딥러닝 논문 요약 및 구현 스터디

**발표자**

이기환

# ELECTRA : PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS

# Introduction

**Masked Language Model**입니다. 이는 어떤 문장이 주어지면 이 문장의 일부분을 Masking하고, 이 masking된 부분을 맞추는 것을 통해 학습을 하는 언어모델입니다. 모델은 마스크를 중심으로 주변 단어들을 살피게 되는데, 이는 다시 말해 문맥을 이해할 수 있게 되는 것 입니다! 그렇기에 자연어처리에서 가장 많이 쓰이는 NLU(Natural Language Understanding 자연어 이해)task를 수행할 수 있게 됩니다. Masked Language Model의 대표 주자는 구글의 BERT와 OpenAI의 GPT-2가 있습니다. 이를 기반으로 엄청난 논문들이 쏟아져 나왔습니다.

하지만 모델의 크기가 너무 크고, 학습 데이터 또한 너무 많기에 필요한 가격이 굉장히 비싸다고 합니다.

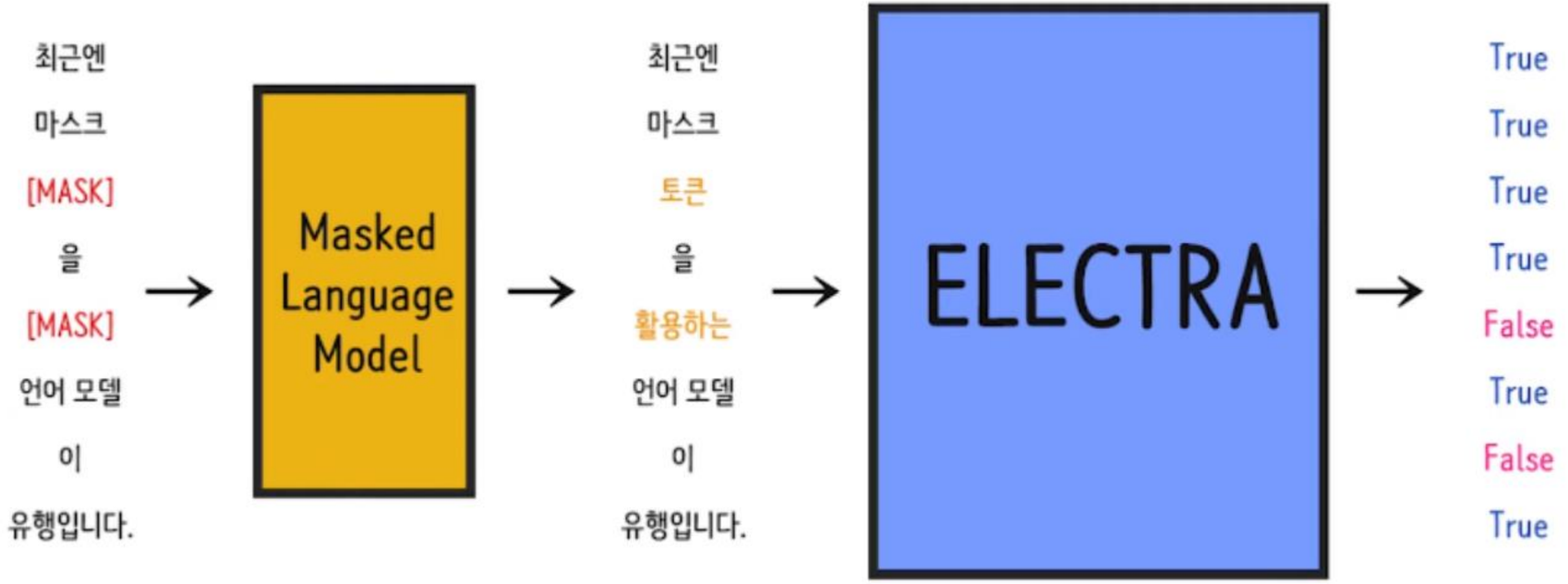
```
* Cost
BERT : $6,912 / GPT-2 : $43,008 / XLNet : $61,440
```

그렇기에 구글에서 본 논문인 ELECTRA를 내면서 연산 효율성을 개선시키는 방안을 제시합니다. (=> ELECTRA 학습 4일만에, GPT-2 학습 120일 연산량을 넘어섰습니다.)

