2021 국립국어원
인공 지능 언어 능력 평가

팀명: eclipse

팀원: 모윤호, 이승수, 정은서, 장지아, 강상우

CONTENTS

Chapter 01

실험준비

^	2	eclipse	eclipse-last	84.02	89.63	91.60	63.19	91.65

1. 실험준비 Data

name	train	dev	test
BoolQ	3655	700	704
CoLA	15876	2032	1060
COPA	3080	500	500
WiC	7748	1166	1246

data가 적다!

data가 적어서 PLM에서도 성능이 잘 안나올 것 같다.

out-domain에도 잘하려면 data를 늘릴 필요성이 있다.

EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks EMNLP(2019)

SR - synonym replacement

RI - random insertion

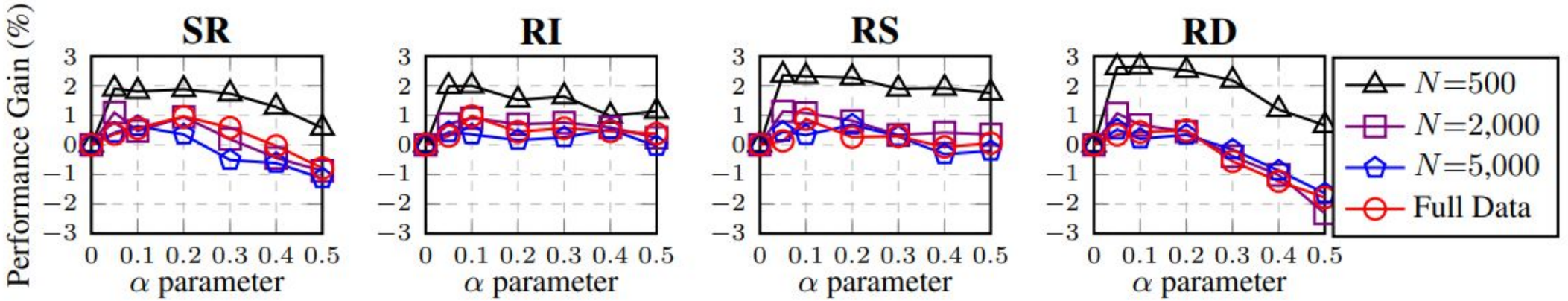
RS - random swap

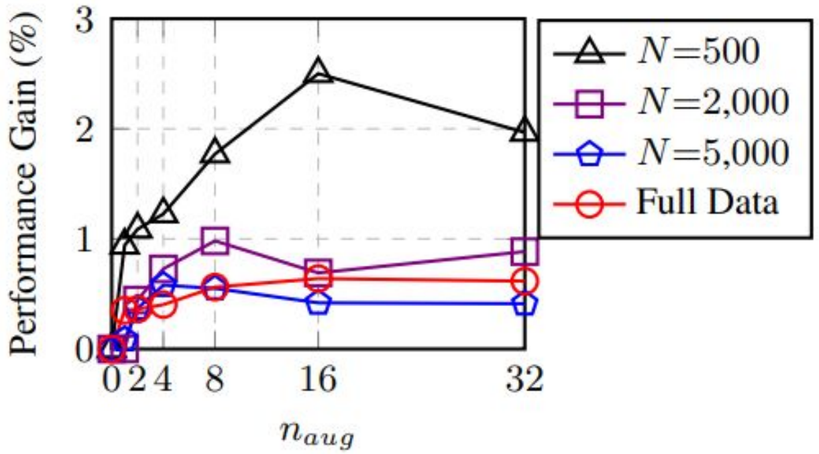
RD - random deletion

Operation	Sentence
None	A sad, superior human comedy played out on the back roads of life.
SR	A ***lamentable***, superior human comedy played out on the ***backward*** road of life.
RI	A sad, superior human comedy played out on ***funniness*** the back roads of life.
RS	A sad, superior human comedy played out on ***roads*** back ***the*** of life.
RD	A sad, superior human out on the roads of life.

https://arxiv.org/pdf/1901.11196.pdf

EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks EMNLP(2019)





N_{train}	α	n_{aug}
500	0.05	16
2,000	0.05	8
5,000	0.1	4
More	0.1	4

Table 3: Recommended usage parameters.

https://arxiv.org/pdf/1901.11196.pdf

AEDA: An Easier Data Augmentation Technique for Text Classification
EMNLP(2021)

punctuations=[".", ",", "!", "?", ";", ":"]

. , ! ? ; :

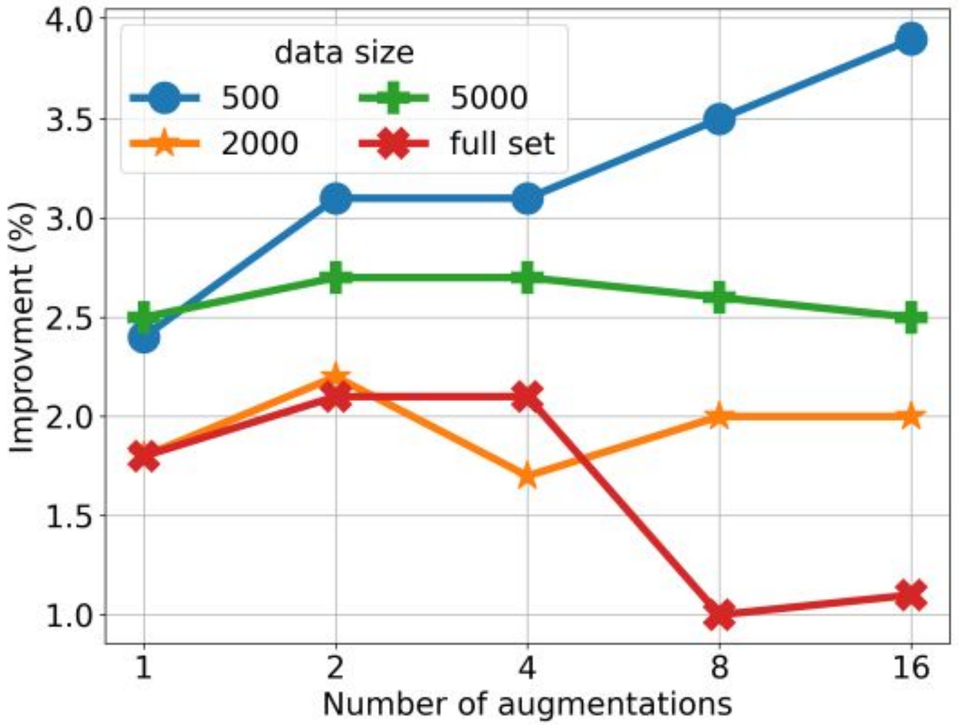
Original	a sad , superior human comedy played out on the back roads of life .
Aug 1	a sad , superior human comedy played out on the back roads ; of life ; .
Aug 2	a , sad . , superior human ; comedy . played . out on the back roads of life .
Aug 3	: a sad ; , superior ! human : comedy , played out ? on the back roads of life .

https://arxiv.org/pdf/2108.13230.pdf

Data Augmentation

AEDA: An Easier Data Augmentation Technique for Text Classification
EMNLP(2021)

Model	SST2	TREC
BERT	91.10	97.00
+EDA	90.99	96.00
+AEDA	**91.76**	**97.20**



Model	Training set size			
	500	2,000	5,000	full set
RNN	73.5	82.6	85.9	87.9
+EDA	76.1	81.3	85.2	86.5
+AEDA	77.8	83.9	87.2	88.6
CNN	76.5	83.8	87.0	87.9
+EDA	77.5	82.2	84.5	86.1
+AEDA	78.5	84.4	86.5	88.1
Average	75.0	83.2	86.5	87.9
+EDA	76.8	81.8	84.9	86.3
+AEDA	**78.2**	**84.2**	**86.9**	**88.4**

https://arxiv.org/pdf/2108.13230.pdf

Open source - KoEDA

```python
from koeda import EDA


eda = EDA(
    morpheme_analyzer="Okt", alpha_sr=0.3, alpha_ri=0.3, alpha_rs=0.3, prob_rd=0.3
)

text = "아버지가 방에 들어가신다"

result = eda(text)
print(result)
# 아버지가 정실에 들어가신다

result = eda(text, p=(0.9, 0.9, 0.9, 0.9), repetition=2)
print(result)
# ['아버지가 객실 아빠 안방 방에 정실 들어가신다', '아버지가 탈의실 방 휴게실 에 안방 탈의실 들어가신다']
```

https://github.com/toriving/KoEDA

Open source - KoEDA

```python
from koeda import AEDA


aeda = AEDA(
    morpheme_analyzer="Okt", punc_ratio=0.3, punctuations=[".", ",", "!", "?", ";", ":"]
)

text = "어머니가 집을 나가신다"

result = aeda(text)
print(result)
# 어머니가 ! 집을 , 나가신다

result = aeda(text, p=0.9, repetition=2)
print(result)
# ['! 어머니가 ! 집 ; 을 ? 나가신다', '. 어머니 ? 가 . 집 , 을 , 나가신다']
```

https://github.com/toriving/KoEDA

EDA, AEDA 결과

문법성이 더 강조되는 TASK - WiC, COLA 에는 성능이 10%정도 하락함.
문맥을 파악하는 BoolQ, COPA에는 성능이 5%정도 하락함.

한국어에는 성능향상에 도움이 되지 못한다는 것을 인지.

