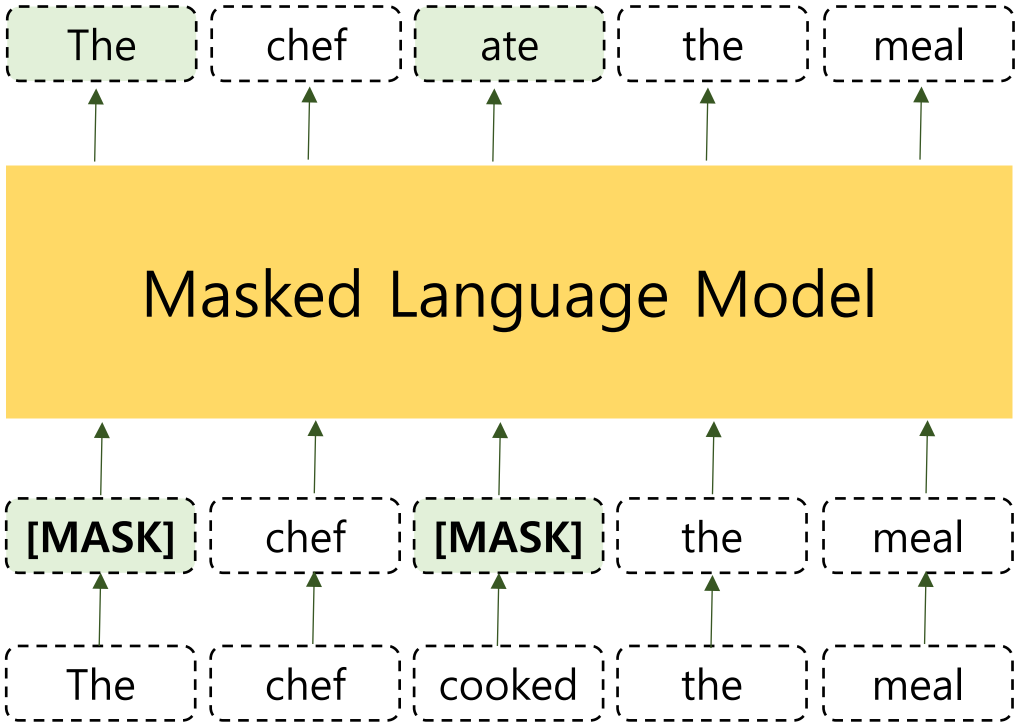
ELECTRA

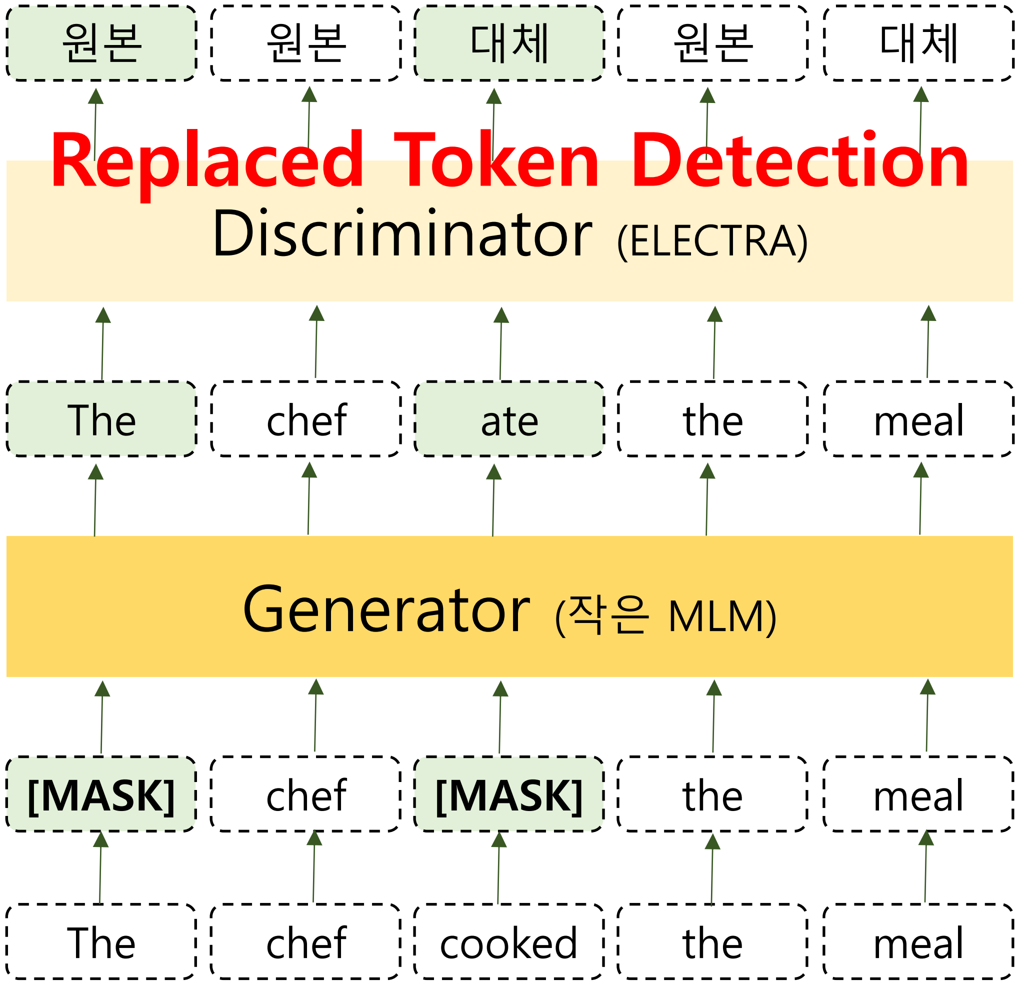
10조
고현지 김병진 차지윤

Efficiently
Learning an
Encoder that
Classifies
Token
Replacements
Accurately

METHOD

BERT vs ELECTRA





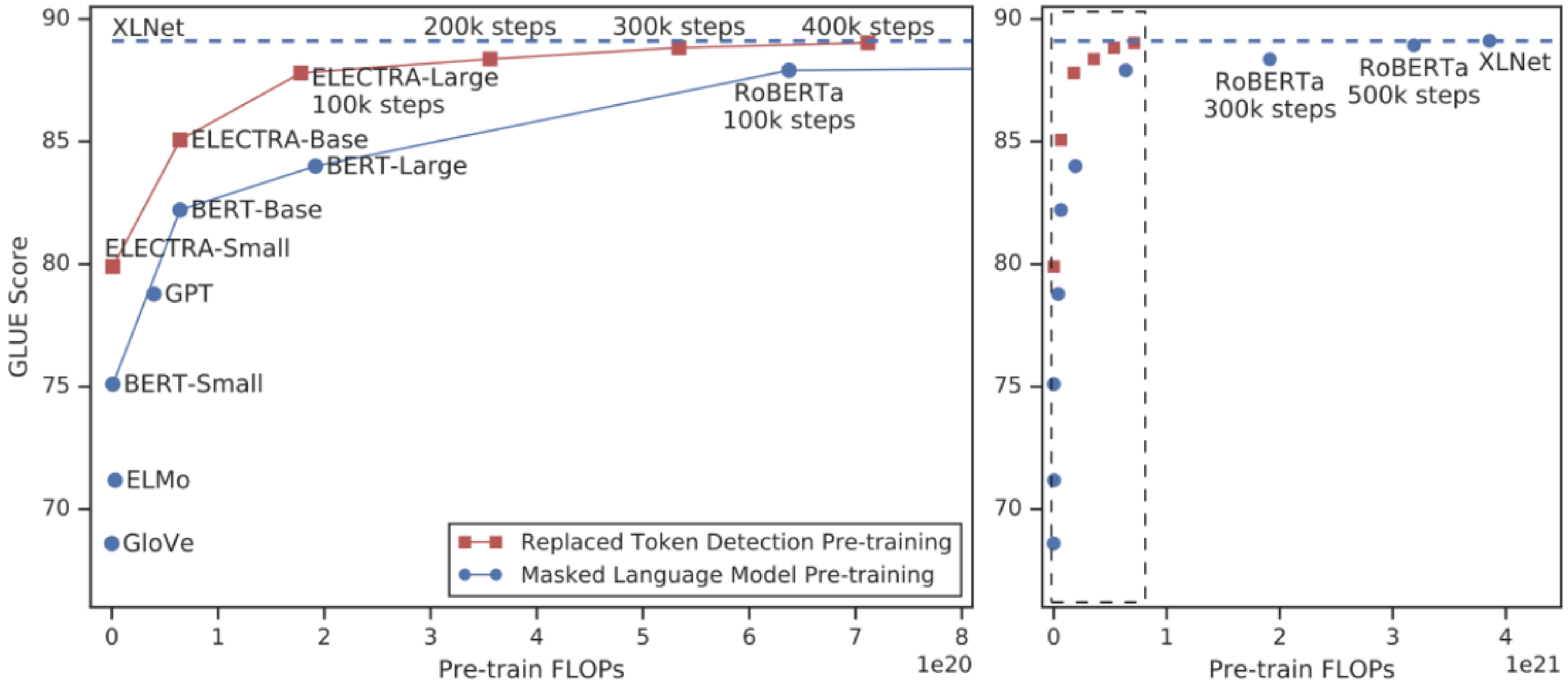
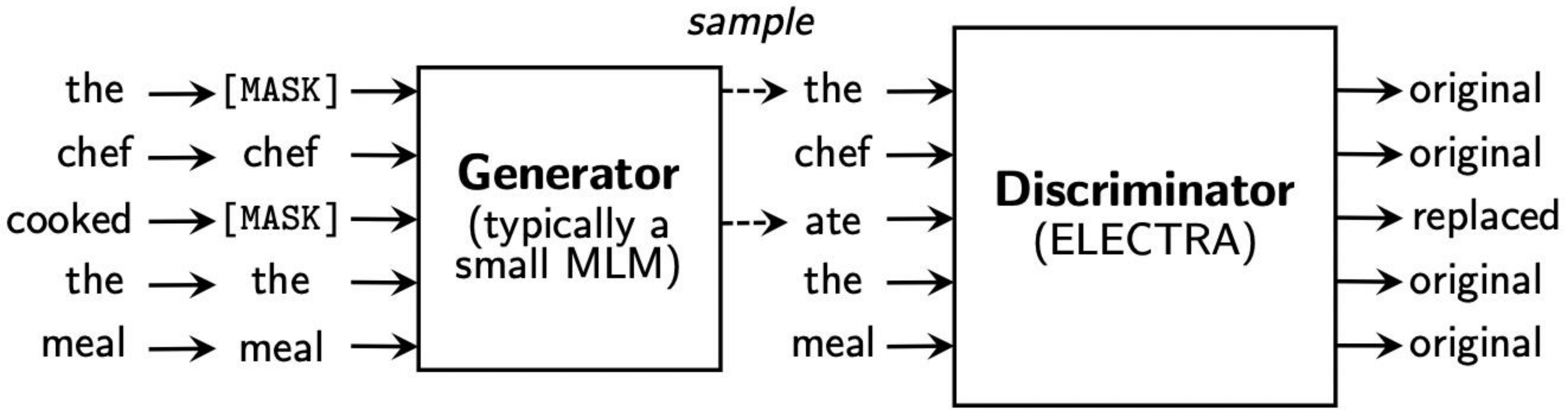
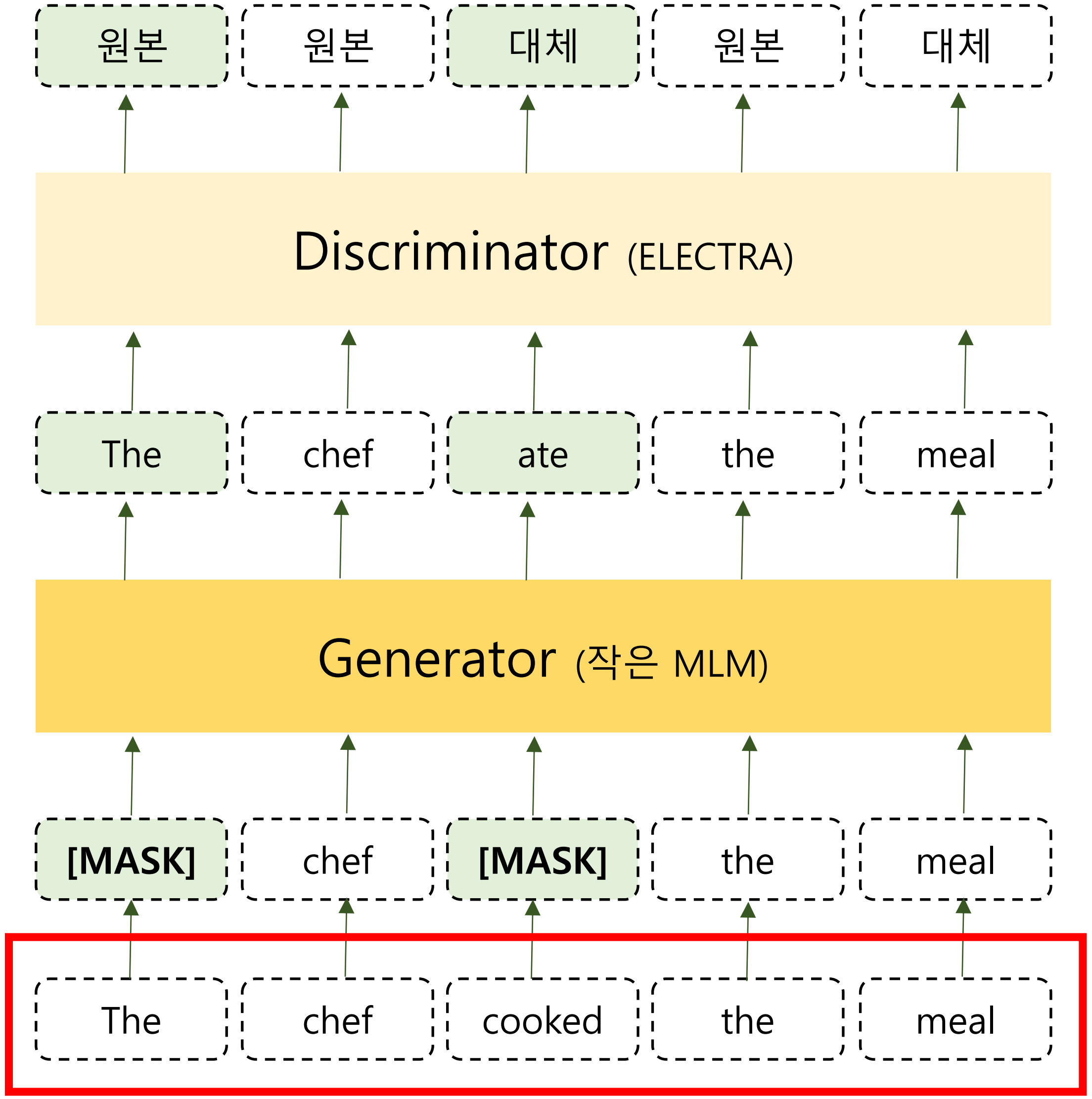


