ELECTRA

Efficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately

김현수

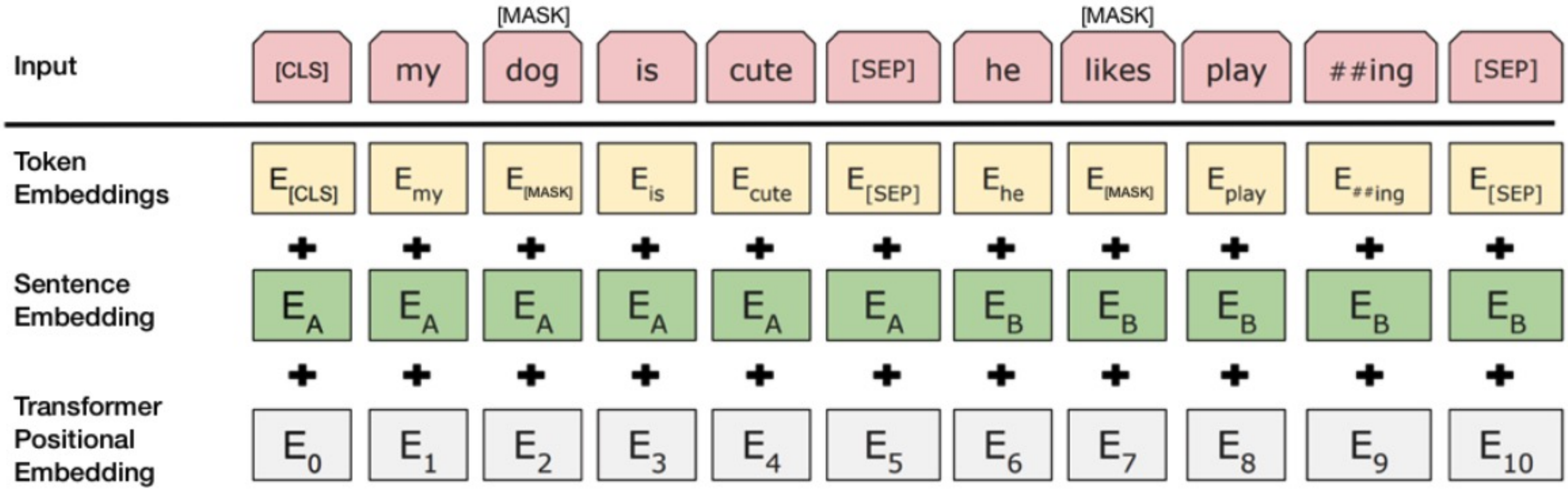
목차

- Abstract
- Introduction
- Method
- Experiments
- Conclusion

목차

Efficiently Learning an Encoder that Classifies Token Replacements Accurately

〈BERT〉

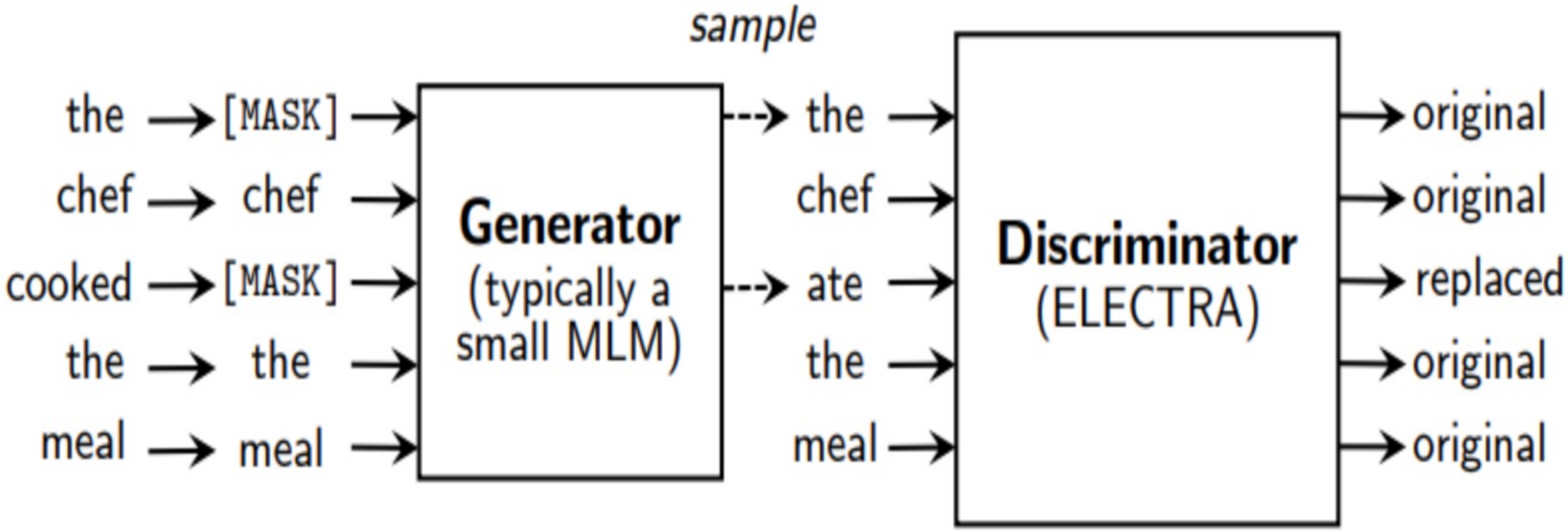


→ MLM(Masked Language Modeling)을 활용한 pre-training method

→ 성능은 좋으나, 너무 많은 연산량 ..

Efficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately

〈ELECTRA〉

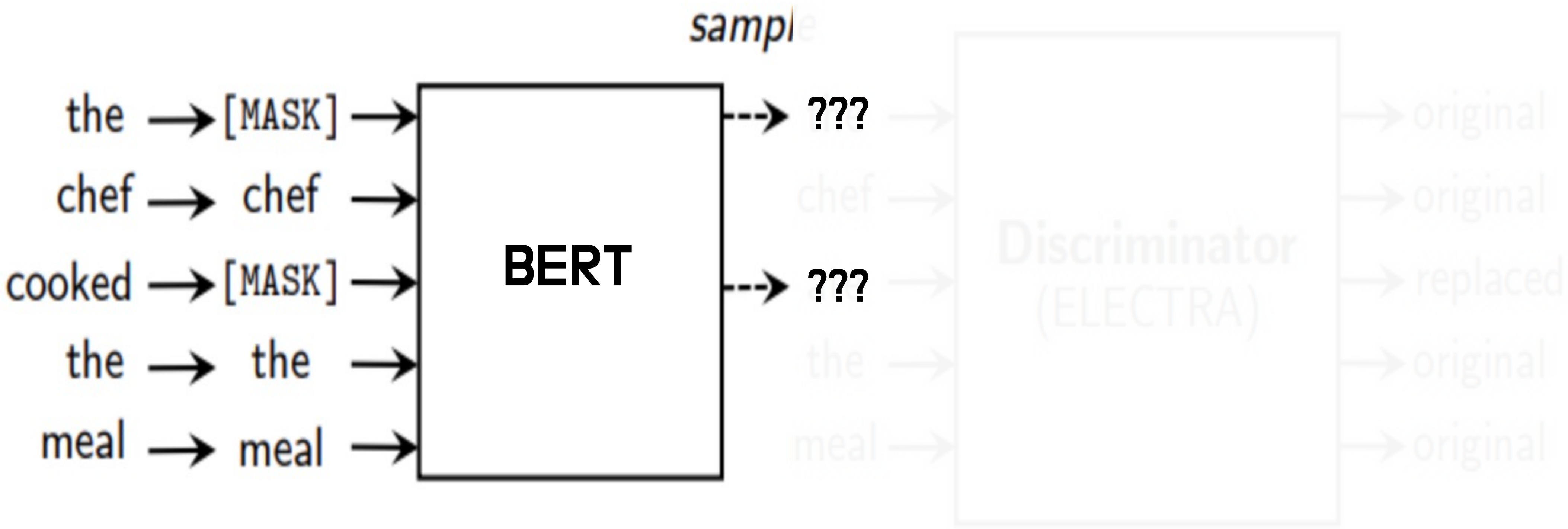


→ **Token Replacements**를 활용한 인코더를 학습하는 **Generator**

(Replaced Token Detection)

Efficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately

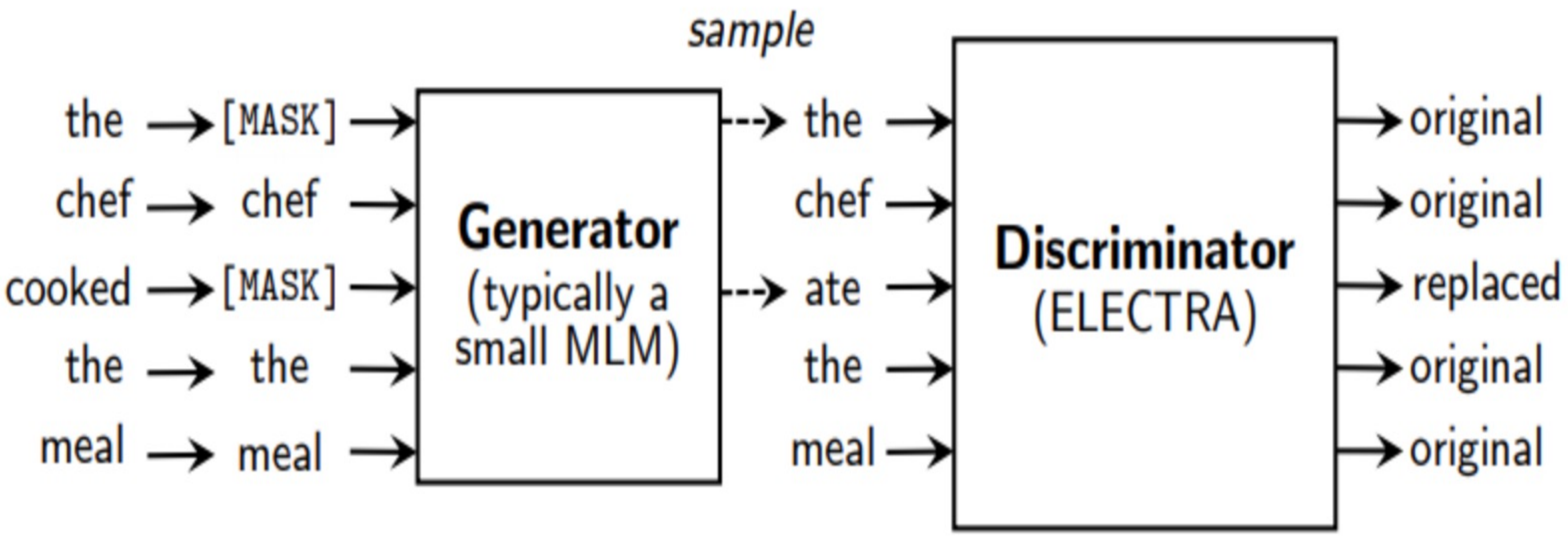
〈BERT〉



→ BERT는 [MASK] 토큰 자체를 예측

→ ELECTRA는 Replaced Token이 original / replaced 인지를 예측

<ELECTRA>



ELECTRA's contribution

- MLM 대신 새로운 pretrain method 제시 (Replaced Token Detection)
- 동일한 사이즈, 데이터, GPU로 BERT보다 성능 향상