Pivot translation

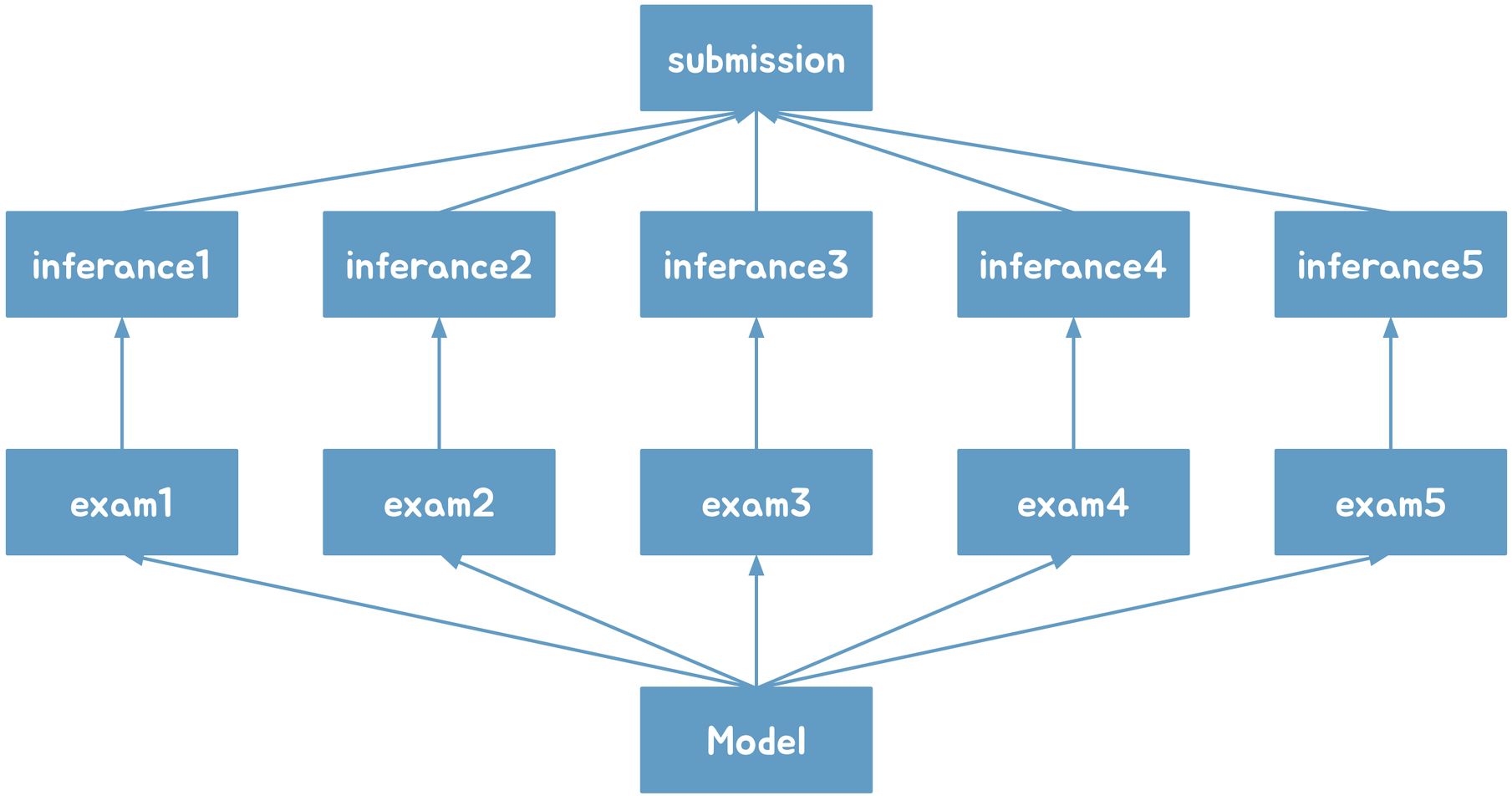
문맥을 파악하는 task(BoolQ, COPA)에 대해서는 확실한 성능 향상이 있었음.

COLA에 대해서는 문법성을 갖는 답변을 생성하기 어려움.
반대로 문법성이 틀린 것에 진행하면 정답인 데이터로 만들어줌. -> negative label 을 positive label로 늘릴때 사용.

WiC에 대해서는 동형이의어의 문법성을 가져가지 못함.

https://github.com/kakaobrain/pororo

Ensemble – hard voting



DeepSpeed

모델을 학습할 때 GPU 사용시에 학습에 항상 필요하지 않는 메모리를 올려놓는 비효율성이 있다.

- 이를 효율적으로 사용 가능하게 한 기법이다.
- 학습 시 fp 32를 16으로 바꾸고 loss scaling 하고 zero_optimization stage 2를 사용하여
  학습 시 batch size를 늘려 속도를 빠르게 하고 모델 파라미터의 크기를 낮추었다.

- DeepSpeed 미적용 시
  - skt/ko-gpt-trinity-1.2B-v0.5 모델과 train data사용시 batch 2로 동작
- DeepSpeed 적용시
  - Batch 8로 동작

GPU RTX3090 1대

https://www.deepspeed.ai/

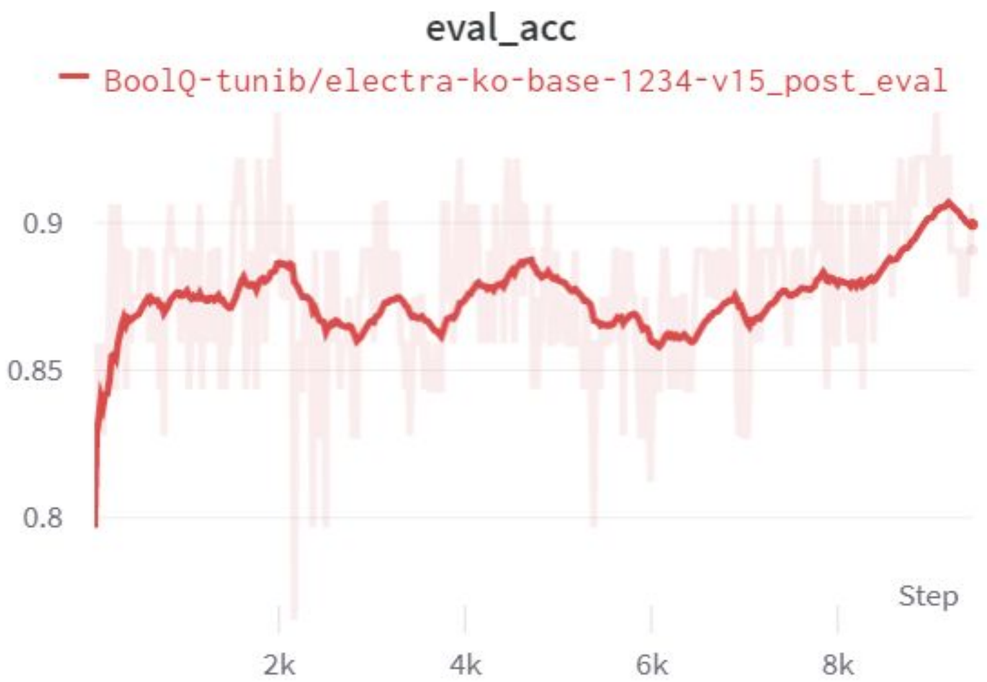
DeepSpeed

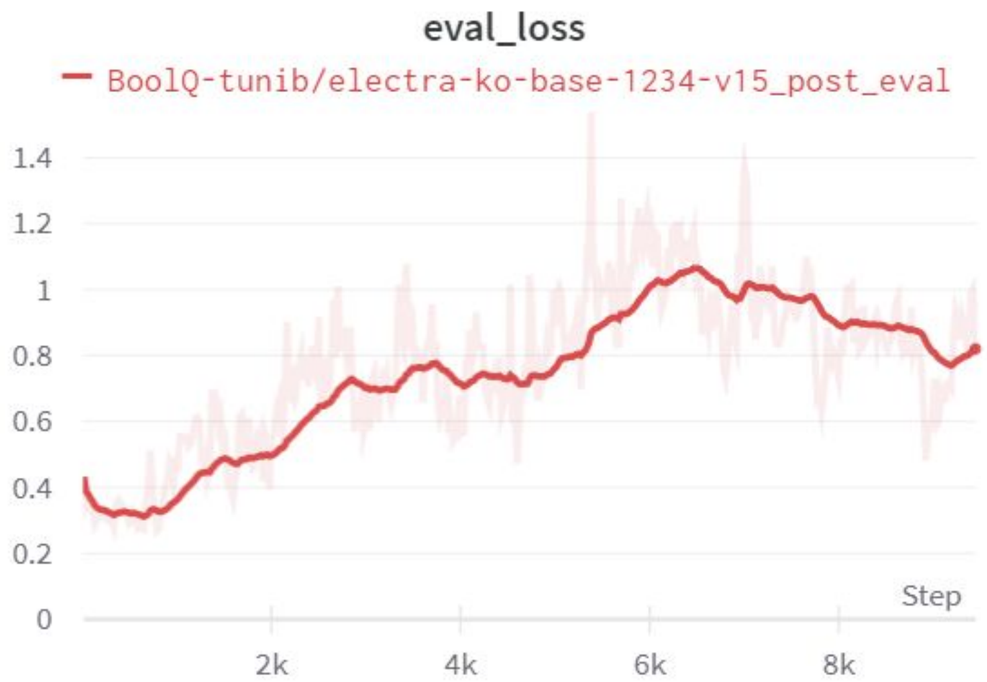
- Optimizer : CPUAdam
- Learing rate : 3e-5
- Gradient_clipping : 1.0
- Weight decay : 3e-7
- Optimizer scheduler : warmupLR

https://www.deepspeed.ai/

1. 실험준비 Model monitoring

Wandb를 사용해서 모델의 로그들(acc, loss, epochs....)을 모니터링 하였다.





https://wandb.ai/site

Chapter 02

Task1. BoolQ

2. BoolQ

모델 구동 과정

모델 구조 huggingface의 Pretrained Language Model(PLM)들을 가져와서 fine-tuning 하였다.

그리고 binary classification 문제임으로 AutoModelForSequenceClassification 를 사용하였다.

모델의 train 사이에 eval step단위로 진행시켜 적은 데이터 안에서 최적의 weight를 찾으려고 하였다.

2. BoolQ

Data Augmentation

기존의 데이터 수가 별로 없어서 BoolQ의 영어 데이터를 번역해서 사용하였다.

kor BoolQ data

train 3655 dev 700 test 704

eng BoolQ data

train 9427 dev 3270 test 3245 -> train, dev만 가져와서 번역 진행

총 12627개 데이터 생성.

추가로 BoolQ train data pivot translation 진행

총 3665 + 3665 + 12,627 = 19,957

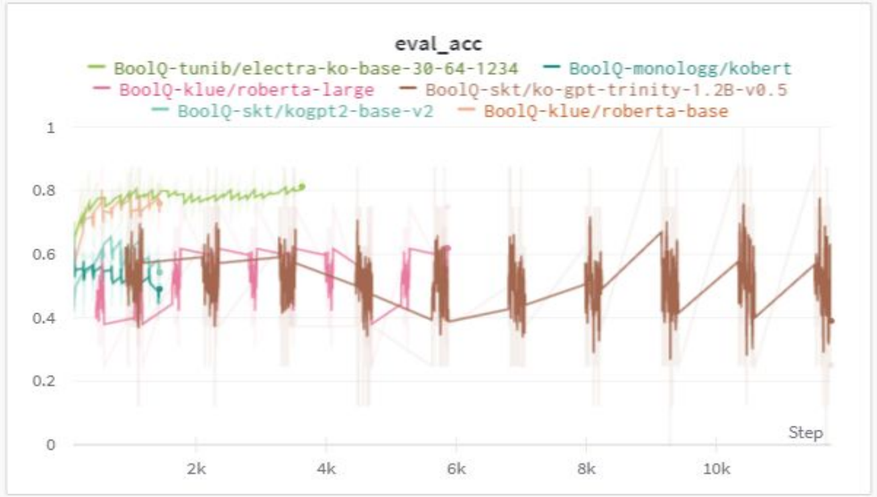
-> out domain data에도 robust

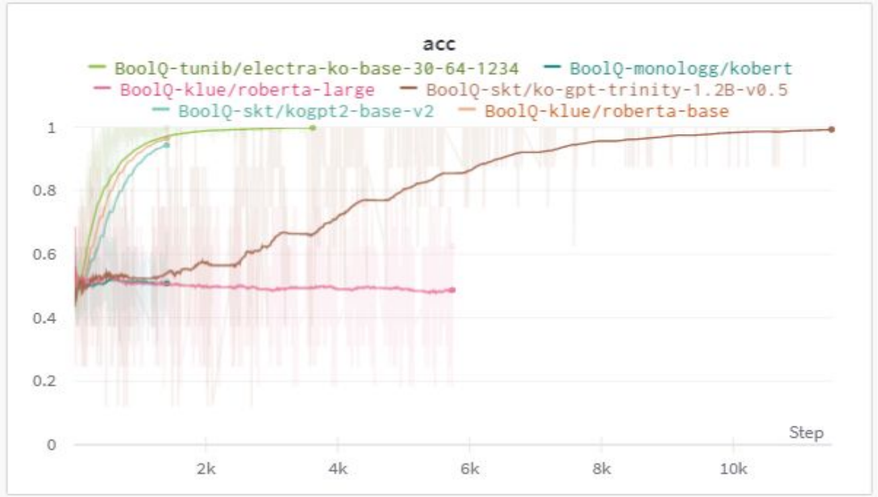
https://github.com/google-research-datasets/boolean-questions

