

The background is a dark blue gradient with abstract circular patterns and binary code (0s and 1s) scattered throughout. The patterns resemble stylized orbits or data paths.

# Korean MRC

Kaggle competition

TEAM GNLP

이다원, 김도연, 양은지, 오희주, 전효택, 이지수, 정명찬



# 목차

## 01 프로젝트 개요

프로젝트 설명, 목표, 제한사항

## 02 프로젝트 진행 프로세스

접근 방식, 전처리, 학습, 후처리, 앙상블, 최적화

## 03 프로젝트 결과

여러실험, 모델 소개

# 01-1 프로젝트 개요

한국어 기계독해 질의응답기 구현

개발 환경

Colab, PyTorch, Transformers,  
Wandb

진행 프로세스

모델 선정 > 평가지표 마련 > 데이터 작업 >  
앙상블 적용 > 최종평가

목표

레반슈타인 거리 3이하  
다양한 방법 학습

# 01-2 levenshtein distance

		m	e	i	l	e	n	s	t	e	i	n
l	0	1	2	3	4	5	6	7	8	9	10	11
e	1	1	2	3	3	4	5	6	7	8	9	10
v	2	2	1	2	3	3	4	5	6	7	8	9
e	3	3	2	2	3	4	4	5	6	7	8	9
n	4	4	3	3	3	3	4	5	6	6	7	8
s	5	5	4	4	4	4	3	4	5	6	7	7
h	6	6	5	5	5	5	4	3	4	5	6	7
t	7	7	6	6	6	6	5	4	4	5	6	7
e	8	8	7	7	7	7	6	5	4	5	6	7
i	9	9	8	8	8	7	7	6	5	4	5	6
n	10	10	9	8	9	8	8	7	6	5	4	5
	11	11	10	9	9	9	8	8	7	6	5	4

글자 단위 레반슈타인 거리

정답파일과 예측파일을 비교

l	e	v		e	n	s		h	t	e	i	n	
o	=	o	+	=	=	=	.	=	=	=	=	=	
m	e	i	l	e	n	s				t	e	i	n

l	e		v	e	n	s		h	t	e	i	n	
o	=	+	o	=	=	=	.	=	=	=	=	=	
m	e	i	l	e	n	s				t	e	i	n

```
def levenshtein_distance(s1,s2, debug=False):
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1, debug)

    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))


        if debug:
            print(current_row[1:])

        previous_row = current_row

    return previous_row[-1]
```



# 01-3 목표 점수



 Community Prediction Competition

## 6th Goorm project 2 Korean MRC

구름 자연어 처리 6기 프로젝트 2차 한국어 기계독해

4 teams · 14 hours to go



## 구름 자연어처리 과정 프로젝트 2

	Oracle	1.19775
	구름 이전 기수 최고 점수	1.71678

Team 2



2.44354

	Blank	5.92888
	Baseline	194.29631

# 01-4 팀 구성

김도연

팀장  
프로젝트 관리  
웹 서비스화

이다원

팀원  
코드보강, 하이퍼 파라미터  
최적화, 앙상블, 최종 평가

오희주

팀원  
성능 평가지표 마련  
모델 종합, 비교

이지수

팀원  
사전학습 모델 선정  
데이터 수집

정명찬

팀원  
사전학습 모델 선정  
데이터 수집, 보강

양은지

팀원  
사전학습 모델 선정  
데이터 수집

전효택

팀원  
성능 평가지표 마련  
모델 종합, 비교

# 02 - 1 진행 프로세스

1/2

## 1. 사전기획

목표 설정  
역할 분담  
코드 이해

1/3 ~ 1/6

## 2. 모델 선정

사전학습 모델 선정  
코드 보강

1/3 ~ 1/6

## 3. 평가지표 마련

레벤슈타인 거리 활용

1/9 ~ 1/11

## 4. 데이터 작업

데이터 수집(aihub)  
데이터 전처리  
데이터 후처리

1/11 ~  
1/13

## 5. 최종 평가

모델 종합, 비교  
양상블 구현

1/16

## 6. 프로젝트 발표

## 02 - 2 학습 데이터 소개

```
with open("/content/drive/MyDrive/[redacted]/goorm_nlp_8th_group3/goorm_nlp_8th_group3/project2/train.json", 'rb') as f:
    input_dict = json.load(f)
    input_dict["data"][0]
```