Figure 1: Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.

METHOD


sample
the
chef
cooked
the
meal
[MASK]
chef
[MASK]
the
meal
Generator
(typically a small MLM)
the
chef
ate
the
meal
Discriminator
(ELECTRA)
original
original
replaced
original
original

Generator G

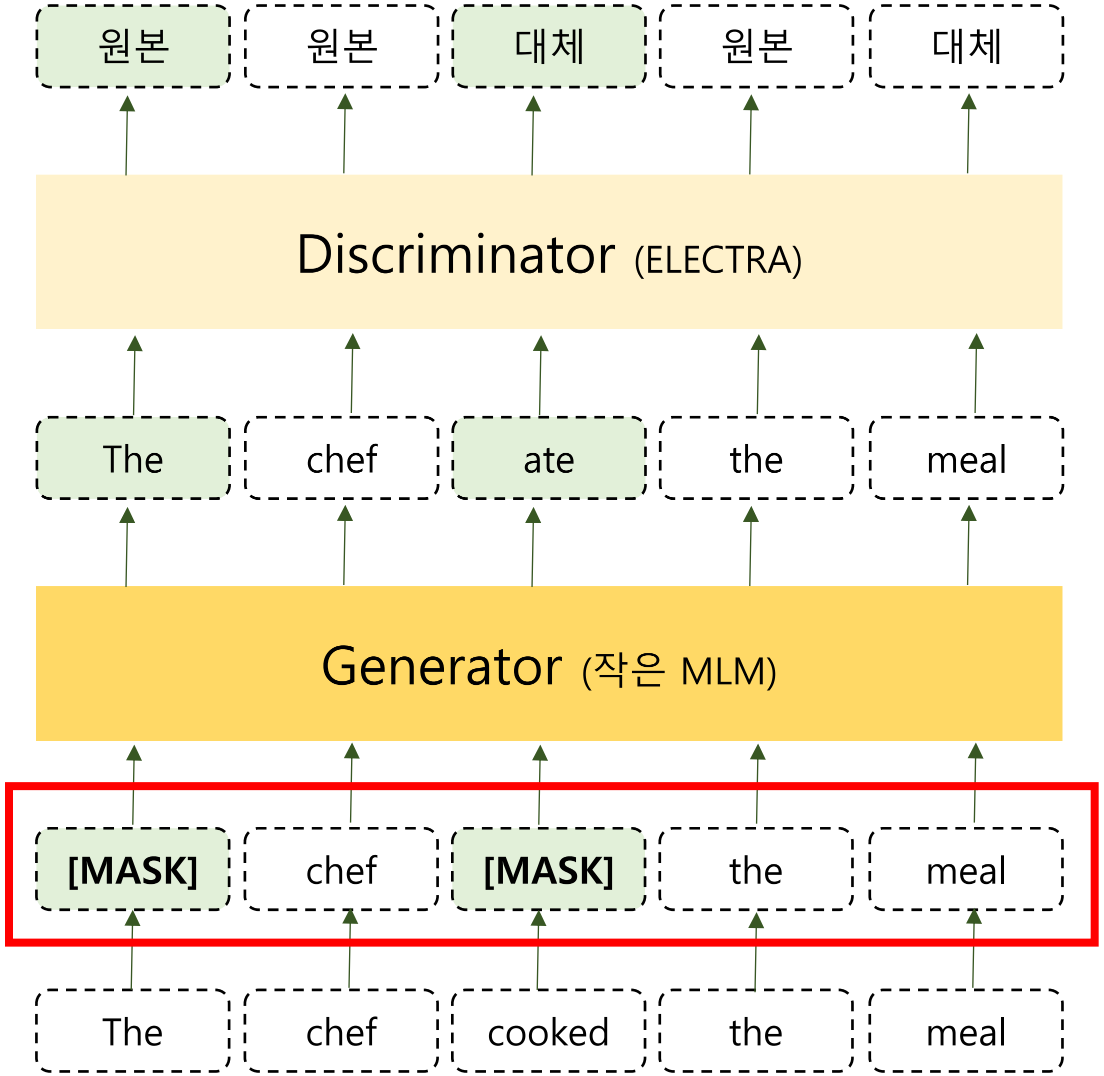


1 $x = [x_1, x_2, ..., x_n]$에 대해서

마스킹할 위치 $m = [m_1, m_2, ..., m_k]$을 랜덤하게 결정

$$mi \sim unif\{1, n\} \;\; for \;\; i = 1 \; to \; k$$

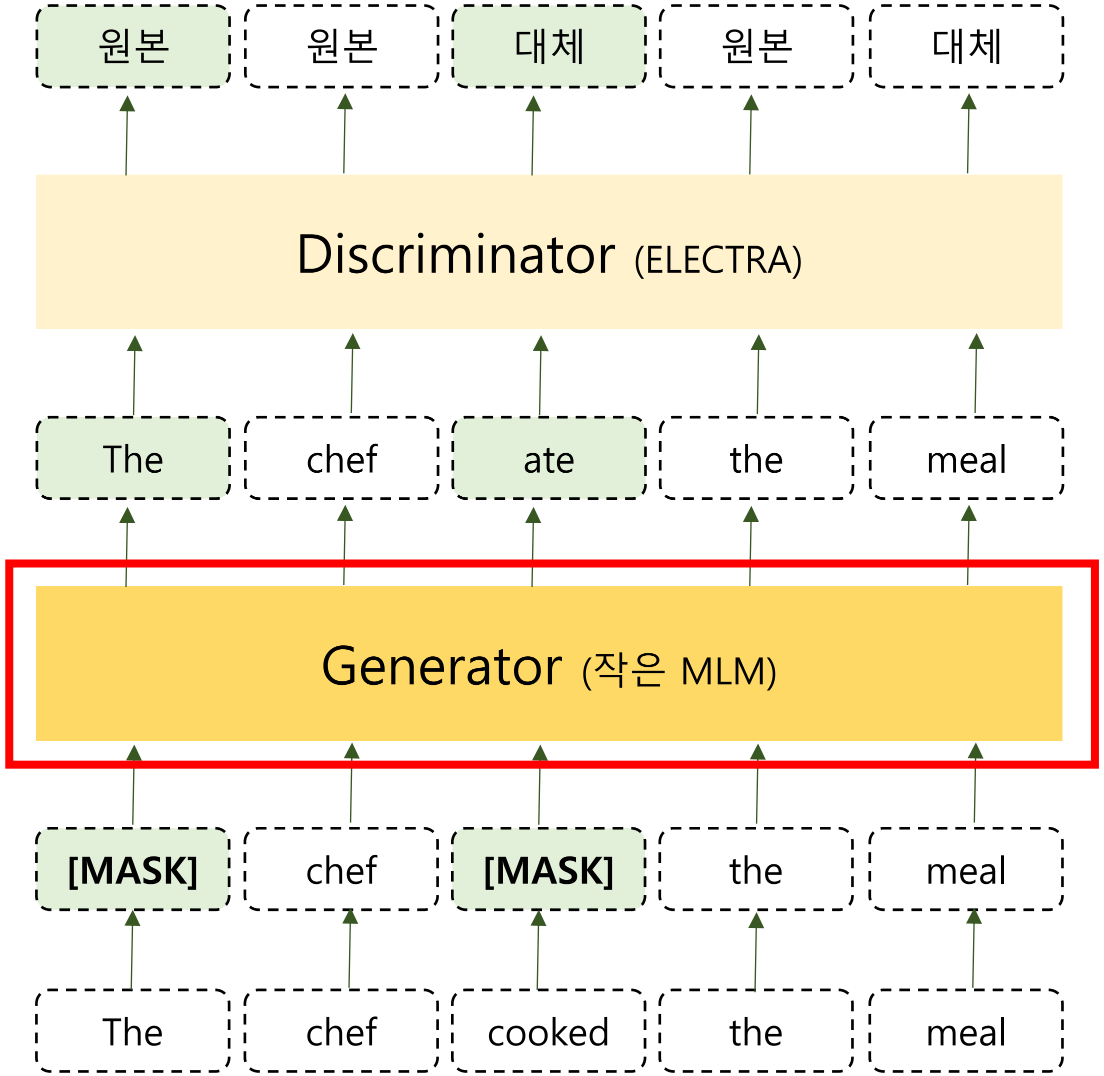
Generator G



2 마스킹된 토큰들은 [MASK] 토큰으로 대체

$$x^{masked} = REPLACE(x, m, [MASK])$$

Generator G



3 마스킹 된 입력 시퀀스 x^{masked}에 대해서

generator는 아래와 같이 원래 토큰이 무엇인지 예측

$$p_G(x_t|x^{masked}) =$$

$$exp(e(x_t)^T h_G(x^{masked})_t) / \sum_{x'} exp(e(x')^T h_G(x^{masked})_t)$$

$p_G(x_t|x^{masked})$

$\Rightarrow$ x^{masked}라는 시퀀스 주어졌을 때 특정 위치 t에 단어 x_t가 들어갈 확률

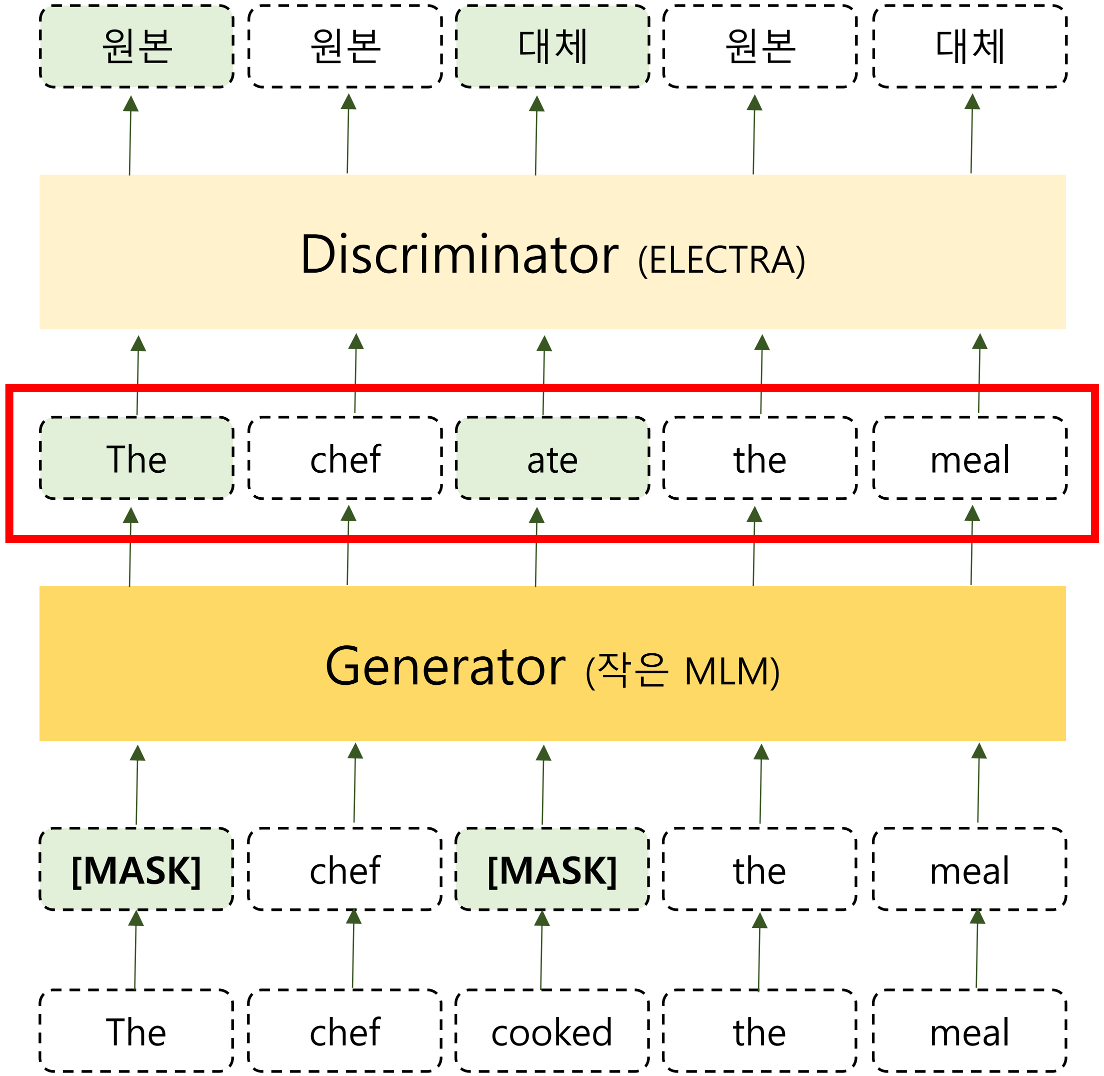
$exp(e \bullet x_i) / \sum_j exp(e \bullet x_j)$

$\Rightarrow$ softmax 함수

$e(x_t)^T h_G(x^{masked})_t$

$\Rightarrow$ token을 임베딩(e)하여 example의 hidden state 값을 곱