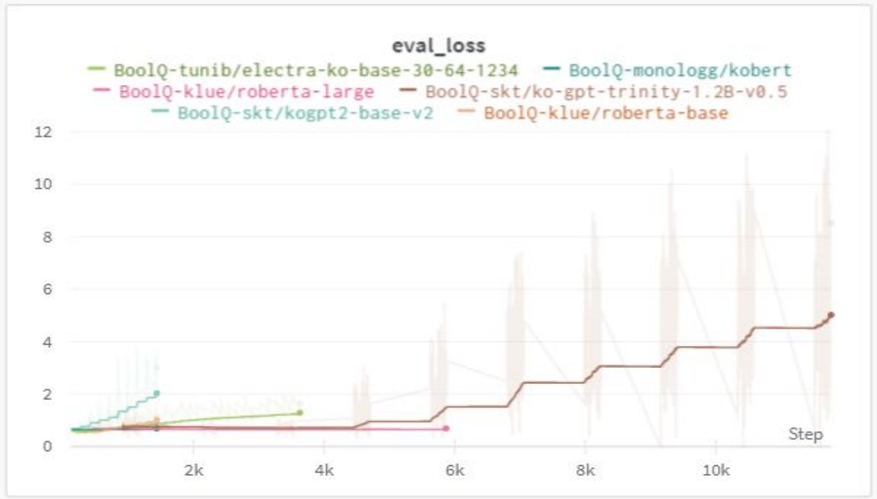
2. BoolQ

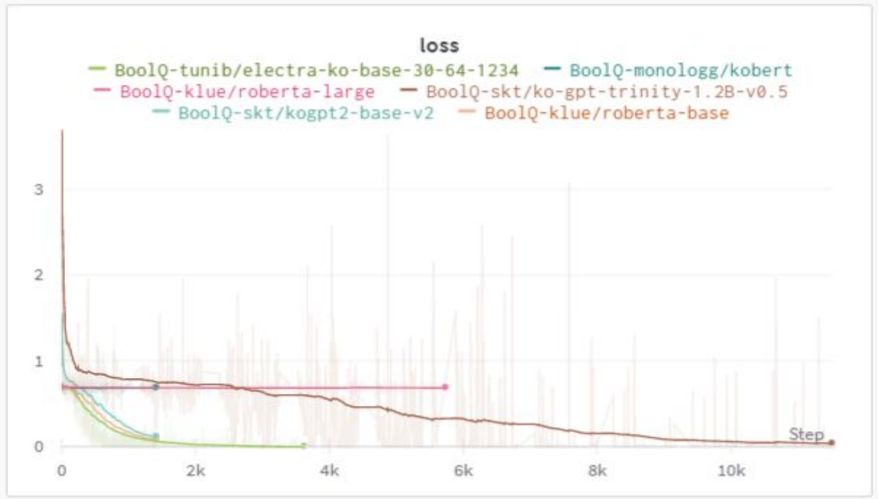
모델 개발 과정

PLM을 어떤것을 사용할지 실험 -> ELECTRA model 선택.



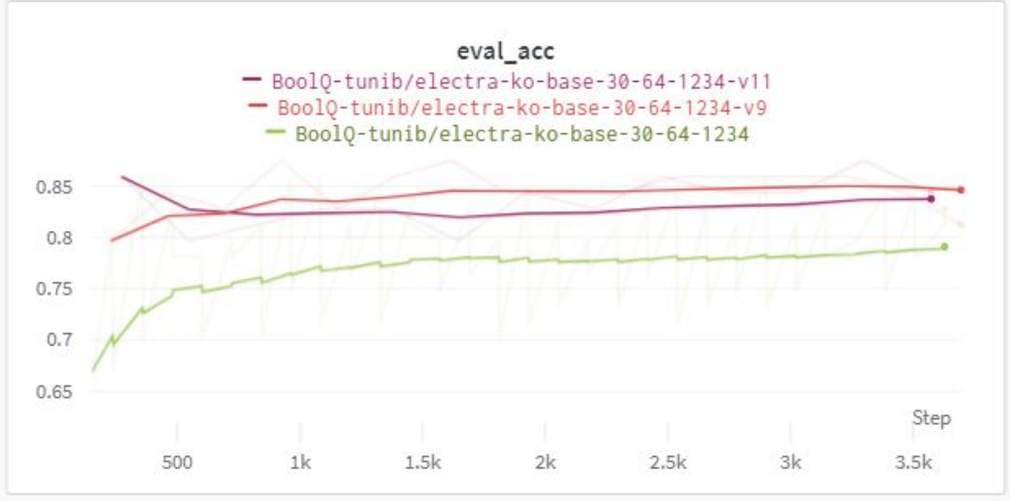


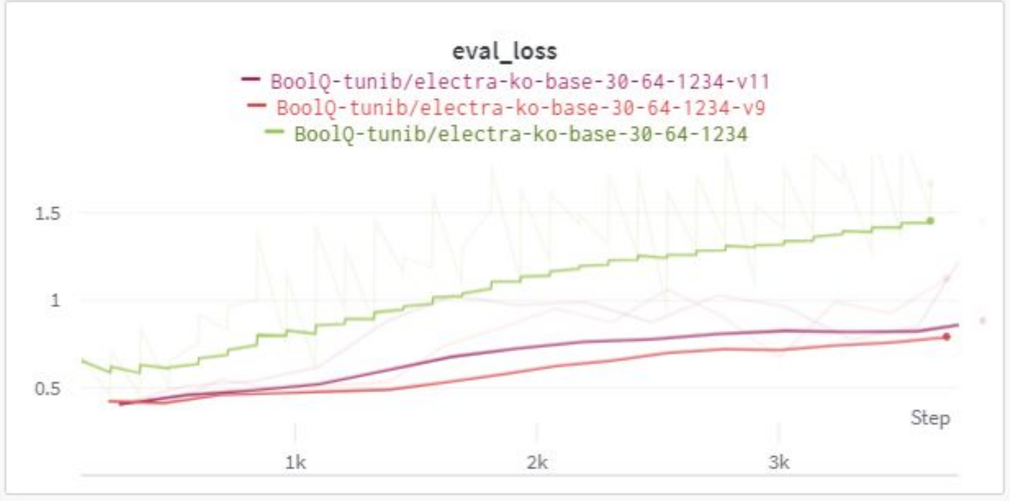


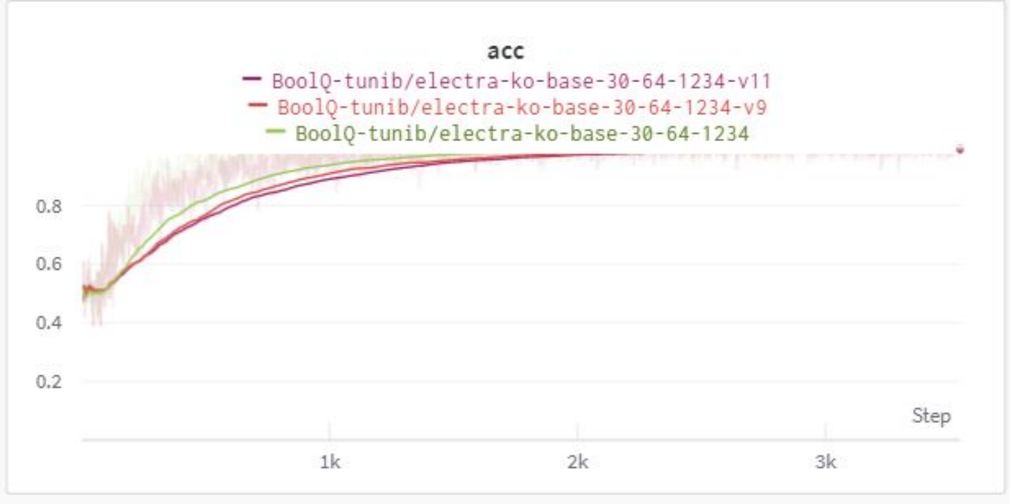


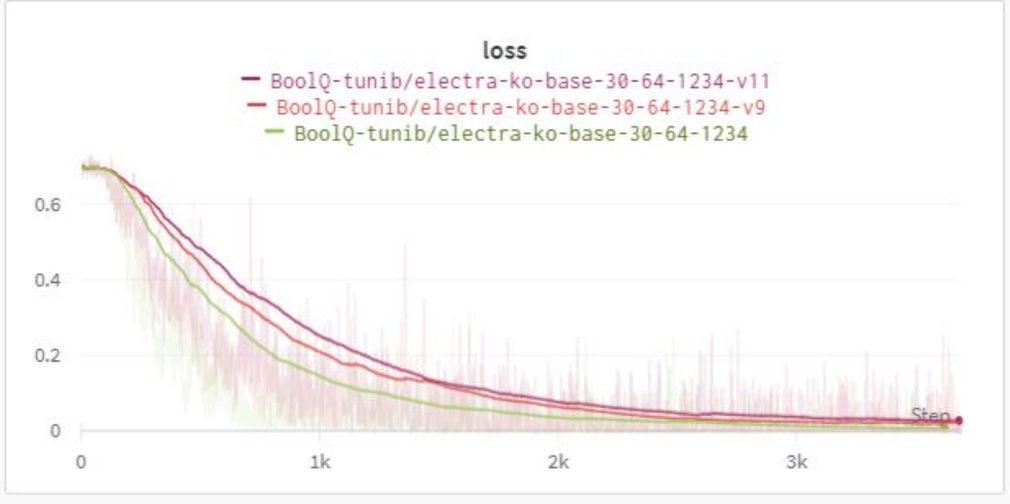
2. BoolQ 모델 개발 과정

base = train v9 = train + pivot translation v11 = train + pivot translation + eng translation data









2. BoolQ

Post Train

다음과 같은 구조로 post-training을 하였다.

