 두개전문장 [SEP] 직전문장 [SEP] 본문장 [SEP]
```

테스트 결과 "[CLS] 본문장 [SEP] 직전문장 [SEP]" 의 형태가 가장 좋은 성능을 보여서 채택하였다.

또한 위의 형태로 변형시에 Dialog 별 첫 문장은 직전 문장이 없는데 이런 경우 [None] 토큰을 넣어 형식을 맞춰 주었다.

```
[CLS] 본문장 [SEP] [None] [SEP]
```

6. 전처리 - Data Class imbalance

Friends 대화 Data Set은 아래처럼 imbalance 하다

anger	513
disgust	240
fear	185
joy	1283
neutral	4752
non-neutral	2017
sadness	351
surprise	1220

이에 따라서 학습이 over-fitting 되면 neutral 로 예측치가 편향되는 현상이 발견되었다. 러닝레이트를 크게 하여 과도하게 반복시 전체 예측치가 neutral 로 예측되기도 하였다.

그러나 이를 위하여 OverSampling 을 하여 재시도를 하였더니 anger, fear, sadness 등 마이너한 라벨의 정답율은 극적으로 올라갔으나 절대 다수를 차지하는 neutral 의 정답률이 떨어지는 현상이 발생했다.

이 부분은 분명히 문제가 있었는데 최종적으로 문제를 해결하지 못하여 아쉬운 부분이며 최종과제 제출은 오버샘플링을 적용하지 않을 때 더 좋은 성능을 보여서 적용하지 않고 제출하였다.

오버샘플링의 비율을 조절하여 전체라벨의 숫자를 맞추는 게 아니라 약간만 오버샘플링 한다거나 오버 샘플링이 아닌 로스 펑션의 종류를 바꿔본다거나 rose, smote 등의 방법을 쓰거나 neutral 에만 under샘플링을 적용한다거나 등등의 방법을 모두 테스트 해보고 가장 좋은 성능을 보이는 것을 채택하였다면 더 좋았을 것이다. 특히 under 샘플링 시에 더 좋은 성능을 보였다면 파인 튜닝 학습시간도 줄어들게 되므로 튜닝 시간도 줄일 수 있었을 것이다.

오버샘플링보다 언더샘플링이 좋은 성능을 보인다는 논문도 있지만 데이터를 버린다는 것이 상식상으로는 맞지 않아서 시간이 좀더 있었다면 오버샘플링 되는 비율을 일종의 하이퍼 파라미터화 하여서 테스트 해봤다면 더 좋은 결과를 낼 수 있었을 듯 하다.

또한 이렇게 모든 라벨의 가중치를 똑같이 하지 않고 일정 비율 만큼만 오버샘플링 한다는 가정하에서 생각할 때 본 테스트의 트레이닝 셋과 캐글 제출 평가용 데이터 셋의 감정 비율이 서로 다르다. 상식적으로도 다르겠지만 테스트 시의 성능이 60%를 넘었는데 제출 평가시에는 최대 58% 였던 것으로 봐서 이는 트레이닝 데이터 셋과

평가용 데이터 셋간에 분명한 의미역 분포 차이가 있다는 것을 의미한다. Domain Adaptation 등 data set간의 차이를 줄여주는 기법을 응용할 수 있다면 더 개선이 가능할 것이다.

7. 모델

허깅페이스사의 Pretraind 모델을 활용하여 전이학습을 수행하였으며 각각의 과제에서 사용한 모델은 다음과 같다