**Q. 그렇다면, 어떤 문제가 낭비적 연산을 불러일으키는가?**

학습 데이터를 Masking시킨 후 이를 언어 모델이 예측하는데, 이를 통해 예측한 값과 정답 값 사이의 Loss를 구하게 됩니다. 여기서 문제가 생기는데, 하나의 example에 대해서 고작 15%만 loss가 발생하고, 이 15%만 학습을 진행하게 되기에 비효율적이라는 것 입니다.

아래를 예시로 보시면, 2개의 토큰에 대해서만 학습이 진행되기에 전체 데이터중 두 단어밖에 학습할 수 없다는 것 입니다. 이렇게 적은 양만 학습한다면, 데이터를 더 모아야 하고, 데이터셋의 크기가 더 커지기에 비용이 많이 들 수 밖에 없다는 것이 논문에서 해결하고자 하는 문제점입니다.



> 이를 **"Replaced Token Detection(RTD)"** 을 통해 개선시키고, 이 개념으로 training한 것이 **"ELECTRA"** 입니다!

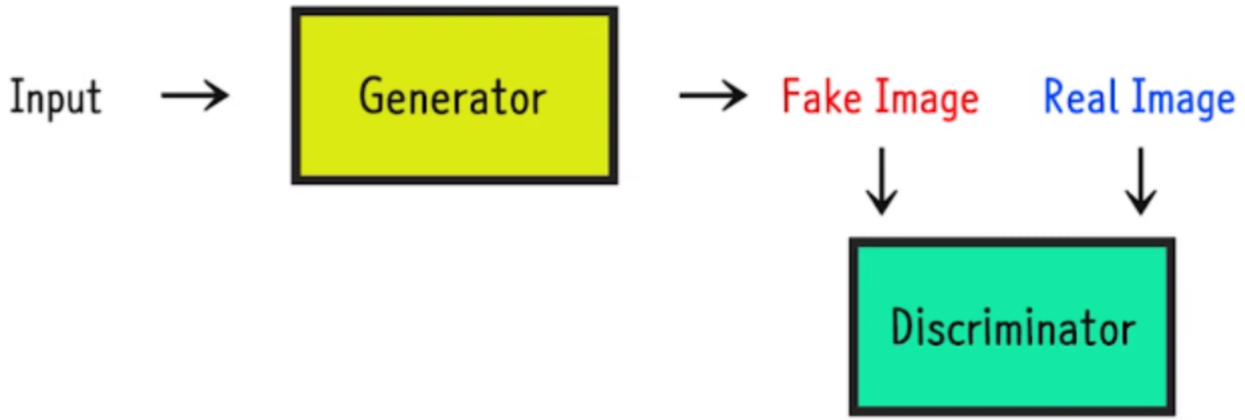
MLM을 통해 생성한 가짜 토큰인지 원본 토큰인지 여부를 판단하는 이진분류를 실시하게 합니다.
여부를 판단하기 위해 이는 모든 토큰을 살펴야 하고, 각 토큰을 볼 때마다 주변 문맥을 다 봐야하기에 MLM이 학습하듯이 문맥을 학습하게 됩니다. 이렇게 모든 토큰에 대해 이진 분류를 진행하는데, 여기서 모든 토큰에 대한 loss를 구하기에 모든 토큰을 학습하는 효과가 있게 되는 것 입니다.

# Method

ELECTRA의 구조는 generator G와 discriminator D가 존재하며 GAN과 비슷합니다.

