Electra :Pre-Training Text Encoders As Discriminators Rather Than gencerators

Kevin Clark, Minh-Thang Luong, Quoc V. Le,
Christopher D. Manning

정영석

Contents

Introduction

1. Research Background

- **Downstream NLP tasks에서 다양한 word representation 방법론이 제안됨**
  - ✓ Denoising Autoencoder
  - ✓ BERT
  - ✓ XLNet
  - → 모두 원래의 입력 문장을 재생성 하는 방향으로 학습함

- **문장의 일부 토큰만 학습하는 기존의 방법론은 전체 문장의 토큰을 학습하지 못하기 때문에 부적절함**
  - → MLM의 경우 Masked 되는 15%의 토큰만 학습에 사용됨

2. Proposal

- **새로운 학습 방법론을 통한 효율적 학습**
  - ✓ MLM → RTD(Replaced Token Detection)
  - ✓ Generator를 통해 생성된 그럴듯한 오류를 학습

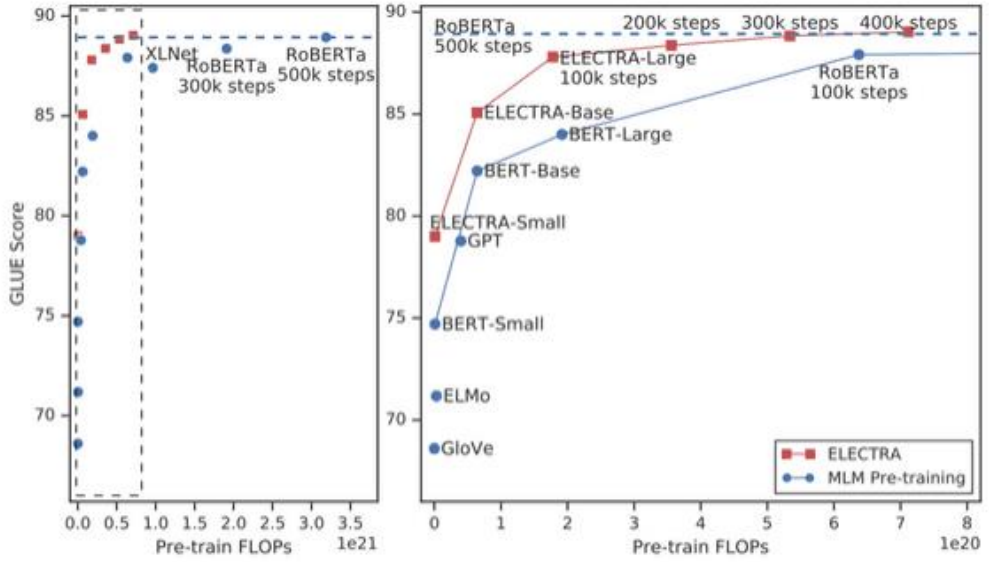


Figure 1: ELECTRA consistently outperforms MLM pre-training given the same compute budget. The right figure is a zoomed-in view of the dashed box.

Related Work

- **관련 연구**

  - Self-Supervised Pre-training for NLP
    - ✓ Masked-language modeling : BERT, large Transformer
    - ✓ Bert's extensions : MASS, UniLM, ERNIE…
    - ✓ Masks attention weight : XLNet
  - Generative Adversarial Networks(GAN)
    - ✓ 대부분의 연구에서 Generator를 학습시키기 위해 Discriminator 사용
    - ✓ 몇몇 연구에서 Discriminator를 downstream task에 적용함 (Rashford et al. 2016)
  - Constrative Learning (자가지도 학습)
    - ✓ Word2Vec