Discriminator D



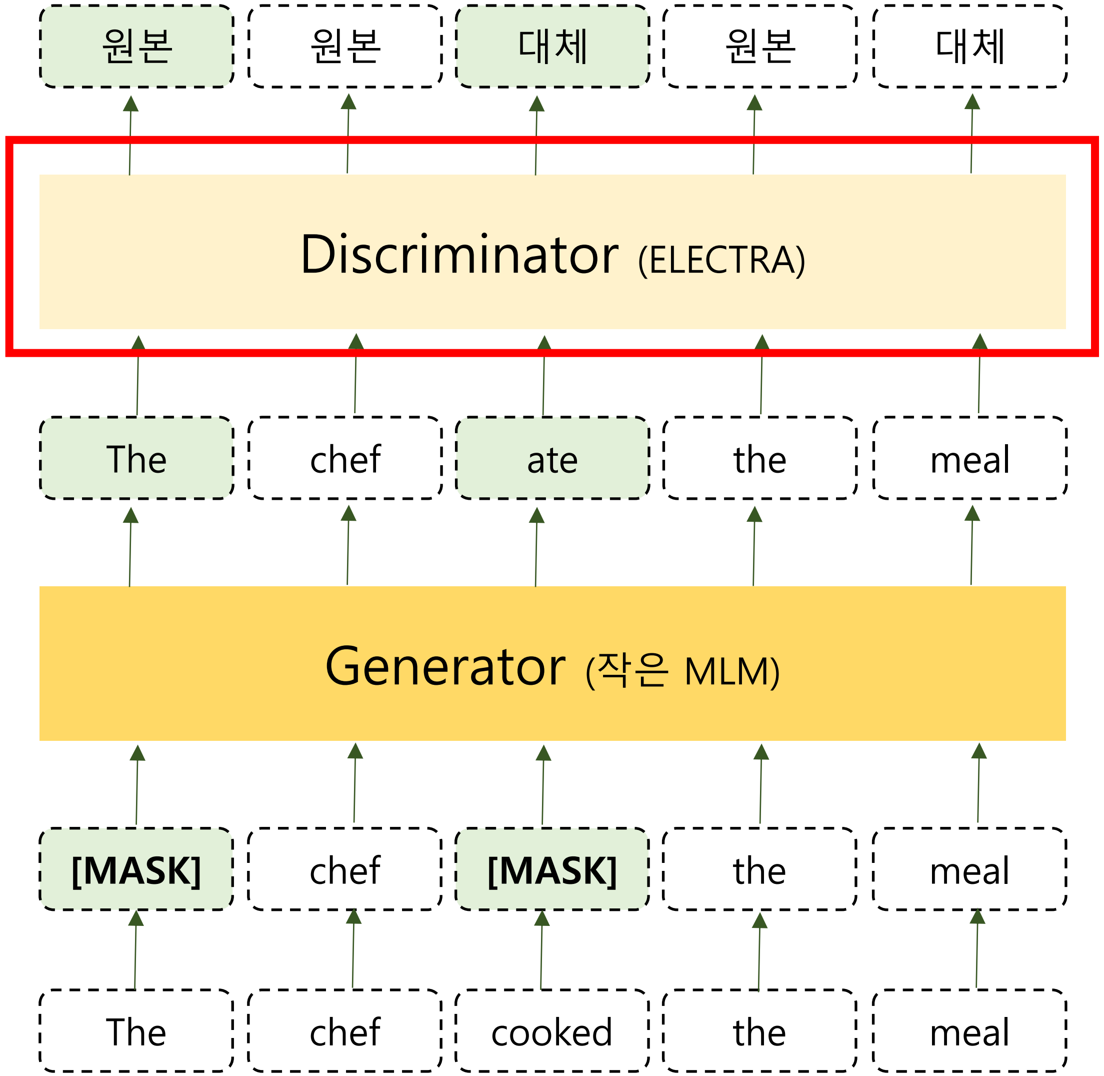
1 Discriminator D input 생성

[MASK]에서 $p_G(x_t|x)$으로 샘플링한 토큰으로 치환(corrupt)

$$x^{corrupt} = REPLACE(x, m, \hat{x})$$

$$\hat{x} \sim p_G(x_i|x^{masked})\ for\ i\ \in m$$

Discriminator D



2 Discriminator D는 $x^{corrupt}$가 원본 입력 토큰과 동일한건지, Generator G가 만들어낸 것인지 예측

$$D(x^{corrupt}, t) = sigmoid(w^T h_D(x^{corrupt})_t)$$

Target Class (이진)

- Original : 원본 문장의 토큰과 같은 토큰
- Replaced : 원본 문장의 토큰과 다른 토큰

Generator G와 Discriminator D 의 학습

Generator G

$$L_{MLM}(x, \Theta_G) = E(\sum_{i \in m} -log p_G(x_i | x^{masked}))$$

Discriminator D

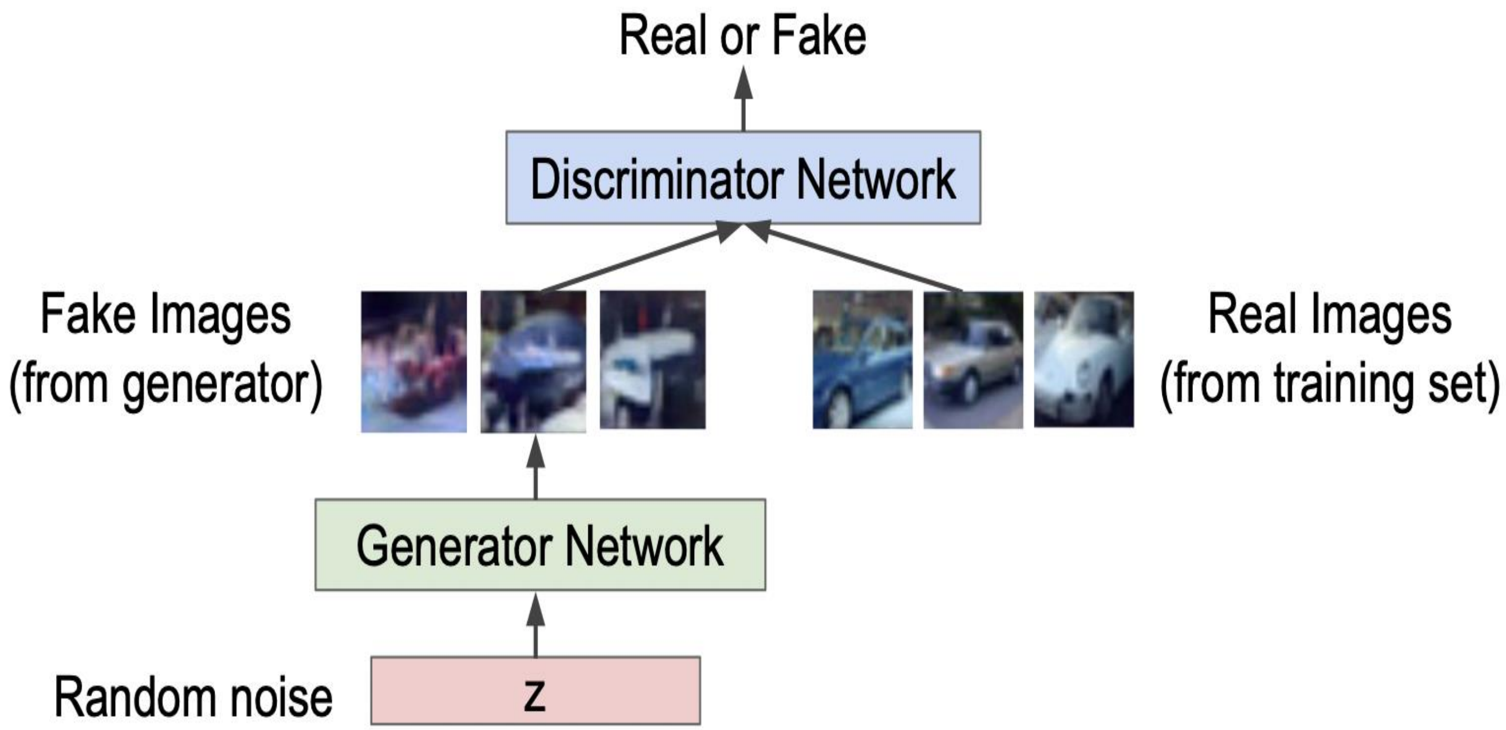
$$L_{Disc}(x, \theta_D) = E(\sum_{t=1}^{n} -(x_t^{corrupt} = x_t) log D(x^{corrupt}, t) - (x_t^{corrupt} \neq x_t) log(1 - D(x^{corrupt}, t)))$$

대용량 코퍼스에 대해서 generator loss와 discriminator loss의 합을 최소화하도록 학습

$$\min_{\theta_G, \theta_D} \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\mathbf{x}, \theta_G) + \lambda \mathcal{L}_{Disc}(\mathbf{x}, \theta_D)$$