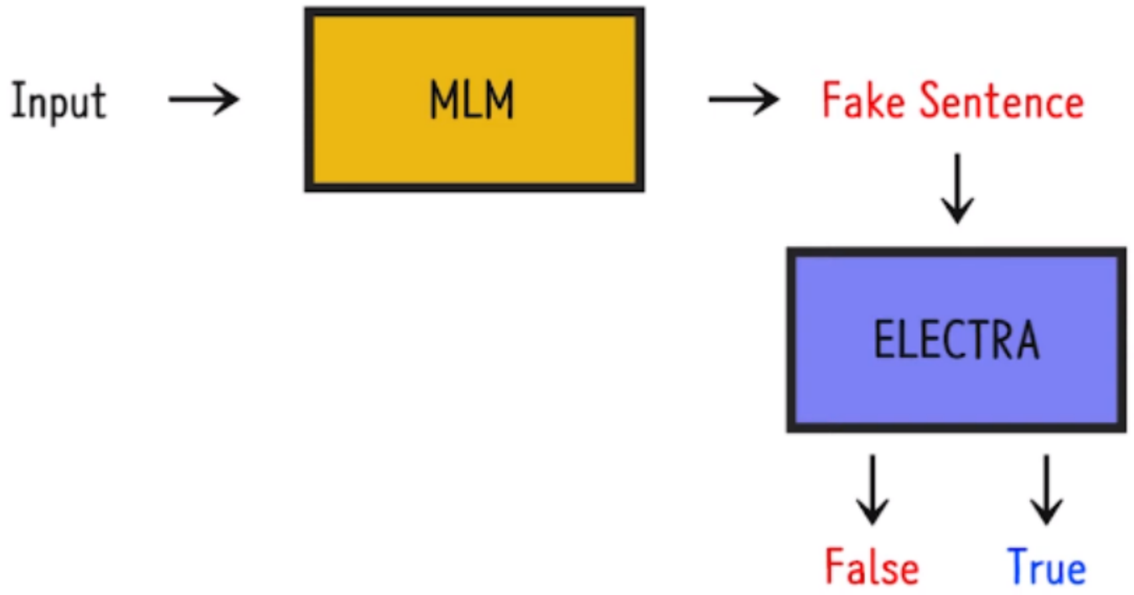
## "GAN : Generative Adversarial Networks"

GAN의 Generator는 랜덤 이미지를 받고 이를 Real image와 비슷하게 Fake image를생성해 냅니다.
GAN의 Discriminator는 받은 image가 Rela인지, 생성자가 만든 Fake인지 판단합니다.
: 이는 서로의 실수를 유발시키며, 적대적으로 동작하며 학습하기에 Adversarial Network라고 합니다.



## "ELECTRA"

그렇다면 ELECTRA를 한번 살펴보겠습니다. 아래와 같은데 이는 GAN과 유사하지만 GAN이라고 부를 수는 없습니다. 가짜 문장을 MLM이 만들고, 이를 ELECTRA가 T/F로 이진분류를 실시하긴 하지만, 가짜 문장을 True라고 했다고 MLM이 받는 보상은 없기 때문입니다. MLM은 ELETRA가 T/F 아무거나 판단해도 상관없이 그냥 문장을 잘 만들고자 하는 것 입니다.



- Generator

  생성자는 BERT의 MLM과 동일합니다.

  1) 입력 시퀀스 x=[x1,x2,...,xn]에 대해 마스킹할 위치의 집합 m=[m1,...mk]을 결정합니다.
  (보통 k=0.15n, 즉 전체 토큰의 15% mask-out)

  2) 결정한 위치에 있는 토큰을 [MASK]로 치환합니다.(x_masked=REPLACE(x,m,[MASK]))

  3) 이렇게 마스킹된 x_masked에 대해 주어진 위치 t에 대하여 어떤 토큰 x_t를 생성할 확률을 output으로 리턴합니다. 이렇게 G는 원래 토큰이 무엇인지 예측하여 생성하는데, 이 생성된 토큰을 x_corrupted라고 표기하고 판별자에게 넘기게 됩니다.