Methodology

- **Methodology**
  - Overview of Electra

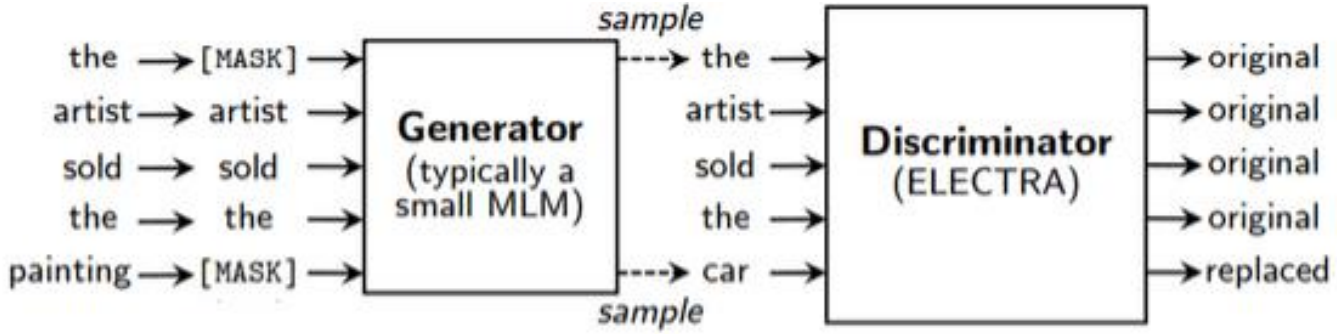
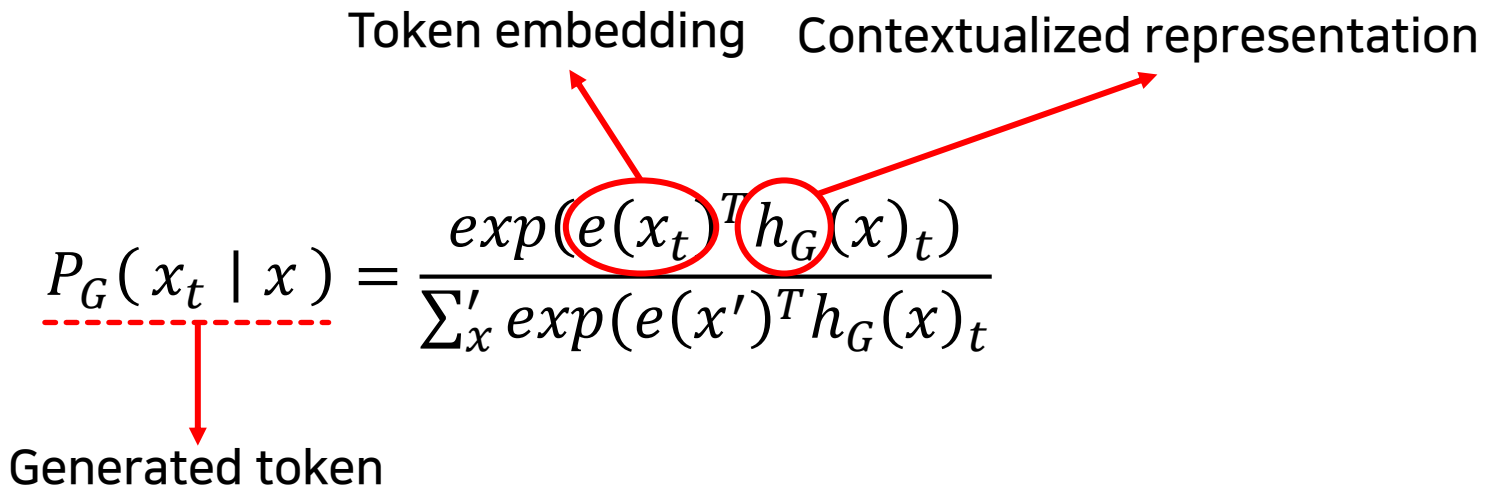


Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator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

Methodology

- **Methodology**
  - Generator output



✓ Generator는 MLM을 통해 그럴듯한 문장들을 생성한다!

Methodology

- **Methodology**
  - Decoder output

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

✓ Real token 여부를 0~1사이의 값으로 표현

  - $x_{corrupt}$ 의 생성 과정

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

→ Masking 여부

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

→ m에 의해 masking 된 입력 값

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

→ Masking된 토큰에 대한 Generator의 예측 값

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

→ Generator에 의해 생성된 예측 값으로 토큰 변환

- **Methodology**
  - Loss Function

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

$$L_{Disc}(x, \theta_G) = E\left(\sum_{i \in m} -1\left(x_t^{corrupt} = x_t\right) \log D(x^{corrput}, t) - 1\left(x_t^{corrput} \neq x_t\right) \log(1 - D(x^{corruput}, t))\right)$$

↓ "Combined loss"

$$\min_{\theta_G,\ \theta_D} \sum_{x \in X} L_{MLM}(x, \theta_G) + \lambda L_{Disc}(x, \theta_G)$$

→ "Discriminator의 Loss는 Generator로 전달되지 않음"