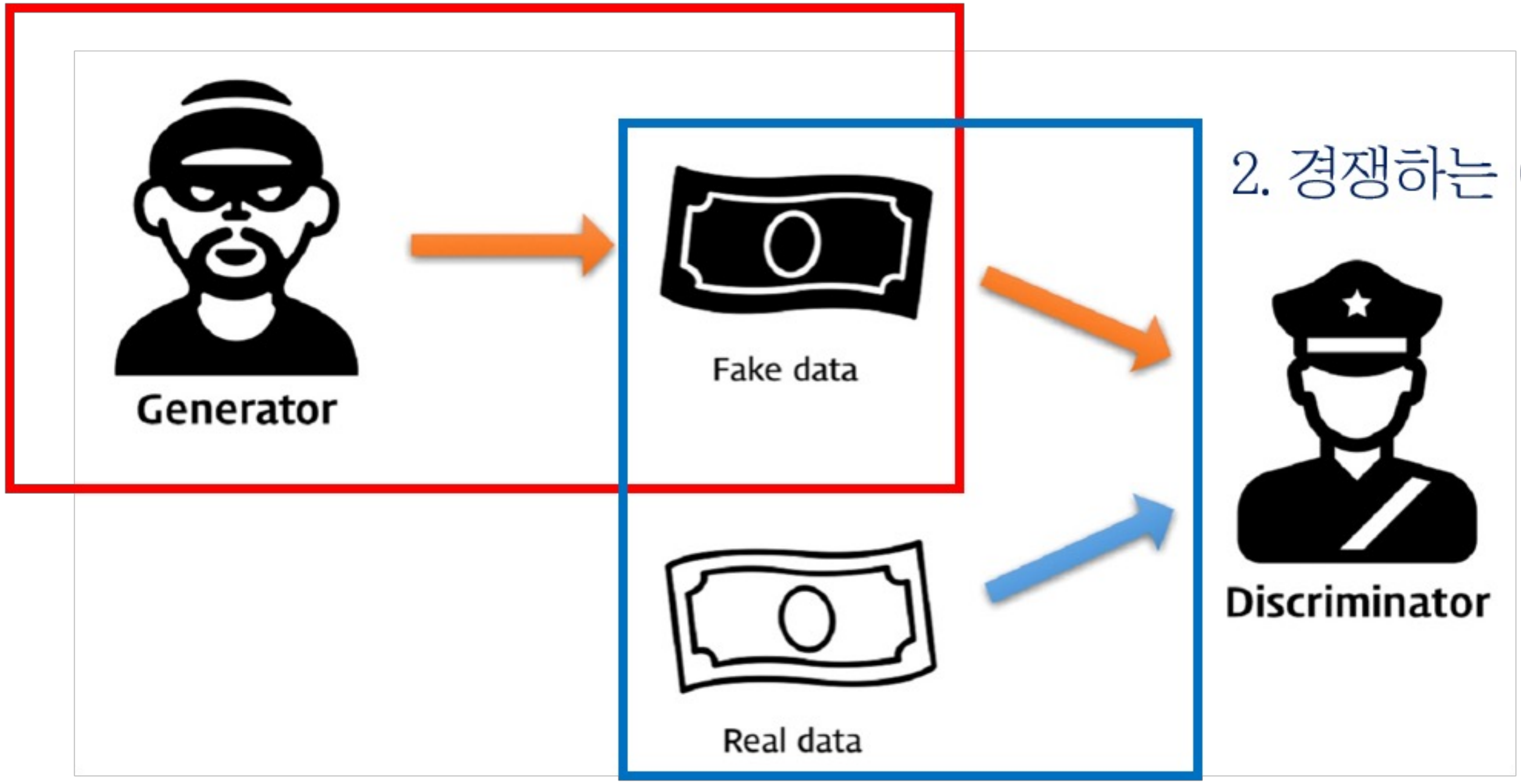
GPT, RoBERTa, XLNet 와 비교해도 훨씬 더 적은 리소스로 비슷한 성능 (혹은 그 이상)

목차

GAN(Generative Adversarial Network)

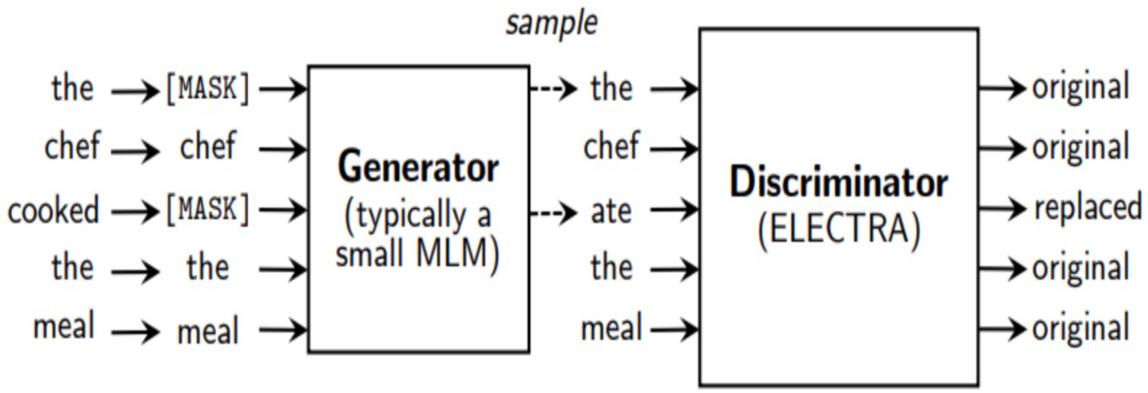
1. 생성하고 (Generative)