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

$$\hat{x}_i \sim p_G(x_i | \boldsymbol{x}^{\text{masked}}) \text{ for } i \in \boldsymbol{m} \qquad \boldsymbol{x}^{\text{corrupt}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \hat{\boldsymbol{x}})$$

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

token embedding

- Discriminator
  판별자는 생성자가 만들어낸 토큰 시퀀스에 대해 각 토큰이 original인지 replace인지 이진 분류를 수행합니다.
  1) generator로 masked-out된 토큰들을 처리한 x_corrupt를 받습니다.
  2) 판별자가 x_corrupt의 어떤 토큰이 original input과 match되는지 예측하도록 학습시킵니다.(이진분류)

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

- Loss function

  최종적으로 ELECTRA는 대용량 코퍼스 x에 대해 G loss와 D loss의 합을 최소화 하도록 학습시킵니다. 이 때, 람다는 50을 사용했으며, 이 파라미터는 binary classificaion인 D loss와 30000 class 분류인 G loss의 스케일을 맞추는 역할을 합니다. G의 sampling 과정이 있기에 D loss는 G로 역전파되지 않고, 위 구조로 pre-training을 마친 뒤 G는 버리고 D만 취해서 downstream task로 fine-tuning을 진행하게 됩니다.

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

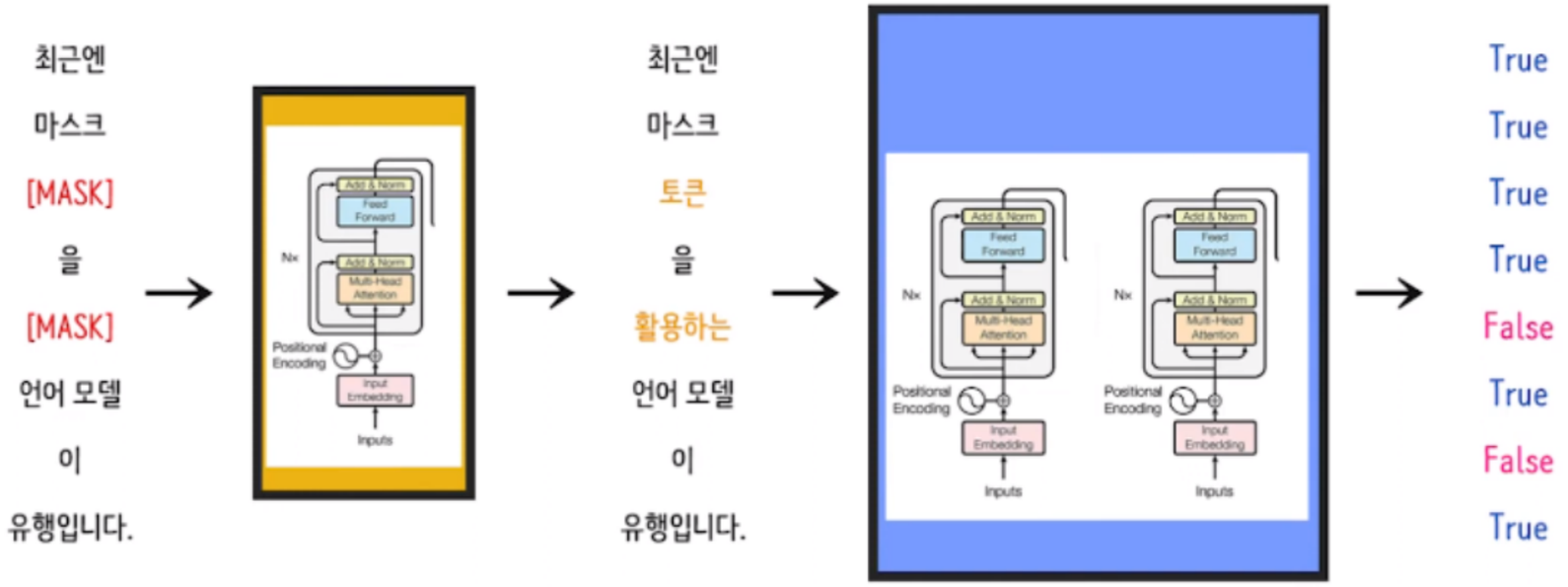
$$\mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D) = \mathbb{E}\left(\sum_{t=1}^{n} -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\boldsymbol{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t))\right)$$