$\lambda = 50$

GAN

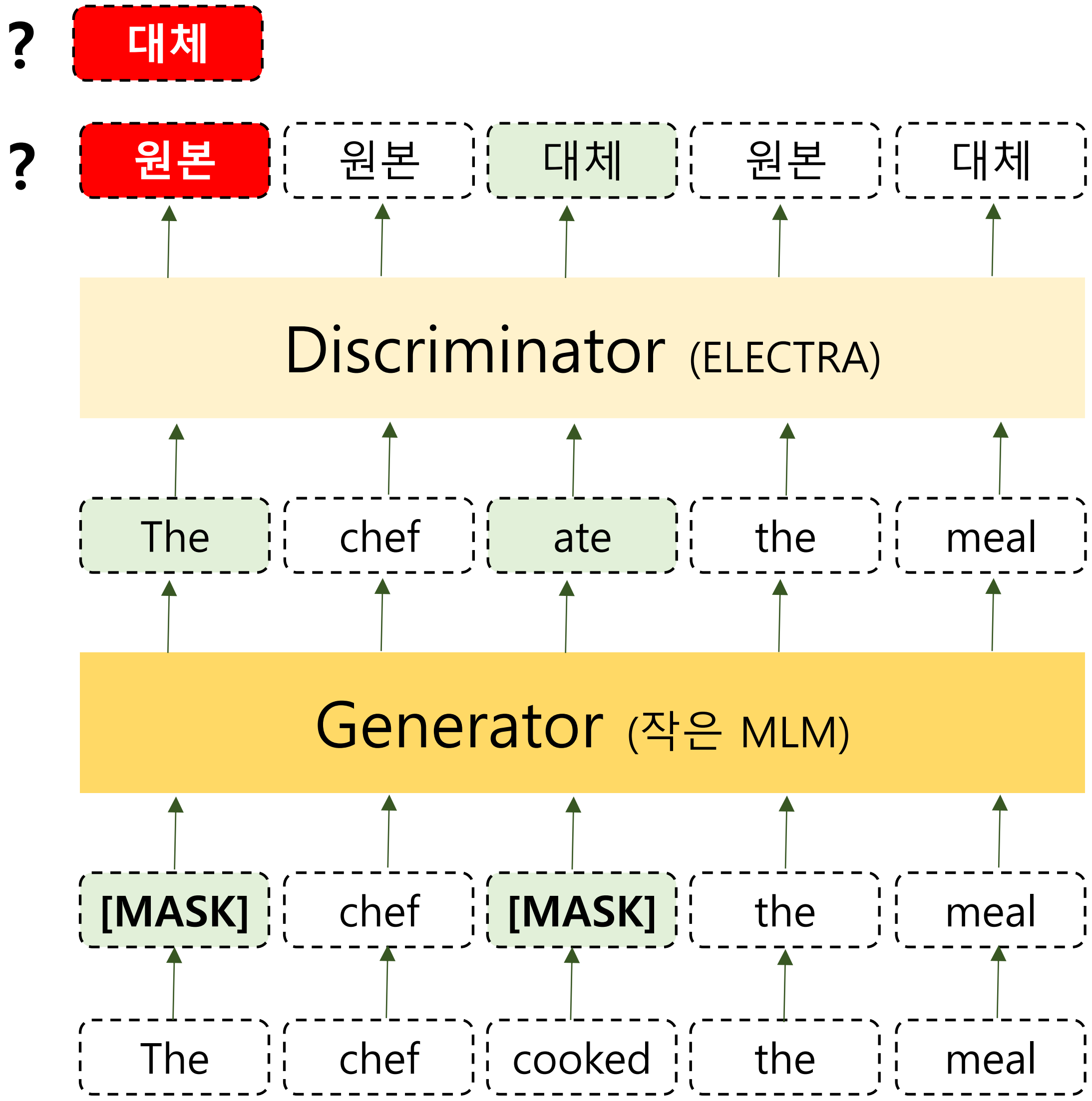


Generator : 진짜같은 image를 만들어내서 Discriminator를 속임
Discriminator : Generator가 만들어낸 이미지인지 실제 이미지인지 구분

GAN vs ELECTRA

1. **Generator가 원래 토큰과 동일한 토큰을 생성한다면?**

GAN : negetive sample (fake)로 간주
ELECTRA : positive sample로 간주



GAN vs ELECTRA

2. 학습 방법?

GAN : generator와 discriminator가 adversarial하게 학습
ELECTRA : generator와 discriminator가 각자 학습

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

GAN vs ELECTRA

3. Generator의 입력으로 노이즈 벡터?

GAN : 넣음
ELECTRA : 넣지 않음

EXPERIMENTS

Experiments

Experiments : SETUP

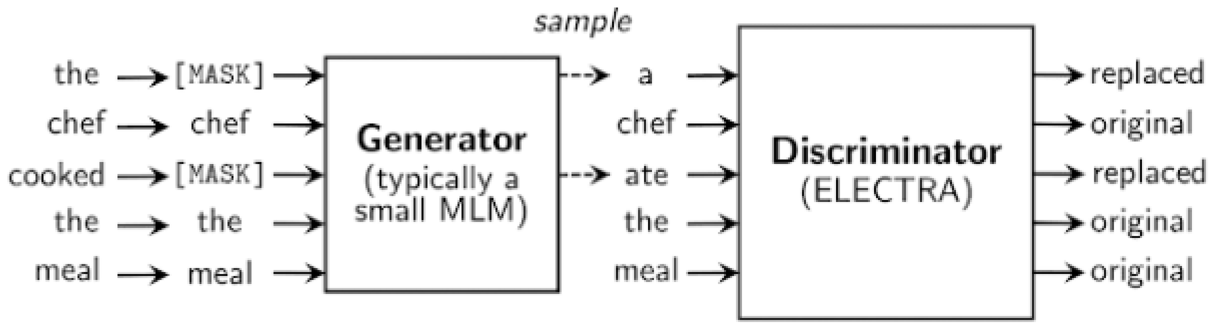
- GLUE - RTE,MNLI,QNLI, MRPC, QQP, STS, SST, and CoLA.
- SQuAD dataset 벤치마크
- BERT와 동일한 사전학습 데이터셋 으로 훈련
  (3.3 Billion tokens from Wikipedia and BooksCorpus)
- 모델 구조와 대부분의 하이퍼파라미터는 BERT와 동일
- fine-tuning 을 위해 간단한 linear classifiers 레이어 추가
- fine-tuning시 작은 평가 데이터셋의 랜덤시드 의존성을 고려해
  동일 체크포인트에서 평가의 정확도는 10회 중앙값 보고

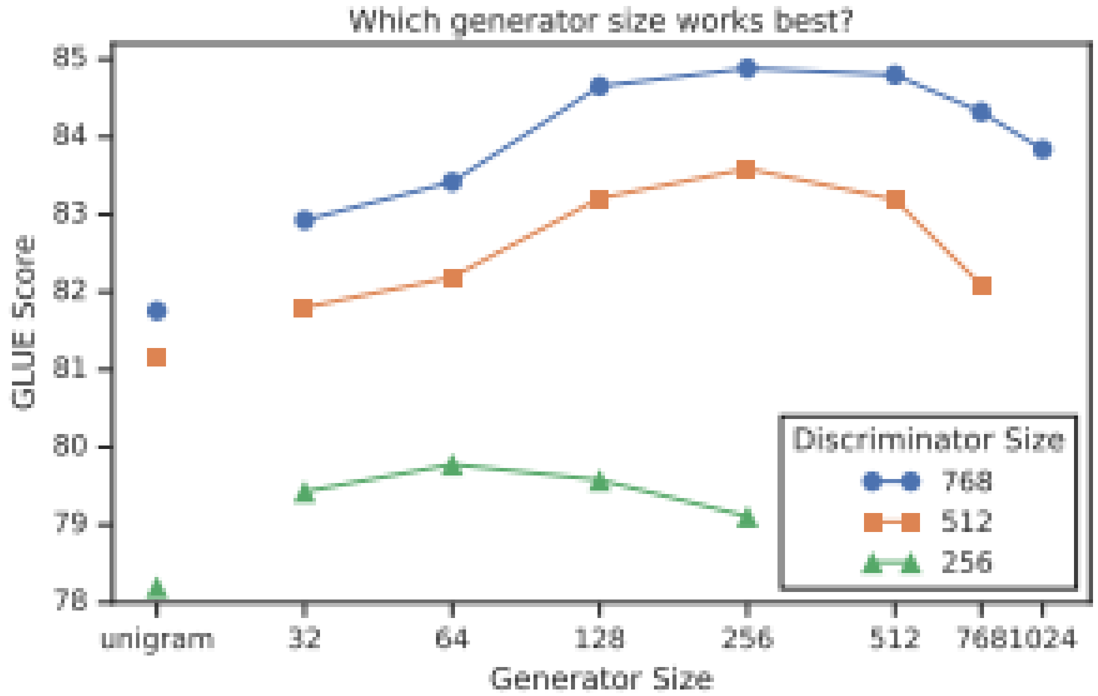
Experiments : MODEL EXTENSIONS

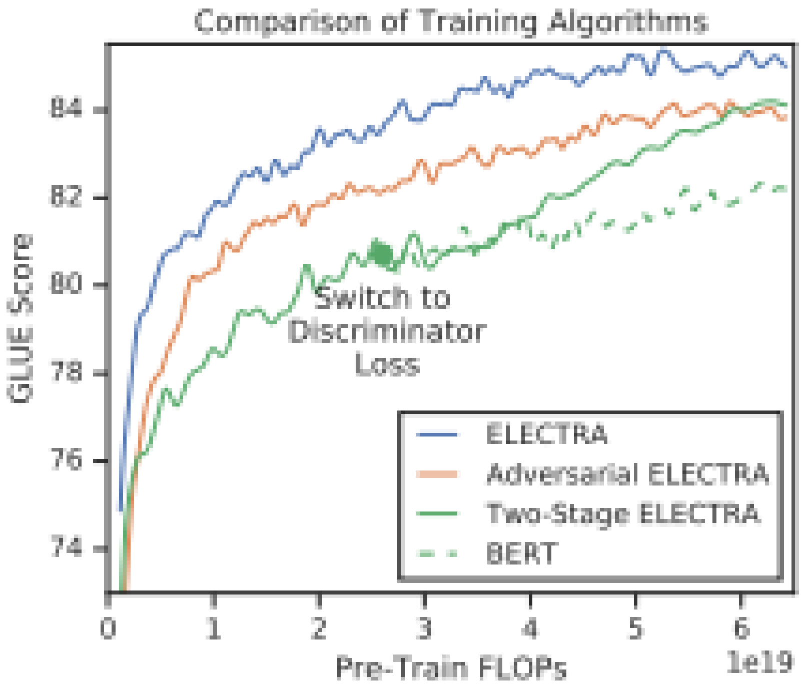
- Weight Sharing
- Smaller Generators
- Training Algorithms