```
국문 : bert-base-multilingual-uncased
영문(튜닝) : bert-base-cased, bert-base-uncased
영문(최종) : bert-large-cased, bert-large-uncased
```

튜닝 과정에서는 반복을 빠르게 수행하기 위하여 bert-base 모델을 사용하였으며 최종 제출본을 만들 때는 bert-large 모델을 사용하였다. 국문의 경우 대소문자가 없으므로 uncased 모델을 사용하였다.

영문과제의 경우는 cased 모델이 더 좋은 성능을 보여서 튜닝을 위한 반복된 하이퍼 파라미터 grid search 시에는 빠른 bert-base-cased 를 사용하였고 최종 과제 제출시에는 ber-large-cased 모델을 사용하였다.

참고로 bert-base는 12-layer, 768-hidden, 12-heads, 110M parameters. bert-large는 24-layer, 1024-hidden, 16-heads, 340M parameters. 이며 이 모델 사이즈에 따라서 GPU 메모리에서의 한번에 처리할 수 있는 적절한 batch사이즈도 달라진다. 이부분은 이후 튜닝 부분에서 조금더 자세히 설명하겠다.

국문 과제를 위하여는 SKT 사의 KoBERT, ETRI의 Korbert를 활용해보고 싶어서 사용허가협약서까지 작성해서 모델을 받았으나 bert-base-multilingual-uncased 만을 사용하여도 튜닝후 적절한 수준의 성능 (89%) 가 나와서 중단하였으나 이후 졸업 과제 등에서 활용해볼 예정이다.

영문과제의 경우는 데이터의 양이 작고 더구나 클래스 수가 많아서 더 데이터의 양이 작다고 볼 수 있다. 따라서 BERT 이후로 좀 더 작은 데이터 양에도 좋은 성능을 보인다는 ELECTRA 등을 테스트 해보고 싶었으나 시간부족으로 BERT 로만 튜닝을 진행하였다.

8. 실험환경

구글의 코랩 환경을 사용하였다. 한달 만원으로 프리미엄 가입시 Tesla P100-PCIE 16GB GPU 를 사용할 수 있는데 싯가 400만원에 육박하는 장비이다. 더구나 하나가 아니라 3~4개의 세션을 띄울 수 있다. 한달이 되기 전에 작업이 끝났으면 구독 취소도 가능하다. 물론 일정 사용량 이상 사용시 제한을 건다.

9. 최대길이 설정 및 Padding

각 Sentence 의 길이가 다르므로 학습속도를 위해 데이터의 손실이 없을 정도인 64로 최대길이를 설정하였다. 워드피스 토크나이저를 거친후의 최대 토큰 수를 계산하여 적용하는 것도 좋겠지만 다른 데이터 셋에 테스트 하는 것도 가정하여 여유를 두었다.

그리고 토큰이 없는 부분은 0으로 채워주는 패딩작업을 하였다. 딥러닝 이후에 데이터 형태가 축소되어 소실되는 경우가 생기지 않도록 0으로 padding 하여 길이를 맞춰준다.

```
array([ 101, 14394, 131, 1145, 146, 1108, 1103, 1553, 1825,
1113, 1139, 1419, 1116, 6468, 1121, 1103, 148, 2162, 118,
126, 1106, 144, 2069, 118, 127, 1449, 119, 102, 164, 7330,
166, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

10. 어텐션 마스크

패딩 부분은 BERT 모델에서 어텐션을 수행할 필요가 없는 의미 없는 부분이다. 따라서 패딩이 아니면 1, 패딩이면 0으로 하는 어텐션 마스크를 두어 수행 속도를 향상시킨다.

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

11. 하이퍼 파라미터

튜닝대상 주요 파라미터로 Batch Size, Epochs, Learning Rate 세가지가 영향도가 큰 요소이다. 또한 앞서 전처리시에 이야기한 담화자, Dialog context, Oversampling 비율등 전처리 방식도 일종의 하이퍼 파라미터로 볼 수 있다. 심지어 모델의 종류도 파라미터로 볼 수 있고 문장의 최대길이, warm up 스텝 도 파라미터로 볼 수 있다.

12. 시간효율적 Bert 튜닝

위와 같이 전처리 및 모델까지 하이퍼 파라미터로 생각할 경우 하이퍼 파라미터 개수 만큼의 차원수를 가지는 튜닝대상 Grid 공간이 생겨버린다. 그런데 튜닝 파라미터 중에는 이산분포를 가지는 것, 연속분포를 가지는 것, 그리고 배치 사이즈 처럼 수행 속도에 영향을 주는 경우, 러닝 레이트 처럼 수행 속도에는 영향을 안주는 경우 등 성격별로 나눌 수 있다

예를 들어 전처리 방식은 러닝 레이트 등과 독립적으

로 어느 것이 최적인지 정해져 있다. 이런 것들은 굳이 모든 케이스를 테스트 해보지 않아도 다른 파라미터를 고정시킨 상태에서 어느 것이 최적인지 해당 파라미터만 일차원적인 테스트로 확인이 가능하다.

반면 러닝레이트-에포크 같은 경우는 독립적이지 않고 서로 연계되어 컨벡스한 형태를 이룬다. 다행인 점은 컨벡스한 형태이기 때문에 모든 케이스를 테스트 해볼 필요가 없고 일종의 경사상승 법 처럼 가장 최고의 정확도를 가지는 파라미터 조합을 찾아 갈 수 있다.