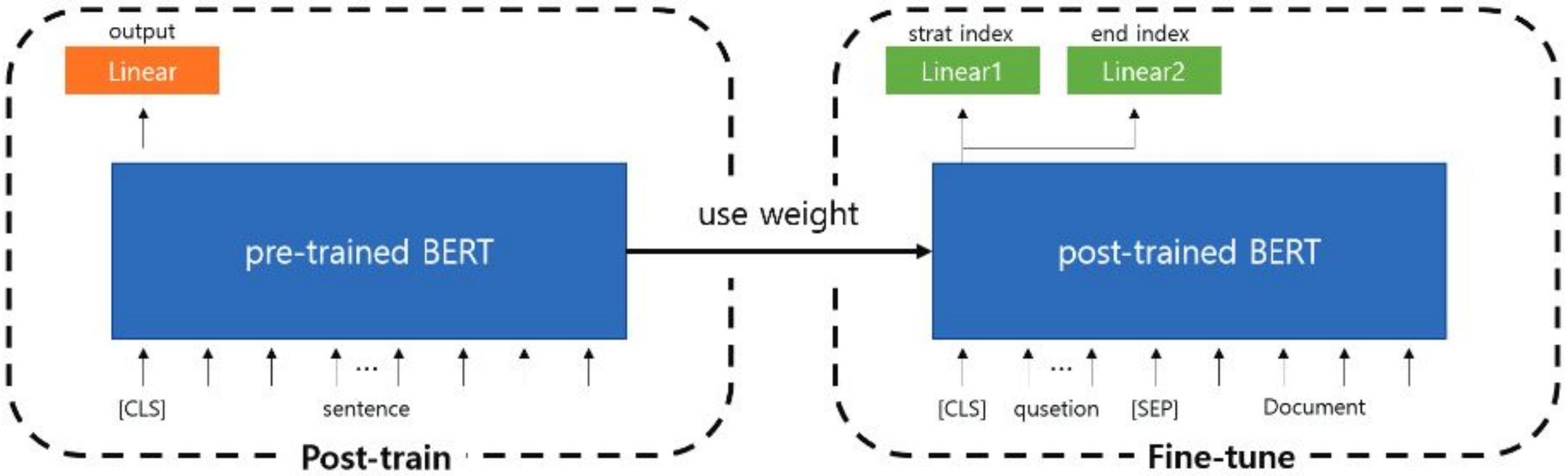


그림 1) 제안 모델의 전체 과정

https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE10582912

2. BoolQ

Data Augmentation

Post train data – AI HUB 기계독해 데이터 10만개 추출.
Train data와 sequence length 를 맞춤.


```
{"data": [{
    "source": 6,
    "paragraphs": [{
        "qas": [{
            "question": "쎈 마이웨이 관련 기자간담회 누가 했어",
            "id": "m4_278529-1",
            "answers":[{
                "answer_start": 0,
                "text": "박영선"
            }],
            "clue":[{
                "clue_start": 4,
                "clue_text": "PD"
            }],
            "classtype": "work_who"
            }],
        "context": "박영선 PD는 18일 오후 서울 양천구 목동 SBS에서 모비딕의 토크 콘텐츠 쎈 마이웨이
                    관련 기자간담회를 열고 출연진에 신뢰를 드러냈다."
    }],
    "title": "1" }
```


-> positive 생성

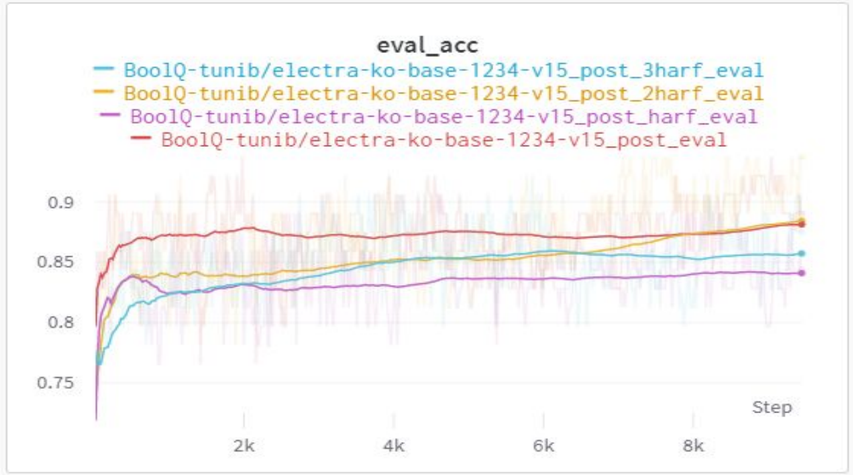

```
"data": [
    {
      "time": "20201029154926",
      "title": "젠더리뷰 2016년 봄호 제40호",
      "agency": "한국여성정책연구원",
      "year": "2016",
      "content_id": "PCY_201707120256306131",
      "KDC": "00",
      "paragraphs": [
        {
"context": "아동학대범죄처벌법에서는 피해아동 및 학대행위자는 피해아동보호명령사건에 대하여 각자 보
          "qas": [
            {
              "question": "판사의 필요 인정이 있으면 법원의 직권으로 변호사를 학대행위자의 누구로
              "id": 997765,
              "is_impossible": false,
              "answers": [
                {
                  "text": "보조인",
                  "answer_start": 357
                }
              ]
            },
```

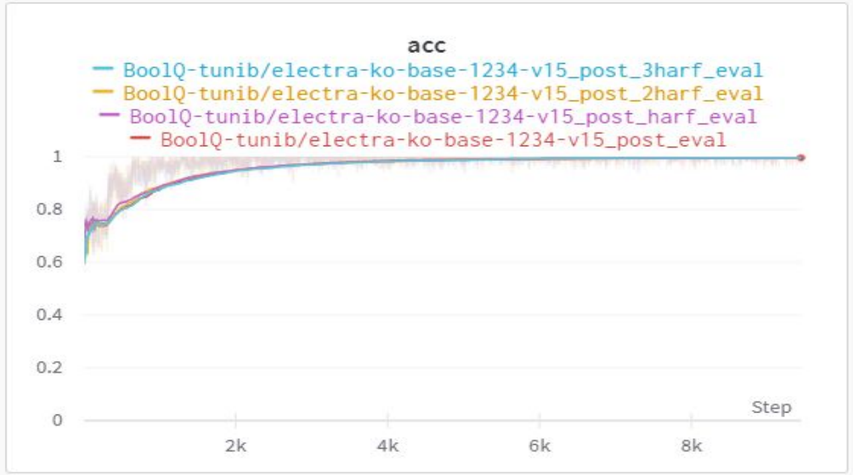

-> negative 생성

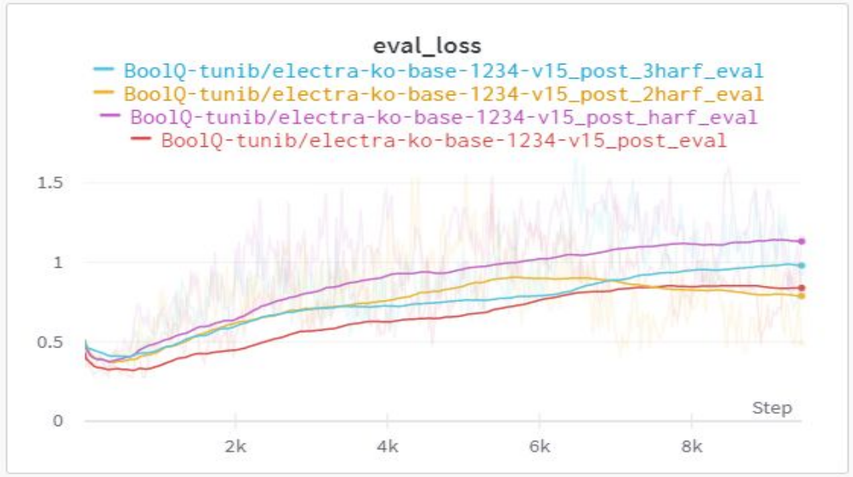
https://aihub.or.kr/aidata/86

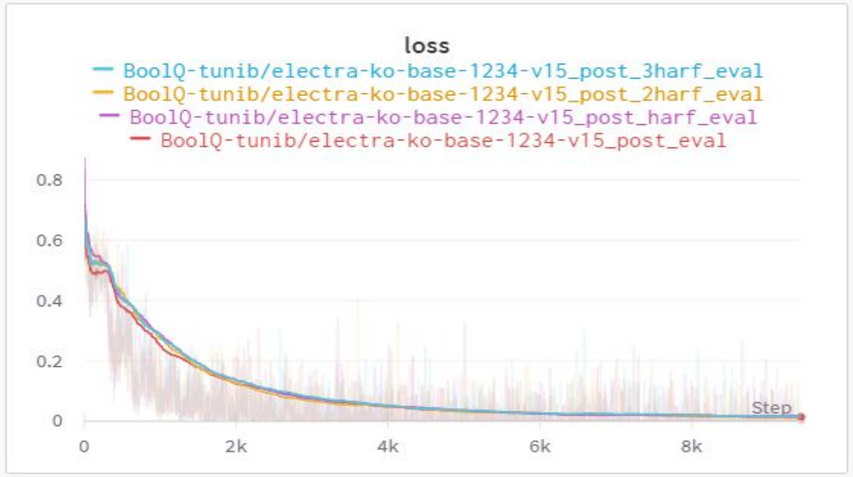
2. BoolQ 모델 개발 과정

Post train data에 따른 성능 차이이다. 10만개를 썼을때 성능이 좋았다! 데이터는 많을수록 좋다!