Python

{'title': '제주도 장마 시작 ... 중부는 이달 말부터',

'paragraphs': [{'context': '올여름 장마가 17일 제주도에서 시작됐다. 서울 등 중부지방은 예년보다 사나흘 정도 늦은 이달 말께 장마가 시작될 전망이다. 17일 기상청에 따르면 제주도 남쪽 먼바다에 있는 장마전선의 영향으로 이달 제주도 산간 및 내륙지역에 호우주의보가 내려지면서 곳곳에 100mm에 육박하는 많은 비가 내렸다. 제주의 장마는 평년보다 2~3일, 지난해보다는 하루 일찍 시작됐다. 장마는 고온다습한 북태평양 기단과 한랭 습윤한 오호츠크해 기단이 만나 형성되는 장마전선에서 내리는 비를 뜻한다. 장마전선은 18일 제주도 먼 남쪽 해상으로 내려갔다가 20일께 다시 북상해 전남 남해안까지 영향을 줄 것으로 보인다. 이에 따라 20~21일 남부지방에도 예년보다 사흘 정도 장마가 일찍 찾아올 전망이다. 그러나 장마전선을 밀어올리는 북태평양 고기압 세력이 약해 서울 등 중부지방은 평년보다 사나흘가량 늦은 이달 말부터 장마가 시작될 것이라는 게 기상청의 설명이다. 장마전선은 이후 한 달가량 한반도 중남부를 오르내리며 곳곳에 비를 뿌릴 전망이다. 최근 30년간 평균치에 따르면 중부지방의 장마 시작일은 6월24~25일이었으며 장마 기간은 32일, 강수일수는 17.2일이었다. 기상청은 올해 장마기간의 평균 강수량이 350~400mm로 평년과 비슷하거나 적을 것으로 내다봤다. 브라질 월드컵 한국과 러시아의 경기가 열리는 18일 오전 서울은 대체로 구름이 많이 끼지만 비는 오지 않을 것으로 예상돼 거리 응원에는 지장이 없을 전망이다.'},

'qas': [{'question': '북태평양 기단과 오호츠크해 기단이 만나 국내에 머무르는 기간은?'},

'answers': [{'text': '한 달가량', 'answer\_start': 478},

{'text': '한 달', 'answer\_start': 478}],

'guid': '798db07f0b9046759deed9d4a35ce31e'}]]],

'news\_category': '종합',

'source': 'hankyung'}

Context 에서 question에 해당하는 answer를 예측합니다.



# 02 - 2 학습 데이터 소개

**기본 data**  
klue mrc  
context: 10434개, question: 12037  
개  
answers: 17663개

**추가 data**  
ai hub(뉴스기사 기계독해- 추출형)  
context: 9000개, question: 18000개  
answers: 18000개



**전처리**  
context 길이 조절

**토크나이저**  
모델별 토크나이저

# 02 - 2 학습 데이터 소개

## 기본 data

klue mrc

context: 10434개, question: 12037  
개

answers: 17663개



```
for group in tqdm(input_dict_add['data']): #딕셔너리 하나씩 꺼낸다.
    for passage in group['paragraphs']: #딕셔너리의 paragraphs
        passage['context'] = re.sub(r'\n', '', passage['context']) #paragraphs의 context
        for qa in passage['qas']: #paragraphs의 qas
            qa['question'] = qa['question'] + '?' #paragraphs의 question
            # for answer in qa['answers']: #question의 answers

            qa['answers']=[qa['answers']] # 기존 데이터셋 처럼 [] 안에 넣기
```