Experiments : MODEL EXTENSIONS

- Weight Sharing
- Smaller Generators

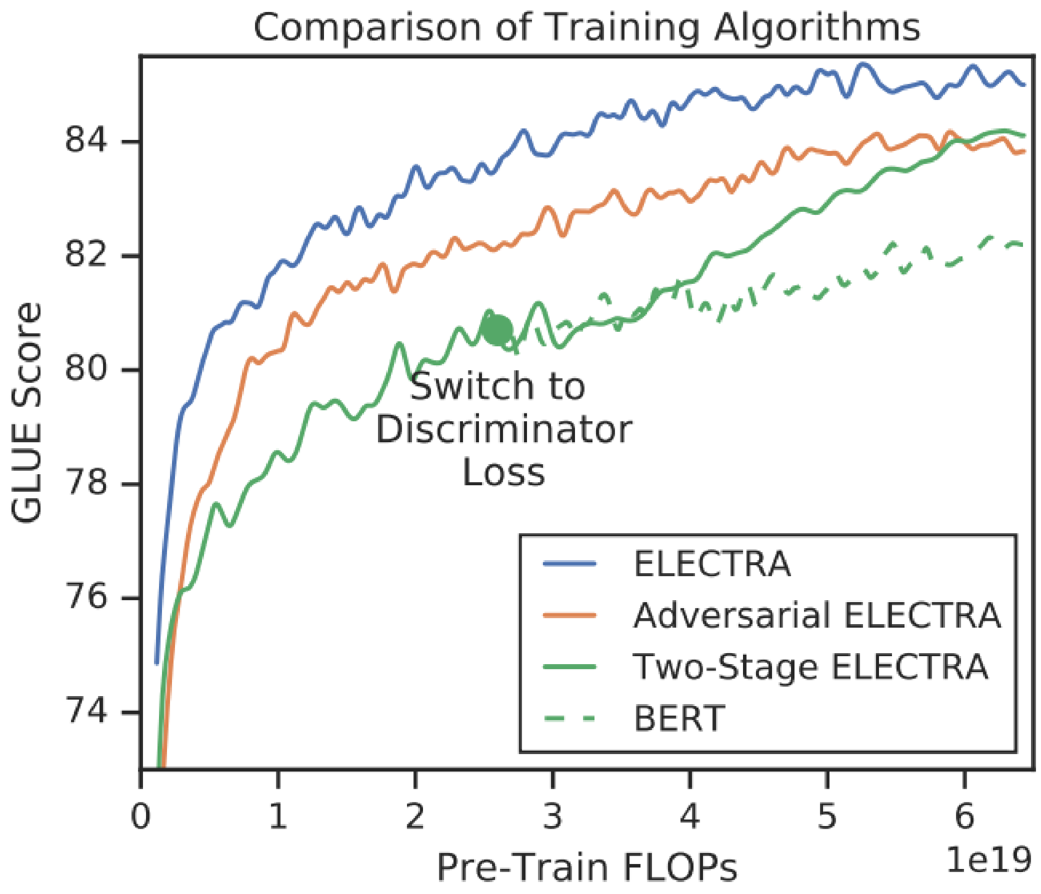






*FLOPS는 1초당 부동 소수점 연산(곱셈)의 명령 실행 횟수를 나타내는 단위

Training Algorithms



- **Adversarial Contrastive Estimation**

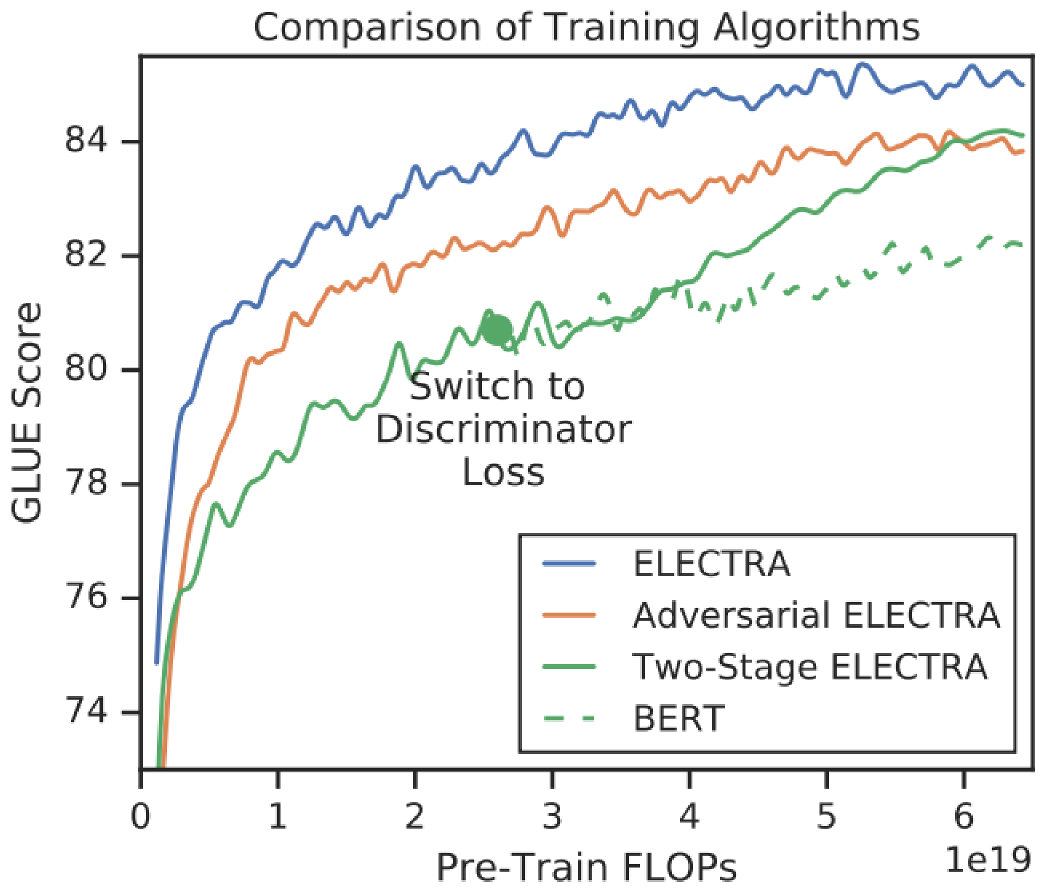
  강화 학습을 이용하여 생성 모델을 GAN과 같이 적대적인 방법으로 학습하는 방법

- **Two-stage training**

  1. n 단계 동안 L MLM G만 훈련
  2. 판별 모델의 weight를 생성 모델의 것으로 초기화하고, 생성모델의 가중치를 고정된 상태로 유지, 판별 모델만 n step 학습 .

*FLOPS는 1초당 부동 소수점 연산(곱셈)의 명령 실행 횟수를 나타내는 단위

Training Algorithms result



- **Adversarial Contrastive Estimation**

생성모델에 대해 판별모델의 목적함수로 옮겨 갈때 성능이 증가하였지만, 기존 방식 보다 성능 낮음

- **Two-stage training problems**

1. poor sample efficiency
2. 생성 모델의 낮은 엔트로피 출력 분포

기존방식 사용

*FLOPS는 1초당 부동 소수점 연산(곱셈)의 명령 실행 횟수를 나타내는 단위

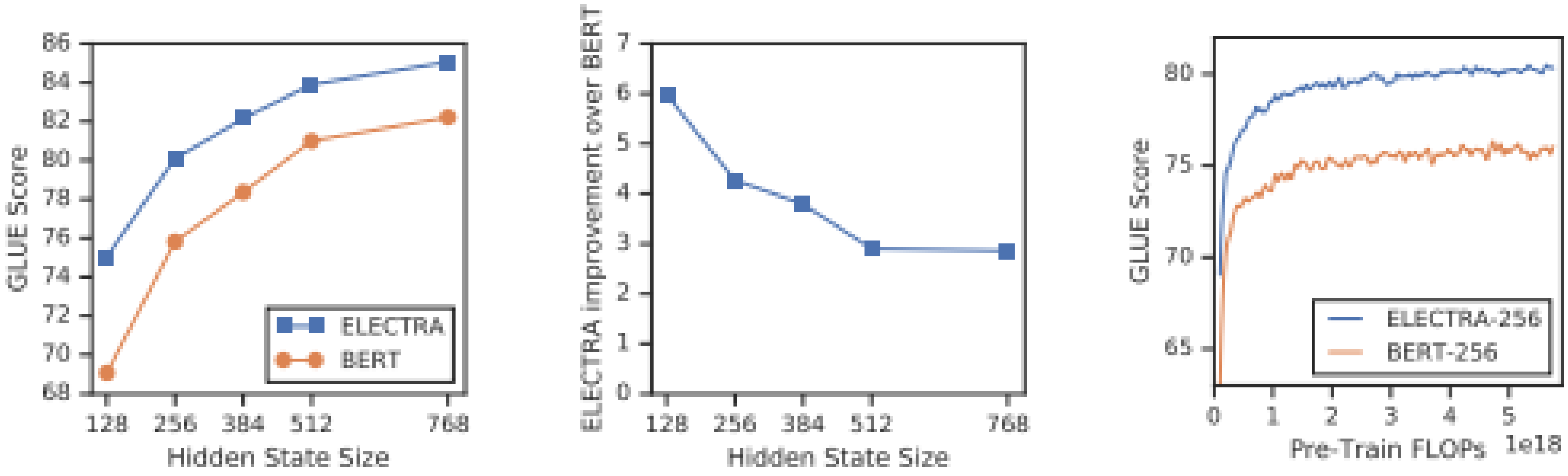
Efficiency Analysis

ELECTRA 15%

원래 ELECTRA와 동일하나, 판별 모델의 loss는 마스킹된 15%의 토큰에서 온 것만을 사용

Model	ELECTRA	All-Tokens MLM	Replace MLM	ELECTRA 15%	BERT
GLUE score	85.0	84.3	82.4	82.4	82.2

Table 5: Compute-efficiency experiments (see text for details).



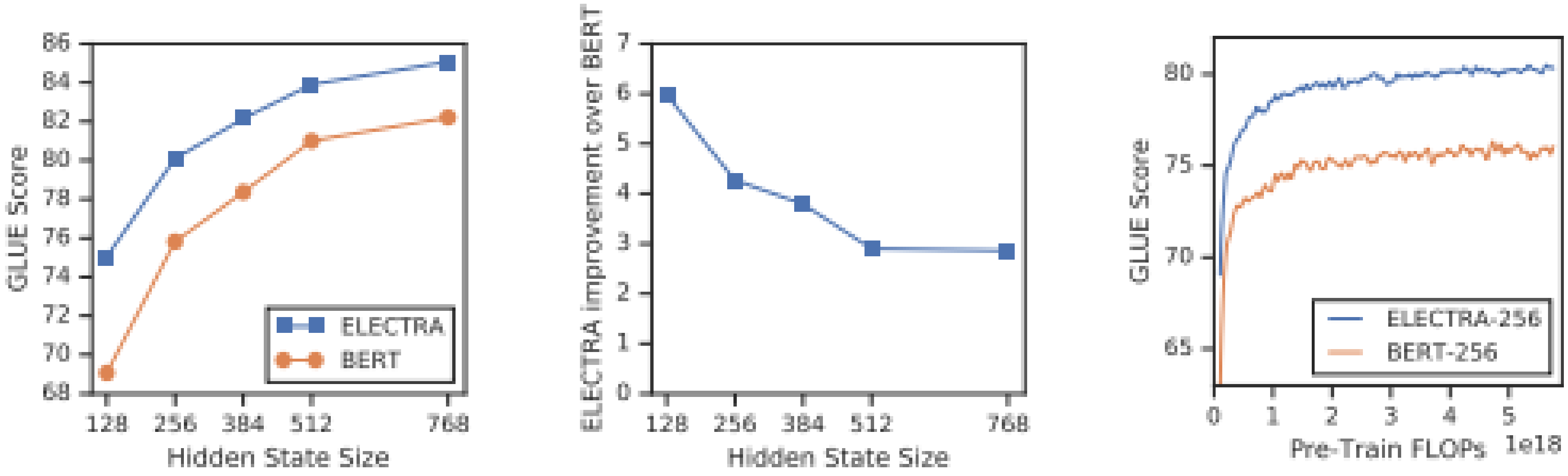
Efficiency Analysis

Replace MLM

MLM과 비슷하지만, 마스킹된 토큰 [MASK]를 인풋으로 받는 대신,
생성 모델이 만들어낸 토큰으로 대체하여 MLM을 진행

Model	ELECTRA	All-Tokens MLM	Replace MLM	ELECTRA 15%	BERT
GLUE score	85.0	84.3	82.4	82.4	82.2

Table 5: Compute-efficiency experiments (see text for details).



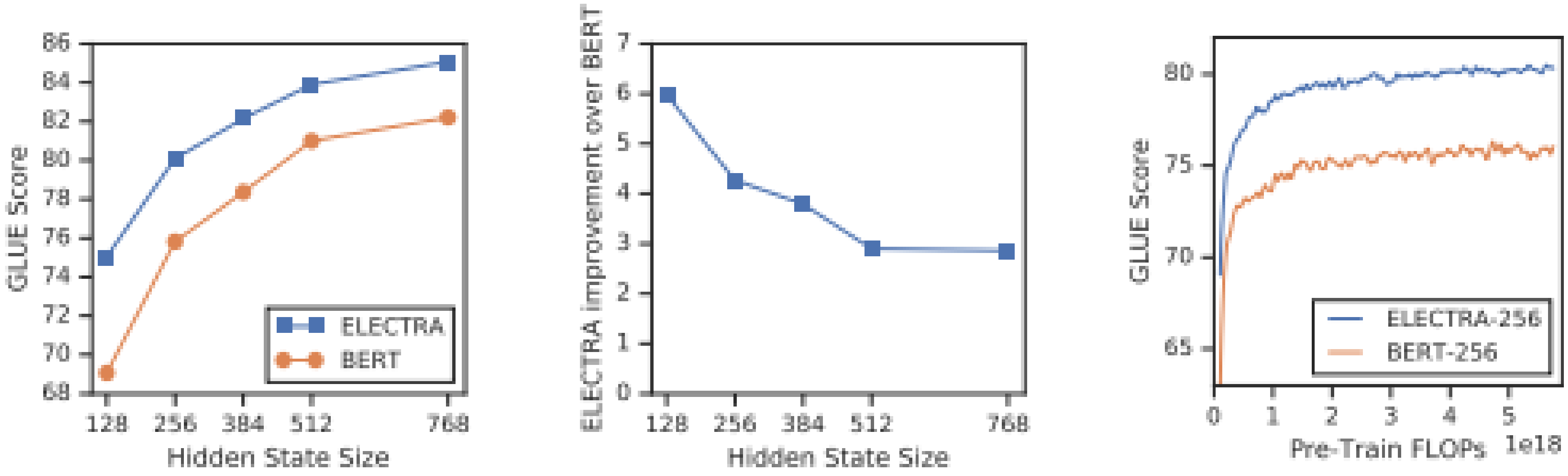
Efficiency Analysis

All-Tokens MLM

BERT와 ELECTRA를 합친 것으로 mask된 토큰이 아닌 모든 토큰을 예측하는 모델이다.

Model	ELECTRA	All-Tokens MLM	Replace MLM	ELECTRA 15%	BERT
GLUE score	85.0	84.3	82.4	82.4	82.2

Table 5: Compute-efficiency experiments (see text for details).