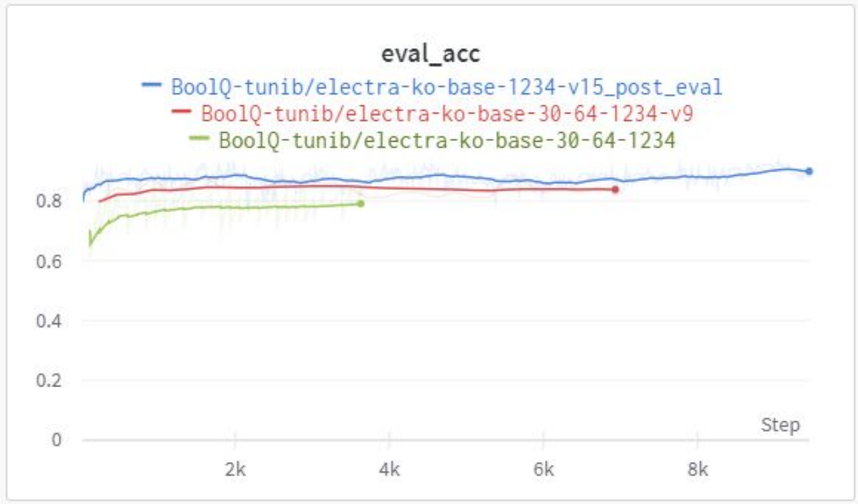


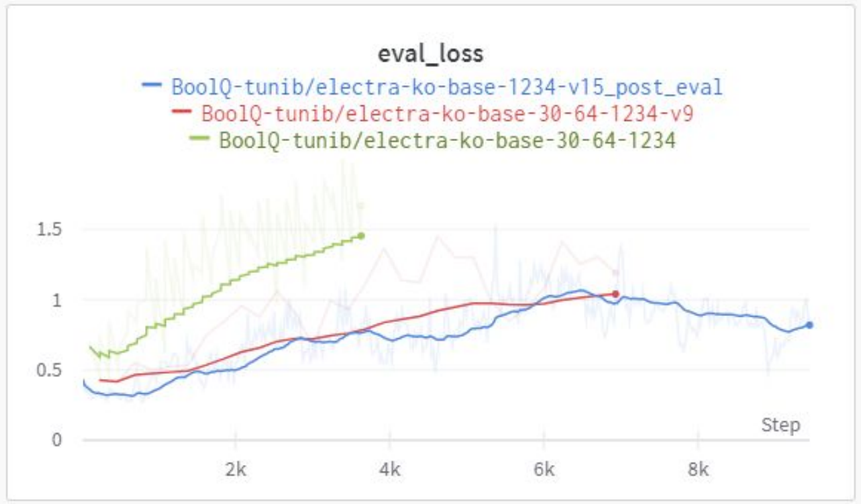


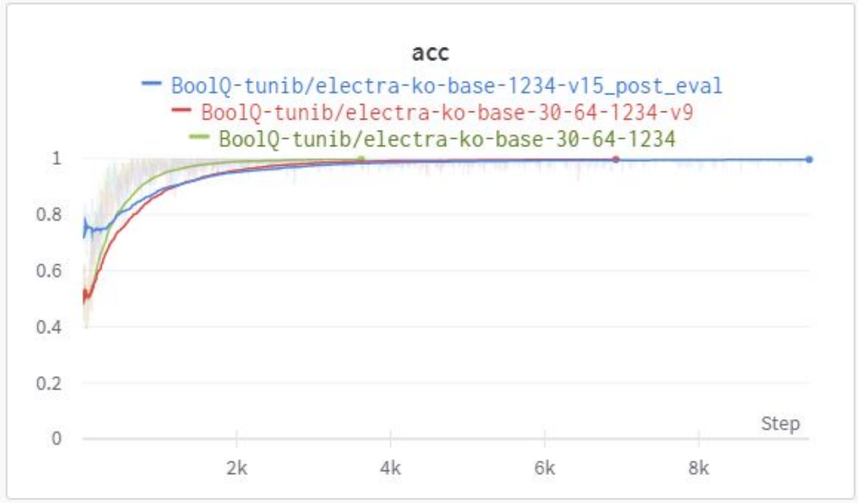


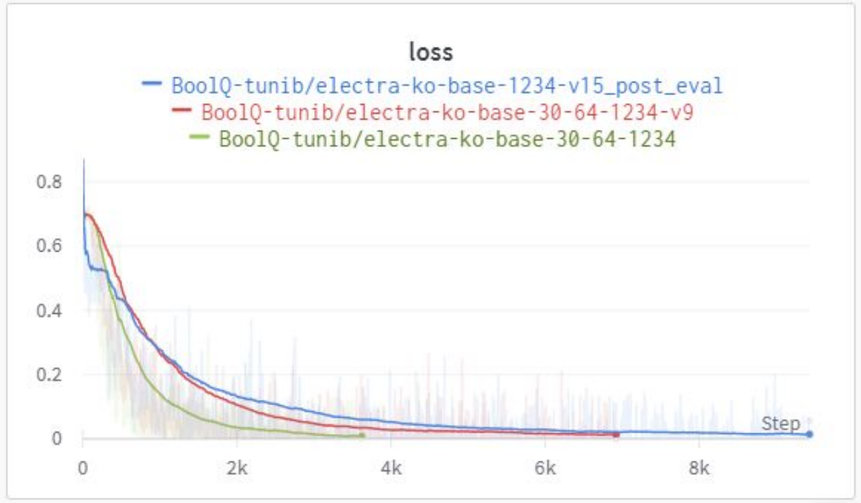
2. BoolQ 모델 개발 과정

Post train을 사용한 것과 하지 않은 것에 대한 차이가 있음을 볼 수 있다.









앙상블 결과

다른 데이터를 활용한 모델 중 준수한 성능의 모델들의 결과로 hard voting 적용.

- **앙상블 후보 별 활용한 학습 데이터**
  각각의 모델과 학습 데이터가 다른 결과들을 앙상블.

# version	활용한 학습데이터 종류	총 학습 데이터 수
tunib electra v15	Post_train 10만 + train + train_pt	10만개 + 7330개
tunib electra v33	Post_train 10만 + train + train_pt + eng_data_translation	10만개 + 7330개 + 12697개
monologg koelectra v30	Post_train 10만 + train + train_pt	10만개 + 7330개
monologg koelectra v35	Post_train 10만 + train + train_pt + eng_data_translation	10만개 + 7330개 + 12697개
monologg koelectra v30	Post_train 10만 + train + train_pt	10만개 + 7330개

위의 5개 test셋에 대한 예측결과를 앙상블한 결과를 최종 산출물로 제출.

ACC = 89.63

Chapter 03

Task2. WiC

3. WiC

모델 구동 과정

모델 구조 huggingface의 Pretrained Language Model(PLM)들을 가져와서 fine-tuning 하였다.

그리고 binary classification 문제임으로 AutoModelForSequenceClassification 를 사용하였다.

모델의 eval을 step단위로 진행시켜 적은 데이터 안에서 최적의 weight를 찾으려고 하였다.

3. WiC

Data Augmentation

추가 데이터

동음이의어 사전을 참고해서 데이터를 생성.

총 2325개의 data를 train data형식과 맞춰서 생성하였다.

기존 데이터

train 7748 dev 1166 test 1246

총 train data - 10,073개

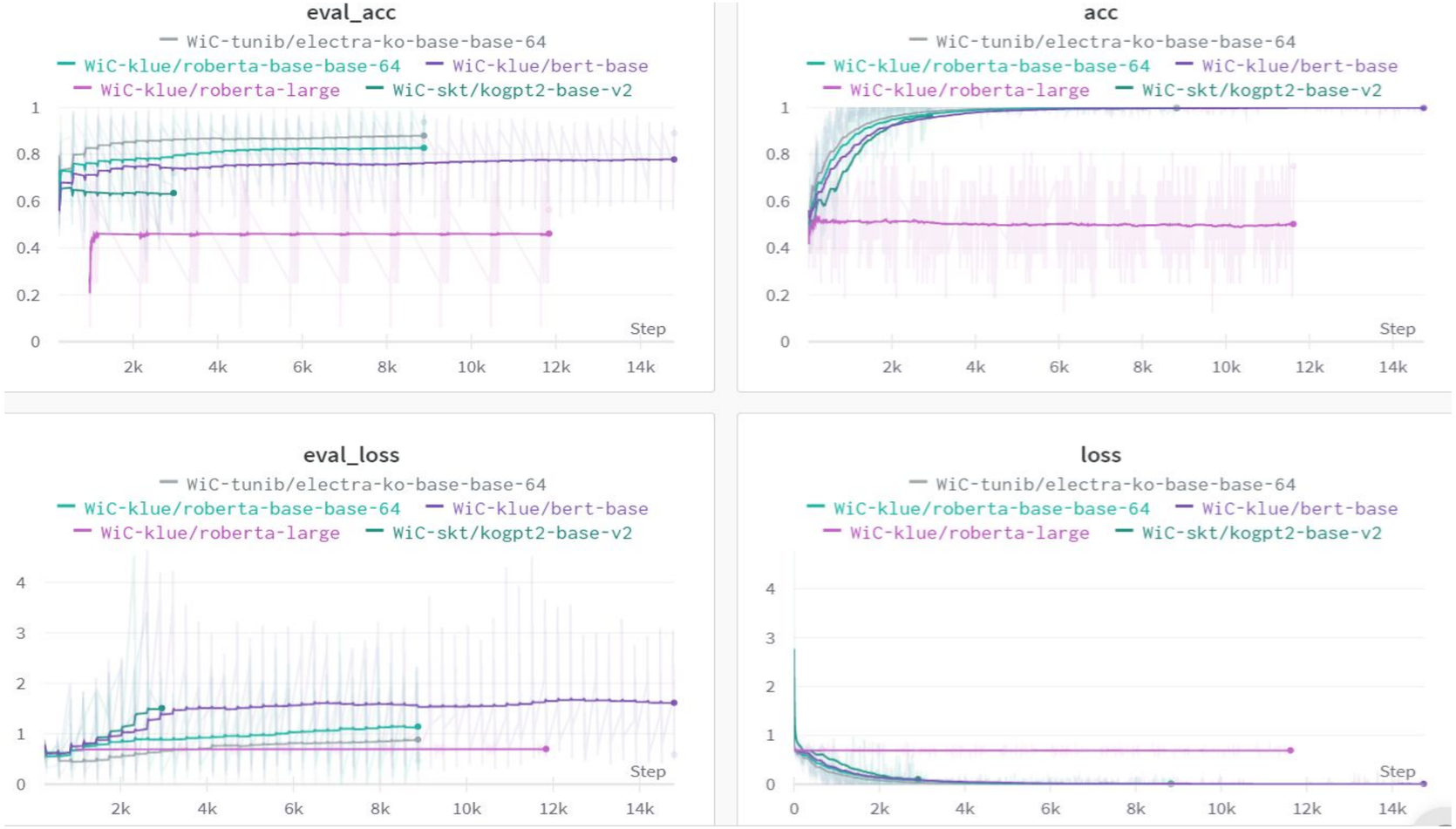
-> out domain data에도 robust

https://lib.seoul.go.kr/search/detail/CAT000001401351

3. WiC

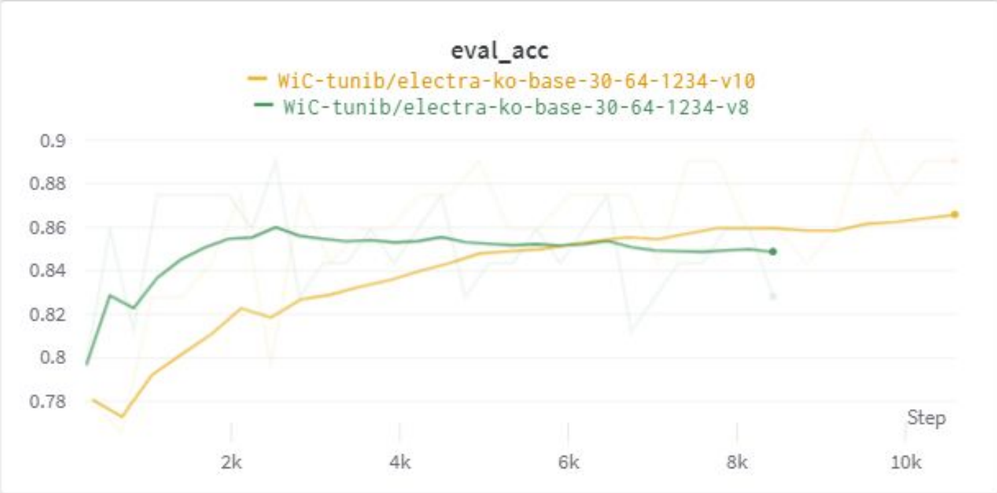
모델 개발 과정

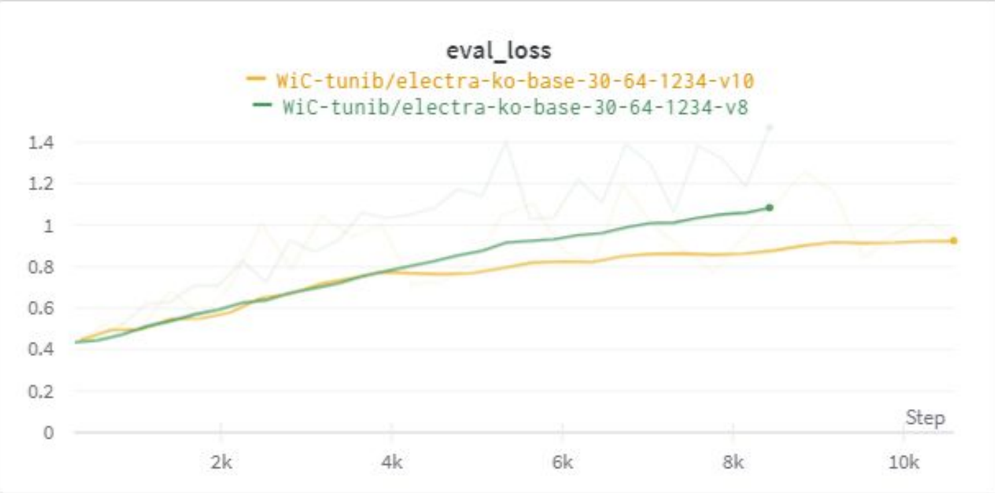
PLM을 어떤것을 사용할지 실험 -> ELECTRA model 선택.