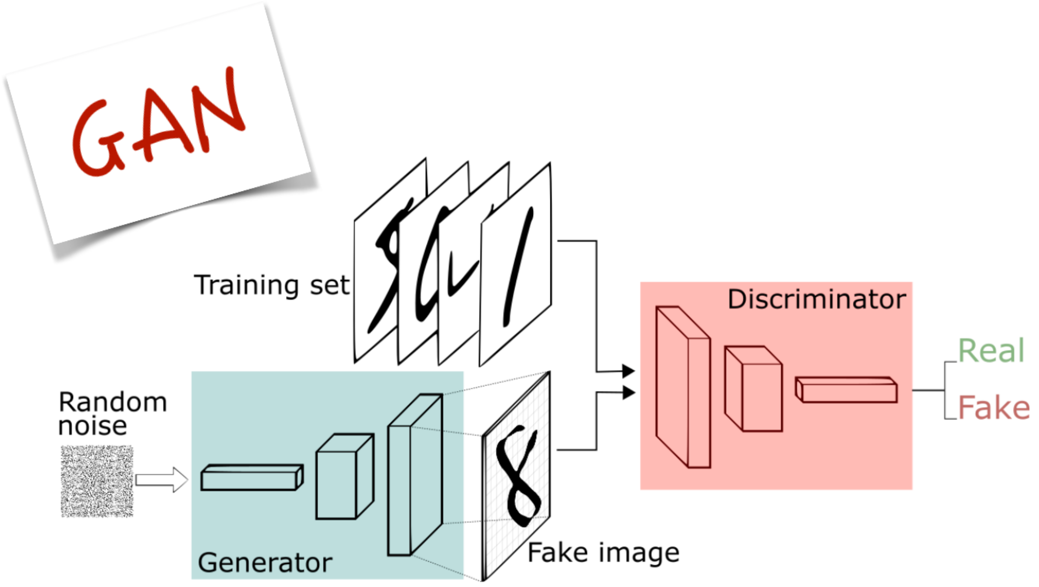
2. 경쟁하는 (Adversarial)



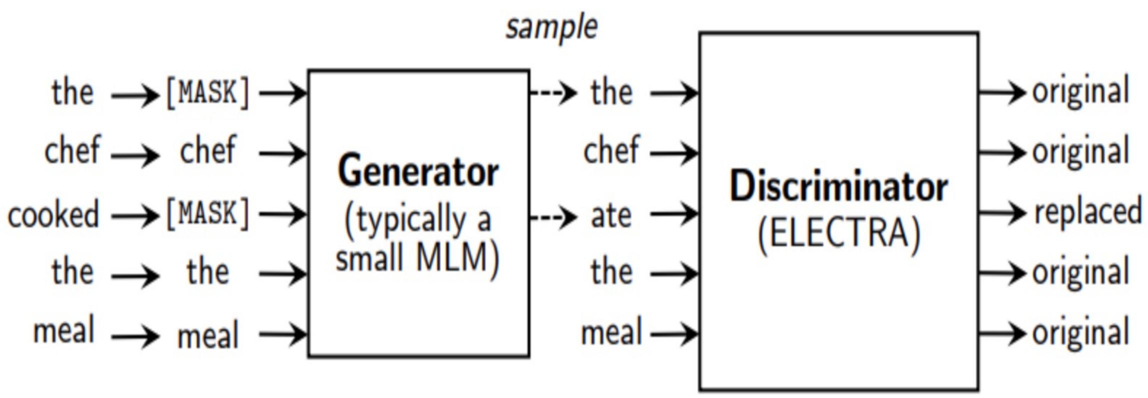
3. 네트워크 (Network)

⟨ELECTRA⟩ vs ⟨GAN⟩





<ELECTRA와 GAN의 차이점>



1. 생성 모델이 원본 토큰을 생성하는 데 성공해내면, 그 토큰은 'fake'가 아닌 'real'으로 간주한다.