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

```
Generator : Maximum likelihood
Discriminator : cross entropy
```

# 모델 아키텍쳐

아래와 같이 표현할 수 있는데, MLM은 Transformer 인코더 1개를 사용하고 ELECTRA는 Transformer 인코더 2개를 사용하였습니다.(일반적으로 MLM이 ELECTRA보다 인코더 개수가 적었을 때 성능이 좋았다고 합니다)



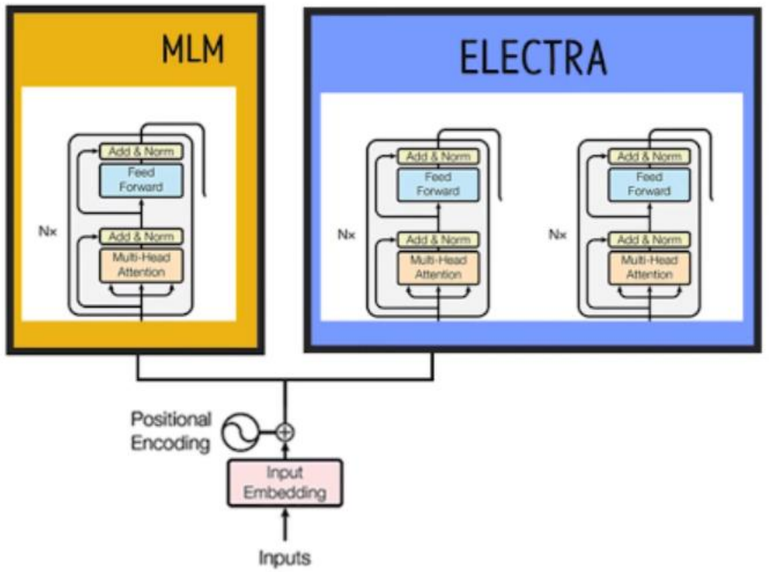
실험 중 하나를 소개하겠습니다.

**Weight Shring**

말 그대로, MLM의 가중치와 ELECTRA의 가중치를 공유하는 것 입니다.
파라미터 수가 너무 많다보니, 파라미터 자체를 줄이는 것 만으로도 normalization되는 효과가 있고, global minimum에 수렴하는 속도가 빠른 효과가 있습니다.
하지만 이는 ELECTRA에는 적합하지 않았는데, 이를 실행하기 위해서는 MLM의 크기와 ELECTRA의 크기를 똑같이 맞춰줘야 하기에 인코더를 2개씩 써야 했기 때문입니다. 이는말이 2개인 것이지, 사실상 하나가 버트정도의 크기라고 생각한다면 버트의 두 배가 되는 것 입니다.
따라서 이를 개선할 방안을 생각해냈는데, 그것이 바로 임베딩만 sharing하는 것 입니다.

**Weight Sharing(Only Embedding share)**

일반적으로 임베딩을 정의할 때, 데이터셋을 기반으로 tokenize를 다 하고 단어장vocab을 구성하고, 임베딩을 구성합니다. 그렇기에 데이터셋의 모든 단어에 대해 공평하게 임베딩하는 기회는 MLM밖에 없습니다. 예를 들어 MLM에서 필요 없다고 생각되는 단어를 제외시켜 버린다면, ELECTRA는 더이상 그 단어를 평생 튜닝할 수 없다는 것 입니다. 그렇기에 임베딩 층만 sharing하며, 모델의 크기는 조절할 수 있게 변경한 것 입니다.