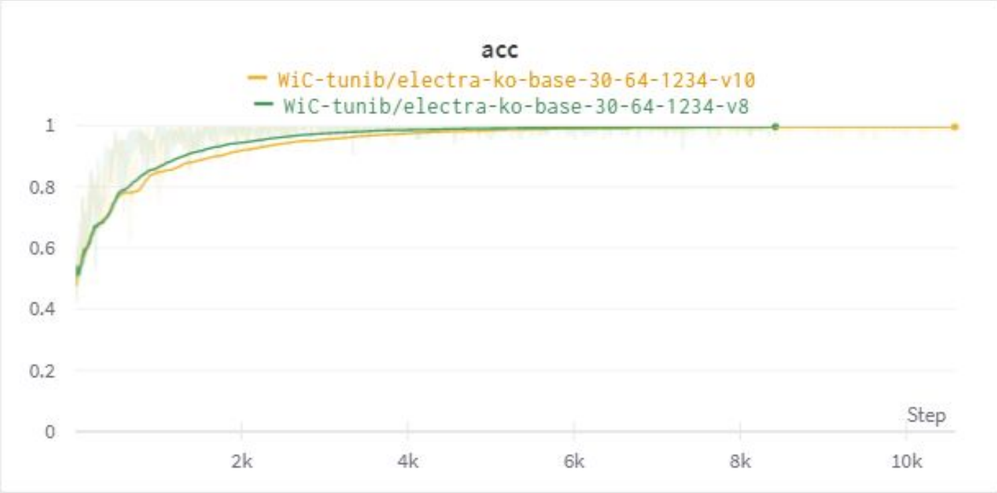


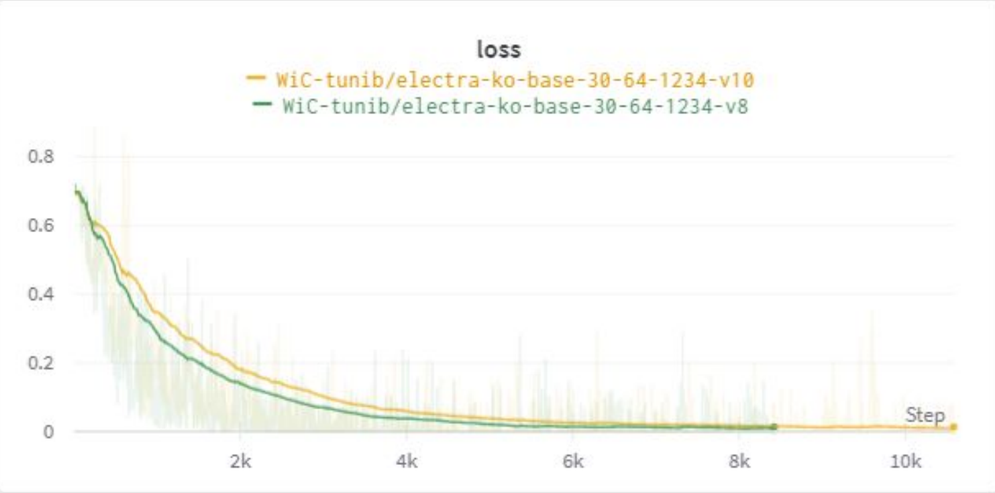
3. WiC 모델 개발 과정

v8 = train data만 사용 vs v10 = train data + 추가데이터









앙상블 결과

같은 데이터를 활용한 모델 중 준수한 성능의 모델의 결과로 hard voting 적용.

- **앙상블 후보 별 활용한 학습 데이터**
  *모델들을 구분짓기 위해 다른 random_seed와 step별 epochs 적용 (rs: random_seed)

# rs/epoch	활용한 학습데이터 종류	총 학습 데이터 수
rs2/11	train + 추가데이터	10,073개
rs4/25	train + 추가데이터	10,073개
rs4/26	train + 추가데이터	10,073개
rs8/11	train + 추가데이터	10,073개
rs1234/16	train + 추가데이터	10,073개

위의 5개 test셋에 대한 예측결과를 앙상블한 결과를 최종 산출물로 제출.

→ **ACC = 91.65**

Chapter 04

Task3. COLA

모델 개발 과정

문법성 판단 과제는 다른 과제에 비해 데이터 수가 많은 편이었지만,

데이터와 과제의 특성 상, 모델 구조를 특별하게 사용하기 힘들고 한국어 문법의 모호함을 고려해

해당 과제에서는 더 많은 데이터가 학습에 필요하다고 생각해 추가적인 학습데이터를 확보했다.

Aa 이름	train	dev	test
BoolQ	3665	700	704
CoLA	15876	2032	1060
COPA	3080	500	500
WiC	7748	1166	1246

모델 개발 과정

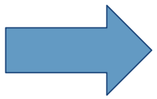
대회 데이터셋과 카카오에서 공개한 korSTS 데이터셋을 활용해

비슷한 길이와 형식의 문법적으로 틀린/맞는 예시 각각 5524개씩 추가 생성했다.

데이터는 작업자가 문법성 여부를 검수했다.

단일모델(train-test)

MCC = 0.5399



단일모델(train+추가데이터-test)

MCC = 0.6102

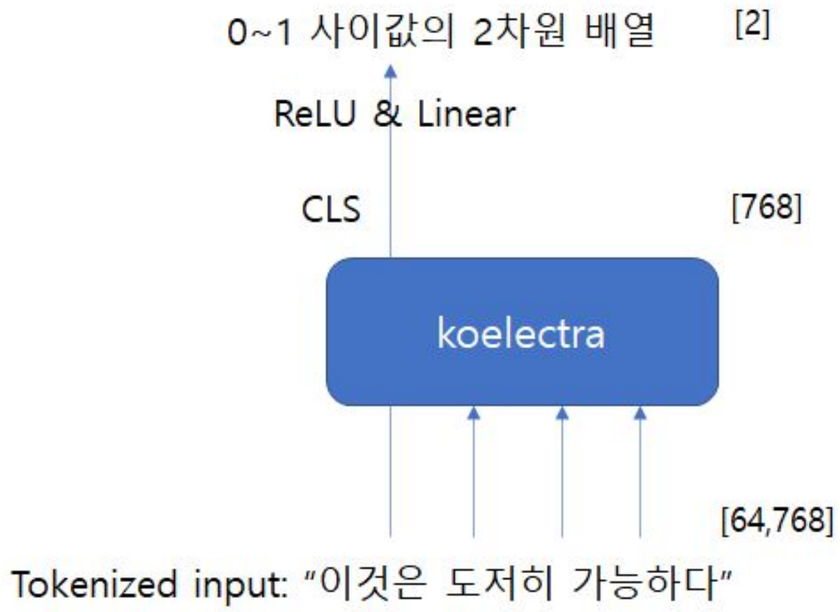
모델의 최적의 학습 시점을 찾기 위해 epoch 내에서 1000개의 데이터 학습 시 마다 모델의 성능평가를 진행하였다.

https://github.com/kakaobrain/KorNLUDatasets

모델 추론 과정

학습한 모델의 pt파일을 불러와서 모델 업데이트 후, 모델출력에서 얻은 0/1의 값을 모델 예측값으로 제공.

- Model Input
  - Dataloader에서 입력 시퀀스 길이는 64로 제한.
  - input data = sentence
- Model Output
  - 모델 출력부에서 CLS 토큰 위치의 출력만 사용.
  - 활성화 함수로 ReLU를 사용하고 nn.Linear(768,2)의 MLP를 통해 선형변환.

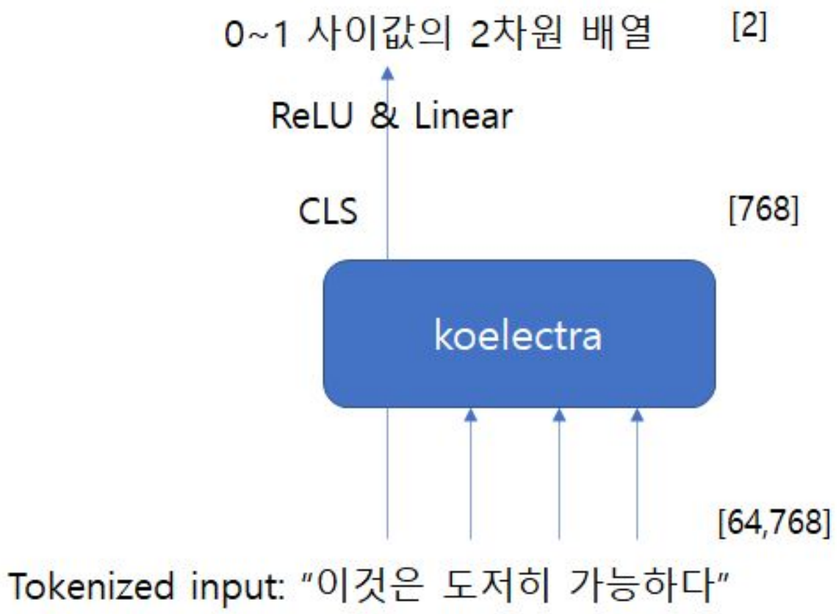


모델 학습 과정

단일 sequence 입력에 대한 binary classification 모델 구조.

2차원 모델 출력에서 nn.Argmax()로 얻은 0/1의 값을 정답라벨과 비교해 역전파했다.

- **Model Output**
  - 모델 출력부에서 CLS 토큰 위치의 출력만 사용.
  - 활성화 함수로 ReLU를 사용하고 nn.Linear(768,2)의 MLP를 통해 선형변환.
  - Loss 함수 = Cross Entropy Loss
  - Optimizer: AdamW(lr=1e-5)



모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- **main: koelectra_COLA.py**
  - main 함수의 역할로, train/eval/inference 과정과 다른 모듈을 불러오는 부분.

