6. ELECTRA Pre-Training Text Encoders as Discriminators rather than Generators

논문 : https://arxiv.org/abs/2003.10555

참고자료 : https://blog.pingpong.us/electra-review/

KoELECTRA : https://github.com/monologg/KoELECTRA

문제제기

MLM(Masked Language Model)은 너무 비효율적이다! (논문에선 BERT 기준으로 문제점 제기)

1. 하나의 샘플(문장)에서 15%만 <MASK>토큰으로 대체하고 학습함 (심지어 <MASK>토큰은 고정됨)
   => 샘플마다 학습하는 양이 작기 때문에 많은 corpus가 필요함
2. <MASK> 토큰은 학습 이외에 사용되지 않음!

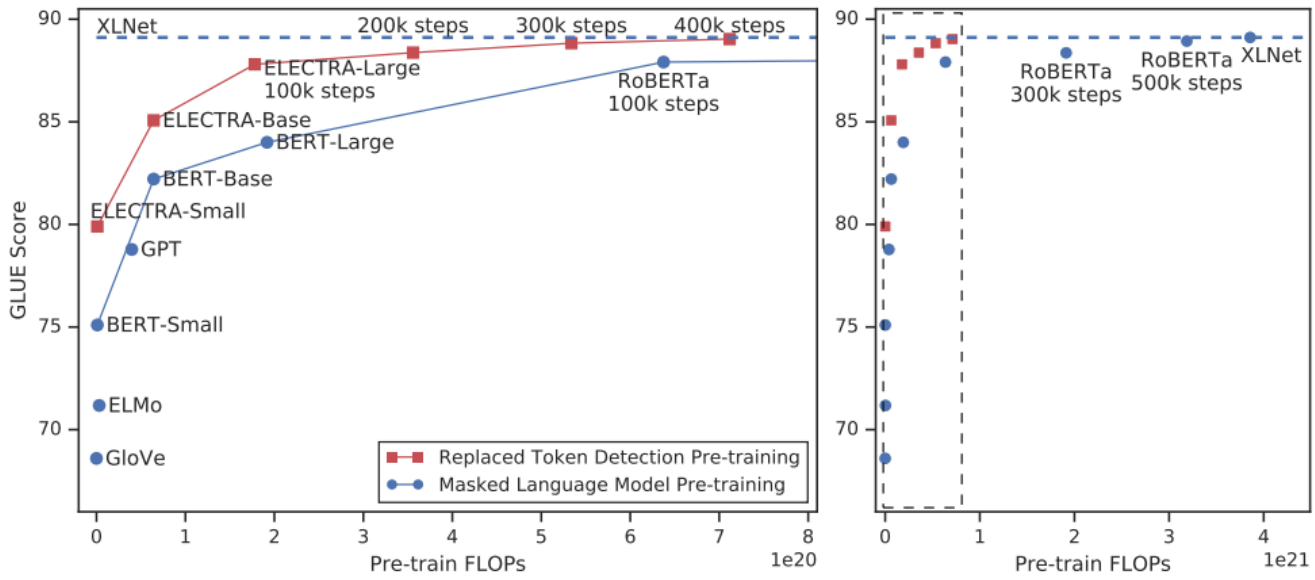

Figure 1: Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.

그림 1 해석

- X축 : Pre-train에서의 계산량
- Y축 : SQuAD Question answering데이터 셋에 대한 benchmark 점수
- 기존 MLM 모델에 비해 성능이 좋음
- XLNet (Permutation Language Model)보다는 성능이 안좋음
- ELECTRA-Small의 경우 1GPU (Tesla V100 GPU) 로 4일만에 학습가능하다고 논문에서 말함