MODEL-DETAIL

Small-model

Model	Train / Infer FLOPs	Speedup	Params	Train Time + Hardware	GLUE
ELMo	3.3e18 / 2.6e10	19x / 1.2x	96M	14d on 3 GTX 1080 GPUs	71.2
GPT	4.0e19 / 3.0e10	1.6x / 0.97x	117M	25d on 8 P6000 GPUs	78.8
BERT-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	75.1
BERT-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	82.2
ELECTRA-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	79.9
50% trained	7.1e17 / 3.7e9	90x / 8x	14M	2d on 1 V100 GPU	79.0
25% trained	3.6e17 / 3.7e9	181x / 8x	14M	1d on 1 V100 GPU	77.7
12.5% trained	1.8e17 / 3.7e9	361x / 8x	14M	12h on 1 V100 GPU	76.0
6.25% trained	8.9e16 / 3.7e9	722x / 8x	14M	6h on 1 V100 GPU	74.1
ELECTRA-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	85.1

ELECTRA-small 결과
같은 계산량의 BERT-small보다 성능이 좋음,
훨씬 계산이 많이 필요했던 GPT보다도 좋음

MODEL-DETAIL

Large Models

Model	Train FLOPs	Params	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	Avg.
BERT	1.9e20 (0.27x)	335M	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4	84.0
RoBERTa-100K	6.4e20 (0.90x)	356M	66.1	95.6	**91.4**	92.2	92.0	89.3	94.0	82.7	87.9
RoBERTa-500K	3.2e21 (4.5x)	356M	68.0	96.4	90.9	92.1	92.2	90.2	94.7	86.6	88.9
XLNet	3.9e21 (5.4x)	360M	69.0	**97.0**	90.8	92.2	92.3	90.8	94.9	85.9	89.1
BERT (ours)	7.1e20 (1x)	335M	67.0	95.9	89.1	91.2	91.5	89.6	93.5	79.5	87.2
ELECTRA-400K	7.1e20 (1x)	335M	**69.3**	96.0	90.6	92.1	**92.4**	90.5	94.5	86.8	89.0
ELECTRA-1.75M	3.1e21 (4.4x)	335M	69.1	96.9	90.8	**92.6**	**92.4**	**90.9**	**95.0**	**88.0**	**89.5**

<GLUE dev set에 대한 결과>

Model	Train FLOPs	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	WNLI	Avg.*	Score
BERT	1.9e20 (0.06x)	60.5	94.9	85.4	86.5	89.3	86.7	92.7	70.1	65.1	79.8	80.5
RoBERTa	3.2e21 (1.02x)	67.8	96.7	89.8	91.9	90.2	90.8	95.4	88.2	89.0	88.1	88.1
ALBERT	3.1e22 (10x)	69.1	**97.1**	**91.2**	92.0	90.5	**91.3**	–	89.2	91.8	89.0	–
XLNet	3.9e21 (1.26x)	70.2	**97.1**	90.5	**92.6**	90.4	90.9	–	88.5	**92.5**	89.1	–
ELECTRA	3.1e21 (1x)	**71.7**	**97.1**	90.7	92.5	**90.8**	**91.3**	**95.8**	**89.8**	**92.5**	**89.5**	**89.4**

큰 모델을 학습시켜 보았을 때,
ELECTRA는 XLNet이나 RoBERTa 사전학습에
필요한 계산량의 1/4만 사용해도 비슷한 성능 결과

PRE-TRAINING, FINE-TUNING

PRE-TRAINING DETAILS

- 대부분 BERT와 동일한 하이퍼파라미터를 사용
- 손실에서 판별자 목적의 가중치인 λ를 50.8로 설정
- 전처리 대신에 즉석에서 결정된 마스킹된 위치와 함께 동적 토큰 마스킹을 사용
- 원본 BERT 논문에서 제안한 Next-sentence-prediction Objective를 사용하지 않음(최근 연구에 따르면 점수가 향상되지 않는 것으로 나타남)
- ELECTRA-Large 모델의 경우 더 높은 마스크 퍼센트 (15 대신 25)를 사용 : Generator가 15% 마스킹으로 높은 정확도를 달성하여 교체된 토큰이 거의 없음을 확인함
- 초기 실험에서 [1e-4, 2e-4, 3e-4, 5e-4] 중 Base 및 Small 모델에 대한 최상의 학습률을 검색하고 [1, 10, 20, 50, 100] 중 λ를 선택

FINE-TUNING DETAILS

- 라지모델은 대부분 Clark et al.의 하이퍼파라미터를 사용.
- RoBERTa가 더 많은 training epoch(3이 아닌 10까지)를 사용한다는 사실을 알게 된 후 ,각 작업에 대해 [10, 3] 중에서 가장 좋은 수의 train epoch를 찾음
- SQuAD의 경우 BERT 및 RoBERTa와 일치하도록 학습 epochs 수를 2로 줄임
- Base모델은 [3e-5, 5e-5, 1e-4, 1.5e-4]에서 학습률 선택, layer-wise learning-rate decay의 경우 [0.9, 0.8, 0.7]에서 선택, 나머지는 라지모델과 동일한 하이퍼파라미터를 사용
- 이에 반해 BERT, XLNet, RoBERTa 등의 GLUE에 대한 선행 연구에서 태스크별로 최적의 하이퍼파라미터를 별도 선택 (동일한 종류의 추가 초매개변수 검색을 수행하면 결과가 약간 향상될 것으로 예상)

Conclusion

- ✓ language representation learning을 위한 새로운 Self-supervised learning task인 **Replaced Token Detection(RTD)**을 제안

- ✓ 핵심은 text encoder가 작은 generator가 만들어낸 고품질의 negative sample과 입력 토큰을 구별하도록 텍스트 인코더를 학습시키는 것

- ✓ MLM과 비교해, 논문의 pre-training objective 계산이 훨씬 효율적이며 downstream task에서 더 좋은 성능을 낼 수 있다는 것을 많은 실험을 통해 확인

- ✓ 이 연구를 통해 컴퓨팅 리소스에 대한 접근 권한이 적은 연구원과 실무자가 적은 컴퓨팅 리소스로도 pre-trained text encoder에 대해 많은 연구개발을 할 수 있길 바람

- ✓ pre-training과 관련된 향후 연구가 절대적 성능 지표만큼 계산량과 파라미터 수 등의 효율성도 함께 고려했으면 하는 바람을 나타냄

Source Code

Source Code

Google-research / electra

https://github.com/google-research/electra

ELECTRA

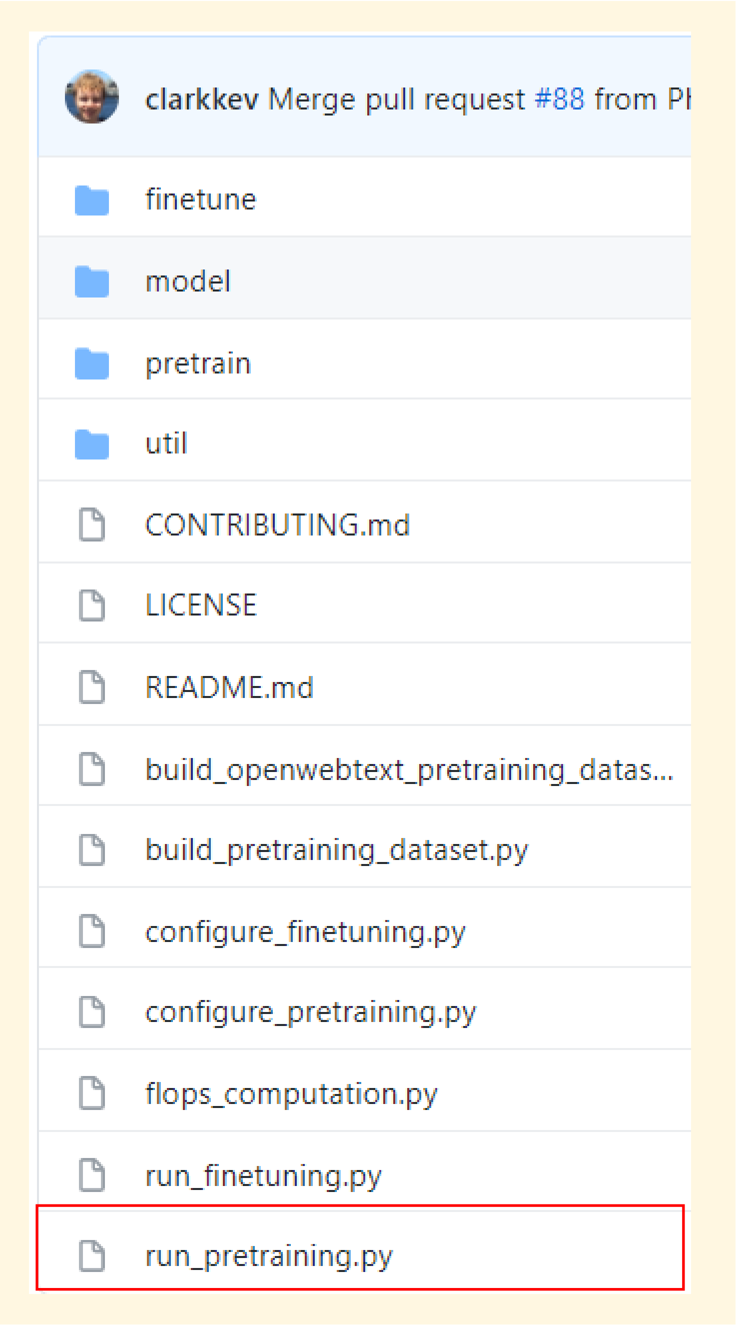
Introduction

ELECTRA is a method for self-supervised language representation learning. It can be used to pre-train transformer networks using relatively little compute. ELECTRA models are trained to distinguish "real" input tokens vs "fake" input tokens generated by another neural network, similar to the discriminator of a GAN. At small scale, ELECTRA achieves strong results even when trained on a single GPU. At large scale, ELECTRA achieves state-of-the-art results on the SQuAD 2.0 dataset.

For a detailed description and experimental results, please refer to our ICLR 2020 paper ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.

This repository contains code to pre-train ELECTRA, including small ELECTRA models on a single GPU. It also supports fine-tuning ELECTRA on downstream tasks including classification tasks (e.g,. GLUE), QA tasks (e.g., SQuAD), and sequence tagging tasks (e.g., text chunking).

This repository also contains code for **Electric**, a version of ELECTRA inspired by energy-based models. Electric provides a more principled view of ELECTRA as a "negative sampling" cloze model. It can also efficiently produce pseudo-likelihood scores for text, which can be used to re-rank the outputs of speech recognition or machine translation systems. For details on Electric, please refer to out EMNLP 2020 paper Pre-Training Transformers as Energy-Based Cloze Models.



Config – configure_pretraining.py

```python
# model settings
self.model_size = "small"  # one of "small", "base", or "large"
# override the default transformer hparams for the provided model size; see
# modeling.BertConfig for the possible hparams and util.training_utils for
# the defaults
self.model_hparam_overrides = (
    kwargs["model_hparam_overrides"]
    if "model_hparam_overrides" in kwargs else {})
self.embedding_size = None  # bert hidden size by default
self.vocab_size = 30522  # number of tokens in the vocabulary
self.do_lower_case = True  # lowercase the input?
```

```python
# batch sizes
self.max_seq_length = 128
self.train_batch_size = 128
self.eval_batch_size = 128
```

```python
# model settings
self.model_size = "small"  # one of "small", "base", or "large"
# override the default transformer hparams for the provided model size; see
# modeling.BertConfig for the possible hparams and util.training_utils for
# the defaults
self.model_hparam_overrides = (
    kwargs["model_hparam_overrides"]
    if "model_hparam_overrides" in kwargs else {})
self.embedding_size = None  # bert hidden size by default
self.vocab_size = 30522  # number of tokens in the vocabulary
self.do_lower_case = True  # lowercase the input?
```

```python
# generator settings
self.uniform_generator = False  # generator is uniform at random
self.two_tower_generator = False  # generator is a two-tower cloze model
self.untied_generator_embeddings = False  # tie generator/discriminator
                                          # token embeddings?
self.untied_generator = True  # tie all generator/discriminator weights?
self.generator_layers = 1.0  # frac of discriminator layers for generator
self.generator_hidden_size = 0.25  # frac of discrim hidden size for gen
self.disallow_correct = False  # force the generator to sample incorrect
                               # tokens (so 15% of tokens are always
                               # fake)
self.temperature = 1.0  # temperature for sampling from generator
```

```python
def get_bert_config(config):
  """Get model hyperparameters based on a pretraining/finetuning config"""
  if config.model_size == "large":
    args = {"hidden_size": 1024, "num_hidden_layers": 24}
  elif config.model_size == "base":
    args = {"hidden_size": 768, "num_hidden_layers": 12}
  elif config.model_size == "small":
    args = {"hidden_size": 256, "num_hidden_layers": 12}
  else:
    raise ValueError("Unknown model size", config.model_size)
  args["vocab_size"] = config.vocab_size
  args.update(**config.model_hparam_overrides)
  # by default the ff size and num attn heads are determined by the hidden size
  args["num_attention_heads"] = max(1, args["hidden_size"] // 64)
  args["intermediate_size"] = 4 * args["hidden_size"]
  args.update(**config.model_hparam_overrides)
  return modeling.BertConfig.from_dict(args)
```

Tokenizer

road
wasn
although
due
major
died
village
third
knew
2016
asked
turned
st
wanted
say
##p
together
received
main
son
served
different
##en
behind
himself
felt
members
power
football
law
voice
play
##in
near
park
history
30
having
2005