- **koelectra_datasets.py**
  - 데이터셋을 읽은 후, 모델의 입력 형태로 데이터 리턴.

- **koelectra_models.py**
  - Dataloader로 입력받은 데이터에 대한 모델의 출력을 리턴.

- **utils.py**
  - 각 모듈에서 사용할 여러 함수와 변수 등을 정의해서 사용.

모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- koelectra_datasets.py
  - 데이터셋을 읽은 후, 모델의 입력 형태로 데이터 리턴.

```
class COLA_dataset(Dataset):
    def __init__(self, data_filename): #str: path of csv
        super(COLA_dataset,self).__init__()
        self.data = pd.read_csv(os.path.join(os.getcwd(),data_filename), sep="\t")
        self.max_tokenizer_length = TOKEN_MAX_LENGTH['COLA']
        self.tokenizer = model_tokenizer.from_pretrained(model_path)
```

```
def __getitem__(self, item): #fetch data sample(row) for given key(item)
    sentData = self.data.iloc[item,:]

    tok = self.tokenizer(sentData['sentence'], padding="max_length", max_length=20, truncation=True)

    label=sentData['acceptability_label']

    input_ids=torch.LongTensor(tok["input_ids"]) #input token index in vacabs
    token_type_ids=torch.LongTensor(tok["token_type_ids"]) #segment token index
    attention_mask=torch.LongTensor(tok["attention_mask"]) #boolean: masking attention(0), not masked(1)

    return input_ids, token_type_ids, attention_mask, label #tensor(400), tensor(400), tensor(400), int
```

'sentence'와 'acceptability_label'을 모델 입력과 라벨로 사용.

모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- koelectra_models.py
  - Dataloader로 입력받은 데이터에 대한 모델의 출력을 리턴.

2차원으로의 linear MLP 정의

CLS 토큰을 사용해
binary classification 진행.

```python
class model_COLA(nn.Module):
    def __init__(self):
        super(model_COLA, self).__init__()
        self.model_PLM = model_class.from_pretrained(model_path)
        self.relu = nn.ReLU() #
        self.linear = nn.Linear(768,2) #CLS: 200->2(binary)

    def forward(self, input_ids, token_type_ids, attention_mask):
        iIds = input_ids.long()
        tok_typeIds = token_type_ids.long()
        attMask = attention_mask.long()

        output=self.model_PLM(input_ids=iIds, token_type_ids=tok_typeIds, attention_mask=attMask)
        output=output['last_hidden_state'][:, 0, :] #CLS token: torch.Size([batch_size, 768])

        self.relu(output)
        output=self.linear(output)

        return output
```

모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- main: koelectra_COLA.py
  - main 함수의 역할로, train/eval/inference 과정과 다른 모듈을 불러오는 부분.

모델 출력 중 큰 값의 index를 모델 예측결과로 사용.

1000개의 학습데이터 단위로 모델 성능평가 진행.

```
output = model(input_ids, token_type_ids, attention_mask)

loss = lf(output,label)
pred = torch.argmax(output,dim=-1)
correct += sum(pred.detach().cpu()==label.detach().cpu())
all_loss.append(loss)
loss.backward() #기울기 계산
optimizer.step() #가중치 업데이트
mini_batch += batch_size
if mini_batch%1000 == 0:
    if bool_save_model:
        filename_modelSave_chkpt = f'{task_name}_{model_name}_rs{random_seed_int}_epoch{countEpoch}
        save_model(model_savePth, filename_modelSave_chkpt, mymodel, optimizer)

    print(f'batch{mini_batch}:',end='')
    result, accuracy = eval_cola(mymodel, eval_loader, bs, device)
    mcc_list.append(result)
    acc_list.append(accuracy)
    model.train() #set model training mode
```

앙상블 결과

다른 데이터를 활용한 모델 중 준수한 성능의 단일 모델의 결과들을 사용해 hard voting을 적용했다.

- **앙상블 후보 별 활용한 학습 데이터**

  *학습 시 사용한 데이터를 간에 구분짓기 위해 활용한 데이터 별 다른 random_seed 적용(rs: random_seed)

# rs/epoch	활용한 학습데이터 종류(데이터 수)	총 학습 데이터 수
rs13/3	train(15876개)	총 15876개
rs73/5	train(15876개) + train데이터의 pivot-translation 데이터(15876개)	총 31752개
rs99/3	train(15876개) + 추가 학습데이터(0/1 라벨의 데이터 각 5244개씩, 총 10488개)	총 26364개
rs2/7	train(15876개) + 추가 학습데이터(10488개) + train데이터의 pivot-translation 데이터(15876개)	총 42240개
rs3/8	train(15876개) + 추가 학습데이터(10488개) + 전체 데이터의 pivot-translation 데이터(15876개 + 10488개)	총 52728개

위의 5개 test셋에 대한 예측결과를 앙상블한 결과를 최종 산출물로 제출.

→ **MCC = 0.6319**

Chapter 05

Task4. COPA

모델 개발 과정

- 전체 과제 중 제공된 데이터의 수가 가장 적어서 추가적인 데이터 확보를 먼저 시도해봤으나,
  알맞은 데이터를 찾기 어려워 SuperGLUE COPA의 데이터를 번역하여 1000개의 추가 데이터를 생성했다.

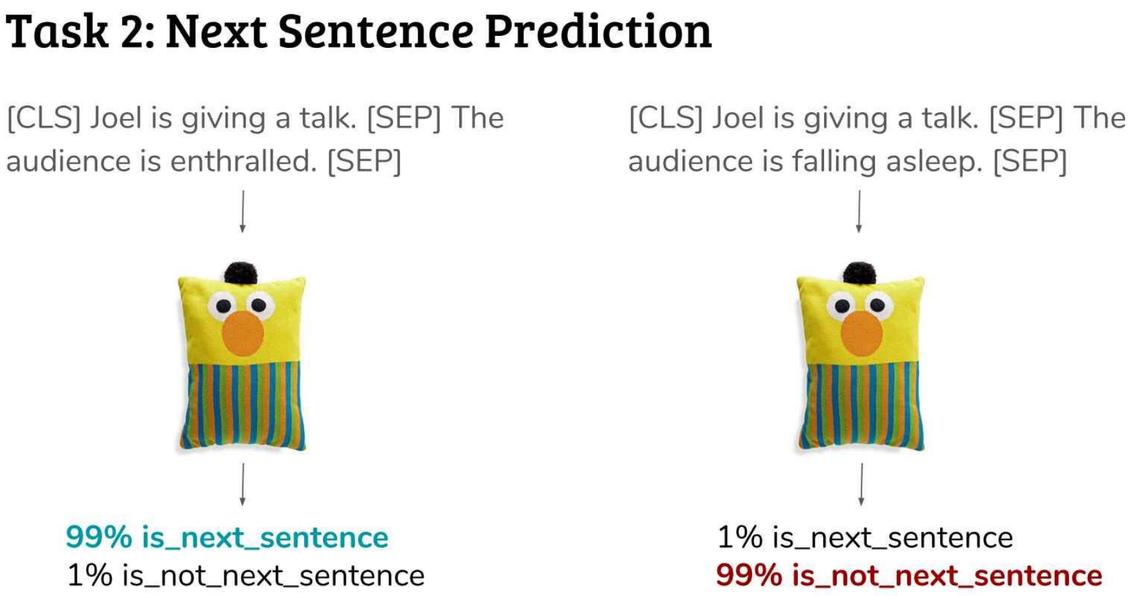
Aa 이름	train	dev	test
BoolQ	3665	700	704
CoLA	15876	2032	1060
COPA	3080	500	500
WiC	7748	1166	1246

COPA 과제의 경우는 다른 과제에 비해 학습데이터 수가 비교적 적어,
epoch 내에서 500개의 데이터 학습 시 마다 모델의 성능평가를 진행하였다.

https://people.ict.usc.edu/~gordon/copa.html

모델 개발 과정: Next Sentence Prediction

- sentece에 대해서 문장 1,2 중 question에 따라 다음 문장으로 왔을 때 더 적합한 문장을 찾는 과제.
- BERT(2018) 등 모델의 사전학습 과제인 NSP(Next Sentence Prediction) task와 유사하다고 생각해 두개의 모델에 문장 1,2를 넣어 sentence 문장에 대한 NSP 결과를 예측하는 구조의 모델을 구성.



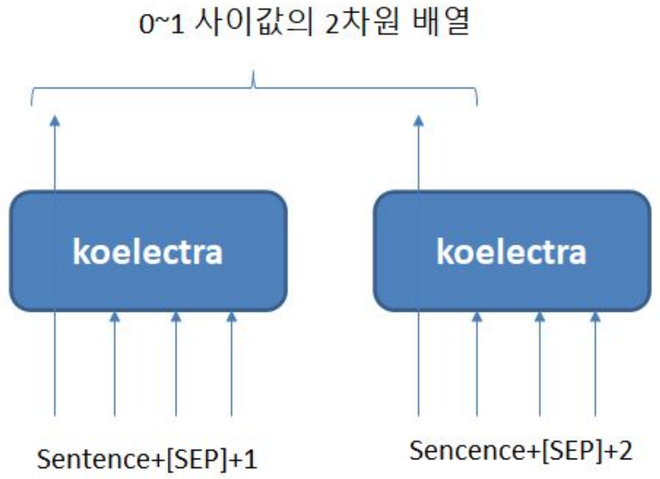
Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

모델 추론과정

학습한 모델의 pt파일을 불러와서 모델을 업데이트한 후, 각 모델의 출력을 concat해서 중 큰 값의 index를 모델 예측값으로 사용.
(nn.Argmax()로 얻은 0/1의 값)

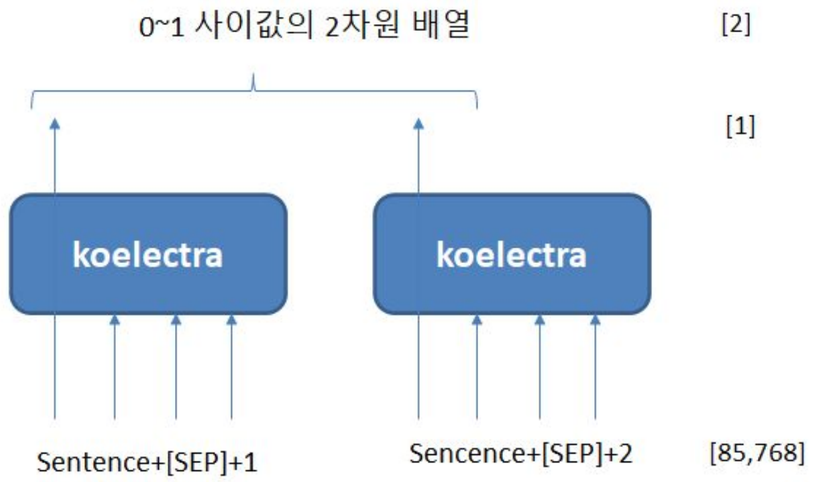
- **Model Input**
  - 데이터에서의 문장(sentence,1,2)에 대해 각각의 시퀀스 길이는 40으로 제한.
  - 시퀀스 두개에 스페셜 토큰의 길이까지 고려해 각 모델의 입력(data) 길이는 <u>총 85로 제한.</u>
  - data = Sentence+[SEP]+1 & Sentence+[SEP]+2
- **Model Output**
  - 활성화 함수로 Sigmoid를 사용하고 nn.Linear(768,1)의 MLP를 통해 선형변환.
  - <u>두 NSP모델의 1차원 출력을 concat하여 사용.</u>



모델 학습과정

두개의 NSP모델의 출력된 확률값을 concat하여 binary classification하는 모델의 구조.