제안기법

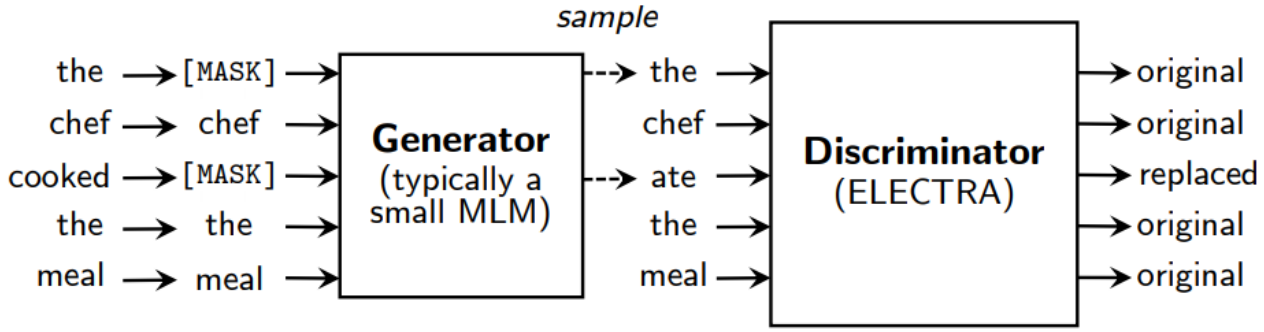

Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator 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 (the ELECTRA model) on downstream tasks.

그림 2 설명 + ELECTRA 설명

- GAN과 비슷하게 학습과정에서 Generator(MLM 기반의 모델)과 Discriminator(ELECTRA)을 사용
- 학습이 끝나면 Discriminator만 사용
- Generator
  - 입력 토큰들중 랜덤으로 <MASK>토큰으로 대체
  - <MASK> 토큰을 원본 토큰으로 맞추도록 학습
  - <MASK> 토큰에 대해서만 Loss 계산
- Discriminator
  - Generator가 생성한 토큰을 입력으로 사용
  - 입력 토큰 중에서 원본 토큰과 생성된 토큰을 구별하는 Loss (Replaced Token Detection, RTD)를 사용
- GAN과의 차이점
  - GAN
    - Generator가 생성한 output을 전부 negative sample (fake data)로 간주함
    - 노이즈 백터가 Generator의 입력으로 사용됨
    - Generator가 Discriminator를 속이기위해 adversarial train
  - ELECTRA
    - Generator가 생성한 output 중 original data와 동일한 토큰에 한해 positive sample로 처리함
    - 원본 토큰에서 일부를 <MASK>토큰으로 대체한 것이 Generator의 입력으로 사용됨
    - Generator/Discriminator가 Maximum likelihood로 학습

수식으로 표현

Generator의 입력 토큰별 softmax 기반의 토큰 생성

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

식 설명

- $\boldsymbol{x} = [x_1, ..., x_n]$: 입력 토큰 (실제 문장)
- $x_t = [\text{MASK}]$: 입력 토큰 중 임의의 위치 t에서 <MASK>토큰으로 대체됨 -> 원본 데이터로 잘 복구하는지 Loss로 사용할 대상
- $h(\boldsymbol{x}) = [h_1, ..., h_n]$: 입력 문장의 contextualized vector representation
- 좌변 : 입력 토큰으로 <MASK> 토큰을 대체할 토큰들의 확률분포
- 우변 : softmax 식

Discriminator의 sigmoid 기반 이진분류(original/replaced)

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

$$m_i \sim \text{unif}\{1, n\} \text{ for } i = 1 \text{ to } k \qquad \boldsymbol{x}^{\text{masked}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \texttt{[MASK]})$$
$$\hat{x}_i \sim p_G(x_i | \boldsymbol{x}^{\text{masked}}) \text{ for } i \in \boldsymbol{m} \qquad \boldsymbol{x}^{\text{corrupt}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \hat{\boldsymbol{x}})$$

데이터 만드는 과정

1. 길이가 n인 원본 토큰들에 k개의 <MASK>토큰으로 변경 => Generator의 입력으로 사용
2. 원본 토큰들에서 <MASK>토큰이 있었던 위치에 Generator의 output으로 교체 => Discriminator의 입력으로 사용

Generator의 Loss Function

- <MASK> 토큰에 한해 Loss 계산

$$\mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) = \mathbb{E}\left(\sum_{i \in \boldsymbol{m}} -\log p_G(x_i | \boldsymbol{x}^{\text{masked}})\right)$$