2. 생성 모델은 판별 모델을 속이려는 적대적인 방식으로 학습하는 것이 아니라 Maximum Likelihood를 통해 학습

-> 1. 생성 모델이 샘플링한 결과에 대해 back-propagation이 어렵기 때문

-> 2. text에 adversarial training을 적용하기 어렵기 때문

3. GAN에서는 생성 모델에 noise 벡터를 인풋으로 주는 반면, ELECTRA는 그렇지 않다.

목차

Method

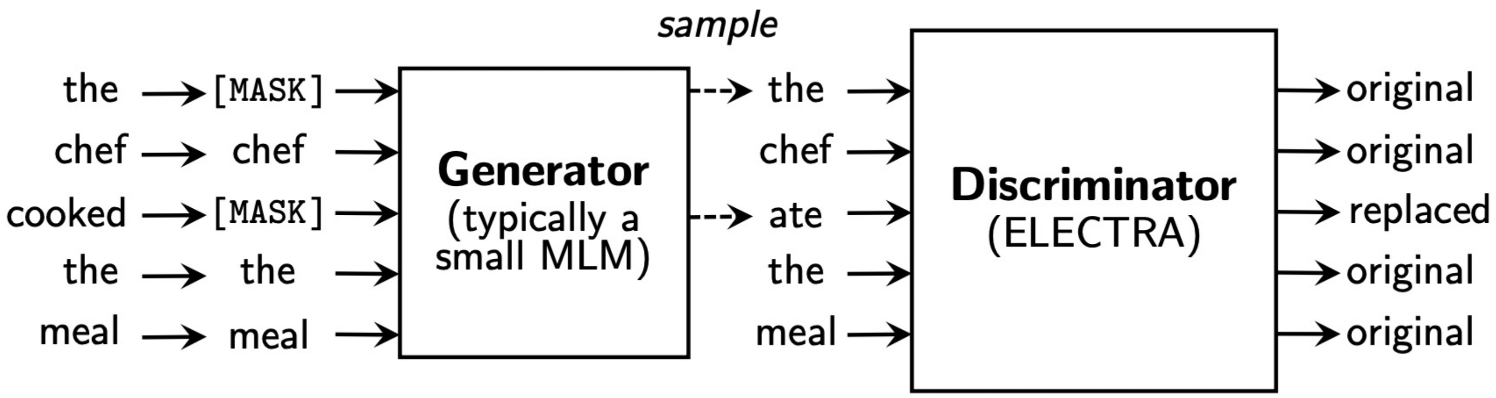
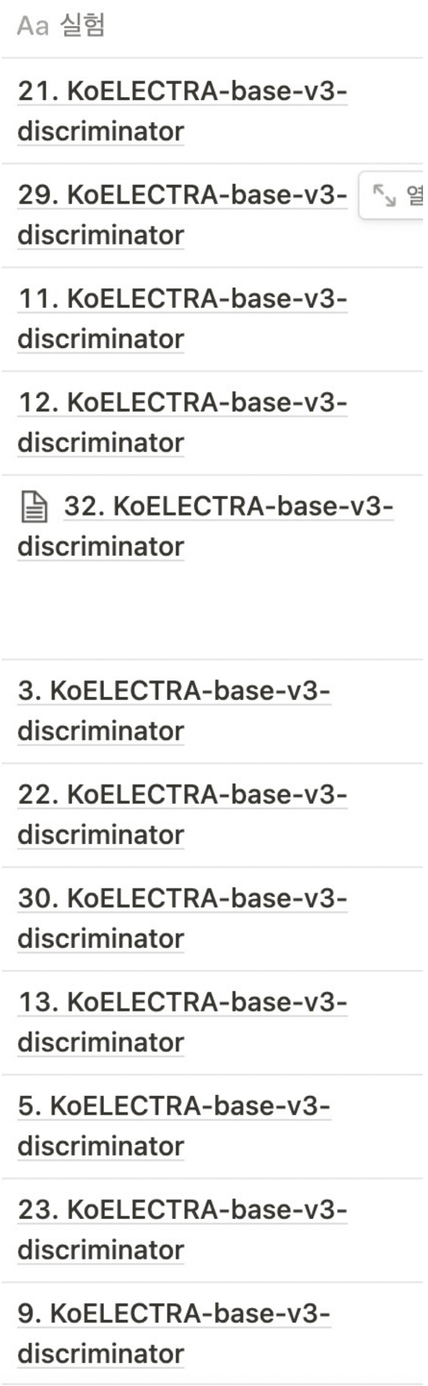
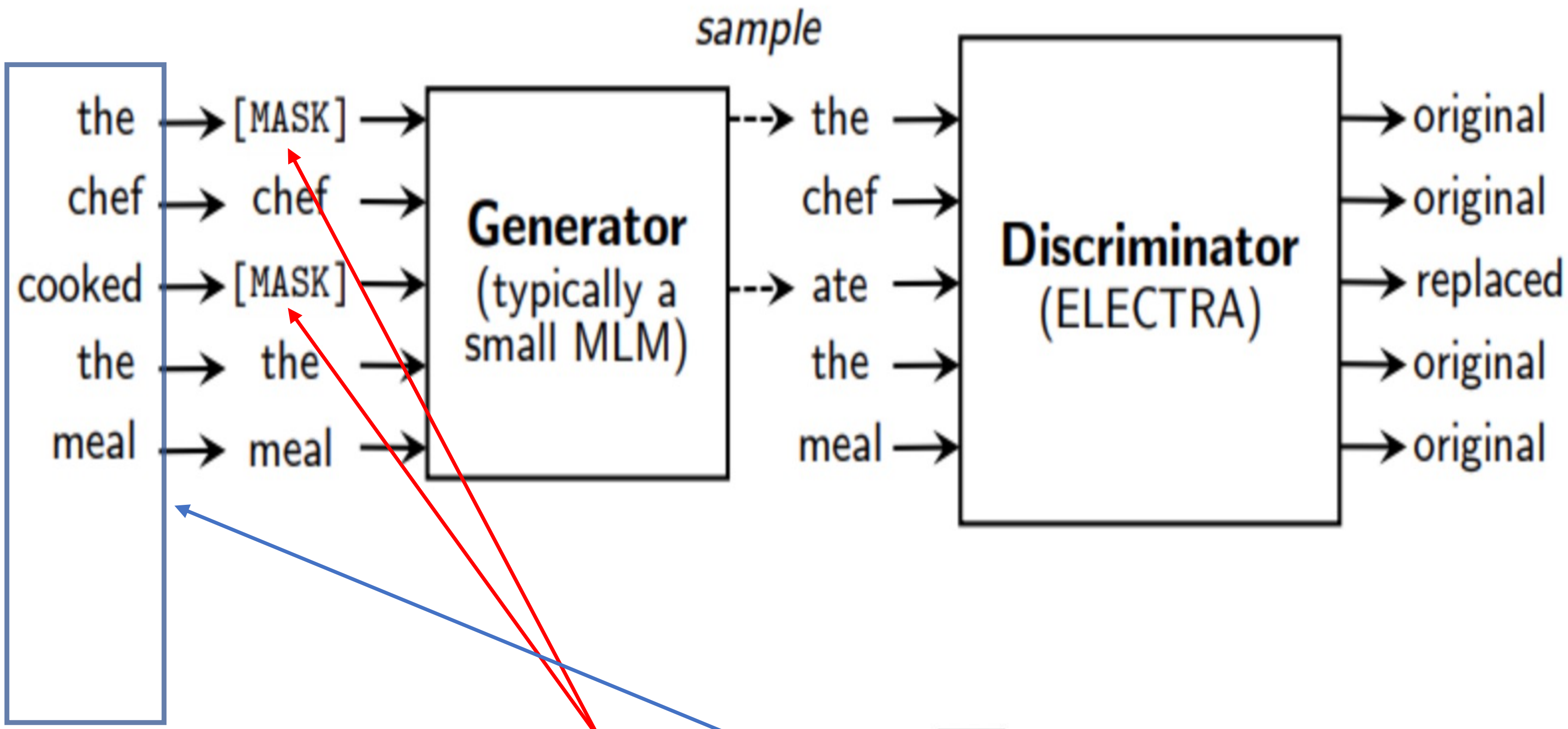


Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator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



Method



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

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

→ Discriminator는 모든 token x에 대해서 original / replaced를 예측

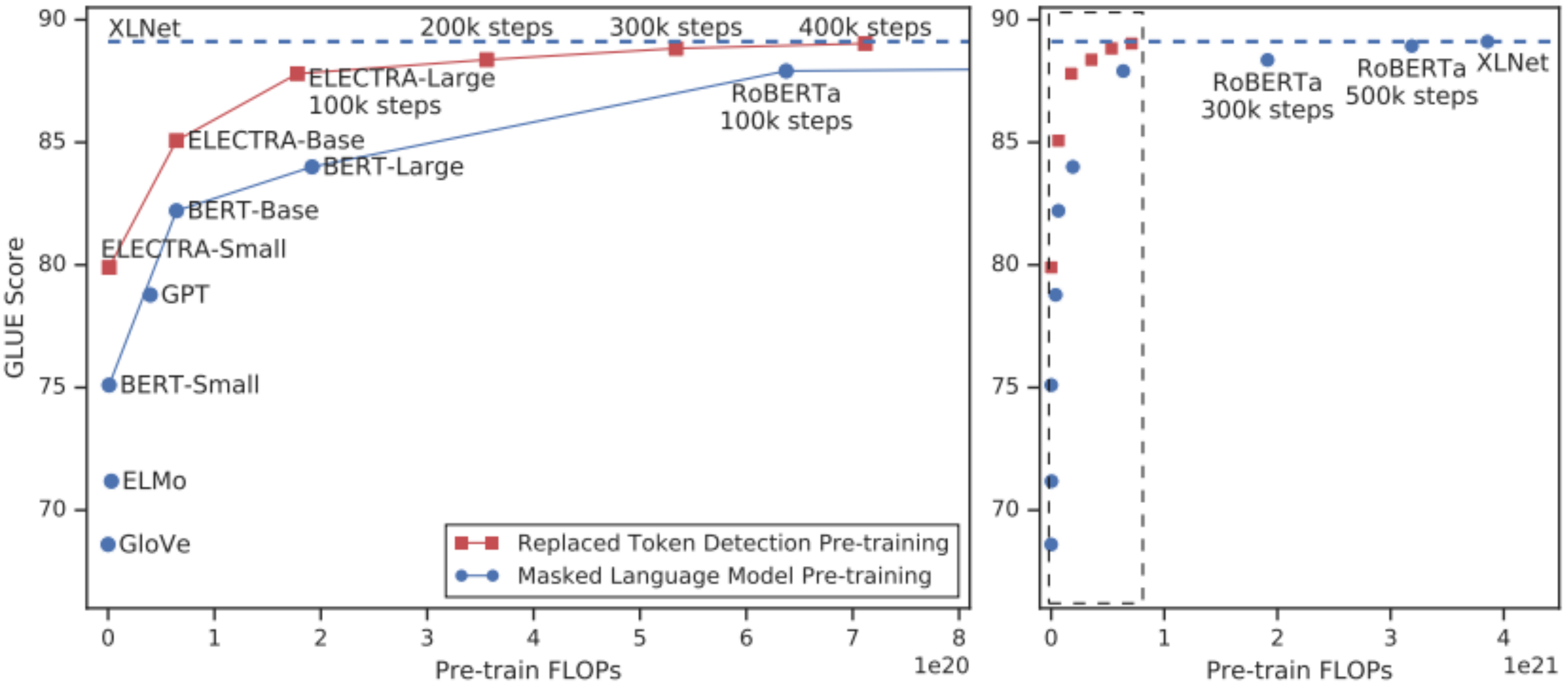
Method

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

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

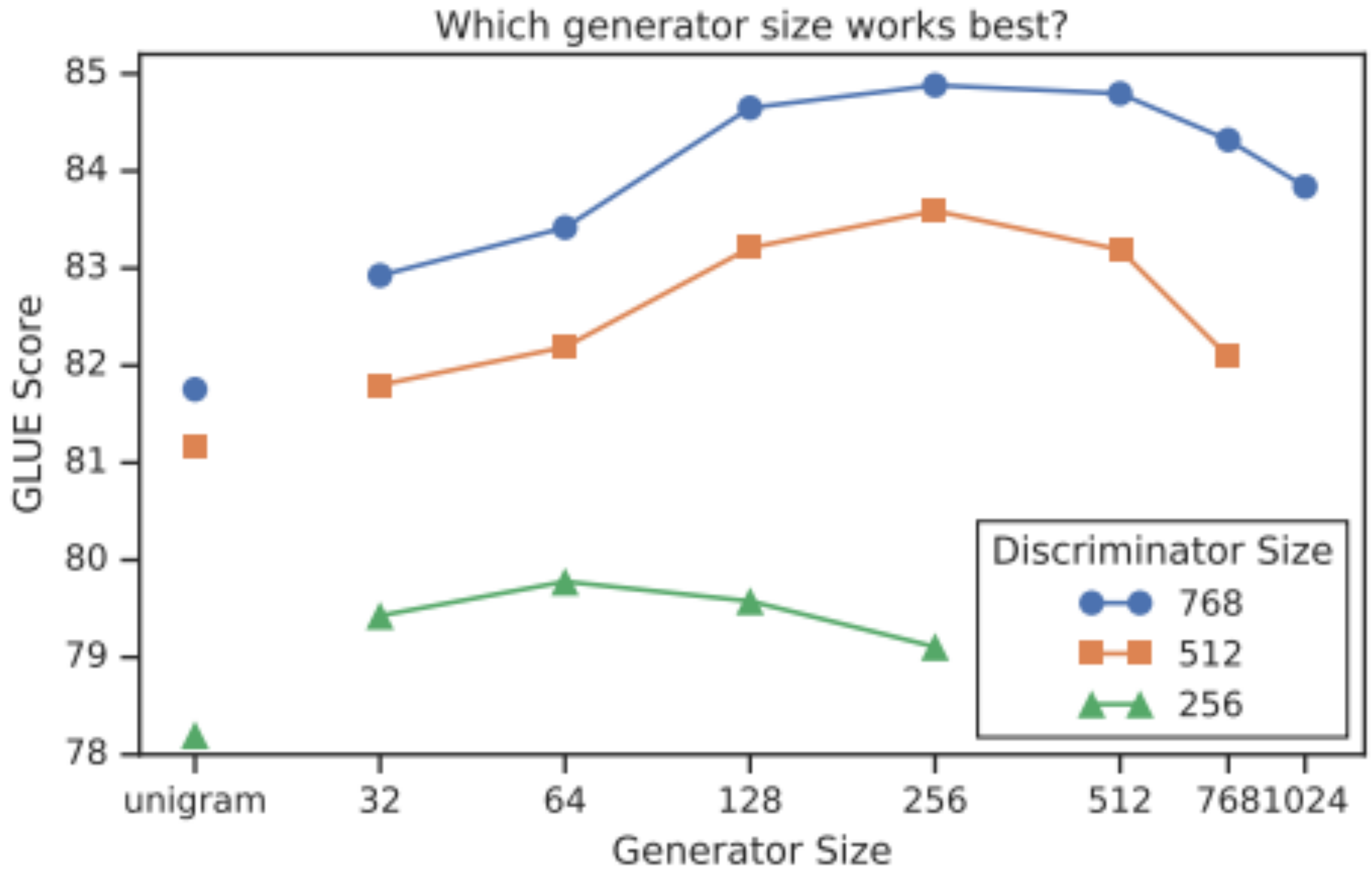
$$\min_{\theta_G, \theta_D} \sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) + \lambda \mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D)$$

[1,10,20,50,100] 중 실험


GLUE Score
Pre-train FLOPs
1e20
1e21
XLNet
200k steps
300k steps
400k steps
ELECTRA-Large
100k steps
ELECTRA-Base
BERT-Large
BERT-Base
ELECTRA-Small
GPT
BERT-Small
ELMo
GloVe
RoBERTa
100k steps
RoBERTa
300k steps
RoBERTa
500k steps
XLNet
Replaced Token Detection Pre-training
Masked Language Model Pre-training