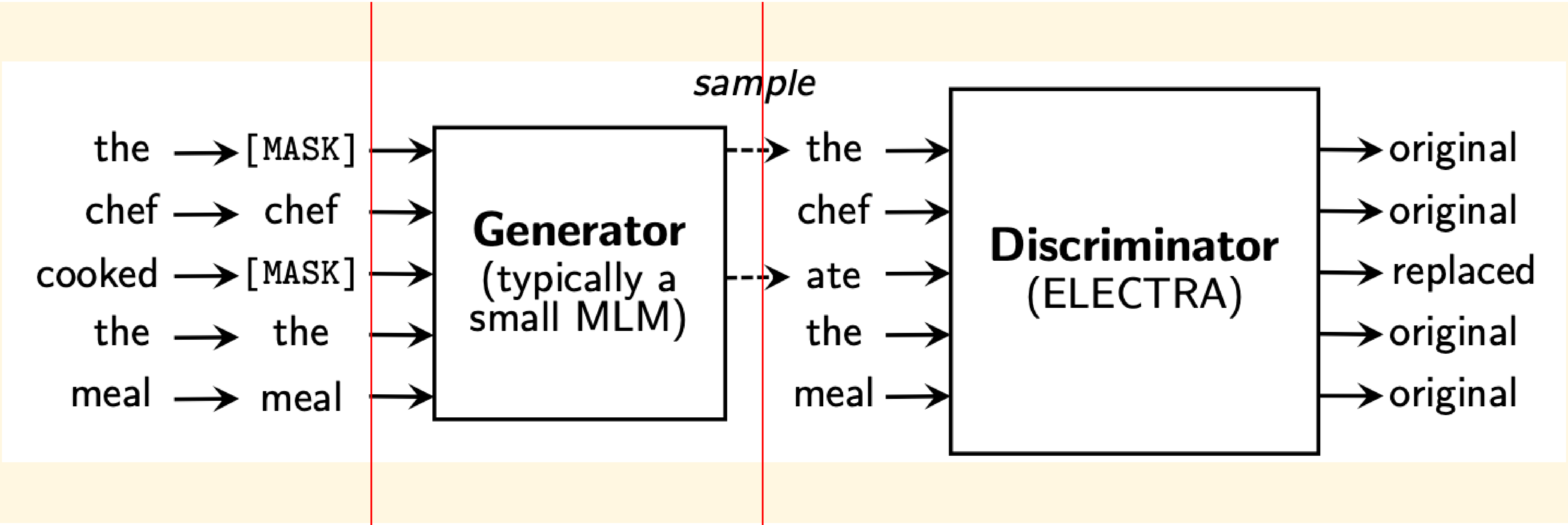
[Vocab]

Wordpiece Vocabulary


```
:1929,"空":1930,"立":1931,"章":1932,"竹":1933,"糸":1934,"美":1935,"義":1936,"耳":1937,"良":1938,"艹":1939,"花":1940,"英":1941,"華":1942,"葉":1943,"藤":1944,"行":1945,"街":194
"郎":1958,"郡":1959,"部":1960,"都":1961,"里":1962,"野":1963,"金":1964,"鈴":1965,"镇":1966,"長":1967,"門":1968,"間":1969,"阝":1970,"阿":1971,"陳":1972,"陽":1973,"雄":1974,"青"
,"(":1987,")":1988,",":1989,"-":1990,".":1991,"/":1992,":":1993,"?":1994,"~":1995,"the":1996,"of":1997,"and":1998,"in":1999,"to":2000,"was":2001,"he":2002,"is":2003
":2014,"##s":2015,"she":2016,"you":2017,"had":2018,"an":2019,"were":2020,"but":2021,"be":2022,"this":2023,"are":2024,"not":2025,"my":2026,"they":2027,"one":2028,"which":2029
,"who":2040,"out":2041,"been":2042,"when":2043,"after":2044,"there":2045,"into":2046,"new":2047,"two":2048,"its":2049,"##a":2050,"time":2051,"would":2052,"no":2053,"what":205
:2064,"if":2065,"like":2066,"back":2067,"them":2068,"only":2069,"some":2070,"could":2071,"##i":2072,"where":2073,"just":2074,"##ing":2075,"during":2076,"before":2077,"##n":20
,"world":2088,"may":2089,"between":2090,"down":2091,"well":2092,"three":2093,"##d":2094,"year":2095,"while":2096,"will":2097,"##ed":2098,"##r":2099,"##y":2100,"later":2101,"
ople":2111,"part":2112,"know":2113,"against":2114,"your":2115,"many":2116,"second":2117,"university":2118,"both":2119,"national":2120,"##er":2121,"these":2122,"don":2123,"kno
didn":2134,"##ly":2135,"team":2136,"american":2137,"because":2138,"de":2139,"##l":2140,"born":2141,"united":2142,"film":2143,"since":2144,"still":2145,"long":2146,"work":2147
ight":2157,"man":2158,"eyes":2159,"house":2160,"season":2161,"war":2162,"states":2163,"including":2164,"took":2165,"life":2166,"north":2167,"same":2168,"each":2169,"called":2
":2179,"won":2180,"area":2181,"here":2182,"going":2183,"10":2184,"away":2185,"series":2186,"left":2187,"home":2188,"music":2189,"best":2190,"make":2191,"hand":2192,"number":2
":2202,"end":2203,"good":2204,"too":2205,"following":2206,"released":2207,"game":2208,"played":2209,"little":2210,"began":2211,"district":2212,"##m":2213,"old":2214,"want":22
,"west":2225,"##u":2226,"face":2227,"think":2228,"##es":2229,"2010":2230,"government":2231,"##h":2232,"march":2233,"came":2234,"small":2235,"general":2236,"town":2237,"june":
246,"along":2247,"international":2248,"2011":2249,"air":2250,"july":2251,"club":2252,"went":2253,"january":2254,"october":2255,"our":2256,"august":2257,"april":2258,"york":22
,"father":2269,"public":2270,"##us":2271,"come":2272,"men":2273,"five":2274,"set":2275,"station":2276,"church":2277,"##c":2278,"next":2279,"former":2280,"november":2281,"roo
tem":2291,"let":2292,"love":2293,"2006":2294,"though":2295,"every":2296,"2014":2297,"look":2298,"song":2299,"water":2300,"century":2301,"without":2302,"body":2303,"black":230
ulation":2313,"river":2314,"named":2315,"band":2316,"white":2317,"started":2318,"##an":2319,"once":2320,"15":2321,"20":2322,"should":2323,"18":2324,"2015":2325,"service":232
335,"children":2336,"february":2337,"book":2338,"why":2339,"11":2340,"door":2341,"need":2342,"president":2343,"order":2344,"final":2345,"road":2346,"wasn":2347,"although":23
2357,"st":2358,"wanted":2359,"say":2360,"##p":2361,"together":2362,"received":2363,"main":2364,"son":2365,"served":2366,"different":2367,"##en":2368,"behind":2369,"himself":2
r":2379,"park":2380,"history":2381,"30":2382,"having":2383,"2005":2384,"16":2385,"##man":2386,"saw":2387,"mother":2388,"##al":2389,"army":2390,"point":2391,"front":2392,"help
oung":2402,"14":2403,"put":2404,"published":2405,"country":2406,"division":2407,"across":2408,"told":2409,"13":2410,"often":2411,"ever":2412,"french":2413,"london":2414,"cen
424,"tell":2425,"among":2426,"species":2427,"really":2428,"according":2429,"central":2430,"half":2431,"2004":2432,"form":2433,"original":2434,"gave":2435,"office":2436,"makin
,"german":2446,"player":2447,"run":2448,"business":2449,"woman":2450,"community":2451,"cup":2452,"might":2453,"million":2454,"land":2455,"2000":2456,"court":2457,"development
67,"become":2468,"sure":2469,"research":2470,"almost":2471,"director":2472,"council":2473,"la":2474,"##2":2475,"career":2476,"things":2477,"using":2478,"island":2479,"##z":24
,"support":2490,"control":2491,"field":2492,"students":2493,"2003":2494,"education":2495,"married":2496,"##b":2497,"nothing":2498,"worked":2499,"others":2500,"record":2501,"
ry":2510,"established":2511,"non":2512,"returned":2513,"feel":2514,"does":2515,"title":2516,"written":2517,"thing":2518,"feet":2519,"william":2520,"far":2521,"co":2522,"asso
"100":2531,"##na":2532,"department":2533,"hall":2534,"role":2535,"various":2536,"production":2537,"21":2538,"19":2539,"heart":2540,"2001":2541,"living":2542,"fire":2543,"vers
:2552,"case":2553,"society":2554,"region":2555,"present":2556,"radio":2557,"period":2558,"looking":2559,"least":2560,"total":2561,"keep":2562,"england":2563,"wife":2564,"pro
n":2574,"##6":2575,"political":2576,"george":2577,"services":2578,"taken":2579,"created":2580,"##7":2581,"further":2582,"able":2583,"reached":2584,"david":2585,"union":2586,"
,"##x":2595,"appeared":2596,"position":2597,"ground":2598,"lead":2599,"rock":2600,"dark":2601,"election":2602,"23":2603,"board":2604,"france":2605,"hair":2606,"course":2607,"
,"moment":2617,"##te":2618,"someone":2619,"##8":2620,"summer":2621,"project":2622,"announced":2623,"san":2624,"less":2625,"wrote":2626,"past":2627,"followed":2628,"##5":2629
ne":2638,"1999":2639,"design":2640,"considered":2641,"northern":2642,"god":2643,"stop":2644,"battle":2645,"toward":2646,"european":2647,"outside":2648,"described":2649,"track
```


[Token]

ELECTRA Structure



Masking

Generator

Discriminator

Pretrain-Model - masking

```
# Mask the input
unmasked_inputs = pretrain_data.features_to_inputs(features)
masked_inputs = pretrain_helpers.mask(
    config, unmasked_inputs, config.mask_prob)
```

[run_pretraining.py]

```
# model inputs - it's a bit nicer to use a namedtuple rather than keep the
# features as a dict
Inputs = collections.namedtuple(
    "Inputs", ["input_ids", "input_mask", "segment_ids", "masked_lm_positions",
               "masked_lm_ids", "masked_lm_weights"])
```

[pretrain_data.py]

```
# Update the input ids
replace_with_mask_positions = masked_lm_positions * tf.cast(
    tf.less(tf.random.uniform([B, N]), 0.85), tf.int32)
inputs_ids, _ = scatter_update(
    inputs.input_ids, tf.fill([B, N], vocab["[MASK]"]),
    replace_with_mask_positions)

return pretrain_data.get_updated_inputs(
    inputs,
    input_ids=tf.stop_gradient(inputs_ids),
    masked_lm_positions=masked_lm_positions,
    masked_lm_ids=masked_lm_ids,
    masked_lm_weights=masked_lm_weights
)
```

[pretrain_helpers.py]

$$mi \sim unif\{1, n\} \;\; for \;\; i = 1 \; to \; k$$

[마스킹 위치 랜덤]

$$x^{masked} = REPLACE(x, m, [MASK])$$

[MASK 토큰으로 대체]

B – Batch Size
L – Sequence Length
D – Depth
N – modeling.get_shape_list(position)[1]

Pretrain-Model – Generator - Config

```
# generator settings
self.uniform_generator = False  # generator is uniform at random
self.two_tower_generator = False  # generator is a two-tower cloze model
self.untied_generator_embeddings = False  # tie generator/discriminator
                                          # token embeddings?
self.untied_generator = True  # tie all generator/discriminator weights?
self.generator_layers = 1.0  # frac of discriminator layers for generator
self.generator_hidden_size = 0.25  # frac of discrim hidden size for gen
self.disallow_correct = False  # force the generator to sample incorrect
                               # tokens (so 15% of tokens are always
                               # fake)
self.temperature = 1.0  # temperature for sampling from generator
```

[Generator setting]

[Generator Config List]

1. Generator is uniform at random
2. Generator is a two-tower cloze model
3. Tie generator / discriminator token embedding
4. Tie all generator / discriminator weights

Pretrain-Model – Generator1

```python
    if config.uniform_generator:
      # simple generator sampling fakes uniformly at random
      mlm_output = self._get_masked_lm_output(masked_inputs, None)
```

$$p_G(x_t|x^{masked}) = exp(e(x_t)^T h_G(x^{masked})_t) / \sum_{x'} exp(e(x')^T h_G(x^{masked})_t)$$