# 모델 사이즈

이제 모델의 크기를 조절할 수 있게 되었습니다!

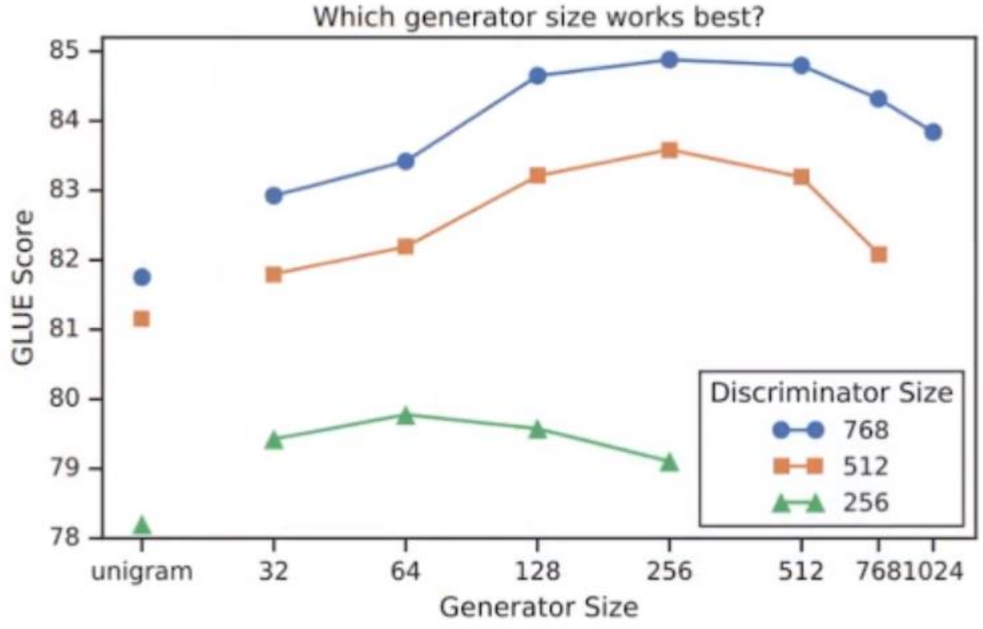
**Q. 그렇다면 어느 정도 크기로 구성할 때 가장 우수할까요?**

아래의 그림을 보시면, 판별자 모델의 크기에 따른 생성자 사이즈별 성능입니다.

판별자가 768사이즈를 가질 때 생성자는 256에서 가장 우수한 성능을 보였고,

판별자가 512면 생성자는 256,

판별자가 256이면 생성자는 64, 의 성능을 보이며 판별자 크기의 약 1/4~1/2 사이일 때 좋은 성능을 나타냄을 확인할 수 있습니다.



# 모델의 훈련

아래의 표는 모델을 훈련을 시키고 각 데이터셋에 대한 평가를 확인한 것 입니다.
맨 아래의 ELECTRA를 확인해보면, 하나의 데이터셋 빼고(이것도 0.2만 차이) 모두 가장 우수한 성능을 보임을 알 수 있습니다. 물론 드라마틱하게 우수한 성능을 띄지는 않지만 조금이라고 성능을 높였다는 점과 연산량을 줄일 수 있다는 점을 고려한다면 꽤나 괜찮은 모델임을 알 수 있습니다.

| 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.

# Conclusion

본 논문은 기존의 연산 비효율성을 지적하고, language representation learning을 위한 새로운 self-supervision task인 Replaced Token Detection(RTD)을 제안했습니다. 또한 임베딩만 공유하고, 생성자와 판별자의 크기를 조정하며 성능 개선과 함께 최적의 효율적 모델을 구상함을 볼 수 있었습니다.

본 논문의 저자는 이 효율적인 모델을 통해 많은 연구자들이 적은 컴퓨팅 리소스로도 언어모델의 사전학습에 대한 많은 연구 및 개발을 할 수 있길 바람을 남겼습니다.