## 추가 data

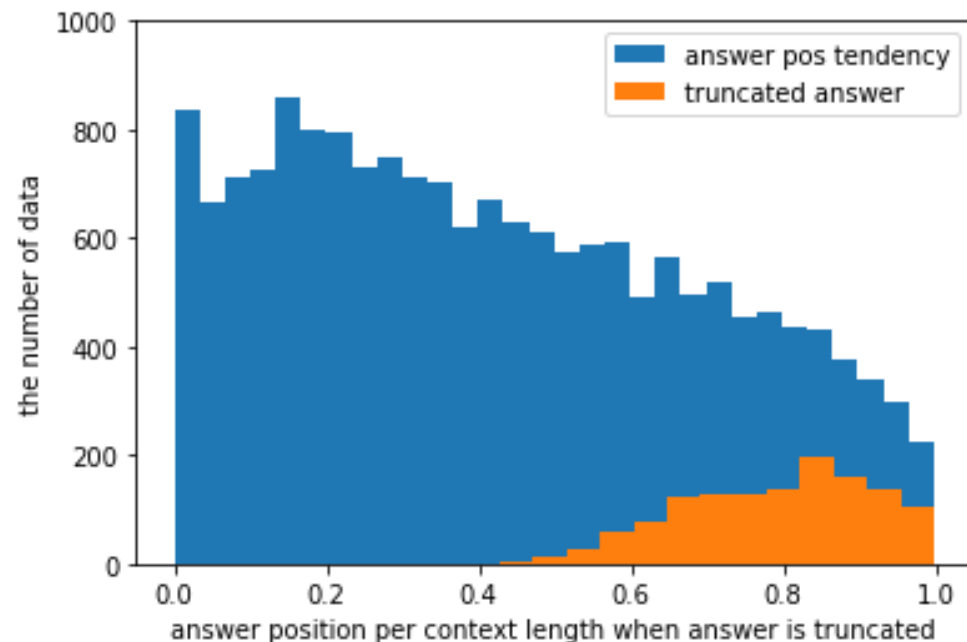
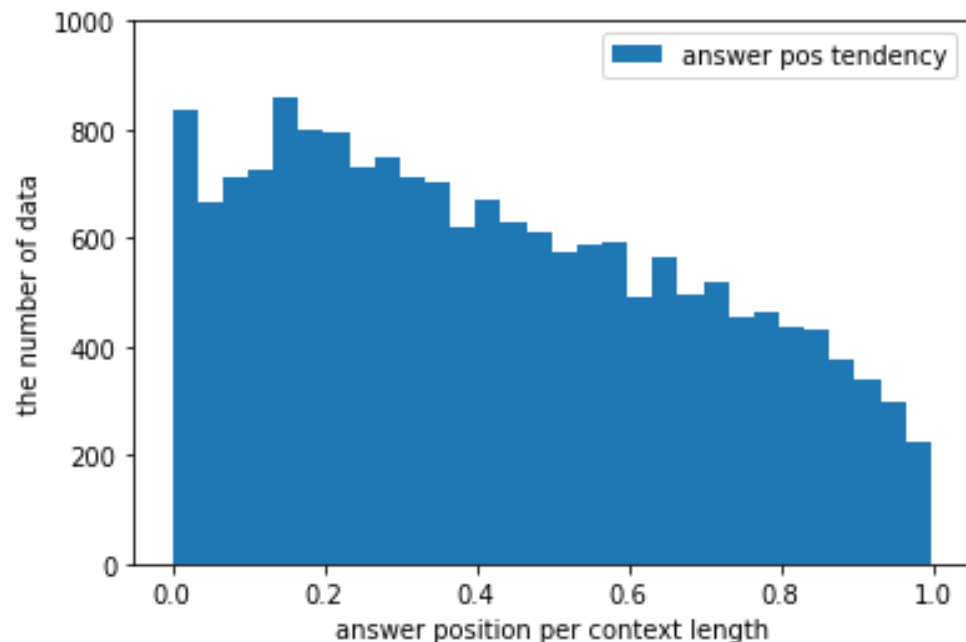
ai hub(뉴스기사 기계독해- 추출형)  
context: 9000개, question: 18000개  
answers: 18000개



```
def split_input_dict(input_dict, ratio = 0.1, seed = 42):
    split_point = int(len(input_dict['data']) * ratio)
    random.seed(seed)
    random.shuffle(input_dict['data'])
    valid_dict = deepcopy(input_dict)
    train_dict = input_dict

    valid_dict['data'] = input_dict['data'][:split_point]
    train_dict['data'] = input_dict['data'][split_point:]
    return train_dict, valid_dict
```