Methodology

- **Methodology**
  - GAN과의 차이점
    - ✓ Discriminator는 Generator가 생성했는지 여부를 판단하지 않음
      - → Generator가 올바른 토큰을 예측한 경우 “Real”로 판단함
    - ✓ Generator는 Discriminator와 적대적으로 학습하는 것이 아닌 [masked] 토큰을 올바르게 예측하기 위해 학습
      - → 적대적으로 학습하는 경우, Generator에서의 sampling과정으로 인해 역전파를 전달하기 어려움
    - ✓ 학습과정에서 Noise vector를 삽입하지 않음

Experiments

- **Experiments**

  - Weight Sharing

    ✓ **Discriminator와 Generator사이의 weight를 공유한 경우의 성능을 비교함**

      ▪ Discriminator와 Generator 사이의 모든 weigh를 공유하는 경우 Discriminator와 Generator의 모델 사이즈는 동일해짐

      → 실험을 통해 Generator의 크기가 Discriminator에 비해 크기가 작아야 성능이 좋아진 것을 발견함

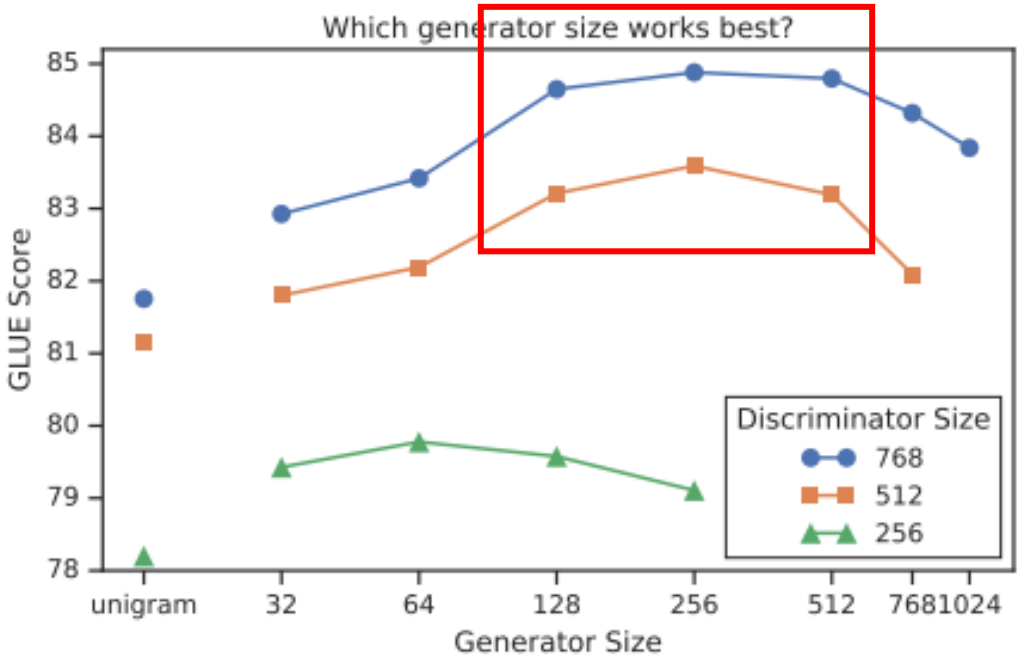
      ➢ Weight sharing에 따른 실험 결과(GLUE)

        - No weight sharing : 83.3
        - Token embedding sharing : 84.4
        - Weight sharing : 83.6

    → **MLM 방법을 통한 학습 방법은 word token을 학습하는데 효율적임, Discriminator의 경우 Generator에 의해 sampling된 토큰만 확인해 학습하기 때문.**