$p_G(x_t|x^{masked})$

$\Rightarrow$ x^{masked}라는 시퀀스 주어졌을 때 특정 위치 t에 단어 x_t가 들어갈 확률

$exp(e \bullet x_i) / \sum_j exp(e \bullet x_j)$

$\Rightarrow$ softmax 함수

$e(x_t)^T h_G(x^{masked})_t$

$\Rightarrow$ token을 임베딩(e)하여 example의 hidden state 값을 곱

```python
  def _get_masked_lm_output(self, inputs: pretrain_data.Inputs, model):
    """Masked language modeling softmax layer."""
    with tf.variable_scope("generator_predictions"):
      if self._config.uniform_generator:
        logits = tf.zeros(self._bert_config.vocab_size)
        logits_tiled = tf.zeros(
            modeling.get_shape_list(inputs.masked_lm_ids) +
            [self._bert_config.vocab_size])
        logits_tiled += tf.reshape(logits, [1, 1, self._bert_config.vocab_size])
        logits = logits_tiled
      else:
        relevant_reprs = pretrain_helpers.gather_positions(
            model.get_sequence_output(), inputs.masked_lm_positions)
        logits = get_token_logits(
            relevant_reprs, model.get_embedding_table(), self._bert_config)
      return get_softmax_output(
          logits, inputs.masked_lm_ids, inputs.masked_lm_weights,
          self._bert_config.vocab_size)
```

```python
def get_softmax_output(logits, targets, weights, vocab_size):
  oh_labels = tf.one_hot(targets, depth=vocab_size, dtype=tf.float32)
  preds = tf.argmax(logits, axis=-1, output_type=tf.int32)
  probs = tf.nn.softmax(logits)
  log_probs = tf.nn.log_softmax(logits)
  label_log_probs = -tf.reduce_sum(log_probs * oh_labels, axis=-1)
  numerator = tf.reduce_sum(weights * label_log_probs)
  denominator = tf.reduce_sum(weights) + 1e-6
  loss = numerator / denominator
  SoftmaxOutput = collections.namedtuple(
      "SoftmaxOutput", ["logits", "probs", "loss", "per_example_loss", "preds",
                        "weights"])
  return SoftmaxOutput(
      logits=logits, probs=probs, per_example_loss=label_log_probs,
      loss=loss, preds=preds, weights=weights)
```

[run_pretraining.py]

Pretrain-Model – Generator2

```python
class TwoTowerClozeTransformer(object):
  """Build a two-tower Transformer used as Electric's generator."""

  def __init__(self, config, bert_config, inputs: pretrain_data.Inputs,
               is_training, embedding_size):
    ltr = build_transformer(
        config, inputs, is_training, bert_config,
        untied_embeddings=config.untied_generator_embeddings,
        embedding_size=(None if config.untied_generator_embeddings
                        else embedding_size),
        scope="generator_ltr", ltr=True)
    rtl = build_transformer(
        config, inputs, is_training, bert_config,
        untied_embeddings=config.untied_generator_embeddings,
        embedding_size=(None if config.untied_generator_embeddings
                        else embedding_size),
        scope="generator_rtl", rtl=True)
    ltr_reprs = ltr.get_sequence_output()
    rtl_reprs = rtl.get_sequence_output()
    self._sequence_output = tf.concat([roll(ltr_reprs, -1),
                                       roll(rtl_reprs, 1)], -1)
    self._embedding_table = ltr.embedding_table

  def get_sequence_output(self):
    return self._sequence_output

  def get_embedding_table(self):
    return self._embedding_table
```

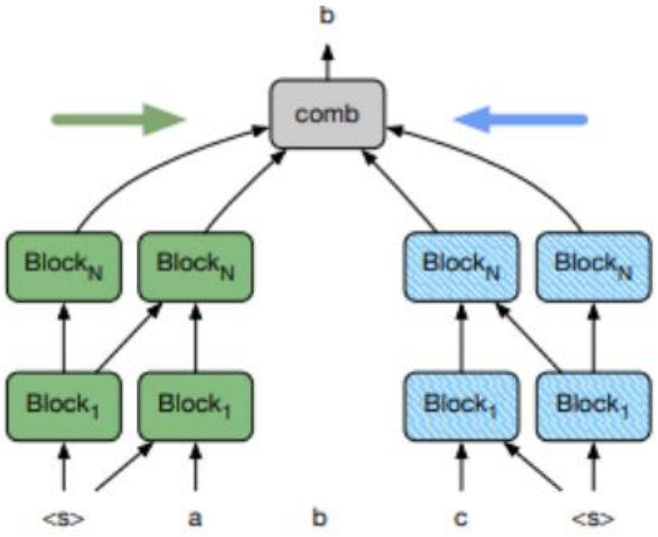


Figure 1: Illustration of the model. Block_i is a standard transformer decoder block. Green blocks operate left to right by masking future time-steps and blue blocks operate right to left. At the top, states are combined with a standard multi-head self-attention module whose output is fed to a classifier that predicts the center token.

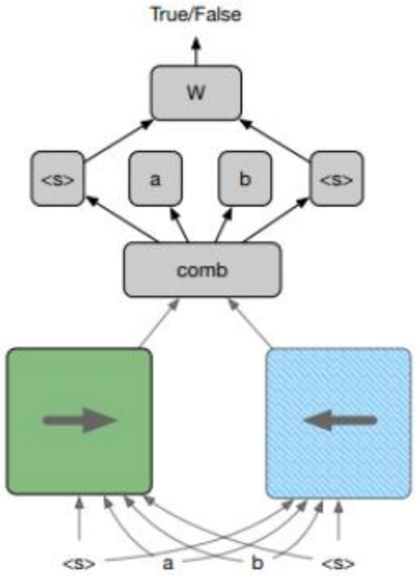


Figure 2: Illustration of fine-tuning for a single-sentence task where the output of the first and last token is fed to a task-specific classifier (W). Masking for the final combination layer (comb) is removed which results in representations based on all forward and backward states (cf. Figure 1).

Cloze-driven Pretraining of Self-attention Networks

Pretrain-Model – Generator3

```
else:
  # small masked language model generator
  generator = build_transformer(
      config, masked_inputs, is_training, generator_config,
      embedding_size=(None if config.untied_generator_embeddings
                      else embedding_size),
      untied_embeddings=config.untied_generator_embeddings,
      scope="generator")
  mlm_output = self._get_masked_lm_output(masked_inputs, generator)
```

```
# generator settings
self.uniform_generator = False  # generator is uniform at random
self.two_tower_generator = False  # generator is a two-tower cloze model
self.untied_generator_embeddings = False  # tie generator/discriminator
                                          # token embeddings?
self.untied_generator = True  # tie all generator/discriminator weights?
self.generator_layers = 1.0  # frac of discriminator layers for generator
self.generator_hidden_size = 0.25  # frac of discrim hidden size for gen
self.disallow_correct = False  # force the generator to sample incorrect
                               # tokens (so 15% of tokens are always
                               # fake)
self.temperature = 1.0  # temperature for sampling from generator
```

[Generator setting]

```
def get_generator_config(config: configure_pretraining.PretrainingConfig,
                         bert_config: modeling.BertConfig):
  """Get model config for the generator network."""
  gen_config = modeling.BertConfig.from_dict(bert_config.to_dict())
  gen_config.hidden_size = int(round(
      bert_config.hidden_size * config.generator_hidden_size))
  gen_config.num_hidden_layers = int(round(
      bert_config.num_hidden_layers * config.generator_layers))
  gen_config.intermediate_size = 4 * gen_config.hidden_size
  gen_config.num_attention_heads = max(1, gen_config.hidden_size // 64)
  return gen_config
```

[run_pretraining.py]

Pretrain-Model – Generator4

```
else:
  # full-sized masked language model generator if using BERT objective or if
  # the generator and discriminator have tied weights
  generator = build_transformer(
      config, masked_inputs, is_training, self._bert_config,
      embedding_size=embedding_size)
  mlm_output = self._get_masked_lm_output(masked_inputs, generator)
```

```
def get_bert_config(config):
  """Get model hyperparameters based on a pretraining/finetuning config"""
  if config.model_size == "large":
    args = {"hidden_size": 1024, "num_hidden_layers": 24}
  elif config.model_size == "base":
    args = {"hidden_size": 768, "num_hidden_layers": 12}
  elif config.model_size == "small":
    args = {"hidden_size": 256, "num_hidden_layers": 12}
  else:
    raise ValueError("Unknown model size", config.model_size)
  args["vocab_size"] = config.vocab_size
  args.update(**config.model_hparam_overrides)
  # by default the ff size and num attn heads are determined by the hidden size
  args["num_attention_heads"] = max(1, args["hidden_size"] // 64)
  args["intermediate_size"] = 4 * args["hidden_size"]
  args.update(**config.model_hparam_overrides)
  return modeling.BertConfig.from_dict(args)
```

Training_utils.py

Pretrain-Model - Generator

Discriminator D input 생성

```
fake_data = self._get_fake_data(masked_inputs, mlm_output.logits)
self.mlm_output = mlm_output
self.total_loss = config.gen_weight * (
    cloze_output.loss if config.two_tower_generator else mlm_output.loss)
```

[Update Generator loss]

[MASK]에서 $p_G(x_t|x)$으로 샘플링한 토큰으로 치환(corrupt)

$$x^{corrupt} = REPLACE(x, m, \hat{x})$$

$$\hat{x} \sim p_G(x_i|x^{masked})\ for\ i\ \in m$$

```
FakedData = collections.namedtuple("FakedData", [
    "inputs", "is_fake_tokens", "sampled_tokens"])
```

[FakedData Structure]

Pretrain-Model - Discriminator

```python
# Discriminator
disc_output = None
if config.electra_objective or config.electric_objective:
  discriminator = build_transformer(
      config, fake_data.inputs, is_training, self._bert_config,
      reuse=not config.untied_generator, embedding_size=embedding_size)
  disc_output = self._get_discriminator_output(
      fake_data.inputs, discriminator, fake_data.is_fake_tokens,
      cloze_output)
  self.total_loss += config.disc_weight * disc_output.loss
```

[run_pretraining.py]

Pretrain-Model - Discriminator

```python
def _get_discriminator_output(
    self, inputs, discriminator, labels, cloze_output=None):
  """Discriminator binary classifier."""
  with tf.variable_scope("discriminator_predictions"):
    hidden = tf.layers.dense(
        discriminator.get_sequence_output(),
        units=self._bert_config.hidden_size,
        activation=modeling.get_activation(self._bert_config.hidden_act),
        kernel_initializer=modeling.create_initializer(
            self._bert_config.initializer_range))
    logits = tf.squeeze(tf.layers.dense(hidden, units=1), -1)
    if self._config.electric_objective:
      log_q = tf.reduce_sum(
          tf.nn.log_softmax(cloze_output.logits) * tf.one_hot(
              inputs.input_ids, depth=self._bert_config.vocab_size,
              dtype=tf.float32), -1)
      log_q = tf.stop_gradient(log_q)
      logits += log_q
      logits += tf.log(self._config.mask_prob / (1 - self._config.mask_prob))

    weights = tf.cast(inputs.input_mask, tf.float32)
    labelsf = tf.cast(labels, tf.float32)
    losses = tf.nn.sigmoid_cross_entropy_with_logits(
        logits=logits, labels=labelsf) * weights
    per_example_loss = (tf.reduce_sum(losses, axis=-1) /
                        (1e-6 + tf.reduce_sum(weights, axis=-1)))
    loss = tf.reduce_sum(losses) / (1e-6 + tf.reduce_sum(weights))
    probs = tf.nn.sigmoid(logits)
    preds = tf.cast(tf.round((tf.sign(logits) + 1) / 2), tf.int32)
    DiscOutput = collections.namedtuple(
        "DiscOutput", ["loss", "per_example_loss", "probs", "preds",
                       "labels"])
```

Target Class (이진)

- Original : 원본 문장의 토큰과 같은 토큰
- Replaced : generator G가 만든 토큰

$$D(x^{corrupt}, t) = sigmoid(w^T h_D(x^{corrupt})_t)$$

[Discriminator 공식]

Finetune-Model

```
def model_fn(features, labels, mode, params):
  """The `model_fn` for TPUEstimator."""
  utils.log("Building model...")
  is_training = (mode == tf.estimator.ModeKeys.TRAIN)
  model = FinetuningModel(
      config, tasks, is_training, features, num_train_steps)
```

```
# Add specific tasks
self.outputs = {"task_id": features["task_id"]}
losses = []
for task in tasks:
  with tf.variable_scope("task_specific/" + task.name):
    task_losses, task_outputs = task.get_prediction_module(
        bert_model, features, is_training, percent_done)
    losses.append(task_losses)
    self.outputs[task.name] = task_outputs
self.loss = tf.reduce_sum(
    tf.stack(losses, -1) *
    tf.one_hot(features["task_id"], len(config.task_names)))
```

- Classification_tasks
- Qa_tasks
- Tagging_tasks
- ....

[run_finetuning.py]

Thank you

Reference

ELECTRA-Code - https://github.com/google-research/electra
Vocab - https://huggingface.co/google/electra-small-generator/resolve/main/vocab.txt
Tokenizer - https://huggingface.co/google/electra-small-generator/resolve/main/tokenizer.json

Paper

ELECTRA - https://arxiv.org/abs/2003.10555

BERT - https://arxiv.org/abs/1810.04805

Attention - https://arxiv.org/abs/1706.03762

Seq2Seq - https://arxiv.org/abs/1409.3215

LSTM / RNN
https://static.googleusercontent.com/media/research.google.com/ko//pubs/archive/43905.pdf

Cloze-driven Pretraining of Self-attention Networks (Two Tower Cloze Transformer)
https://arxiv.org/abs/1903.07785