시간 효율적인 Bert 튜닝의 핵심은 먼저 독립적인 파라미터와 강한 상관관계를 가지는 파라미터를 구분하고 완전히 독립적인 파라미터 들은 비교적 단순한 모델을 사용하여 최적의 값을 빠르게 찾아내어 고정해 놓고 독립적이지 않은 파라미터들의 최적조합을 찾아가는데 있다.

또한, 반복 수행 속도에 영향을 크게 주는 파라미터들은 먼저 자원을 최대한 활용할 수 있는 값으로 고정해 놓고 나머지 파라미터들을 튜닝해 나가는 방식으로 진행하였다.

본 과제에 위 방법론을 적용한 상세 사례는 다음과 같다. (1) 먼저 가벼운 bert-base 로 전처리 방식의 best case들을 찾아 고정한다. (2) 문장의 Max length 는 64로 고정한다. (3) Batch 사이즈는 base 모델의 경우 32, large 모델 사용시는 10으로 고정한다. (4) 웜업 스텝은 0.2로 고정한다.(5) cased와 uncased 도 비교해본다.(6) 오버샘플 ON/OFF (7) 이후 Epoch와 러닝레이트의 두개의 파라미터를 중심으로 2차원적인 Grid Search를 수행한다.

(1),(2),(3),(4),(5),(6) 까지는 1차원 적으로 값을 변화시키며 테스트 해본 후 고정하였고 (7) 번 단계에서는 Epoch와 러닝레이트의 2차원 적인 Grid Search를 수행하였다.

13. 튜닝 결과 - 국문 Binary Class 감정분석

아래처럼 20회이상 에포크와 러닝레이트를 바꿔가며 테스트 하였고 실제 가장 좋은 결과를 보인 조합은 러닝레이트 2e-5 에 12 epoch로 수행한 결과가 89% 로 가장 좋은 결과를 보였다.

러닝레이트가 4E-05 기준으로 epoch 2 이하는 언더피팅 상태이고 epoch 8 이상은 오버피팅 상태인 것을 볼 수 있다.

만약 테스트 시간이 조금더 있다면 2E-5 를 기준으로 전후하여 3E-5, 1E-5로 러닝레이트를 바꾸어서 에포크를 증가시키며 테스트 할 수 있을 것이다.

Learning Rate	1	2	3	4	5	6	7	8	9	10
4E-05	85.9	86.9	87.3	87.1	87.0	87.0	87.3	86.9	86.8	86.8
2E-05	85.5	86.7	87.3	87.5	87.6	87.2	87.3	87.4	87.5	87.4

14. 튜닝 결과 - 영문 Multi Class 감정분석

러닝레이트 - Epoch Grid 가 완벽히 컨벡스한 모양을 가질 것이라고 생각하였는데 약간의 local minima 도 있을 수 있는 모양으로 나타났다. 그에 따른 영향인지 캐글에서도 public Set 에 대하여는 4E-6 epoch 3 조합이 가장 좋은 성능을 보였으나 Private Set 에 대하여는 6E-6 epoch 3 조합이 더 좋은 성능을 보였다. 물론 차이가 크지 않아 랜덤 오차에 의한 영향도 일부 있다.

아래 표에서 눈에 띄는 점은 첫번째 국문 과제에 비하여 오버피팅되어 성능이 저하되는 수준이 더 급격히 나타난다는 것과 러닝레이트 조정에 무관하게 일정 에포크 이상이 되면 성능이 저하되는 현상이다.

이러한 성능저하는 Data의 class imblance 에 따라서 발생하는 현상으로 보인다.

Learning Rate	1	2	3	4	5	6	7	8	9	10
2E-05	55.8	56.7	56.4	57.1	55.4	56.4	55.4	55.1	55.0	55.1
6E-06	54.2	57.4	58.0	58.0	57.4	57.4	56.9	56.3	55.7	56.4
5E-06	51.9	55.4	57.5	57.4	57.2	57.4	56.9	57.0	55.9	55.8
4E-06	51.4	55.3	57.6	57.2	57.2	57.0	57.2	57.0	56.4	57.1
3E-06	42.0	53.9	56.1	56.9	57.8	57.7	57.7	56.9	56.9	56.6
2E-06	38.3	50.5	54.3	55.8	56.3	57.4	56.9	57.4	56.0	56.7
1E-06	31.4	41.0	41.8	49.1	51.9	54.8	55.1	55.0	55.9	56.3

15. 튜닝 결과 - 영문 OverSample 적용시

모든 class를 동일한 비중으로 OverSample 시 아래처럼 더 성능이 나빠지는 현상이 발생하였다. 모두 동일 동일 하는 것은 오히려 역효과가 있으며 오버샘플 비중을 조정해가며 갭을 줄여가며 최적값을 찾는 방식을 적용한다면 더 좋은 결과를 얻을 수 있을 것이다.