Experiments

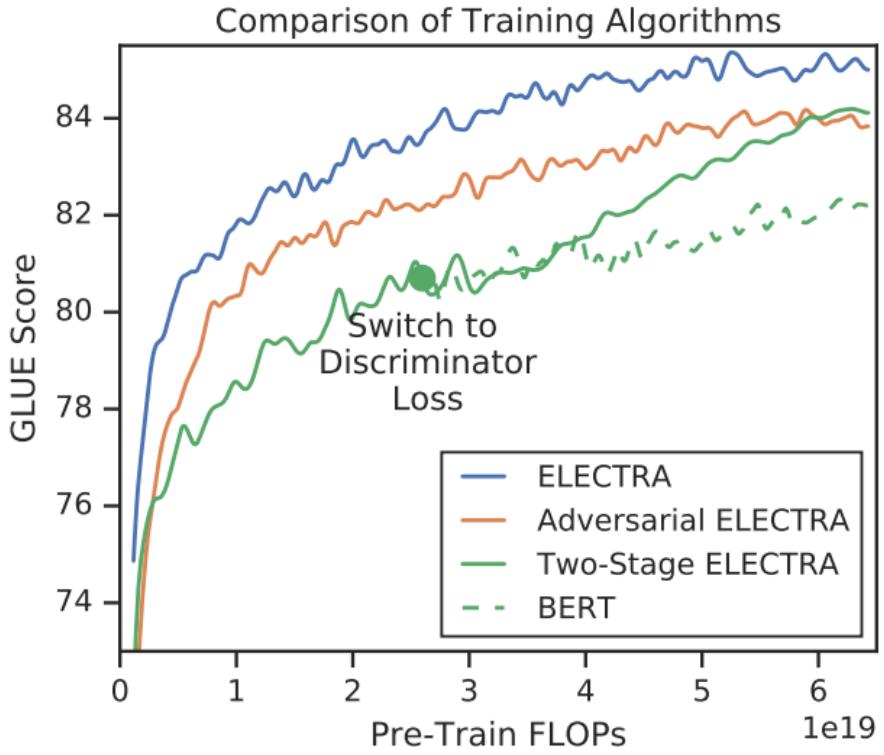
- **Experiments**
  - Small Generator



→ **너무 강한(strong) Generator는 Discriminator 에게 너무 어려운 task를 생성하게 될 수 있음**

Experiments

- **Experiments**
  - Training Algorithm



✓ Adversarial Electra의 성능이 비교적 낮은 이유

1. Generator의 MLM성능이 58%로 매우 안좋음
2. Generator에서 생성된 결과가 cross-entropy가 Electra에 비해 낮음

Experiments

- **Experiments**
  - Small Models

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.

- BERT모델에 비해 훨씬 좋은 성능을 보였으며,
- 파라미터 수가 8배 정도 차이나는 GPT 보다 빠르게 학습하며 성능도 좋음

Experiments

- **Experiments**
  - Large 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.

Experiments

- **Experiments**
  - Efficiency Analysis

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).

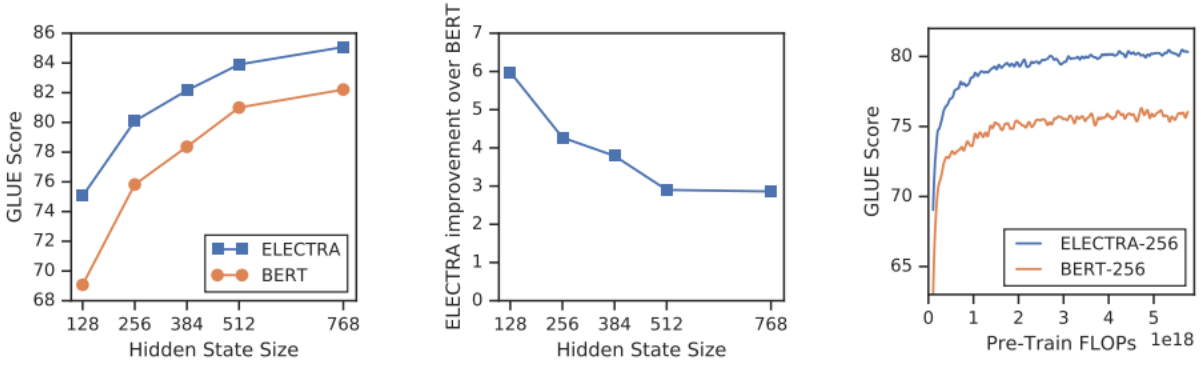


Figure 4: Left and Center: Comparison of BERT and ELECTRA for different model sizes. Right: A small ELECTRA model converges to higher downstream accuracy than BERT, showing the improvement comes from more than just faster training.

✓ All-Tokens MLM

모든 토큰에 대해 Generator의 결과를 MLM함

✓ Replaced MLM

Generator에 입력 시 [Mask]인 토큰에 대해서 학습

✓ Electra 15%

discriminator loss를 입력 토큰의 15%만으로 학습