- Model Output
  - 활성화 함수로 Sigmoid를 사용해 0~1 사이의 값으로 변환.
  - nn.Linear(768,1)의 MLP를 통해 선형변환한 1차원 확률을 concat하여 사용.
  - label = Answer
  - Loss 함수 = Cross Entropy Loss
  - Optimizer: Adam(lr=2e-5, eps=1e-8)

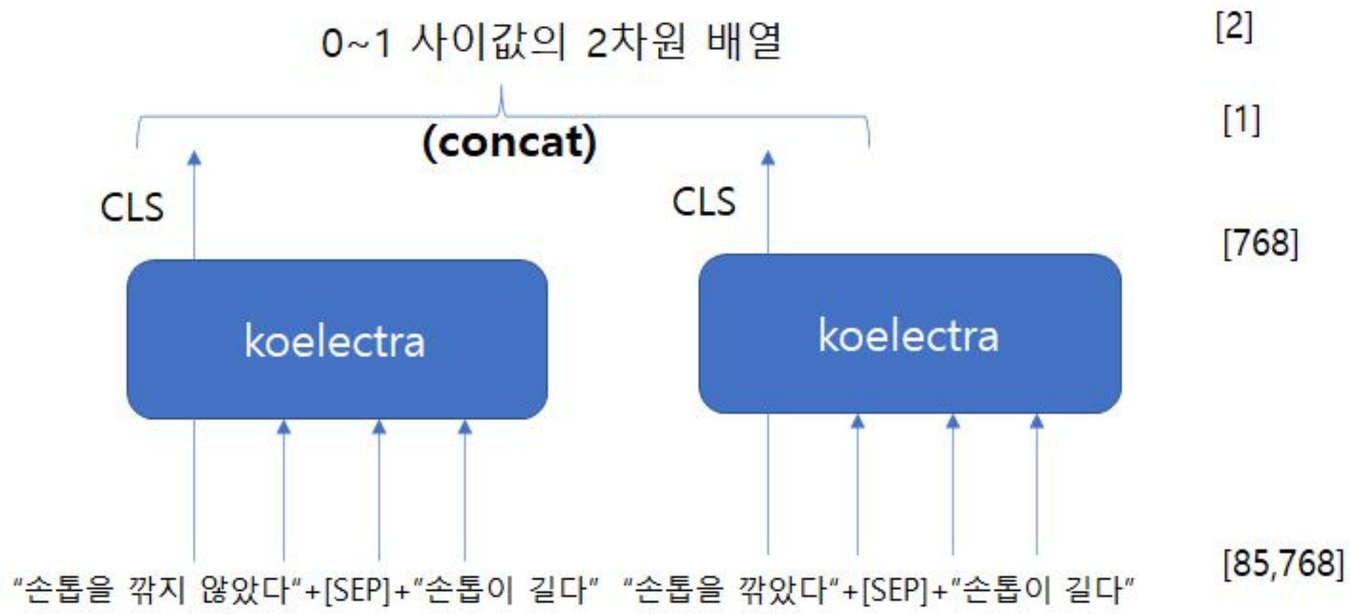


Example

데이터에서 컬럼 'question'에 따라서 'sentence+SEP+1' 또는 'sentence+SEP+2'의 multi-sequence 입력 순서를 바꿔서 넣어줌.

- 'question'='원인' 일 경우의 예시:
  - sentence: 손톱이 길다.
  - Question: 원인
  - 1: 손톱을 깎지 않았다.
  - 2: 손톱을 깎았다.
  - Answer: 1

*COPA의 경우는
Dataloader에서 'answer' 1/2을 0/1로
변환해서 모델에 정답 라벨로 제공.



모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- **main: koelectra_COPA.py**
  - main 함수의 역할로, train/eval/inference 과정과 다른 모듈을 불러오는 부분.

- **koelectra_datasets.py**
  - 데이터셋을 읽은 후, 모델의 입력 형태로 데이터를 리턴.

- **koelectra_models.py**
  - Dataloader로 입력받은 데이터에 대한 모델의 출력을 리턴.

- **utils.py**
  - 각 모듈에서 사용할 여러 함수와 변수 등을 정의해서 사용.

모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- **koelectra_datasets.py**
  - 데이터셋을 읽은 후, 모델의 입력 형태로 데이터를 리턴.

```
def __init__(self, data_filename): #str: path of csv
    super(COPA_biSentence,self).__init__()
    self.data = pd.read_csv(os.path.join(os.getcwd(),data_filename), sep="\t")
    self.tokenizer = model_tokenizer.from_pretrained(model_path)
    self.max_tokenizer_length = TOKEN_MAX_LENGTH['COPA'] #40
```

```
return input_ids1, token_type_ids1, attention_mask1, input_ids2, token_type_ids2, attention_mask2, label_int
```

5. COPA

모듈 구조

결과일 경우,
sentence 뒤에 sent1 또는 sent2를 연결.

원인일 경우,
sentence 앞에 sent1 또는 sent2를 연결.

정답 라벨이 1 또는 2로 주어지므로,
0 또는 1로 치환해서 모델의 정답으로 사용.

```python
def __getitem__(self, item): #fetch data sample(row) for given key(item)
    sentData = self.data.iloc[item,:]
    #max_tokenizer_length = TOKEN_MAX_LENGTH['COPA']

    sentence = sentData['sentence'][:self.max_tokenizer_length]
    sent1 = sentData['1'][:self.max_tokenizer_length]
    sent2 = sentData['2'][:self.max_tokenizer_length]
    qType = sentData['question'] #'원인', '결과'

    input1_str, input2_str = '',''
    if qType == '결과':
        input1_str = sentence+self.tokenizer.sep_token+sent1 #[CLS:2]+S+[SEP:3]+S1+[SEP]+[PAD:1]
        input2_str = sentence+self.tokenizer.sep_token+sent2 #[CLS:2]+S+[SEP:3]+S2+[SEP]+[PAD:1]
    elif qType == '원인':
        input1_str = sent1+self.tokenizer.sep_token+sentence #[CLS:2]+S+[SEP:3]+S1+[SEP]+[PAD:1]
        input2_str = sent2+self.tokenizer.sep_token+sentence #[CLS:2]+S+[SEP:3]+S2+[SEP]+[PAD:1]
    else:
        pass
    tok1 = self.tokenizer(input1_str, padding="max_length", max_length=self.max_tokenizer_length*2, truncation=True)
    tok2 = self.tokenizer(input2_str, padding="max_length", max_length=self.max_tokenizer_length*2, truncation=True)

    label = sentData['Answer']
    label_int = 0
    if label==1:
        label_int=0
    elif label==2:
        label_int=1
    else:
        pass
```

```python
return input_ids1, token_type_ids1, attention_mask1, input_ids2, token_type_ids2, attention_mask2, label_int
```

모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- **koelectra_models.py**
  - Dataloader로 입력받은 데이터에 대한 모델의 출력을 리턴.

1차원의 확률값으로 변환하는 linear MLP 정의.

모델 출력을 1차원의 확률값으로 반환.

```
class model_COPA_biSent(nn.Module):
    def __init__(self):
        super(model_COPA_biSent, self).__init__()
        self.model_PLM = model_class.from_pretrained(model_path)

        self.sigmoid = nn.Sigmoid()
        self.linear = nn.Linear(768,1) #CLS: 768->2(binary)

    def forward(self, input_ids, token_type_ids, attention_mask):
        iIds = input_ids.long()
        tok_typeIds = token_type_ids.long()
        attMask = attention_mask.long()

        output=self.model_PLM(input_ids=iIds, token_type_ids=tok_typeIds, attention_mask=attMask)

        #CLS토큰임베딩&LinearFC 사용하는 경우
        output = output['last_hidden_state'][:, 0, :] #cls_embedding: torch.Size([batch_size, 768])
        self.sigmoid(output)
        output = self.linear(output) #tensor: 768 -> 1 #linear classifier(확률)

        return output
```

모듈 구조

전체 과제의 모델과 데이터 로더 부분을 포함하는 부분과 기타 함수와 변수를 정의한 utils, 그리고 메인으로 구성.

- main: koelectra_COPA.py
  - main 함수의 역할로, train/eval/inference 과정과 다른 모듈을 불러오는 부분.

2차원 배열의 값 중 큰 값을 갖는 인덱스를 모델 출력으로 사용.

500개의 학습데이터 단위로 모델성능 평가 진행.

```
output1 = model(input_ids1, token_type_ids1, attention_mask1)
output2 = model(input_ids2, token_type_ids2, attention_mask2)
output = torch.cat([output1, output2], dim=1) #(bs,1)*2 -> (bs*2)

loss = lf(output,label)
pred = torch.argmax(output,dim=-1)

correct += sum(pred.detach().cpu()==label.detach().cpu())
all_loss.append(loss)
loss.backward() #기울기 계산
optimizer.step() #가중치 업데이트
mini_batch += batch_size
if mini_batch%500 == 0:
    if bool_save_model:
        filename_modelSave_chkpt = f'{task_name}_{model_name}_rs{random_seed_int}_epoch{countEpoch}
        save_model(model_savePth, filename_modelSave_chkpt, mymodel, optimizer)

    print(f'batch{mini_batch}:',end='')
    accuracy = eval_copa_biModel(mymodel, eval_loader, bs, device)
    acc_list.append(accuracy)
    model.train() #set model training mode
```

앙상블 결과

다른 데이터를 활용한 모델 중 준수한 성능의 단일 모델의 결과로 hard voting 적용.

- **앙상블 후보 별 활용한 학습 데이터**
  *학습 시 사용한 데이터를 구분짓기 위해 활용한 데이터 별 다른 random_seed 적용 (rs: random_seed)

# rs/epoch (데이터셋 이름)	활용한 학습데이터 종류(데이터 수)	총 학습 데이터 수
rs6/20	train(3080개)	총 3080개
rs12/11	train(3080개) + train데이터의 pivot-translation 데이터(3080개)	총 6160개
rs13/16	train(3080개) + 추가 학습데이터(1000개)	총 4080개
rs103/14	train(3080개) + 추가 학습데이터(1000개) + train데이터의 pivot-translation 데이터(3080개)	총 7160개
rs131/12	train(3080개) + 추가 학습데이터(1000개) + 전체 데이터의 pivot-translation 데이터(4080개)	총 8160개

위의 5개 test셋에 대한 예측결과를 앙상블한 결과를 최종 산출물로 제출.

→ **ACC = 0.9160**

감사합니다