Discriminator의 Loss Function

- 전체 토큰에 대해 Loss 계산

$$\mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D) = \mathbb{E}\left(\sum_{t=1}^{n} -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\boldsymbol{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t))\right)$$

Joint Learning을 통해 학습

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

실험 및 검증

Weight sharing (그림 없음)

- Generator와 Discriminator에 사용되는 모델자체는 동일(트렌스포머의 인코더)하므로 weight sharing을 하면 어떨까?
- 500k step만큼 학습후 GLUE score 확인
  - 쉐어링하지 않음 : 83.5
  - 임베딩레이어만 공유 : 84.3
  - 모든 가중치를 공유 : 84.4
- 논문의 실험 해석
  - Discriminator는 입력으로 들어온 토큰만 학습하는 반면, generator는 출력 레이어에서 softmax를 통해 사전에 있는 모든 토큰에 대해서 밀도 있게 학습할 수 있음
  - ELECTRA는 결국 discriminator만을 취해서 사용하는데, 이때 generator와 임베딩을 공유해서 학습한 경우의 discriminator는 훨씬 효과적으로 학습했을 것이고 결과적으로 좋은 성능을 기록한 것으로 보임
- 단점 : Generator와 Discriminator의 사이즈가 동일해야함

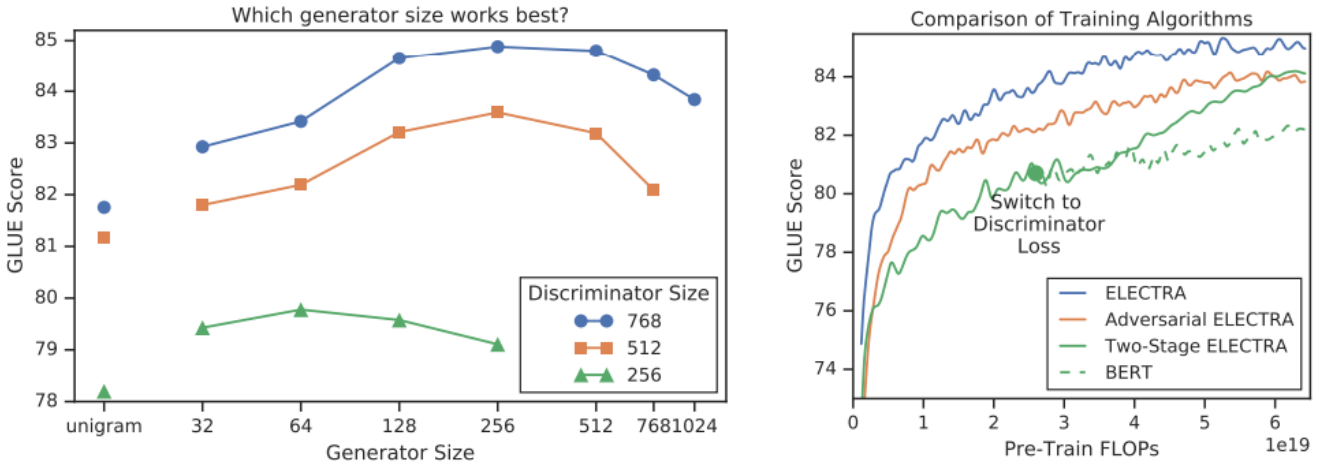

Figure 3: Left: GLUE scores for different generator/discriminator sizes (number of hidden units). Interestingly, having a generator smaller than the discriminator improves results. Right: Comparison of different training algorithms. As our focus is on efficiency, the x-axis shows FLOPs rather than train steps (e.g., ELECTRA is trained for fewer steps than BERT because it includes the generator).