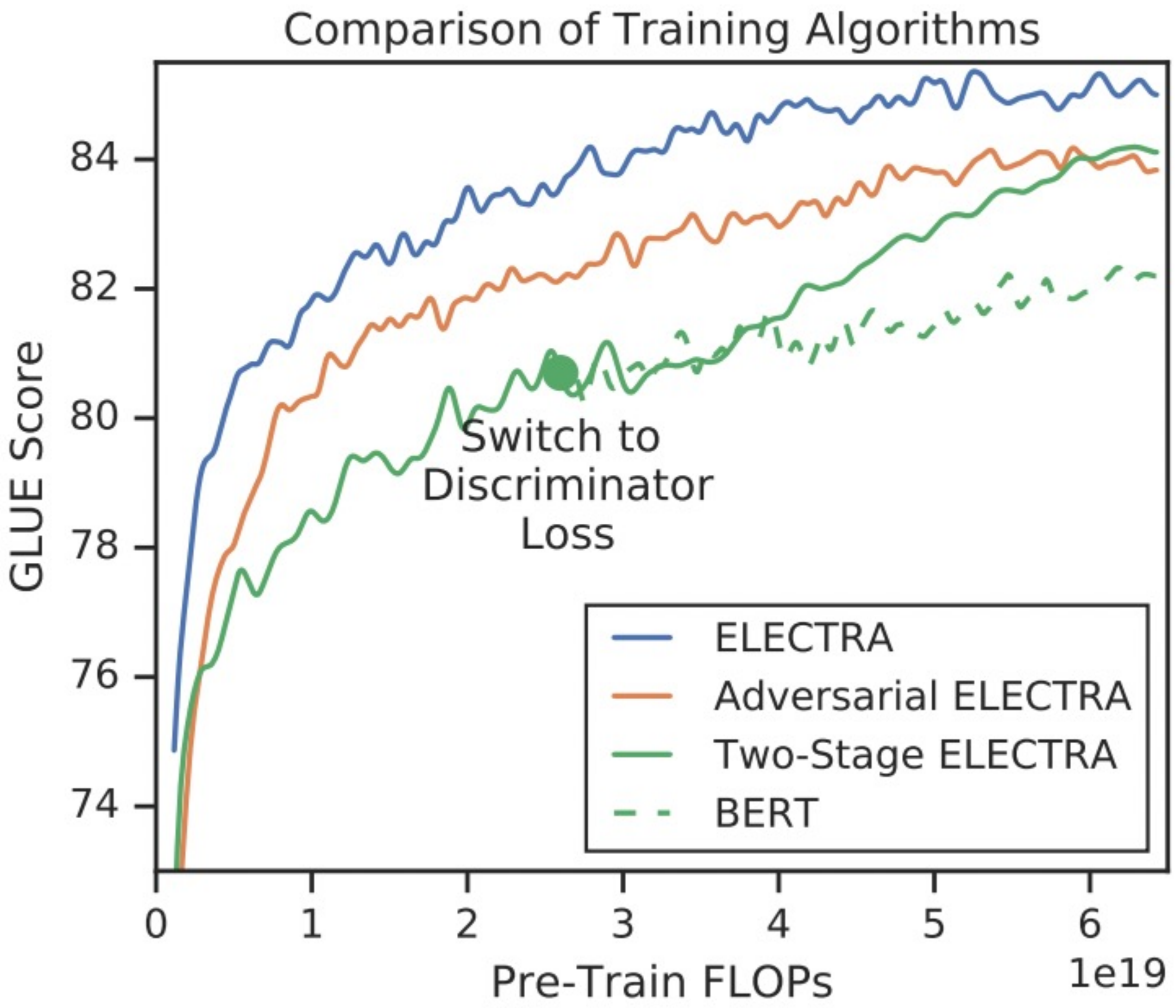
목차

Weight Sharing & Smaller Generators



Generator이 너무 뛰어나면 Discriminator가 판별하기 어려움 → 성능 하락

Training Algorithms



MLM은 과연 비효율적인가?

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

1. All-Tokens MLM : BERT인데, 모든 토큰에 대해 MLM 진행 (원래는 MASK token만)

2. Replace MLM : BERT인데, MASK token 대신 replaced token으로 MLM 진행

3. ELECTRA 15% : ELECTRA인데, loss를 전체가 아닌 마스킹된 15%의 토큰에서 온 것 만을 사용해 계산

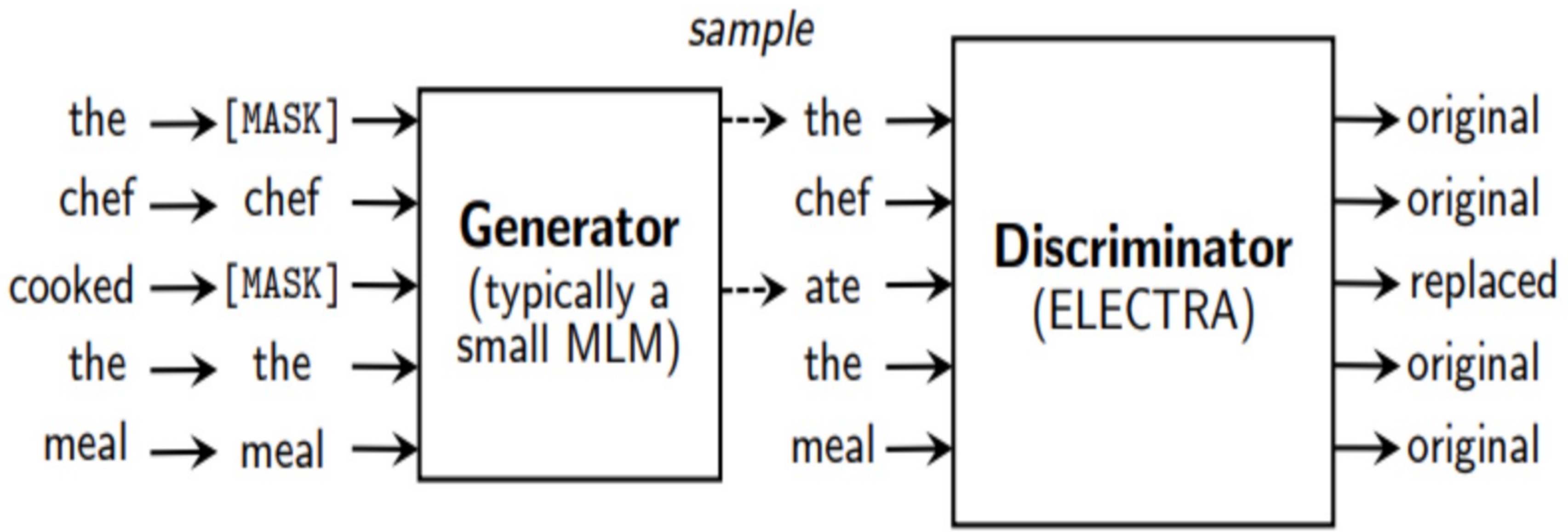
→ 모든 토큰에 대해 loss를 계산하는 것이 효과적이다.

BERT는 MASK 토큰 때문에 성능이 안 좋았을 수 있다.

목차

- Abstract
- Introduction
- Method
- Experiments
- **Conclusion**

Conclusion



- MLM 대신 새로운 pretrain method 제시 (Replaced Token Detection)
- 동일한 사이즈, 데이터, GPU로 BERT보다 성능 향상

GPT, RoBERTa, XLNet 와 비교해도 훨씬 더 적은 리소스로 비슷한 성능 (혹은 그 이상)

감사합니다