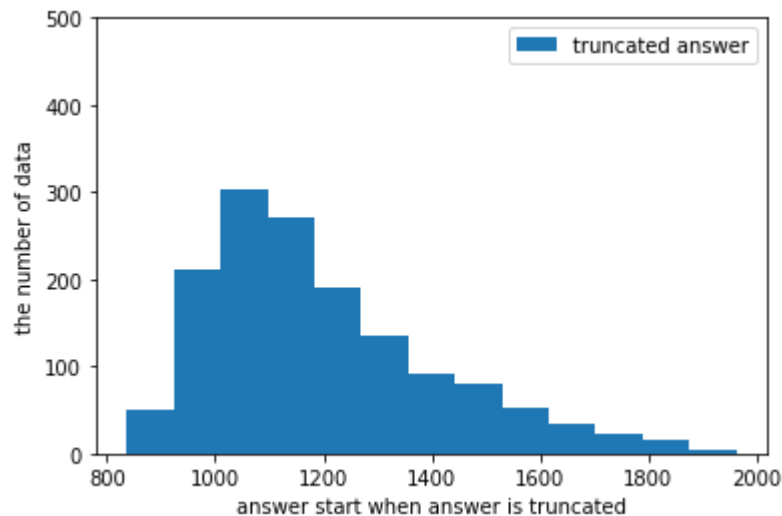
## 02 - 3 전처리



context가 길어서 토큰화 과정에서 정답이 잘리는 현상이 발생했습니다.

answer\_start의 앞 뒤로 context를 미리 slicing 했습니다.

## 02 - 3 전처리



minimum : 837  
maximum : 1962

```
### context 넘겨서 자르기 ###  
target_context = context  
if answer['answer_start'] > 1400:  
    target_context = target_context[1000:]  
    answer['answer_start'] -= 1000  
elif answer['answer_start'] > 1200:  
    target_context = target_context[800:]  
    answer['answer_start'] -= 800  
elif answer['answer_start'] > 1000:  
    target_context = target_context[600:]  
    answer['answer_start'] -= 600  
elif answer['answer_start'] > 800:  
    target_context = target_context[400:]  
    answer['answer_start'] -= 400  
  
valid_contexts.append(target_context)    #answers의 한 answer당 해당하는 context 저장  
### context 넘겨서 자르기 ###  
valid_questions.append(question)    #answers의 한 answer당 해당하는 question 저장  
valid_answers.append(answer)        #answers의 한 answer 저장
```

Answer\_Start를 최대한 토큰화 후에 존재할 수 있도록 슬라이싱을 했습니다.

## 02 - 4 학습

```
if do_accumulation:
    (loss / ACCUMULATION).backward()

    step += 1
    if step % ACCUMULATION:
        continue

    clip_grad_norm_(model.parameters(), max_norm=1.)
    optimizer.step()
    optimizer.zero_grad(set_to_none=True)
```

Gradient accumulation으로 oom문제를 완화했습니다.

Batch size를 늘려서 학습한 것과 비슷한 효과가 있습니다.

Validation set으로 최적의 모델을 선택했습니다.

학습이 적절한 방향으로 이루어 지지 않을 때 학습을 종료하는 early stop기능이 있습니다.

```
if total_valid_loss < lowest_total_valid_loss:
    print(f"lowest_total_valid_loss: {total_valid_loss} epoch : {epoch} iteration : {iteration}")
    torch.save(model.state_dict(), './output_model_best')
    lowest_total_valid_loss = total_valid_loss
    early_stop_stack = 0
else:
    early_stop_stack += 1
    if early_stop_stack == 3:
        print("SYSTEM HALT")
        return 0
```