Smaller Generators (그림 3 왼쪽)

- 계산량을 줄일려고 ELECTRA 제안했는데 Weight sharing해서 동일 모델 두개를 계산해야되네?
  => Generator 모델의 크기를 줄여야 계산량이 줄어듦
- Generator와 Discriminator의 사이즈는 얼마가 적당할까?
- 500k step 만큼 학습후 GLUE score 확인 (그림 3의 왼쪽)
  - Generator는 Discriminator의 1/4~1/2정도가 적당하다~
- 논문의 실험 해석
  - Generator가 너무 강력하면 discriminator의 태스크가 너무 어려워져서 이런 현상이 나타나는 듯함
  - 게다가 discriminator의 파라미터를 실제 데이터 분포가 아닌 generator를 모델링하는 데 사용할 수도 있음 (Weight Sharing에서 검증)

Training Algorithim (그림 3 오른쪽)

- 위 실험 두개는 500k Step만큼 학습했는데 얼만큼 학습하면 BERT보다 좋을까? + 어떤 학습방법이 효과적일까?
- 실험에서 사용한 3가지 학습방법
  - Generator와 discriminator를 joint learning으로 학습 (논문이 주장한 방법)
  - Two-stage 학습
    - Generator를 먼저 학습
    - Generator 모델의 파라미터로 Discriminator 모델을 초기화
    - Discriminator만 학습 (Generator 파라미터 고정)
  - Adversarial 학습 : GAN과 같은 구조로 만들어서 adversarial training
- Genereator가 Maximum likelihood로 학습하는게 Adversarial 학습보다 좋은 이유 (Language GANs Falling Short 논문에서 이미 주장)
  - MLM 자체가 구림
  - 학습된 generator가 만드는 분포의 엔트로피가 낮음
    => Softmax 분포는 하나의 토큰에 확률이 쏠려있고, 샘플링할때 다양성이 떨어짐

Hyperparameter	Small	Base	Large
Number of layers	12	12	24
Hidden Size	256	768	1024
FFN inner hidden size	1024	3072	4096
Attention heads	4	12	16
Attention head size	64	64	64
Embedding Size	128	768	1024
Generator Size (multiplier for hidden-size, FFN-size, and num-attention-heads)	1/4	1/3	1/4
Mask percent	15	15	25
Learning Rate Decay	Linear	Linear	Linear
Warmup steps	10000	10000	10000
Learning Rate	5e-4	2e-4	2e-4
Adam ϵ	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
Attention Dropout	0.1	0.1	0.1
Dropout	0.1	0.1	0.1
Weight Decay	0.01	0.01	0.01
Batch Size	128	256	2048
Train Steps (BERT/ELECTRA)	1.45M/1M	1M/766K	464K/400K

Table 6: Pre-train hyperparameters. We also train an ELECTRA-Large model for 1.75M steps (other hyperparameters are identical).

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

Table 1: Comparison of small models on the GLUE dev set. BERT-Small/Base are our implementation and use the same hyperparameters as ELECTRA-Small/Base. Infer FLOPs assumes single length-128 input. Training times should be taken with a grain of salt as they are for different hardware and with sometimes un-optimized code. ELECTRA performs well even when trained on a single GPU, scoring 5 GLUE points higher than a comparable BERT model and even outscoring the much larger GPT model.

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**

Table 2: Comparison of large models on the GLUE dev set. ELECTRA and RoBERTa are shown for different numbers of pre-training steps, indicated by the numbers after the dashes. ELECTRA performs comparably to XLNet and RoBERTa when using less than 1/4 of their pre-training compute and outperforms them when given a similar amount of pre-training compute. BERT dev results are from Clark et al. (2019).

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**

Table 3: GLUE test-set results for large models. Models in this table incorporate additional tricks such as ensembling to improve scores (see Appendix B for details). Some models do not have QNLI scores because they treat QNLI as a ranking task, which has recently been disallowed by the GLUE benchmark. To compare against these models, we report the average score excluding QNLI (Avg.*) in addition to the GLUE leaderboard score (Score). "ELECTRA" and "RoBERTa" refer to the fully-trained ELECTRA-1.75M and RoBERTa-500K models.