아래 확인시는 2차원적으로 모든 케이스를 조사하지 않고 1차원적으로 가장 좋은 케이스 파악후 해당항목을 고정하고 다른 변수를 변화시키며 관찰하는 1차원 반복 방식으로 최적값을 찾았다.

Learning Rate	1	2	3	4	5	6	7	8	9	10
3E-05			53.3							
2E-05			54.4							
1E-05			54.4							
9E-06			54.7							
7E-06			54.0							
5E-06			53.8							
3E-06	50.1	53.7	54.0	53.4	54.1	54.1	53.5	54.3	53.7	53.6
1E-06			48.3							

5. 결론

BERT 의 성능 튜닝을 위한 주요 파라미터로는 배치사이즈, 러닝레이트, 에포크 수가 있고 이중 배치 사이즈는 실험환경에서 GPU 의 메모리가 허용하는 최대크기의 60~80% 정도 크기로 고정하고 러닝레이트와 에포크를 중

점적으로 Grid Search 하며 최적의 조합을 찾는 방법이 효율적이다.

이러한 본 과제의 최적 파라미터 조합법을 찾기위하여 백번 이상의 과정을 반복하여 수작업으로 수행하였으나 향 후 기회가 된다면 이러한 최적의 조합역시 피쳐값들의 조합과 Evaluation Function 값을 가지는 딥러닝의 과제로 볼 수 있기 때문에 하이퍼 파라미터를 찾는 것 자체를 모델화 시켜보고 싶다. Gradient Descent 기법을 응용하여 최적값을 찾는 것을 자동화 할 수 있고, 또한 비슷한 과제가 있을 때 최초 파라미터 세팅을 감이 아닌 합리적인 모델을 통해 산출할 수 있을 것이다. 예를 들어 국문/영문, 최대 길이수, 워드 벡터의 평균값과 분산, 레이어 수 등을 입력하면 적절한 파라미터를 모델이 알려주는 것이다.

또는 유전 알고리즘을 활용해 최적의 파라미터를 찾는 방법도 있을 것이다.

본 연구를 통해서 강력하고 효율적인 자연어 처리 모델인 BERT 를 전이학습을 통해 활용하는 기초를 학습하였고 국문/영문 언어의 종류를 달리 하여 적용하는 방법, Binary/Multi class 분류를 다르게 하여 적용하는 법, 화자 및 컨텍스트 처리법 등을 익힐 수 있었다.

특히 의미 있었던 것은 비록 기초적인 형태지만 제한된 시간과 자원 상에서 어떻게 성능을 튜닝할 것이며 성능 튜닝의 단계는 어떻게 되는지 어떤 파라미터를 중점적으로 튜닝해야 하는지에 대하여 알게 된 것이다.

종합하여 결론을 내리면 BERT 의 성능 튜닝을 위한 주요 파라미터로는 배치사이즈, 러닝레이트, 에포크 수가 있고 이중 배치 사이즈는 실험환경에서 GPU 의 메모리가 허용하는 최대크기의 60~80% 정도 크기로 고정하고 러닝레이트와 에포크를 중점적으로 Grid Search 하며 최적의 조합을 찾는 방법이 시간 효율적이다. 그 외 패딩과 어텐션 마스크를 사용하여 학습시간을 줄일 수 있으며 최대 토큰 길이 등도 학습시간에 많은 영향을 주므로 적절히 설정해야 한다. 파라미터간 독립성을 파악하여 특정 파라미터를 튜닝할 때 독립적인 파라미터 들은 최대한 빠른 수행이 가능한 조합으로 설정해 놓는 방식의 테스트용-최종제출용 파라미터 이중화를 적극 활용해야 한다.

참고문헌

[1] Jacob Devlin, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

[2] Huggingface.Inc.Transformer https://huggingface.co/transformers/main_classes/tokenizer.html

[3] Muhammad AbdulMageed and Lyle Ungar. EmoNet: Fine-Grained Emotion Detection with Gated Recurrent Neural Networks

[4] Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, Yi-Shin Chen, CARER: Contextualized Affect Representations for Emotion Recognition