# 02 - 5 앙상블

monologg\_koelectra-small-v2-distilled-korquad-384

albert-kor-base

nlpotato/roberta\_large\_origin\_added\_korquad

```
def to_list(tensor):
    return tensor.detach().cpu().tolist()

def logits_voting(A_model, B_model, C_model):
    # 각 모델마다 input_ids, start_logits, end_logits값 저장. 순서대로 저장 리스트에 + guides // context 필요.
    # 모델 logit 확률 top 5 구해서 모델 1 top1 모델 2 top1 모델3 top1 모델1 top2 순으로 체크.
    # print(list_dict_all[0]["start_logits"]) pickle
    # A_model

    ind = 0
    RESULT = []

    if len(A_model) != len(B_model) or len(A_model) != len(C_model) or len(B_model) != len(C_model):
        print('error')
    else:
        llogit = len(A_model)

    while 1:
        # input_ids
        A_input_ids = A_model[ind]['input_ids']
        B_input_ids = B_model[ind]['input_ids']
        C_input_ids = C_model[ind]['input_ids']

        # guid
        A_guid = A_model[ind]['guid']
        B_guid = A_model[ind]['guid']
        C_guid = A_model[ind]['guid']

        # logit 값 순서대로 불러오기기
        A_start_logit = A_model[ind]['start_logits']
        A_end_logit = A_model[ind]['end_logits']
        B_start_logit = B_model[ind]['start_logits']
        B_end_logit = B_model[ind]['end_logits']
        C_start_logit = C_model[ind]['start_logits']
        C_end_logit = C_model[ind]['end_logits']

        # 로짓값 -> 확률 큰 순서대로 리스트형태 + 인덱스 -> top 5 자르기기
        A_start_idx_and_logit = (sorted(enumerate(A_start_logit), key=lambda x: x[1], reverse=True))[:5]
        A_end_idx_and_logit = (sorted(enumerate(A_end_logit), key=lambda x: x[1], reverse=True))[:5]
        B_start_idx_and_logit = (sorted(enumerate(B_start_logit), key=lambda x: x[1], reverse=True))[:5]
        B_end_idx_and_logit = (sorted(enumerate(B_end_logit), key=lambda x: x[1], reverse=True))[:5]
        C_start_idx_and_logit = (sorted(enumerate(C_start_logit), key=lambda x: x[1], reverse=True))[:5]
        C_end_idx_and_logit = (sorted(enumerate(C_end_logit), key=lambda x: x[1], reverse=True))[:5]
```

정규 표현식 라이브러리로 특수문자를 제거한 후 questionanswering 모델이 예측한 정답의 시작을 예측하는 logit 정답의 끝을 예측하는 Logit를 pickle형태로 저장합니다. 3개의 사전 학습 모델을 fine tuning한 모델에서 start logit과 end logit의 최댓값의 인덱스가 가까운 정답을 선택합니다.

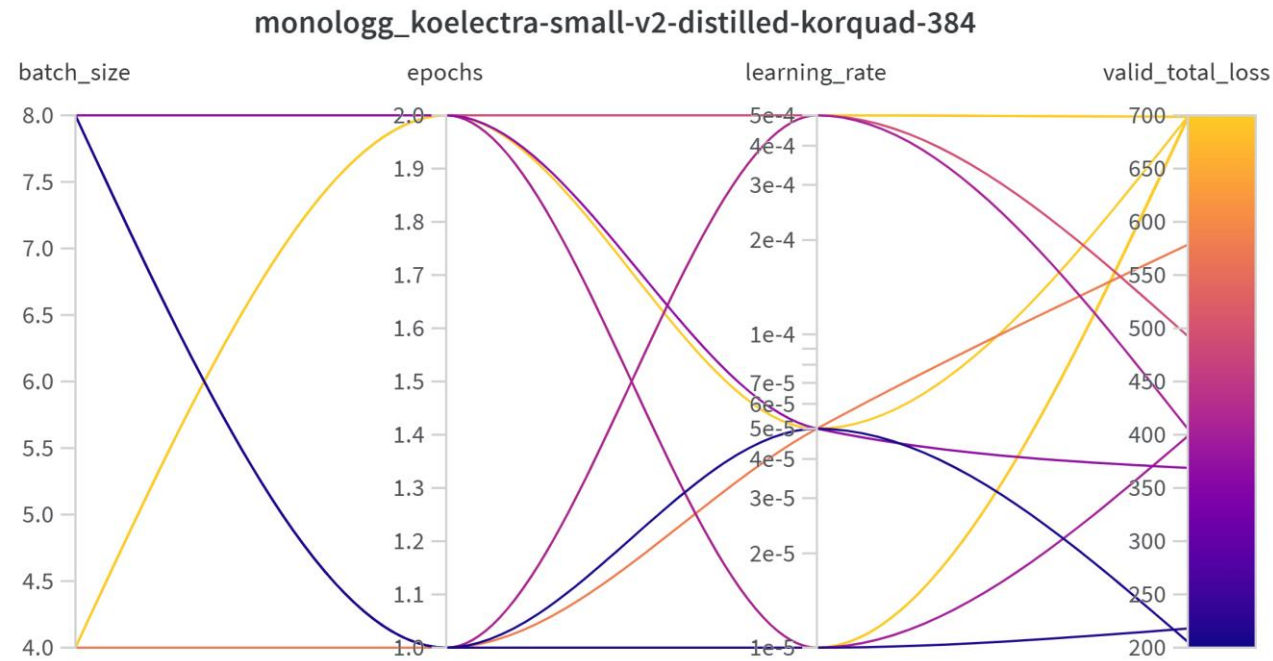


## 02 - 6 후처리



정규 표현식 라이브러리로 특수문자를 제거한 후 8글자를 넘지 않도록 슬라이싱 하였습니다.