Model	Train FLOPs	Params	SQuAD 1.1 dev		SQuAD 2.0 dev		SQuAD 2.0 test	
			EM	F1	EM	F1	EM	F1
BERT-Base	6.4e19 (0.09x)	110M	80.8	88.5	–	–	–	–
BERT	1.9e20 (0.27x)	335M	84.1	90.9	79.0	81.8	80.0	83.0
SpanBERT	7.1e20 (1x)	335M	88.8	94.6	85.7	88.7	85.7	88.7
XLNet-Base	6.6e19 (0.09x)	117M	81.3	–	78.5	–	–	–
XLNet	3.9e21 (5.4x)	360M	**89.7**	**95.1**	87.9	**90.6**	87.9	90.7
RoBERTa-100K	6.4e20 (0.90x)	356M	–	94.0	–	87.7	–	–
RoBERTa-500K	3.2e21 (4.5x)	356M	88.9	94.6	86.5	89.4	86.8	89.8
ALBERT	3.1e22 (44x)	235M	89.3	94.8	87.4	90.2	88.1	90.9
BERT (ours)	7.1e20 (1x)	335M	88.0	93.7	84.7	87.5	–	–
ELECTRA-Base	6.4e19 (0.09x)	110M	84.5	90.8	80.5	83.3	–	–
ELECTRA-400K	7.1e20 (1x)	335M	88.7	94.2	86.9	89.6	–	–
ELECTRA-1.75M	3.1e21 (4.4x)	335M	**89.7**	94.9	**88.0**	**90.6**	**88.7**	**91.4**

Table 4: Results on the SQuAD for non-ensemble models.

Small Model + Large Model(표 1, 2, 3, 6)

- 타 기법처럼 여러가지 크기의 모델을 실험 (파라미터 정보는 표 6)
- 결론
  - 표 1(작은 모델 & 적은 step 비교) : 더 빠르게 학습하고 성능도 좋다~
  - 표 2, 표 3, 표 4(큰 모델끼리 비교) : train step이 더 적은데 비슷한 성능을 낸다~

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).

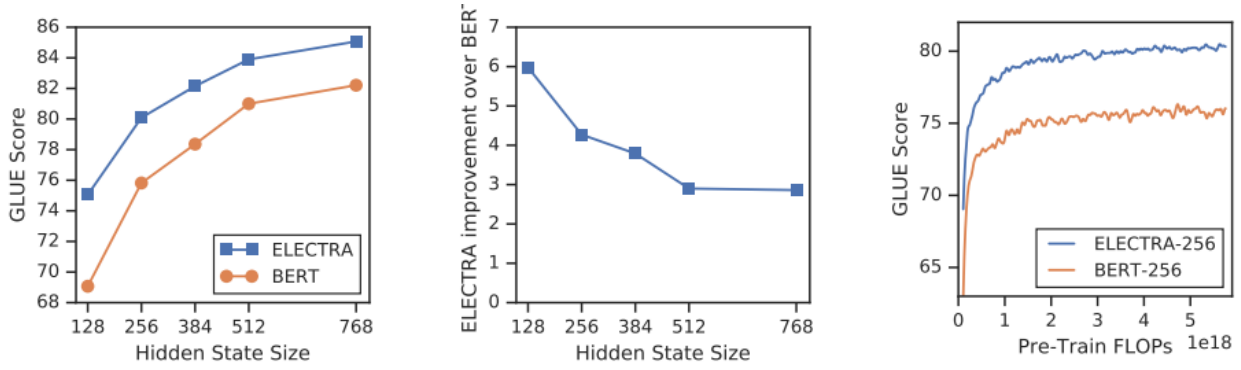

Figure 4: Left and Center: Comparison of BERT and ELECTRA for different model sizes. Right: A small ELECTRA model converges to higher downstream accuracy than BERT, showing the improvement comes from more than just faster training.

Efficiency Analysis (표 5, 그림 4)

- ELECTRA를 여러형태로 변형해서 결과 살펴보기 (표 5)
  - ELECTRA 15% : ELECTRA도 <MASK>토큰 위치에 대해서만 Loss를 계산하면 어떨까?
  - Replace MLM : Discriminator를 MLM으로 학습하면 어떨까?
    (<MASK>토큰대신 Generator가 생성한 토큰들을 입력으로 사용)
  - All-Tokens MLM : Discriminator의 입력을 만들 때, Generator가 출력한 모든 토큰을 사용하면 어떨까?
- 모델사이즈가 변화되도 ELECTRA가 무조건 좋을까? (그림 4)
  - BERT 모델과 ELECTRA비교