# 02 - 7 하이퍼 파라미터 최적화



Wandb sweep으로 하이퍼 파라미터를 최적화 했습니다.

# 03 - 1 모델 실험

- roberta\_large\_acc.csv > 2.51 (nlpotato/roberta\_large\_origin\_added\_korquad)
- nlp04org\_and\_korquad.csv -> 2.55 (nlp04/org\_and\_korquad)
- lmqqmbart-large-cc25-koquad-qg.csv -> 3.32 (Facebook dataset)
- koelectra -> 3.19 (monologg/koelectra-base-v3-finetuned-korquad)
- baseline -> 128.3
- bertbase -> 3.85 (sangrimlee/bert-base-multilingual-cased-korquad)
- bertbase11 -> 21.2 (sangrimlee/bert-base-multilingual-cased-korquad)
- albert -> 3.39 (kykim/albert-kor-base)
- blank -> 5.10
- kobert\_morp\_korquad -> 6.16 (jack-oh/kobert\_morp\_korquad)
- roberta\_origin\_added -> 2.62 (nlpotato/roberta\_large-ssm\_wiki\_e2-origin\_added\_korquad\_e5)
- xlm-roberta-large -> 3.03 (hanmaroo/xlm\_roberta\_large\_korquad\_v2)1
- roberta\_acc\_logit -> 2.54 (nlpotato/roberta\_large\_origin\_added\_korquad)
- roberta\_accumulation -> 4.09 (nlpotato/roberta\_large\_origin\_added\_korquad)
- roberta\_finetuned -> 2.63 (nlpotato/roberta\_large\_origin\_added\_korquad)
- roberta\_large\_origin -> 2.60 (nlpotato/roberta\_large\_origin\_added\_korquad)
- roberta\_preprocess -> 3.67 (nlpotato/roberta\_large\_origin\_added\_korquad) -

## 03 - 2 모델 소개

1

monologg\_koelectra-small-  
v2-distilled-korquad-384

3.19

2

albert-kor-base

3.39



2.44

3

nlpotato/roberta\_large\_origin\_  
added\_korquad

2.54

# 03 - 2 모델 소개

1

monologg\_koelectra-small-v2-distilled-korquad-384

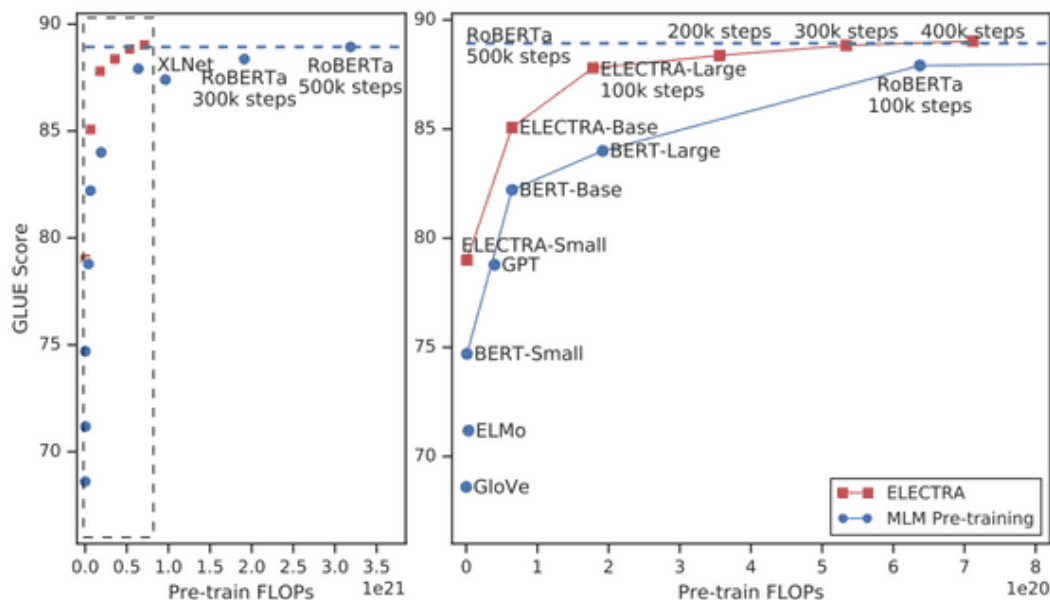


Figure 1: ELECTRA consistently outperforms MLM pre-training given the same compute budget. The right figure is a zoomed-in view of the dashed box.

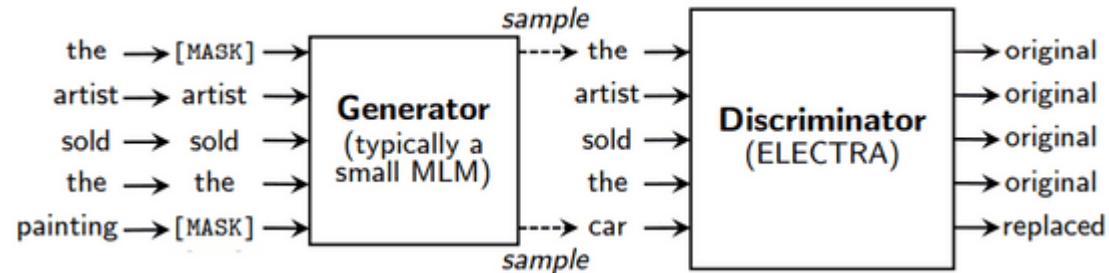


Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator on downstream tasks.

Generator와 discriminator사용

기존 MLM방식보다 효율적

# 03 - 2 모델 소개



albert-kor-base

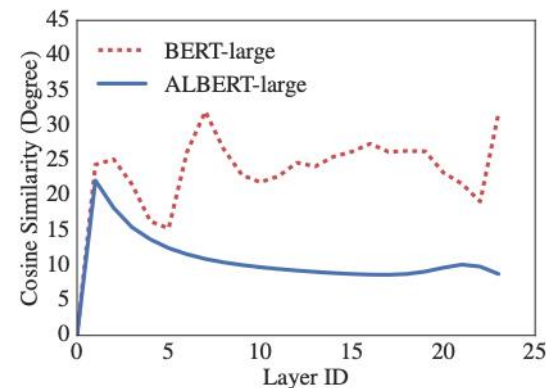
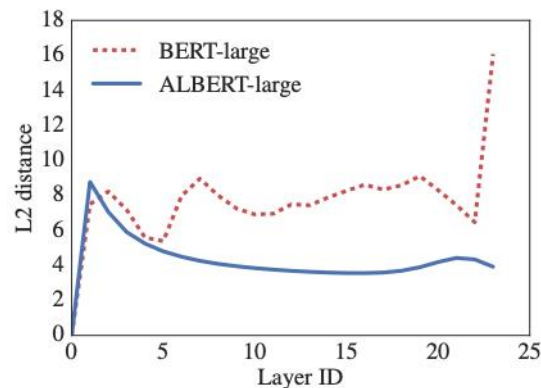


Figure 1: The L2 distances and cosine similarity (in terms of degree) of the input and output embedding of each layer for BERT-large and ALBERT-large.

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	<b>94.1/88.3</b>	<b>88.1/85.1</b>	<b>88.0</b>	<b>95.2</b>	<b>82.3</b>	<b>88.7</b>	0.3x

Parameter sharing

순서 맞추는 방식

Bert보다 메모리 소모 적음

Table 2: Dev set results for models pretrained over BOOKCORPUS and Wikipedia for 125k steps. Here and everywhere else, the Avg column is computed by averaging the scores of the downstream tasks to its left (the two numbers of F1 and EM for each SQuAD are first averaged).



# 03 - 2 모델 소개



nlpotato/roberta\_large\_origin\_added\_korquad

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	<b>94.6/89.4</b>	<b>90.2</b>	<b>96.4</b>
BERT <sub>LARGE</sub>						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
XLNet <sub>LARGE</sub>						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB  $\rightarrow$  160GB of text) and pretrain for longer (100K  $\rightarrow$  300K  $\rightarrow$  500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of BERT<sub>LARGE</sub>. Results for BERT<sub>LARGE</sub> and XLNet<sub>LARGE</sub> are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.

